THE MANIPULATION OF SCHEMATIC CORRESPONDENCES WITH THE QUANTIFICATION OF UNCERTAINTY IN DATASPACES

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy in the Faculty of Engineering and Physical Sciences

2013

By
Lu Mao
School of Computer Science
## Contents

Abstract ............................................ 7

Declaration ......................................... 8

Copyright ........................................... 9

Acknowledgements .................................... 10

1 Introduction ........................................ 1
   1.1 The Data Integration Problem ................. 2
       1.1.1 Schematic Heterogeneities ............... 2
       1.1.2 Reconciling Schematic Heterogeneities  5
       1.1.3 Functional Decomposition of Dataspaces  7
       1.1.4 Uncertainty in Dataspaces ............... 9
   1.2 Model Management for Dataspaces ............. 10
   1.3 Research Aim, Objectives and Contributions  15
   1.4 Thesis Organisation ............................ 16

2 Related Work ...................................... 18
   2.1 Initialisation of Data Integration Systems .... 19
       2.1.1 Schema Matching ......................... 19
       2.1.2 Schema Mapping .......................... 23
   2.2 User Feedback .................................. 27
   2.3 Uncertainty in Databases Systems and Integration 28
   2.4 Model Management .............................. 35
   2.5 Conclusion .................................... 41

3 View Generation over Schematic Correspondences 43
   3.1 Query Translation with Schema Mapping ......... 45
5.4 User Feedback to Degree of Belief .......................... 146
  5.4.1 The Trustworthiness of User Feedback ................. 149
5.5 Scenario Study: Incremental Update of Degree of Belief .... 154
5.6 Propagation of Uncertainty by Model Management .......... 158
5.7 Conclusion .................................................. 162

6 Conclusion .................................................. 164
  6.1 Summary of Research Contributions ....................... 164
    6.1.1 Schema Mapping and Model Management on Expressive
          Schematic Correspondences .............................. 165
    6.1.2 Quantification and Propagation of Uncertainty in Schematic
          Correspondences ......................................... 166
  6.2 Limitations and Future Research .......................... 167

Bibliography ................................................ 169

Word Count: 52,478
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Examples of schematic heterogeneities</td>
<td>6</td>
</tr>
<tr>
<td>3.1</td>
<td>Heterogeneous data sources</td>
<td>50</td>
</tr>
<tr>
<td>3.2</td>
<td>An example of query unfolding</td>
<td>67</td>
</tr>
<tr>
<td>3.3</td>
<td>Algorithm for generating mappings</td>
<td>69</td>
</tr>
<tr>
<td>3.4</td>
<td>Subroutine for generating GAV mappings</td>
<td>71</td>
</tr>
<tr>
<td>4.1</td>
<td>Algorithm for deriving a reconciled schema</td>
<td>94</td>
</tr>
<tr>
<td>4.2</td>
<td>Subroutine for deriving a reconciled superabstract</td>
<td>95</td>
</tr>
<tr>
<td>4.3</td>
<td>Subroutine for reconciling many-to-many superabstract correspondence</td>
<td>96</td>
</tr>
<tr>
<td>4.4</td>
<td>Subroutine for copying a superabstract to the reconciled schema</td>
<td>98</td>
</tr>
<tr>
<td>4.5</td>
<td>Subroutine for deriving reconciled superrelationship</td>
<td>99</td>
</tr>
<tr>
<td>4.6</td>
<td>Two source schema</td>
<td>102</td>
</tr>
<tr>
<td>4.7</td>
<td>The derived reconciled schema</td>
<td>105</td>
</tr>
<tr>
<td>4.8</td>
<td>The algorithm for composing two sets of schematic correspondences</td>
<td>110</td>
</tr>
<tr>
<td>4.9</td>
<td>The algorithm for composing two superabstract correspondences</td>
<td>111</td>
</tr>
<tr>
<td>4.10</td>
<td>The algorithm for composing two superabstract correspondences (Continued)</td>
<td>112</td>
</tr>
<tr>
<td>4.11</td>
<td>The algorithm for composing two superlexicals correspondences</td>
<td>115</td>
</tr>
<tr>
<td>4.12</td>
<td>The algorithm for eliciting missing constructs</td>
<td>121</td>
</tr>
<tr>
<td>4.13</td>
<td>The algorithm for copying constructs to a schema</td>
<td>122</td>
</tr>
<tr>
<td>5.1</td>
<td>Incremental update of degree of belief</td>
<td>132</td>
</tr>
<tr>
<td>5.2</td>
<td>Procedure for classifying matches into true positives and false positives</td>
<td>136</td>
</tr>
<tr>
<td>5.3</td>
<td>Deriving score distributions for true positive matches and false positive matches</td>
<td>138</td>
</tr>
<tr>
<td>5.4</td>
<td>Degree of belief update with MatchBench data</td>
<td>141</td>
</tr>
</tbody>
</table>
5.5 Score distribution for true positive matches with estimated curve. 141
5.6 Score distribution for false positive matches with estimated curve. 142
5.7 Update of degree of belief with Bayes’s theorem and the estimated functions. 143
5.8 Average errors in $K$ with respect to support. 152
5.9 Degree of belief update for different priors for a fixed, high level of support. 153
5.10 Degree of belief update for a fixed, high prior for different levels of support. 153
5.11 Incrementally assimilating evidence into degree of belief, Scenario 1. 157
5.12 Incrementally assimilating evidence into a degree of belief, Scenario 2. 157
5.13 Incrementally assimilating evidence into a degree of belief, Scenario 3. 158
Abstract

Dataspaces aim to remove upfront cost in the generation of the schema mappings that reconcile schematic heterogeneities, and to incrementally improve the generated mappings based on user feedback. The reconciliation of schematic heterogeneities is a crucial step for translating queries between a mediating schema and data sources. The generation of schema mappings depends on the elicitation of conceptually equivalent schema constructs and information on schematic heterogeneities. Furthermore, many dataspace operations manipulate associations between schemas, for example for generating a global schema to mediate user queries. With a view to minimizing upfront costs associated with understanding the relationships between schemas, many schema matching algorithms and tools have been developed for postulating equivalent schema constructs. However, they derive simple associations between schema constructs, and do not provide rich information on schematic heterogeneities. Without manual refinement, the elicitation of conceptually equivalent schema constructs and schematic heterogeneities may create uncertainties that must be managed.

The schematic correspondences captures a wide range of one-to-one and many-to-many schematic heterogeneities. This thesis investigates the use of schematic correspondences as a central component in a dataspace management system. To support query evaluation in a dataspace in which relationships between schemas are represented using schematic correspondences, we propose a mechanism for automatically generating schema mappings from the schematic correspondences. We then characterise model management operators, which can underpin the bootstrapping and maintenance of dataspaces, over schematic correspondences. To support the management of uncertainty in dataspaces, we propose techniques for quantifying uncertainty in the equivalence of schema constructs from evidence in the form of similarity scores and user feedback, and provide a flexible framework for incrementally updating the uncertainties in the light of new evidence.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s policy on presentation of Theses
Acknowledgements

I am grateful to my supervisors, Professor Norman Paton and Dr. Alvaro Fernandes for their guideline on a broad perspective for my research in the past four years.

Meanwhile, I would like to thank my parents for their financial support, and grandparents in China for the enduring encouragement they give to me in the past four years.
Chapter 1

Introduction

Dataspaces [HFM06] aim to provide unified data access services over a set of heterogeneous data sources with minimal up-front cost. Users pose queries on an unified interface, which is answered by data from a set of data sources. They leverage user feedback [BPE+10] to incrementally improve the quality of service. Data integration systems [Len02], still a popular research theme, were proposed more than twenty years ago and have aimed to fully resolve heterogeneities at the initial design stage so that the quality of the query service is at a high level from the start. However, this approach to quality is based on manual intervention to achieve the goal of total resolution of heterogeneities and is, therefore, costly. To reflect the difference between the two approaches, the idea of dataspaces is also referred to as pay-as-you-go data integration [HFM06, SDH08]. In other words, we replace as much manual intervention with automated technologies as we can, and we pay later, in the form of user feedback, to the extent that is required to obtain satisfactory quality. Research outcomes of data integration [FHH+09, BMC06] have been incorporated into commercial software for supporting business information integration [MLM+01, BBF+10] where the quality of data access services play a vital role. The most prominent application domains for dataspaces are personal information management [DSB09] and web-scale data integration [MCD+07].

Dataspaces provide users with data access services that exhibit reduced quality at the start. They must account for and mitigate the impact of the inevitably low quality at the bootstrapping stage, and this is the general problem domain in which the research contributions described in this dissertation are situated.
CHAPTER 1. INTRODUCTION

The next section briefly introduces the core operational components of dataspaces. We will illustrate with a simple example how those components cooperate to realise with the vision of dataspaces [HFM06]. Each functional component has a specific role at the different stages of the dataspaces life-cycle proposed by Hedeler et al. [HBF+09, HBM+10]. We structure the next section according to such stages, viz. bootstrapping, usage and improvement. We then present the research issues and challenges tackled in this thesis and introduce our approach to solving them. Finally, we describe the aim, objectives and contributions of this thesis.

1.1 The Data Integration Problem

1.1.1 Schematic Heterogeneities

One of the most crucial issues to be addressed for providing unified data access services is to reconcile schematic heterogeneities between data sources. For example, assume that there is a global schema for describing to users the concepts can be queried and for mediating users’ queries to data sources. Assume that it contains two relations:

CREATE TABLE taughtStudent
(
    sid int,
    name varchar(30),
    grade varchar(1)
)

CREATE TABLE researchStudent
(
    sid int,
    name varchar(30),
    grade varchar(1),
    research varchar(100)
)

Assume that one of the data sources, $S$, is described with the schema:
CREATE TABLE Student
(
    sid int,
    name varchar(30),
    grade varchar(1),
    research varchar(100),
    type varchar(30)
)

Both the global and source schema describe information on the same concept, but with different schematic representations, i.e. the relations in the global schema can be construed as partitioning instances of students into taught students and research students, whilst, in the source, both types of students are represented by a single relation.

A user may pose the following query \( q \) on the global schema:

\[
\text{SELECT } * \text{ FROM researchStudent WHERE research = 'mathematics'}
\]

To obtain information on postgraduate students doing research in mathematics. The aim is to answer \( q \) with data stored in the sources. For the query \( q \), we can retrieve the same number of tuples from the source schema \( S \), which denoting students whose \( type \) is research student and where \( research \) is equal to mathematics. This can be expressed as the following query \( q' \) on the source \( S \):

\[
\text{SELECT } * \text{ FROM Student WHERE type = 'researchStudent' AND research = 'mathematics'}
\]

One expects the query, \( q' \), that should be obtained by translation from from \( q \). To reach this point, two conventional tasks are performed [Len02]:

- **Schema matching**: schema constructs from different schemas that describe conceptually equivalent concepts, or attributes of a concept, need to be associated. Research contributions from data integration have delivered several tools [DR02, BMC06, MRB03] or algorithms [RB01] for postulating the equivalence of constructs based on similarity comparison of features (such as the name or type) of schema constructs and of instance data.
• **Schema mapping:** the associations between equivalent schema constructs are then taken as input to postulate mappings, in the form of executable expressions, between the corresponding schema. A schema mapping specifies how to populate constructs of one schema with tuples from another schema. In the above example, a mapping can specify the relation $researchStudent$ in the global schema as a view over the source relation $Student$:

```
CREATE VIEW researchStudentLocal AS{
    SELECT *
    FROM Student
    WHERE type = 'research'
}
```

so that the query $q$ on the global schema can be translated to a query on the data source by unfolding the FROM clause with the view:

```
SELECT * FROM AS researchStudent{
    SELECT *
    FROM Student
    WHERE type = 'research'
} WHERE research = 'mathematics'
```

This query is equivalent to the desired query $q'$ against the source $S$.

The generated schema mappings aim to reconcile schematic heterogeneities because the translated query must not retrieve data that is inconsistent with the global schema. For example, if the attributes $sid$ in $researchStudent$ and $sid$ in the source relation $Student$ have different types, the lack of reconciliation would generate an error when populating $researchStudent$ with values drawn from $Student$. Moreover, if the mapping did not reconcile the fact that $researchStudent$ represents a subset only of $Student$, the query $q$ would return tuples denoting taught students.

In order to reconcile schematic heterogeneities, the generation of schema mappings must be well informed. In the initialisation stage of data integration systems, similarity associations derived from schema matching that associate equivalent schema constructs are used as hints for users or domain experts to specify mappings that reconcile schematic heterogeneities between associated constructs.
Dataspaces aim to remove this upfront cost by automating the generation of mappings, a task sometimes referred to as bootstrapping [HFM06]. Although the quality of such schema mappings is likely to be low due to them being derived automatically, user feedback can be solicited from users of the data access service [BPE+10, CVDN09, TJM+08, MSD08, ACMT08, HRO06]. For example, if a dataspace system does not know how to select tuples from the *Student* relation for populating the *researchStudent* relation, it can apply best efforts to postulate a set of mappings. For example, one such mapping might be:

```
CREATE VIEW researchStudent AS
    SELECT sid, name, grade, research
    FROM Student
```

in other words, the mapping simply specifies that all tuples from *Student* can be used to populate *researchStudent*. With this mapping all tuples from *Student* will be returned by a query posed on *researchStudent*. When a user observes the returned tuples (a mix of research students and taught students), he or she can indicate which tuples are correctly returned and which are not. Based on this user feedback, the mapping can be incrementally refined [BPE+10].

We have motivated the importance of reconciling schematic heterogeneities to providing unified data access service over heterogeneous data sources, and of incorporating user feedback for overcoming the lost quality caused by the removal of upfront cost in deriving schema mappings. In the following sections, we elaborate on the challenges involved in realising the vision of dataspaces, and on the aims of the research reported in this dissertation.

### 1.1.2 Reconciling Schematic Heterogeneities

There can exist many types of schematic heterogeneities between schemas. We have seen an example in the previous section. Figure 1.1 shows other representations of the same concept.

The relation *Student* in the source schema $S_1$ has attributes *address* and *country* which are missing in the relations *researchStudent* and *taughtStudent* of the global schema. On the other hand, the attribute *grade* in *taughtStudent* is missing in *Student*.

In $S_2$ and $S_3$, the name of a student is represented by two attributes *firstname* and *surname*, whereas in *researchStudent* and *taughtStudent*, it is represented
Figure 1.1: Examples of schematic heterogeneities
by a single attribute name.

In S4, information on students is partitioned into local students and overseas students. Meanwhile, in S5, information on the accommodation of a student is stored separately in the relation Accommodation.

Schema mapping generation needs to operate on an informative model that can express a wide range of schematic heterogeneities. For example, it is important to specify how to combine data values of firstname and surname in order to populate the attribute name, and how to select tuples of Student to populate researchStudent. Unless information that indicate how to reconcile schematic heterogeneities is available, one cannot hope to derive schema mappings for translating queries for data without error.

Associations derived from automated schema matching tools or algorithms [RB01, MRB03, MGRM02, MCD+07] are usually expressed as a tuple $<c_1, c_2, s>$ postulating that constructs $c_1$ and $c_2$ are similar to a level of $s$. Thus, they can express one-to-one correspondences e.g. between pairs of attributes or pairs of relations. However, they cannot express e.g., how to combine data values of attributes firstname and surname to populate the attribute name. Several proposals have been made for richer models of schematic correspondences [KQCJ07, CVDN09] that can express more complex associations between constructs, e.g. using the string concatenation function for combining data values as exemplified above. However, they fall short of expressing how tuples from one relation are selected to populate other relations, or how tuples from several relations are combined to populate other relations. Kim et al. have proposed a classification of a wide range of schematic heterogeneities. In this dissertation, we propose a new model of schematic correspondence that represents such heterogeneities to support automated schema mapping generation and other dataspace operations.

1.1.3 Functional Decomposition of Dataspaces

It is useful to generalise the main tasks that a dataspace management system is responsible for into a set of generic operators (as is proposed in the Dataspace Project [MBHR05, HBM+10]) so that interactions within dataspace management can be expressed as a sequence of operator invocations. The functionalities of some operators used at the bootstrapping and maintenance stages of dataspaces management are described in the following list:
• **Match:** is used to elicit associations between schema constructs.

• **Merge:** is used to derive a reconciled schema from two input schemas.

• **Schema Mapping:** is used to generate schema mappings over a set of associations between constructs derived by *Match*.

• **Compose:** is used to derive associations between constructs based on existing associations.

• **Diff:** is used to elicit missing schema constructs between two schemas.

These operators are structured as algorithmic programmes to achieve shared goals [BM07]. We now illustrate, using simple scenarios, how the application of a selection of such operators can underpin the decomposition of dataspaces functionalities.

Consider the UK university system. Assume firstly that every UK university maintains information about their students stored in their local database. Assume further that each data source of Figure 1.1 represents students in a different university. Assume now that, the Scottish government would like to study the enrolment and performance of overseas students in Scottish universities. Assume finally that $S_1$, $S_2$ and $S_3$ of Figure 1.1 represent three different universities in Scotland, and that the Scottish government incorporates existing solutions to support their data analysis proposal. Firstly, schema matching tools and algorithms are deployed to derive associations between pairs of source schema. With algorithms proposed in the *schema merging* literature [PB03, BDK92, QK07, PB08, PB09], a global schema, as well as associations between the global schema and each source schema, can be automatically derived as a query interface. Finally, based on the derived associations, *schema mappings* can be generated between each data source and the global schema. Given the derived mappings and the global schema, queries can be answered. Meanwhile, feedback can be solicited on results retrieved by the queries for refining the generated mappings.

Assume now that, the UK Department of Education would like to conduct similar evaluation at the national level. It applies *schema merging* to automatically derive a mediating schema directly upon the mediating schemas used by individual local governments. Alternatively, it can select a set of universities
across the country and invoke the same functional components: *schematic association derivation* for every pair of data sources, *global schema derivation*, and *schema mapping generation* to bootstrap the unified data access service. Furthermore, while using the dataspaces, feedback can be given to improve the data access services.

If the database schemas maintained by individual universities are changed (if, for example, a relation is deleted or inserted, or attributes of some relations are deleted or added), we need to propagate such changes to the global schema and update any generated mappings that have associations with the changed schemas. This problem is referred to as schema evolution [Ber03]. To address this problem, we need operations such as *elicit the difference* between schemas to identify their changes, and *compose new associations* between the new schema and all the schemas associated to the old schema. Sometimes, a change on the global schema has to be propagated to all the mappings that are mapped to it [Len02, Hal01]

All the functional operators mentioned above derive or manipulate schemas and associations. Some of them were originally proposed in the model management literature [Ber03], others were proposed specially for dataspaces [HBM+10]. As we have observed in the previous sections, the goal of providing unified data access service, makes it crucial to reconcile schematic heterogeneities for query translation. Therefore, it is important for reusable functional operators to be characterised over schematic correspondences and thereby underpin the generation of schema mappings of greater expressiveness than usual.

### 1.1.4 Uncertainty in Dataspaces

Associations between conceptually equivalent schema constructs are postulated based on the results of similarity comparisons between schematic properties (such as name or type of schema constructs) or between data instances [RB01]. Uncertainty in eliciting conceptually equivalent constructs is likely to exist, specially in the situation where the supporting resources (e.g., dictionaries for comparing construct names or data instances) or similarity comparison mechanisms are insufficient. Without manual refinement, dataspaces must account for and mitigate such uncertainty. One approach is to retain all the plausible associations postulated by algorithms or tools [DHY09a, SDH08]. For example, schema matching algorithms, in accordance to the type of elicitation methods they apply, may yield
different similarity scores in associating relations to the Student relation:

\[ (<\text{Student}, \text{researchStudent}, 0.7>) \]
\[ (<\text{Student}, \text{taughtStudent}, 0.6>) \]
\[ (<\text{Student}, \text{masterStudent}, 0.3>) \]

These associations are then taken as input to the generation of schema mappings by automated tools. User queries posed on Student should be answered by tuples retrieved by mappings derived from plausible associations. Dong et al. [DHY09a] proposed an approach for ranking query results based on the degree of confidence in the equivalence of a pair of relations. In their approach, the degree of confidence is represented as probability. However, no strategies were discussed on how and where the uncertainty is derived from.

The derived degree of confidence should be able to be incrementally updated in accordance to newly available information that is relevant to the judgement of equivalence between schema constructs. For example, another schema matching algorithms may later on return

\[ (<\text{Student}, \text{researchStudent}, 0.2>) \]

This would call into question the previous conclusion that researchStudent and Student are most likely equivalent.

Likewise, user feedback on query results may be used to compute the proportion of tuples are correctly returned. Given this kind of information, the question is then what adjustment to the previous degree of confidence should be made.

Finally, when correspondences derived from these associations are manipulated by model management operators, new correspondences might be derived by the operators, which are likely to be used for schema mapping generation. The question then arises as to what degree of confidence one should assign to derived correspondences and mappings.

### 1.2 Model Management for Dataspaces

In the previous section, we pointed out some of the challenges involved in realising the vision of dataspaces [HFM06]. Before discussing the aim and objectives of
this research in the next section, we firstly present an overview of our approach to addressing these challenges.

Dataspaces aim to eliminate the upfront cost resulting from human involvement in the generation of schema mappings. We have observed that schema mappings are executable expressions aimed at reconciling schematic heterogeneities, that specify how to populate constructs of one schema with data conforming to another schema. Schema matching is a crucial step towards the generation of mappings. Tools and algorithms have been developed for postulating similarity between schema constructs [RB01], however, most of existing schema matching algorithms [RB01] return simple one-to-one associations, which are insufficient to directly support the reconciliation of many-to-many schematic heterogeneities, as exemplified in previous sections.

In an early study conducted by Kim et al. [KCGS95], a wide range of reconcilable schematic heterogeneities were classified and some of the corresponding resolution techniques were studied. This work aimed to provide guidance for the manual mapping generation. Kim et al.’s classification includes:

- **One-to-one entity-to-entity name conflict.** Two entities in different schemas are conceptually equivalent (in other words, they represent the same concept), but are assigned with different names. Likewise, two entities are conceptually different with the same name.

- **One-to-one entity-to-entity missing attributes.** Two entities are conceptually equivalent, but some attributes in one are missing in the other one.

- **One-to-one entity-to-entity constraint conflicts.** Two entities are conceptually equivalent, but has different attributes as their keys.

- **One-to-one entity-to-entity entity inclusion.** The extent of one entity is included in the extent of another conceptually equivalent entity.

- **Many-to-many entities-to-entities conflicts.** The concept is represented by different numbers of entities in different schemas.

- **One-to-one attribute-to-attribute name conflicts.** An attribute of a concept is represented by two attributes in different schemas with different or the same name.
• **One-to-one attribute-to-attribute data type conflicts.** An attribute of a concept is represented by two attributes in different schemas with different data types.

• **Many-to-many attributes-to-attributes conflicts.** An attribute of a concept is represented by two sets of attributes in different schemas.

• **Entity-to-attribute conflicts.** A concept is represented by entities in one side, but is represented by attributes of the other side. This can be construed as a special case of attribute-to-attribute conflicts, since every entity constitutes of one or many attributes.

Models of schematic correspondence proposed in existing research work are short for expressing information on all the schematic heterogeneities classified by Kim et al. The model for schematic correspondences proposed in the GeRoMeSuite system [KQCJ07, KQL07, QK07] can express many-to-one schematic heterogeneities. The correspondences indicate the function needed for combining tuples or data values of the many side to populate the one side. However, the correspondences cannot express how tuples should be combined. If the many side is construed as a vertical partitioning of the one side, we should use join and a join needs to be informed with join conditions (i.e. a conjunction or disjunction of binary comparisons between a set of attribute values of the tuples to be combined).

Apart from vertical partitioning, the many side may also be construed as the horizontal partitioning of the one side (e.g. researchStudent and taughtStudent to Student). In this case, there is a need to specify which predicates define which partitions. Other proposals for a schematic correspondence model (e.g., [MRMM05, MRB03, FHH+09, ABBG09c, DHY09a, PB03]) come short of expressing Kim et al.’s schematic heterogeneities (especially many-to-many heterogeneities). Based on this observation, we further extended the many-to-many entity-to-entity heterogeneities to more specific types, i.e. vertical partitioning to vertical partitioning heterogeneities, horizontal partitioning to horizontal partitioning heterogeneities and vertical partitioning to horizontal partitioning heterogeneities. One aim of this research is to propose a model that can capture the extended Kim et al.’s schematic heterogeneities. In particular, the model should express information needed for the reconciliation of such schematic heterogeneities. Notice that Kim et al. also classified heterogeneities at the extent level, which are not discussed in this thesis, since the scope of this research is on
expressing and reconciling schematic heterogeneities.

As mentioned in Section 1.1.3, several existing research work have studied the incorporation of model management to underpin the functional decomposition of systems that aim at integrating heterogeneous data sources [MBHR05, GBM08, BM07, HBM+10]. Model management operators were originally characterised over simple associations returned by schema matching. In order for them to underpin dataspaces, they must be redesigned to operate on a richer model of schematic correspondences so that the correspondences derived from the operators can be directly taken as input for automated generation of schema mappings [BM07, HBM+10]. The schematic correspondence model proposed in this research captures Kim et al.’s schematic heterogeneities. We have studied the characterisation of model management operators over such a richer correspondence model.

A schematic correspondence describes the heterogeneity that holds between two, or two sets of, schema constructs that are postulated to be conceptually equivalent. The equivalence relationship between schema constructs is postulated by schema matching [HBM+10] based on evidence from similarity comparison of schematic representations or of data instances [RB01]. Most automated tools for schema mapping generation provide assistance for users or experts to refine the results of schema matching [FHH+09, BMC06, ATV08, YMHF01, BMP+08, CT06]. In the case of dataspaces, since manual refinement is avoided (because it creates an upfront cost), it is hard to be certain as to the degree to which schema matching tools or algorithms plausibly associate conceptually equivalent constructs. Despite the existence of uncertainty, dataspaces aim to provide users with best-effort unified data access services. In this context, dataspaces need to account for and mitigate uncertainty that may result from postulations by matchers of conceptually equivalent constructs.

We anticipate that quantified uncertainty can inform the ranking of query results retrieved using schema mappings that were derived from uncertain associations between schema constructs. In recent years, some research has focused on query evaluation over uncertain correspondences [DHY09a, GMS09, TCY92, SDH08]. In such work, query evaluation mechanisms have been proposed and their complexity has been studied. Previous work on probabilistic databases (in other words, on uncertain data values) has also proposed mechanisms for query evaluation and result ranking [Koc08, RDS07, DS07a, JXW+08]. Uncertainty is
mostly represented as probability [DHY09a, GMSS09, TCY92, SDH08, Koc08, RDS07, DS07a, JXW+08, IJ84, GT06]. To the best of our knowledge, most of the existing research contributions on uncertainty rely on the assumption that uncertainty has been previously quantified into probability from certain sources of evidence. In other words, it is rare for authors to provide an explicit, detailed, and precise account of how uncertainty is quantified to give rise to a probability value. In dataspaces, judgments as to the equivalence of two constructs can be arrived at from the results of similarity comparison returned by automated schema matching algorithms and from feedback given from users. Therefore, one of the aims of our research is to provide an account as to how similarity scores and user feedback give rise to probability values.

In this research, we quantify the uncertainty in the hypothesis that two constructs are equivalent is true as a subjective degree of belief. A probability value, interpreted as a measure of subjective degree of belief, represents one’s assessment of the likelihood of the outcome of a hypothesis based on the available evidence. The more evidence we have, the more certain we are. In other words, we continuously learn from experience to reduce the uncertainty on the outcome of the hypothesis. In reasoning with evidence, Bayes’ theorem is the theoretical foundation for updating belief in a hypothesis from a given prior belief to derive a posterior belief [Sha07, J92]. The derived posterior belief can then become a prior belief to be updated again, using Bayes’ theorem with the next available evidence. In dataspaces, results of similarity comparison and user feedback may become available in different orders. Thus, Bayes’ theorem is an appropriate foundation for incrementally assimilating evidence into degrees of belief in the context of dataspaces.

Apart from schema matching, other model management operators also postulate associations between schema constructs. Correspondence composition [Ber03, BM07] postulates correspondences between schema constructs from existing correspondences rather than from scores returned by similarity comparison algorithms. The merge of two schemas yields a unified schema as well as the correspondences that associate constructs of the output schema and constructs of the input schemas. The output correspondences can be taken as input to generate schema mappings. Thus, it is useful that model management operators can propagate uncertainty. In this research, we study the propagation of uncertainty as to the equivalence of constructs guaranteed by the application of model management
1.3 Research Aim, Objectives and Contributions

The aim of this research is to develop automated mechanisms for manipulating schematic correspondences, which is complemented with a proposed framework for the quantification and propagation of uncertainty in dataspaces. To achieve this aim, the following objectives were set out:

1. We firstly designed and implemented mechanisms for the manipulation of schematic correspondences and the generation of schema mappings based on defining high level model of schematic correspondences which expresses heterogeneities classified by Kim et al. [KCGS95]. The application of developed mechanism was demonstrated with scenario studies from the bootstrapping and maintenance of dataspaces.

2. We then developed a Bayesian framework for incrementally updating uncertainty in the equivalence of schema constructs based on defining functions to quantify uncertainty from evidence in the form of similarity scores and user feedback. The flexibility of the framework for been deployed by dataspaces was demonstrated with scenario studies.

The following research contributions are reported in this dissertation:

1. A high-level model of schematic correspondences that capture the schematic heterogeneities classified by Kim et al. [KCGS95] as well as the information needed to support the automatic generation of mappings that reconcile such heterogeneities.

2. A set of mechanisms for schema mapping generation, schema merging, correspondence composition and eliciting difference between schema over schematic correspondences. The scenario studies are provided for demonstrating the use of the mechanisms in supporting the bootstrapping and maintenance of dataspaces.

3. A framework for incrementally updating degree of belief based on the Bayesian theory which incorporates:
(a) A new method for deriving a degree of belief from similarity scores based on experimental data on the performance of schema matchers in associating schema constructs that are known to be equivalent and not equivalent.

(b) A new function for converting user feedback into degree of belief based on data from experimental studies on user feedback reported in [BPE+10].

1.4 Thesis Organisation

The rest of the thesis is organised as follow:

- In Chapter 2, we present the technical background that underpins this research: schema matching, correspondence inference, schema mapping generation, mapping refinement with user feedback, uncertainty management in database and information integration systems, and model management. For each, we discuss related work.

- In Chapter 3, we firstly study research results on query translation using schema mappings that are of different formalisms (i.e., GAV, LAV and GLAV [Len02]). We then review and refine Kim et al.’s schematic heterogeneities and reconciliation techniques. Following from this exploration, a model of schematic correspondences is defined. Based on the schematic correspondence model, we describe a new mapping generation algorithm. Finally, we present scenario studies that demonstrate the effectiveness the algorithm in generating mappings between schemas with different heterogeneities.

- In Chapter 4, we firstly motivate the use of model management operations to underpin the bootstrapping and maintenance of dataspaces. In particular, we show how to incorporate the functionality of model management operators for deriving a global schema and mappings over a set of source schemas. We then present and demonstrate the proposed algorithms for merging two schemas, for composing schematic correspondences and for eliciting the differences between two schemas.

- In Chapter 5, we firstly review research on evaluating schema matching tools, and on soliciting user feedback for refining schema mappings. We then
review the basis for updating degrees of belief using Bayes’ theorem. We then proceed to introduce the methodology for updating degree of belief on the equivalence of schema constructs. Secondly, we describe the new method for deriving a degree of belief from similarity scores based on experimental data from a schema matching benchmark. Thirdly, we present the new function for converting user feedback into degree of belief. Lastly, we present scenario studies on incrementally updating degree of belief using Bayes’ theorem in order to demonstrate the flexibility of the framework.

- In Chapter 6, we review the contributions of this research and opportunities for future work.
Chapter 2

Related Work

We have discussed in the introduction chapter that different kinds of uncertainty encountered in dataspaces can originate either at the stage of schema matching or at the stage of user feedback. At schema matching, it can be caused by either the large space of data sources or the shortcomings of similarity comparison methods that are available at a particular time. At user feedback, users may provide feedback on a small number of tuples returned by a query which may be not sufficient to infer the real performance of the dataspace [BPE+10]. In such context of pay-as-you-go data integration where schema mappings are constructed and refined incrementally, the existence of uncertainty is almost inevitable. The uncertainty on correspondences has an impact on operations that manipulate the correspondences. Therefore, dataspaces must find solutions for acknowledging and managing such an impact.

This chapter sets out to study existing research on the management of uncertainty in database systems and integration, as well as the readiness of model management prototypes and proposals for serving dataspaces. Before going into the main objectives, we firstly analyse existing proposals or solutions for the problems that exist in the path from schema association elicitation to schema mapping generation. From this analysis, we observed that most of the existing work on schema matching can only identify simple types of schematic heterogeneities.

Uncertainty management in database and data integration systems is a complex problem and it is yet to be comprehensively solved after three decades of research investment [MM10]. In Section 2.1, we motivate the need for studying uncertainty, identify a number of problem dimensions and examine the degree of coverage from representative research contributions that address the management
CHAPTER 2. RELATED WORK

of uncertainty.

We then examine a set of model management prototype systems. The main focus is on the model of correspondences. In particular, we analyse the level of information provided by the correspondences. If these systems are to be adopted for supporting dataspaces, they need to have mechanisms for managing uncertainty. The conclusion to be drawn from this analysis is that although several techniques and prototype systems have implemented a wide range of operators to operate on rich information on semantic relationships between schema constructs, very few of them have provided mechanisms for managing uncertainty to mitigate the inadequacy in schema matching solutions.

2.1 Initialisation of Data Integration Systems

In this section, we review existing proposals on a wide-range of topics in providing data integration systems, which includes schema matching, inferring schematic correspondences, schema mapping generation and soliciting feedback from users for improving the quality of the integration service.

The ultimate purpose of this study is to ascertain whether it is possible to go from schema matching to schema mapping without manual refinement for removing uncertainty sourced from schema matching, and still obtain schema mappings that reconcile all schematic heterogeneities that may exist between a pair of schemas. If this is not possible, then it demonstrates the importance of accounting for and managing uncertainty.

2.1.1 Schema Matching

Dataspaces provide unified access services over a set of data sources. Users of such services can pose queries on a global schema that are to be answered with data residing in one or more data sources. Data stored in the data sources are specified by schemas which are instances of meta-modelling languages, e.g. relational, XML or ER. Each participating data source may be developed and maintained by different users or organisations. In such a situation, heterogeneities may exist among the data sources. In other words, conceptually equivalent data are represented or specified in different ways. The fundamental challenge of providing a unified data access service is to identify and reconcile different kinds of heterogeneities [KCGS95] between schema constructs that represent and specify
conceptually equivalent data entities. Schema matching is the process of eliciting associations between equivalent schema constructs, which is usually the starting step in the construction of any data integration system. Moreover, associating equivalent schema constructs is the key task in model management systems [BM07, HBM+10, PB03, SRM08, KQCJ07], which can underpin various tasks involving schema and mapping manipulation in the initialisation and maintenance of information systems. The associations returned by schema matching are input to operations for merging schemas [BM07, PB03] and for composing mappings [BM07, BGMN08] between schemas.

In early-generation data integration systems, schema matching is usually done manually [KCGS95], perhaps supported by a graphical user interface [FHH+09]. Manually eliciting associations between schema constructs is a tedious and time-consuming task. Sometimes, it requires the coordinated effort of both domain experts and technical experts to jointly manage the process. After more than ten years of research, some automated schema matching tools [DR02] and commercial systems [FHH+09] have been developed to minimize such manual effort, but they generate associations between schema constructs with simple and inadequate information on schematic heterogeneities.

A schema matching tool usually deploys multiple matching algorithms (often referred to as matchers) for different application domains, and for matching schema of different types of modelling languages. Primitive matchers of different types are invoked for examining one or more aspects of schema constructs as follows:

- **Element matchers:** It is likely that conceptually equivalent schema constructs may be specified in the same or similar ways. For example, two conceptually equivalent attributes in different schemas may be assigned with the same name or the same types. Schematic specification on individual constructs is useful evidence to postulate the equivalence of the constructs. Apart from string characters of constructs names, other examples of such information are synonyms or antonyms of construct names (assisted by external resources, e.g. dictionaries) or data types.

- **Instance matchers:** It is also likely that conceptually equivalent schema constructs may store the same or similar sets of data instances. Instances represented by the constructs can give alternative evidence for judging the equivalence of the constructs.
**Structure matchers:** as well as examining the individual constructs themselves, in the case of complex schema structures, such as XML documents, the equivalence of two constructs can be judged based on their structural context [MGMR02]. The operation of structure matchers builds on the results of prior element matchers.

Since the result of primitive matchers are important information for the inference of semantic relationship between constructs, advanced matchers [MGMR02, MBR01, DR02] have been developed which are composed of different types of primitive matchers [RB01]. Furthermore, the matching process can be customised to suit different application domains.

A matcher may associate one construct in one schema to another construct in another schema or a set of constructs, or associate a set of constructs in one schema to a set of constructs in another schema. They may also be referred to as *one-to-one associations* or *1:1 associations*, *one-to-many associations* or *1:n associations*, and *many-to-many associations* or *m:n associations*, respectively. The matching result is usually represented by the tuples, respectively:

\[
< c_1, c_2, s >, < c_1, C_2, s >, < C_1, C_2, s >
\]

where \( c_i \) denotes a single construct, \( C_i \) denotes a set of constructs and \( s \) is a numerical score within a range (usually \([0, 1]\)) that measures the degree of similarity.

We will briefly introduce some representative systems for schema matching:

**Cupid:** a matcher that is hard-coded with element and structural matching algorithms [MBR01]. Its matching approach was evaluated on relational and XML schemas. At the element level, it examines synonyms, abbreviations, and acronyms of construct names supported with dictionaries, and produces a score denoting the degree of similarity between two constructs. Element level matching is succeeded by bottom-up structural level matching. In other words, the degree of similarity of one construct to another construct in another schema depends on the degree of similarity of their child constructs. The similarity of two constructs is derived based on the similarity at the element level and at the structural level.

**Similarity Flooding:** Melink *et al.*[MGMR02] proposed the matching
algorithm called similarity flooding. The prototype implementation of similarity flooding was later integrated into a generic model management system called Rondo [MRB03]. The matching strategy used in similarity flooding is similar to the one used in Cupid. The difference is that similarity flooding decides the similarity of two constructs based on not just constructs that are directly contained within them, i.e. child constructs, but also the constructs which they share the same parent construct with, i.e. sibling constructs.

- **COMA:** The COMA schema matching system was proposed as a platform for flexibly combining a range of matchers to suit different demands [DR02]. The combined matchers are selected from a library of primitive and composite matchers. COMA provides various mechanisms for combining results of individual matchers and evaluating the performance of different matchers.

- **LSD:** Machine learning techniques were integrated into the composite matcher LSD [DDH01] to produce 1:1 associations between schema constructs in a set of source schemas and a global schema. LSD trains a set of instance matchers during a preprocessing step. The matchers learn from user-supplied associations to discover characteristic instance patterns and matching rules. The learnt patterns and rules are later used to match other source schema with the global schema.

The performance of composite matchers relies on the performance inherited from the primitive matchers they are built on. In turn, the performance of primitive matchers depend on the availability of schema matching resources e.g., dictionaries, schema information and data instances. In certain situations where high quality of schema matching is crucial, e.g. in the commercial context, schema matching is a cooperative task combining effort from domain experts and technical experts. In other situations, such as personal information management systems, where low quality of schema matching at some periods of the life cycle can be tolerated, automated schema matching tools are deployed to generate a set of candidate associations between two schemas, and provide assistance for users to refine the associations. As information integration on the web and large-scale scientific collaborations became increasingly demanding, new challenges have emerged [HFM06]. The number of data sources can be very large. The complexity of domain-specific schemas can be high, e.g. in biological data
CHAPTER 2. RELATED WORK

sources [BY06, LMMS+07], and data sources are constantly evolving. Under these kinds of situations, the need for human involvement inevitably becomes a major limitation. It is difficult to remove all the uncertainty at the schema matching phase if automation is applied, despite many advances embodied in the systems listed above.

2.1.2 Schema Mapping

The ultimate goal of dataspaces is to evaluate user queries posed on a global schema over data stored in heterogeneous sources. The global schema provides a unified representation of the source schemas and is either created by users or automatically derived by merging the source schemas [PB03, PB09]. Schema mappings are constructed to reconcile the conflicts between source schemas and the global schema. Schema mappings are the key components in the process of query translation. Without reconciling schematic heterogeneities, a user query posed against the global schema may be answered with incorrect data entities retrieved from different data sources. The mapping must specify how data entities stored in one data source are represented by the global schema. Schema mapping generation is a key component in other problems in information system management. For example, in the problem of data migration [FHH+09], schema mappings need to be provided as an executable expression to transform data from an old database to be stored in a new database.

Given a set of associations between equivalent constructs, the next problem in the construction of dataspaces is how to infer and reconcile schematic heterogeneities between associated constructs. Associations produced by schema matching are evidence of similarity between schema constructs. Schematic correspondences go one step further by also providing information on schematic heterogeneities between the associated constructs. Manual or automated mapping generation must understand and characterise the types of schematic heterogeneities that may exist. In early data integration systems, domain experts and technical experts interacted in order to derive schema mappings from the output of schema matching. More recently, many research prototypes and commercial systems [BMC06, FHH+09, MLM+01, BBF+10, Stu, Map, Car06] have been developed to ease the generation of schema mappings. For example, IBM’s Clio system [FHH+09, YMHF01] provides a graphical interface for specifying and refining mappings between two schema. The schema to be mapped are firstly
imported and converted into an internal representation. A set of matchers (or manual input) are then applied to characterise 1:1 and m:n associations between the schemas. Based on the associations, algorithms [MHH00] are applied to derive mappings represented as SQL queries between source and target schemas. The mapping generation algorithm can reconcile heterogeneities such as one-to-one construct name conflicts, as well as many-to-one vertical partitioning conflicts. The vertical partitioning conflicts are inferred based on schema associations and reasoning with referential constraints provided in the source schema. In dataspaces, one aims to fully automate mapping generation. The incorporation of a mapping design, generation and refining tool is not normally required. Under such an assumption, automatically generated mappings can not be expected to correctly reconcile all the heterogeneities that may exist.

This dissertation is largely inspired by [HBM+10], which identified and specified a set of generic functional components for dataspace systems. The functional design of [HBM+10] was based on an extension of the model management vision [BM07]. The schematic correspondence model proposed by [HBM+10] is in turn inspired by the schematic heterogeneities identified by Kim et al. [KCGS95]. It is useful to characterise model management operators that can be used to bootstrap and maintain dataspaces, but taking schematic correspondences as input so that schema mappings can be generated over the manipulated correspondences. The research conducted by Kim et al. [KCGS95] produced a classification of a wide-range of schematic heterogeneities between object-relational schemas. Apart from the classification, Kim et al. also explored a set of resolution techniques for each type of classified heterogeneities for the purpose of guiding the manual construction of a multidatabase system [KCGS95]. The classified schematic heterogeneities are listed below with a corresponding approach to reconcile them by mapping for query translation:

- **One-to-one entity-to-entity name conflict.** Two entities in different schemas are conceptually equivalent (in other words, they represent the same concept), but are assigned different names. Likewise, two entities maybe conceptually different but have the same name. If they are different, we should not derive mappings between them even though they share the same name. If they are equivalent, mappings should be derived.

- **One-to-one entity-to-entity missing attributes.** Two entities are conceptually equivalent, but some attributes in one are missing in the other
one. If we define a mapping to populate a target entity that has attributes that are missing in the source entity, the mapping should provide null value to the missing attributes. On the other hand, if we define a mapping to populate the source entity, the attributes of the target entity that are missing in the source should not be projected by the mapping.

- **One-to-one entity-to-entity constraint conflicts.** Two entities are conceptually equivalent, but have different attributes as their keys. As for missing attribute conflicts, if one of the key attributes in the target entity is missing in the source entity, the mapping should populate as null the missing key attribute.

- **One-to-one entity-to-entity entity inclusion.** The extent of one entity is included in the extent of another conceptually equivalent entity. In this case, the mapping should express how tuples of one entity should be selected to populate the other entity.

- **Many-to-many entities-to-entities conflicts.** The concept is represented by different numbers of entities in different schemas representing the same concept. In this case, the mapping should specify how to combine entities of one side to populate entities of the other side.

- **One-to-one attribute-to-attribute name conflicts.** An attribute of a concept is represented by attributes in different schemas with different names or different attributes of a concept are represented by attributes in different schema with the same name. As for one-to-one entity-to-entity name conflicts, regardless if the attributes have the same or different names, we should not specify how to map between conceptually different attributes.

- **One-to-one attribute-to-attribute data type conflicts.** An attribute of a concept is represented by two attributes in different schemas with different data types. In this case, the mapping should indicate the functions to be used to cast between the data types.

- **Many-to-many attributes-to-attributes conflicts.** An attribute of a concept is represented by two sets of attributes in different schemas. Mappings should indicate functions to be used to calculate values of attributes in one side from values of attributes of the other side.
• **Entity-to-attribute conflicts.** A concept is represented by entities in one side, but is represented by attributes of the other side. This can be construed as a special case of attribute-to-attribute conflict, since every entity constitutes of one or many attributes.

Building on Kim et al.’s work, this dissertation describes a high-level model of schematic correspondences that capture the schematic heterogeneities in [KCGS95], and describes how the manual reconciliation techniques can be captured as an algorithm to generate schema mappings that reconcile such schematic heterogeneities.

There have been research proposals focused on experimental evaluation of schema matching systems [DR07, DMR02, ATV08, LSDR07], that study the effectiveness of schema matching algorithms in eliciting similarities on different kinds of schemas and data sources. The *MatchBench* benchmark [HBM+10] proposed a systematic test of the performance of schema matching algorithms in diagnosing Kim et al.’s schematic heterogeneities. One lesson learnt from the experimental study with *MatchBench* is that instance level matching makes a significant contribution to the correct identification of conceptually equivalent constructs. Moreover, even with advanced composite matchers [DR02, MGMR02], it is quite hard to identify all the schematic heterogeneities required for mapping generation [KCGS95, HBM+10]. Therefore, the management of uncertainty plays a crucial role to mitigate the negative impact generated by such limitations in schema matching tools and algorithms.

Several proposals exist for refining and verifying automatically generated mappings [YMHF01, BMP+08, CT06]. For instance, Bonifati et al. [BMP+08] developed the Spicy system. The system executes the executable expression of the mapping to populate the target schema with a sample of data instances represented by the source schema. The transformed data instances are compared with instances represented by the target schema. Therefore, this approach relies on the assumption that a target instance is always available. Chiticariu et al. [CT06] proposed a schema mapping debugger system that provides a step-by-step walk through for guiding users to understand the derivation of mappings between two schemas. Such an approach may generate significant upfront cost.
CHAPTER 2. RELATED WORK

2.2 User Feedback

The refinement and verification of mappings at the initialisation stage is inconsistent with the principles of dataspaces [HFM06], since it may generate significant up-front cost. An alternative is to leverage feedback provided by users while accessing the data. Belhajjame et al. [CVDN09] proposed a solution that annotates mappings with user feedback. Based on the feedback, the precision and recall of an evaluating mapping are estimated. Precision is the ratio of the number of records correctly retrieved by the mappings to the number of retrieved records. Recalled is the ratio of the number of correctly retrieved records to the number of records that should be retrieved. For example, assume a user is interested to query a relational table book in the global schema for records on books. A schema association may associate book to two tables, science_book and economics_book, in the source schema S1, and to the table flight_booking in the other source schema, S2 (e.g. since their names have similar sub-strings). Based on this information, schema mapping generation may derive candidate mappings that associate book respectively to the table science_book, to economics_book, to the union of the tables science_book and economics_book (of S1), and to flight_booking (of S2). The user is shown a table of records retrieved by all the candidate mappings which he or she can use to point out which records are correctly or incorrectly retrieved. Likewise, he or she can suggest records that should be retrieved and yet have not been shown in the result. Precision and recall can be leveraged as evidence for selecting and ranking mappings for future query translation between the global schema and the source schema, as well as to update the confidence in the similarity of constructs. Since we can not expect users to provide feedback on all the records returned to them, Belhajjame et al. conducted experimental studies on approximating the real precision and recall of the mappings in respect to the amount feedback users may provide.

Talukdar et al. [TJM+08] studied a different type of feedback that users may offer. The authors developed the Q system, which assists users in posing a query on a unified interface for data stored in a set of data sources. The user provides feedback by ordering records returned by different alternative queries. In the previous example, a query posed on book in the global schema will be translated to queries posed on S1 and S2 which return records about scientific and economic books. Users can order the records based on their requirements.

As well as query results, users can provide feedback on other artifacts in
dataspaces. McCann et al. [MSD08] developed a system aimed at reducing the workload of refining results of schema matching by seeking feedback from users. User are prompted with questions such as \textit{is attribute name the same as the attribute book\_name} or \textit{is attribute published\_date of type Date}. Alexe et al. [ACMT08] developed the MUSE system to assist domain experts in designing and refining a set of candidate mappings as executable expressions between two schemas. Based on their knowledge of the target schema, participating domain experts provide answers to a set of yes-or-no questions to determine join predicates in reconciling many-to-many heterogeneities. Moreover, for evaluating whether mappings can correctly map source data, the experts are asked to select data instance sets generated by executing all the plausible candidate mappings on the source using a graphical interface. These approaches to soliciting feedback from users ease the workload involved in refining the artifacts, but create significant upfront cost.

Feedback from users of dataspaces provide valuable evidence to update the confidence acquired at the bootstrapping stage regarding the postulation of conceptual equivalence between schema constructs, as well as for reconciling schematic heterogeneities that may exist between them. However, feedback from users may cause uncertainty. For example, users may provide feedback only on a small set of tuples mapped by a mapping, or users may have only limited knowledge of the target domain. Experiments conducted in [CVDN09] show that the more feedback users provide, the smaller their divergence to the actual precision and recall of the mappings. This dissertation adopts the model of feedback proposed in [CVDN09]. We then propose a methodology for incrementally assimilating evidence in the form of user feedback and similarity scores from schema matching algorithms into a degree of belief in schematic correspondences between schema constructs.

In the next section, we present a set of challenges and approaches in the management of uncertainty for information system and integration.

## 2.3 Uncertainty in Databases Systems and Integration

The management of uncertainty is a long-established research topic in database systems [Gra77, IJ84, CP87, Koc08, RDS07, DS07a, ABS+06, JXW+08] and in
CHAPTER 2. RELATED WORK

data integration [SDH08, HRO06, DHY09a, GMSS09, TCY92, DS07b]. The term “uncertain” is typically used to express that one has limited information or knowledge in support of a conclusion. A single database may store imprecise information about real world entities. This may be caused, for example, by the fact that data stored in the database was inserted by users that had limited knowledge about the information domain. For example, a user may not be certain about the attribute value of a particular data record. To mitigate the impact uncertainty may cause, many researchers have proposed extensions to the relational data model to facilitate users entering multiple candidate attribute values [IJ84, CP87, Koc08, RDS07] for a particular data record. Each plausible attribute value is associated with a confidence value. In most cases, the confidence values are drawn from probability distributions, i.e. the sum of all the confidence values must be 1. Uncertainty of this kind is frequently encountered in acquiring location information about moving objects [CXP+04]. Such information is continually updated at a certain pre-set time point. Between two update time points, the location information of the object becomes uncertain.

Furthermore, users may not be sure about the existence of a data entity [ABS+06, JXW+08]. For example, a user may not be sure whether a book should be stored into a relation called computer_science_book. Agrawal et al. [ABS+06] developed the Trio database system that allows users to assign confidence values to data records he or she enters. These two kinds of uncertainty are classified and named respectively as attribute-level uncertainty and tuple-level uncertainty, and databases that store such uncertain data represented by the extended relational model are named as probabilistic databases. The majority of subsequent research activities were dedicated to studying query evaluation over probabilistic database [Koc08, RDS07, DS07a, TCY92]. Most of the research contributions depend on the assumption that the confidence values, or their probability distributions, are given in advance. Uncertainty on data needs to be incrementally updated when different kinds of evidence becomes available.

Furthermore, there are research proposals regarding the representation of uncertain data. A set of approaches for extending the relational model has been published for dealing with attribute-level uncertainty [BGMP92, Fuh90, SBHW06], tuple-level uncertainty [BSHW06, SBHW06, GT06]. Sarma et al. [SBH+09] surveyed representation approaches for uncertain data. They considered various properties, such as completeness (i.e. whether the model for uncertain data can
represent any sets of plausible data instances), closure (i.e. whether the model can represent the result of an operation on uncertain data instances) and expressive power. However, they did not consider methods for quantifying uncertainty, or for updating uncertainty.

In data integration, apart from data level uncertainty, new types of uncertainty may exist in the elicitation of associations between schema constructs, the inference of schematic correspondences, and the generation of mappings. Automated schema matching algorithms may be supported with limited resources, e.g. dictionaries, data instances or schema information. A technical expert may not understand the true semantics of domain specific schema constructs e.g. of a biological database. Likewise, without technical experts, domain experts may lack knowledge of the model and language used for expressing schematic correspondences.

The vision of dataspaces [HFM06] benefits from full automation at the bootstrapping stage. This means that, ideally, no manual intervention should be required for eliciting schema associations, inferring schematic correspondences and schema mappings specification or refinement. Starting at schema matching, there may exist many plausible associations between a schema construct in one schema and constructs in the other schema. For example, assume that in schema $S_1$, there is one relation $\text{Student}(\text{sid}, \text{name})$ that stores data records about students (where $\text{sid}$ is the key). In the other schema, $S_2$, there is a conceptually equivalent relation $\text{Student}(\text{student id}, \text{firstname}, \text{surname})$. Two relations are equivalent, if and only if they represent the same set of entities. Although the relations are conceptually equivalent, they are assigned with different numbers of attributes. There are several plausible ways to associate the attribute $\text{name}$ to the attributes $\text{firstname}$ and $\text{surname}$, which depends on the type of schema matching algorithms is applied. For example, a matcher that compares string character of relation names may yield the following associations, among others:

1. $\text{name}$ matches the attribute $\text{firstname}$.

2. $\text{name}$ matches the attribute $\text{surname}$.

3. $\text{name}$ matches the concatenation of $\text{surname}$ and $\text{firstname}$.

Existing schema matching tools [MGMR02, DR02] usually return a set of associations and provide assistance for users to select between them. Based on
these associations, several plausible types of schematic heterogeneities that are classified by Kim et al. [KCGS95] may be inferred:

1. In the case where name is associated to the attribute firstname, this is construed as a different name for the same attribute [KCGS95]. Assume that sid is associated to student.id, then surname (of $S_2$.student) becomes a missing attribute in $S_1$.student.

2. If name is associated to the attribute surname, this is construed as a different name for the same attribute [KCGS95]. Assume that sid is matched to student.id, then firstname becomes a missing attribute in $S_1$.student.

3. If name is associated to the concatenation of surname and firstname, this is construed as a many-to-many attribute heterogeneity [KCGS95].

Notice that, in this research, we define model of schematic correspondence based on schematic heterogeneities classified by Kim et al. [KCGS95], although there may potentially exist other types of schematic heterogeneities (for example, there can be more specific types of many-to-many attribute-to-attribute heterogeneities, e.g. the value of an attribute in one schema is the concatenation of values delimited by ’.’ of a set of attributes in another schema, or the sum of values of a set of attributes in another schema, etc). Our aim is to demonstrate the use of an expressive model of schematic correspondences to capture schematic heterogeneities for mapping generation so that, given the model of schematic correspondences, automated mechanisms for schema mapping generation can be developed. The inference of schematic heterogeneities is not in the scope of this thesis. We assume that some functional components in a dataspace systems (such as the inferCorrespondences components developed in [HBM+12]) can infer heterogeneities classified by Kim et al. [KCGS95] (reviewed in Section 2.1.2). In Chapter 3 (Sections 3.2.2 and 3.2.3), we will discuss the corresponding mappings generated by our mapping generation algorithm for each type of schematic correspondence.

Given information on plausible schematic heterogeneities between schema constructs, corresponding candidate schema mappings can be generated to express constructs of $S_2$ as a view over constructs of $S_1$.

Besides issues from probabilistic databases, we anticipate a set of specific issues on uncertainty management for data integration and dataspaces, as follows:
CHAPTER 2. RELATED WORK

1. **The quantification of uncertainty.** Uncertainty on equivalence judgments (e.g. whether two schema constructs are conceptually equivalent [MGMR02, MR08], or whether there is a vertical partitioning on constructs, $c_1$ and $c_2$ in schema $S_1$ with respect to $c_3$ in schema $S_2$) is usually caused by insufficiency of the relevant knowledge (e.g. lack of information on integrity constraints to infer many-to-many schema correspondences, lack of data instances to support instance-level matching, or lack of domain specific dictionaries to understand the semantics of construct names). The extent of perceived information support for or against a hypothetical assessment [Sha07, J92, Ram26] can be quantified as the degree of uncertainty. Quantified uncertainty needs to be represented and measured. There are several approaches in data integration for representing uncertainty direct as similarity scores [SDH08, MGMR02, DR02]; probability [DHY09a, MRMM05, GMSS09, WLB07, MM08].

2. **The update of the degree of uncertainty.** The degree of uncertainty needs to be updated in response to a new evidence. For example, when a new schema matching algorithm or matching resources (e.g. new data instances are inserted) becomes available, or information on the empirical performance of schema mapping or schema matching algorithms stemming from relevant benchmarks [ATV08, DMR02, LSDR07] is obtained. Methods have been proposed for updating the degree of uncertainty on the equivalence between schema constructs based on aggregating similarity comparison scores from schema matching algorithms [GMSS09, MM08, WLB07, NS05, NS07]. In dataspaces, evidence, such as similarity comparison result [HFM06] or user feedback [BPE+10, CVDN09] can become available in no particular order [HBF+09]. Thus, an incremental approach for updating uncertainty is required for a pay-as-you-go style of data integration.

3. **The adaptation of operations to uncertain inputs.** Algorithms for conventional database and data integration operations need to be adapted to cope with the uncertain context they operate in. In data integration, query translation between two schema constructs needs to be adapted to retrieve data records based on all the plausible schema mappings between the constructs [DHY09a, GMSS09], as well as to rank the retrieved data records [DHY09a, GMSS09] based on the degree of uncertainty attributed
to the mappings. Dong et al. [DHY09a] proposed the notion of probabilistic mappings. They assumed that the quantified degree of uncertainty was given. The authors defined two semantics for the "probabilistic mappings". Firstly, among the set of plausible mappings, there is only one correct mapping. In other words there is only one correct way to specify how data records represented by the source relation can be represented under the target relation. This is not very realistic. Secondly, among the set of plausible mappings, there is more than one correct mapping between the relations. For example, assume that a target relation is defined as

\[ \text{Student}(\text{sid, name}) \]

whilst a source relation is defined as

\[ \text{Student}(\text{sid, givenname, surname}) \]

One mapping may specify that the \textit{name} attribute of the target relation is to be populated by data values derived from concatenating \textit{givenname} and \textit{surname} (of the source schema) delimited by a space in between,

\[ \text{name} \leftarrow \text{givenname} + "\ " + \text{surname} \]

The other mapping may specify a different order on concatenating \textit{givenname} and \textit{surname}:

\[ \text{name} \leftarrow \text{surname} + "\ " + \text{givenname} \]

This may happen because, for instance, the target relation may represent students from different parts of the world. Students from far east countries express their names differently from European students, with the surname arriving before given name. Dong et al. [DHY09a] also studied query evaluation and complexity for \textit{selection-project-join} queries over probabilistic mappings. The proposed query evaluation algorithm is mainly an extension of existing query evaluation algorithms on GLaV mappings [Len02, Hal01] with the additional mechanisms for query result ranking. Following on from Dong et al. [DHY09a], Gal et al. [GMSS09] further studied query evaluation and complexity for other types of queries such as aggregation over
probabilistic mappings.

However, operations that manipulate schematic correspondences also need to be adapted. For example, the derivation of a global schema over a set of source schema may need to be automated based on schema associations identified between the source schema. Sarma et al. [SDH08] proposed an adaptation for automated global schema generation to cope with uncertainty that may exist in schema associations elicited by schema matching. Their algorithms generate a single global schema as well as probabilistic mappings over a set of source schemas. The schema merging mechanisms proposed by Magnani et al. [MRMM05] derive a set of plausible merged schemas over two source schemas based on uncertain associations between constructs of the source schemas. The set of merged schemas is presented for refinement by users or experts.

4. The representation of uncertainty in the output of data integration operations. We have mentioned several proposed solutions for representing uncertain data instances and query results in probabilistic databases [BGMP92, Fuh90, SBHW06, AKG91, DS07a, Gra84, LLRS97, AKG91, LLRS97]. There are many kinds of uncertain objects in data integration generated from automated processes [FHH+09, YMHF01, BMC06] or even manual set-up [BY06, LMMS+07]. For example, there may be many plausible associations between two sets of constructs in different schemas. Moreover, there may be many plausible mappings between two (or two sets of) constructs [DHY09a, GMSS09]. Likewise, there may be many plausible mediating schemas [MRMM05] that reconcile a set of heterogeneous data sources. In data integration, the elimination of uncertainty is performed by manual intervention [Len02, SDH08]. To assist manual effort, a set of representation mechanisms have been proposed to explicitly present uncertain schema associations [RB01, FHH+09, BMC06], uncertain mappings [FHH+09, ATV08, CT06, DHY09b] and uncertain mediating schemas [SDH08, MRMM05].

To sum up, we note that most existing proposals for uncertainty management in database and integration system have focused on the definition and representation of uncertain data and uncertain correspondences between schema constructs,
as well as on the adaptation of query evaluation mechanisms and query complexity analysis in the presence of uncertainty. The judgement of equivalence between schema constructs can be supported by different kinds of information. There has been research on strategies for combining similarity scores at the schema matching stage to quantify the uncertainty. No proposal has studied the assimilation of user feedback. In dataspaces, information such as similarity comparison results and feedback from users are available in no given order at different points of the lifecycle. Therefore, an incremental strategy for assimilating similarity scores from different kinds of schema matching tools as well user feedback is more suitable for adopting dataspaces. The contributions of this dissertation provide support that complement existing research results on query evaluation and result ranking over uncertain correspondences [SDH08, DHY09a, GMSS09]. Furthermore, the leverage of user feedback for updating uncertainty in the judgement of equivalence between schema constructs can enhance the effectiveness of user feedback for underpinning the improvement of information integration.

2.4 Model Management

The generalisation of schema and correspondence manipulation operation for use in the initialisation and maintenance of information systems has been widely studied. These have mostly focused on one part of the problem. The manipulation usually involves specifying transformations for translating schemas in different modelling languages, and specifying transformation for data migration between data sources. Some motivating scenarios are:

1. Companies may regularly upgrade their database systems. Data stored in an old database needs to be migrated to a new database by which it is to be replaced. In advance, schema mappings are designed and expressed as executable expressions between schema constructs of the old and new database, which encode requirements on how data need to be transformed.

2. When two companies decide to merge to become a single company, they may also need to merge their information systems. Such a task requires the generation of a merged schema which subsumes the information in the schemas to be merged. Mappings need to be specified between the merged
schema and source schemas so that data stored in the old database can be transformed and loaded into the new database.

3. Relational databases are typically designed with ER diagrams. An ER diagram needs to be translated into a relational schema expressed in SQL. Later, a user may decide to modify the relational schema e.g. by adding or removing attributes. Any changes made on the relational schema need to be propagated to the original ER diagram to maintain consistency.

Based on an analysis of many initialisation and maintenance scenarios of information systems in which schemas and mappings are first-class types, Berstein et al. [Ber03] envisioned a set of generic and reusable manipulation operations which are frequently in the demand regardless of scenarios. Likewise, the bootstrapping and maintenance of dataspaces can be functionally decomposed into a set of generic operators underpinned by model management [MBHR05, HBM+10]. For example, to set-up a dataspace system, Match is repeatedly invoked to elicit associations between schema constructs of each participating data source and constructs of the global schema. Secondly, Merge is repeatedly invoked [PB09] to automatically obtain a global schema served as a unified query interface.

As well as from schemas and correspondences, dataspaces require schema mappings to translate queries between the global schema and each source schema [MBHR05, HBM+10]. Moreover, the source or global schemas may be updated by, for example, adding or removing constructs. To maintain the consistency, corresponding schematic correspondences need to be changed accordingly (an example of such case will be presented in Chapter 4). In this case, Compose and Diff are used [MBHR05]. Hedeler et al.[HBM+10] has proposed a set of generic operators for the management of dataspaces. Based on the stages of the dataspace life-cycle where they might be needed, the identified operators are classified into bootstrapping, usage, improvement and maintenance stages. Most operators used in the bootstrapping and maintenance of dataspaces are functionally identical to model management operators, but manipulate correspondences that capture the schematic heterogeneities [KCGS95]. In addition to a set of model management based operators, there are operators that are specific for generating schema mappings, evaluating and translating queries, as well as operators manipulate user feedback.

Most model management operators rely on results of the Match operator. Dataspaces require high automation at the bootstrapping and maintenance stages.
To be adopted into the context of dataspaces, there are some anticipated extensions need to be made on model management operators among others:

1. Dataspaces require schema mappings to translate queries between the global and the source schemas. Thus, it is useful to complement model management operators with mapping generation mechanisms.

2. Schema mappings need to reconcile heterogeneities between the global and the source schemas. One approach is to characterise model management operators over schema mappings. However, mappings are represented as executable expressions. There sometimes requires a step to parse the expression to elicit associations between schema constructs. For example, a SQL view needs to be parsed in order to understand there is an association between an attribute of the source schema and an attribute of the schema created by the view. Whereas, associations between schema constructs are directly indicated by schematic correspondences. Some model management operators need to be informed by associations between schema constructs. For example, the Compose operator drives an association between two constructs if they are both associated to another construct. Thus, it is useful to characterise model management operators over schematic correspondences, and schema mapping generation mechanisms can be directly applied to generate mappings from correspondences derived by model management operators. Although there has been the proposal of an inferenceCorrespondence operator to derive schematic correspondences from results of schema matching [HBM+10], characterising model management operators over schematic correspondences, rather than associations derived from the Match operator, may reduce the number of InferenceCorrespondence invocation. For example, without characterising Merge over schematic correspondences, Merge will derive two sets of schema associations, Match\(_{1-3}\) and Match\(_{2-3}\), between the merged schema, \(S_3\), and each of the two source schema \(S_1\) and \(S_2\) respectively, based on the set of schema associations, Match\(_{1-2}\).

\[
\text{Match}(S_1, S_2) \rightarrow \text{Match}_{1-2}
\]

\[
\text{Merge}(S_1, S_2, \text{Match}_{1-2}) \rightarrow < S_3, \text{Match}_{1-3}, \text{Match}_{2-3} >
\]

The two sets of derived schema associations, Match\(_{1-3}\) and Match\(_{2-3}\), then
need to be input to $InferenceCorrespondence$ to derive schematic correspondences, $CR_{1-3}$ and $CR_{2-3}$.

$$InferenceCorrespondence(Match_{1-3}) \rightarrow CR_{1-3}$$

$$InferenceCorrespondence(Match_{2-3}) \rightarrow CR_{2-3}$$

Therefore, in total, there are two $InferenceCorrespondence$ invocations to complete the overall task. If the $InferenceCorrespondence$ is invoked on the input set of schema association in advance to apply $Merge$, and if $Merge$ is characterised over schematic correspondences, then the same task is simplified to one $InferenceCorrespondence$ invocation.

$$Match(S_1, S_2) \rightarrow Match_{1-2}$$

$$InferenceCorrespondence(Match_{1-2}) \rightarrow CR_{1-2}$$

$$Merge(S_1, S_2, CR_{1-3}) \rightarrow <S_3, CR_{1-3}, CR_{2-3}>$$

3. There can be several types of heterogeneities that may exist between schemas [KCGS95]. Associations elicited by advanced schema matching algorithms only capture simple one-to-one schematic heterogeneities [MGMR02, DR02], quite short of the wide-range of schematic heterogeneities identified by Kim et al. [KCGS95], which include missing attribute between two entity types, vertical partitioning, and horizontal partitioning heterogeneities between two sets of entity types. The inference of schematic correspondences may take as input uncertain schema associations from schema matching algorithms. Such uncertainty must be propagated to the inferred schematic correspondences, and to schematic correspondences derived by model management operators which manipulate the former. Thus, model management operators must be adapted to propagate degrees of uncertainty in a principled manner.

Several research prototypes [MRB03, SRM08, KQCJ07, ABBG09c] have implemented the model management vision. Many other proposals have concentrated on specific model management operators [DR02, MGMR02, MBR01, MR08, PB03, PB09, BDK92, MRMM05, BGMN08]. What follows, a review of proposed correspondence models, mapping generation and correspondence manipulation.
mechanisms, as well as uncertainty awareness, is presented.

The AutoMed system [SRM08] implemented the Match and Merge operator. The Match operator elicits a set of one-to-one correspondences between a pair of constructs in two schemas. A composite matcher is used (based on construct name string comparison and data instance comparison algorithms [Riz04]) to hypothesise whether the sets of data instances represented by two constructs are equivalent, intersecting, subsuming or disjoint, without quantifying its degree of uncertainty. The elicited correspondences are refined by manual effort assisted by a graphical user interface. The Merge operator acts on refined correspondences elicited by Match to obtain a reconciled schema without generating correspondences between the reconciled schemas and each source schema. A mapping model, BAV [RM05], was proposed based on the Global-Local-as-View concept [Len02]. Each BAV mapping specifies the steps needed for transforming a source schema to a target schema. The transformation task contains a sequence of operations such as delete a construct, add a construct or rename a construct. The generation of BAV mappings from one-to-one correspondences is achieved semi-automatically. In this dissertation, we characterise schematic correspondences that capture the heterogeneities in [KCGS95] and that can be input to Merge, Compose, Diff and a complementary mapping generation operator ViewGen. So, firstly, schema mappings can be generated to support query translation and, secondly, other operators can be used to underpin the bootstrapping as well as the maintenance of a dataspace. Moreover, we have studied and proposed a set of rules for propagating degrees of uncertainty on the equivalence of constructs, from inputs to outputs.

Based on the correspondence model defined in [Riz04], Magnani et al. [MRMM05] conducted an independent study of schema merging algorithms with uncertainty management. They developed a method based on the Dempster-Shafer rules for combining degrees of belief given by experts in assessing the associations between schema constructs. The proposed schema merging algorithm enumerates different sets of plausible correspondences to generate a set of merged schemas between two source schemas. In comparison to our contributions, the authors did not propose strategies for quantifying the degree of uncertainty from user feedback.

The GeRoMeSuite system [KQCJ07, KQL07, QK07] implemented the Merge and Compose operators. The system adopted element-level and structural matchers developed in [MBR01] to generate schematic correspondences that inform
one-to-one and many-to-many heterogeneities between schema constructs. The \textit{Merge} operator was designed over such correspondences to generate a reconciled schema over two source schemas. The system provides a graphical user interface to support semi-automatic schema mapping generation \cite{KQ+09}. The mappings are represented in the GLaV mapping model, based on which \textit{Compose} was designed. The downside of such a mapping model is that it is difficult for domain experts to understand and improve the mappings \cite{KQ+09, BGMN08}. The system does not provide uncertainty management in the case where entire automation of bootstrapping is needed.

The Rondo system \cite{MRB03}, is developed by Microsoft, was the first general prototype implementation of model management. It implemented a wide-range of model management operators over simple one-to-one correspondences that associate constructs in relational schemas. The schema matcher adopted by Rondo \cite{MGMR02} is composed of element-level, instance-level and structural comparison algorithms. The generation of mappings requires manual intervention, which may incur significant upfront cost. There are no mechanisms in Rondo for managing the impact of uncertainty stemming from schema matching \cite{MRB03}.

Atzeni \textit{et al.} developed the MISM model management system \cite{ABBG09d, ABBG09a} and implemented the \textit{ModelGen} \cite{ACG07a}, \textit{Diff} and \textit{Merge} \cite{ABBG09a}, as well as schema mapping generation mechanisms \cite{ABBG09d}. A schema conforming to a meta-model type, such as relational, XML schema or ER, can be translated into a schema of any other meta-model type using the \textit{ModelGen} operator. The translation mechanisms are driven by proposed dictionaries of schema constructs as well as a set of predefined one-to-one construct translation rules. Atzeni \textit{et al.} proposed a supermodel whose constructs subsume the modelling capabilities of specific concrete models, such as XML schema, relational, object-relational and ER. There are different sets of translation rules, each of which specify the associations between different types of construct of one specific model to corresponding types of construct in the supermodel. Based on the translation rules, a schema in a specific model is firstly translated, one construct at a time, to a schema in supermodel which is then translated to a schema in the target-specific model.

The \textit{Diff} and \textit{Merge} operations of MISM are characterised by one-to-one construct translation rules. Every schema of a specific model is firstly translated into the supermodel. Associations between constructs of the translated schema are
elicited with a primitive $Match$ operator which performs construct name string comparison. The $Diff$ and $Merge$ are then applied on the translated schema. Given two schemas, $S_1$ and $S_2$, the $Merge$ operator firstly adds constructs of $S_1$ to the merged schema, and then adds constructs of $S_2$ that are missing in $S_1$ into the merged schema. Thus, the $Merge$ operator does not create correspondences for constructs of the output schema and constructs of the input schemas. In comparison to the contributions of this dissertation, we characterised the model management operators, $Merge$, $Compose$, $Diff$, and schema mapping generation over correspondences that capture richer schematic heterogeneities. More importantly, we postulate that uncertainty, in the form of multiple plausible correspondences between a pair of schema constructs, may exist in the input of the operators. Each correspondence is assigned a degree of uncertainty in judging the equivalence of associated constructs. To manage the uncertainty, we propose a set of rules for propagating the input uncertainty to correspondences derived by the operators.

In general, most of the model management prototype systems have implemented a wide range of operators aimed at solving the problem of schema evolution or the initialisation of data integration systems. However, they are inadequate for supporting dataspaces [HFM06, HBF+09]. Firstly, some of the systems do not provide mechanisms for generating schema mappings that reconcile schematic heterogeneities and are necessary for mediating query translation. Moreover, the designed and implemented operators do not take into consideration the existence of uncertainty at the $Match$ operator. To manage uncertainty, we have developed a methodology for incrementally updating the degree of uncertainty in the judgement of the equivalence of constructs (associated by correspondences), as well as proposing propagation rules for deriving the degree of uncertainty for correspondences output by model management operators, on the basis of the degree of uncertainty provided as input to the operators.

### 2.5 Conclusion

In this chapter, we have reviewed the work on schema matching and observed that schema matching tools usually incorporate a range of similarity comparison methods and, in general, produce simple one-to-one associations between schema constructs. Mappings generated over such associations can only reconcile limited
types of schematic heterogeneities. Although there is ongoing work on eliciting schematic correspondences from the associations identified by matchers, uncertainty cannot be fully removed from generating correspondences that carry rich information on schematic heterogeneities.

We have analysed several issues on uncertainty management in dataspaces. By examining related work on each of the issue, we noted that support is lacking for quantifying uncertainty on equivalence judgments resulting from matchers, as well as on other forms of evidence, such as user feedback, that are part of the dataspaces vision.

In recent work on dataspaces, Hedeler et al. [HBM+10] have used model management as the foundation of dataspaces, so that the design and specification of bootstrapping and maintenance of schema and mapping evolution for dataspaces can then be reduced to sequences of operation invocations. However, most model management operators [Ber03] manipulate the result of schema matches. Furthermore, whilst some implemented prototype systems have adopted correspondence manipulation which carry rich information on schematic heterogeneities, very few of these have addressed uncertainty management, so that they cannot account for the shortcomings stemming from correspondence elicitation.

We have discussed the technical context and related work of this dissertation. Follow from these, Chapter 3 describes our approach for automating schema mapping generation over schematic correspondences based on a proposed high-level modelling of schematic correspondences. Chapter 4 describes the characterisation of model management operations over such model of schematic correspondences. In Chapter 5, we present the proposed approach for quantifying similarity scores and user feedback into degree of belief and a strategy for incrementally assimilate different kinds of evidence into degree of belief founded on the Bayes theorem.
Chapter 3

View Generation over Schematic Correspondences

Having introduced the concept of schematic correspondence in the previous chapters, this chapter defines an algorithm for generating schema mappings with schematic correspondences as input. The addition of an intermediate step geared towards eliciting schematic correspondences between the conventional data integration initialisation steps, i.e. schema matching and schema mapping, has been widely adopted in the literature [HBM+10, MRMM05, SRM08, QK07, PB03].

Schematic correspondences describe schematic heterogeneities based on associations between conceptually equivalent schema constructs postulated by schema matching, and provide the basis for the reconciliation of heterogeneities by schema mapping. In this research, we propose a model of schematic correspondence designed for expressing the heterogeneities between schematic representations of superabstractions. Between schematic representations, two schemas, for example, may use two superabstractions with different sets of attributes, or partition the attributes into more than one superabstraction, for representing the same concept.

To contend with the heterogeneities, schema mapping needs to apply some reconciliation method, e.g., when populating $r_2$ with tuples of $r_1$, the null value is assigned to the missing attribute. Hedeler et al. [BM07, HBM+10] proposed a set of generic operations for dataspaces that functionally support the bootstrapping and maintenance phases, of dataspace management. Among others, the operators inferCorrespondences and ViewGen have, respectively, the goals of generating correspondences that capture schematic heterogeneities and converting
the correspondences into mappings that reconcile the heterogeneities. The generated view can then be used to translate and evaluate queries between schemas. This chapter focuses on the ViewGen operator.

Schematic heterogeneities were originally studied and classified by Kim et al. [KCGS95]. A set of resolution techniques associated with each type of heterogeneity was proposed for the purpose of guiding manual integration of object-relational databases. The first objective of this chapter is to review Kim et al.'s schematic heterogeneities. We observe that the classification did not provide a sufficiently concrete characterisation of certain types of heterogeneities. Moreover, the background modelling language assumed in their classification was object-relational. In dataspaces, data can be modelled with different methods, such as relational or XML. Methods adopted by different modelling languages for describing real world concepts share similar purposes. For example, a concept can either be represented as an entity type in the ER model or as a relation in the relational model. We can define a general modelling construct type that subsumes the ER entity type and the relation. Atzeni et al. [ACG07b] studied the generalisation of the modelling capabilities of specific formalisms, and proposed the MIDST supermodel, which contains a set of generic modelling constructs. Among others, an ER entity type, an Object-Relational typed table and an XML root element are generalised as an Abstract; a Relational attribute and an ER attribute are generalised as a Lexical; an ER relationship is generalised as an AggregationOfAbstracts; an Object-Relational generalisation and an ER generalisation are generalised as a Generalisation; an XML complex element and an Object-Relational structured column are generalised as a StructOfAttributes. The usefulness of these generic constructs, as argued and demonstrated by Atzeni et al., is that they can play the role of an intermediate model for mediating the translation of specific models from one type to another [ACG07b]. For example, a rule can be encoded as a Datalog program that translates an Abstract (an ER entity type) into an Aggregation (a Relational table). Hedeler et al. further generalised the MIDST supermodel to one with the following constructs: superabstract (subsumes Abstract and Aggregation), superlexical (subsumes Lexical) and superrelationship (subsumes AggregationOfAbstracts, Generalisation and ForeignKey). In this dissertation, we define a model of schematic correspondences on constructs of this supermodel, so that, given model translation rules, the schematic correspondences can be used to associate constructs of specific models. Based on this model
of schematic correspondences, we have designed algorithms for generating mappings. We finally demonstrate, with a case study, how schematic correspondences enable the automation of schema mapping generation. The work discussed in this chapter was published in [MBPF09].

3.1 Query Translation with Schema Mapping

Data sources participating in dataspaces may be developed and maintained by independent parties. As such, conceptually equivalent entity types or data values may be specified and structured in different ways. The intensional description of a data source is specified in a schema (e.g. as a set of statements in some data definition language). Translations of queries between two schema rely on information and techniques for reconciling heterogeneities between the schema. The following example is used to illustrate manual query translation with information on schematic heterogeneities.

Different departments of a university may maintain database systems for archiving information of their students. Among others, Manchester University School of Computer Science may specify the concept undergraduate students in the following relational schema, taken as source:

\[ \text{Student}(\text{sid, givenname, surname, annual\_grade}) \]

The university may need to regularly evaluate the performance of undergraduate students studying in the individual department. It queries information of students in each department through the following global schema as a unified interface:

\[ \text{UG\_Student}(\text{sid, fullname, gender, annual\_grade}) \]

and a SQL query \( Q \) is specified on the global schema as:

\begin{verbatim}
SELECT fullname, gender, annual_grade
FROM UG_Student
\end{verbatim}

\( Q \) can not be compiled against the source schema. To translate \( Q \) to a query that can be compiled against the source schema, schematic heterogeneities must be captured and reconciled in advance. We can observe the following heterogeneities between the source and global schema:
1. They use different strings, i.e. “UG_Student” and “Student” to name relations that represent the same concept.

2. The gender attribute in the target relation UG_Student is missing in the source relation Student.

3. The names of students are decomposed into two attributes, givenname and surname, in Student, but are represented with a single attribute, name, in UG_Student.

Given the information on schematic heterogeneities, one can manually translate $Q$ onto a query $Q'$ against the source schema using conflict resolution techniques. Firstly, to reconcile the relation name conflict, the FROM clause has to be renamed. Secondly, since there is no attribute in Student denoting gender, the null value should be used to mark this fact. Lastly, the fullname attribute should be populated with values that concatenate givenname and surname (with a white-space character in the middle). Therefore, the derived form of $Q'$ is:

```sql
SELECT CONCATENATE(givenname, ' ', surname), null, annual_grade
FROM Student
```

where we assume that the function CONCATENATE is given and the domains of its input arguments match the domain of givenname and surname, respectively.

This example the process of reconciling schematic heterogeneities using query translation. The resolution of schematic heterogeneities can be encoded in schema mappings for supporting automated query translation. The essential form of a mapping is that of an executable expression in a query language that can correctly populate a target schema with data values drawn from a source schema. In the previous example, one can capture the reconciliation process in the mapping:

```sql
CREATE VIEW UG_Student
AS (sid, fullname, gender, annual_grade)
SELECT sid, CONCATENATE(givenname, ' ', surname), null, annual_grade
FROM Student
```
Q':
SELECT fullname, gender, annual_grade
FROM AS Student'
{
    SELECT sid, CONCATENATE(givenname, ' ', surname), null,
            annual_grade
    FROM Student
}

Apart from heterogeneities between a pair of two constructs, there may exists heterogeneities between a pair of two sets of constructs. For example, assume that university partitions the conceptual view of students into local and overseas students:

\[\text{Local\_UG\_Student}(\text{sid}, \text{fullname}, \text{gender}, \text{annual\_grade})\]

\[\text{Overseas\_UG\_Student}(\text{sid}, \text{fullname}, \text{gender}, \text{annual\_grade}, \text{country})\]

Assume that the computer science department added an extra attribute to the relation to represent the country where a student comes from:

\[\text{Student}(\text{sid}, \text{givenname}, \text{surname}, \text{annual\_grade}, \text{country})\]

Assume that the university is interested in analysing the performance of students from the UK (local students), it issues the following query, \(Q_1\):

SELECT fullname, gender, annual_grade
FROM Local\_UG\_Student

In order to translate \(Q_1\) to a query to retrieves data from the computer science department that compiles to \(Q_1\) (i.e. students studying computer science from the UK only, rather than student from all countries in the world), the mapping needs to specify which tuples of \text{Student} should be populated to \text{Local\_UG\_Student}. For example, the mapping includes a \textit{where clause} to specify a \textit{selection condition}:

CREATE VIEW Local\_UG\_Student
AS (sid, fullname, gender, annual_grade)
SELECT sid, CONCATENATE(givenname, ' ', surname), null,
        annual_grade
FROM Student
Likewise, information students of the Maths department is partitioned in two relations:

\[\text{Student}(\text{sid}, \text{givenname}, \text{surname}, \text{annual\_grade})\]

\[\text{Address}(\text{aid}, \text{street}, \text{postalCode}, \text{country}, \text{sid})\]

Given the same query \(Q_1\), in order to retrieve tuples of students studying mathematics from the correct countries, tuples in the Student and Address relations need to be joined based on the foreign key \(\text{sid}\) in Address. Thus, the mapping need to specify the correct join condition:

\[
\text{CREATE VIEW Local\_UG\_Student}
\text{AS (sid, fullname, gender, annual\_grade)}
\text{SELECT S.sid, CONCATENATE(S.givenname, ' ', S.surname), null,}
\text{S.annual\_grade}
\text{FROM Student S, Address A}
\text{WHERE A.sid = S.sid AND A.country = 'UK'}
\]

In these examples, a construct in the global schema is associated to a view over constructs in the source schema. In data integration, this mapping formalism is called Global as View (or GAV in short) [Len02, GMPQ+97a]. Alternatively, mappings may associate a construct in the source schema to a view over constructs in the global schema (i.e. LAV [Len02, Hal01]). Lastly, a mapping may associate a query over constructs of the global schema to a query over constructs of the source schema. The last two formalisms are respectively called local as view (LAV) [Len02] and Global and Local as View (GLAV) [Len02, FLM99]. One of the enduring contributions made by data integration is techniques for query rewriting with mappings formalisms [Hal01, Len02, GMPQ+97a, FLM99], which were surveyed in [Hal01].

Data integration systems that developed query evaluation mechanisms over the LAV formalism are [LRO96b, MFK01] among others. Mappings denoted as views are implemented in the form of XQuery [MFK01] or Datalog [LRO96b]. Queries posed on a global schema are rewritten based on selecting a set of views which relate constructs of the global schema to constructs of sources from which the data can be drawn. Many techniques were proposed for evaluating specific
forms of queries, such as conjunctive queries [LRO96a], XQuery [CGLV99], and aggregation queries [CNS99], and for many query complexity has been studied. Systems that are implemented over GAV mappings include TSIMMIS [GMPQ+97b], Garlic [CHN+95] and Squirrel [ZHKF95]. Queries posed on constructs of the global schema are evaluated based on modifying the queries with views which express each construct of the global schema in terms of constructs of sources. Embley et al. [XE04, EXD04a] developed and implemented techniques for evaluating queries over GLAV. Most of the previous work on query evaluation over mappings for data integration rely on the assumption that mappings are available. Although Embley et al. complemented their query evaluation techniques with automated mapping generation mechanisms [EXD04b], the mappings are generated with simple associations provided by automated schema matching algorithms. Schema mapping generation tools are mostly semi-automatic [FHH+09, BMC06, ACMT08, ATV08, BMP+08, KQ+09], providing assistance to experts in refining associations generated by schema matching algorithms. Based on the refined matches, mapping generation tools infer and reconcile specific types of schematic heterogeneities (such as different names for the same constructs and many-to-many vertical partitioning). The generated mappings are implemented in view queries which are then refined by experts. These approaches cannot be adopted in dataspaces given the purpose of minimising upfront costs. In dataspaces, the assumption is that the global resources draw from a great number of autonomous sources which, therefore, often exhibit many types of heterogeneities. In addition, the schemas that describe the sources may be continually updated due to evolving data access requirements. To alleviate these challenges and at the same time comply to the dataspaces aim of minimising upfront costs, the generation of mappings has to anticipate and automatically reconcile schematic heterogeneities as comprehensively as possible.

3.2 Schematic Correspondences

We have shown above examples illustrating the translating of queries between two schemas. Schema mappings are the crucial mediator in reconciling schematic heterogeneities caused by the use of different schematic representations for describing conceptually equivalent information.
In the example depicted in Figure 3.1, a set of data sources use the relational model for describing the concept of university students. The global schema contains a single table with a set of attributes representing each dimension of the concept, such as student name or address. On the other hand, each source schema represents the same concept in different ways. Source1 allocates a table with a different name, UniStudent; Source2 represents names of students with two attributes, i.e. firstname and surname, and uses an attribute for study type which is not specified in the global schema; Source3 represents the student concept with separate tables, namely OverseasStudent and LocalStudent; and, finally, Source4 considers the attributes (address and country) of students as a separate concept, i.e. Accommodation, which is related to the student concept with the aid attribute.

**3.2.1 Classification of Schematic Heterogeneities**

Kim *et al.* [KCGS95] classify a wide range of schematic heterogeneities that may exist between object-relational schemas. Some types of heterogeneity are associated with reconciliation techniques for guiding the manual generation of a global schema and of mappings between the global and the source schemas. At the most general class of the classification, heterogeneities are classified based on
the nature of objects that are heterogeneous, e.g. heterogeneities between entity
types, between attributes, or between entity types and attributes. Subsequently,
these heterogeneities are further classified into one construct to one construct,
one construct to many constructs or many constructs to many constructs, i.e.
the same concept is represented by different number of constructs. Between
two constructs in different schemas, there can be name conflicts, type conflicts
(between attributes), and missing attributes (between entity types). Between two
sets of entity types in different schemas, the representation of a concept in one
schema can be construed as vertical partitioning (or horizontal partitioning) of
the representation in another schema. Between one construct in one schema and
a set of constructs in the other schema, the later may adopt vertical or horizontal
partitioning method to represent the concept.

Kim and al. identified a wide range of schematic heterogeneities and proposed
reconciliation techniques. Nevertheless, there are some limitations:

- It did not identify horizontal partitioning as part of many-to-many hetero-
geneties.

- It did not provide technique for reconciling conflict between extentional
domain of entity types.

- It does not convey sufficient information for deriving views that express the
mappings between schemas. Consider, for example, a correspondence that
connects the relations Student and Accommodation in Source4 of Figure
3.1 to the relation, Student, in the global schema. To be able to derive
the view for populating the relation Student using tuples in both Student
and Accommodation (of Source4), information specifying how the tuples
in Student and Accommodation are to be combined is needed.

In the following, we augment the characterisation of heterogeneities by ex-
tending the classification proposed by Kim and al. to support the automated
generation of mappings. In doing so, we define a model of schematic correspon-
dences that capture heterogeneities over the supermodel proposed by Atzeni and
al. [ACG07b]. Relations such as UniStudent and Accommodation, in Figure
3.1, are captured as superabstractions, whilst attributes, such as typeofstudy and
surname are captured as superlexicals.

For the rest of this dissertation, a schema is denoted with $S$, and is defined
as $S = SA \cup SL \cup SR$, i.e. the union of the sets of superabstracts, superlexicals and superrelationships, respectively.

**Definition** A superlexical is defined as a 4-tuple $<\text{name}, \text{domain}, \text{sa}, \text{isKey}>$, where

- **domain** is the domain of the data values (e.g. int, varchar),
- **name** is the identifier of the superlexical,
- and **isKey** is a boolean indicating whether the superlexical is a key of superabstract $sa$.

**Definition** A superabstract is defined as a pair $<\text{name}, SL>$, where

- **name** is the identifier of the superabstract,
- and **SL** is a list superlexicals identifiers.

**Definition** A superrelationship is defined as a 4-tuple $<\text{name}, rt, sa_1, sa_2>$, where

- **rt** denotes the relationship type (e.g. association or generalisation),
- **sa_1** and **sa_2** are references to the superabstracts related by the superrelationship,
- and **name** is the identifier of the superrelationship.

In this research, we proposed a model of schematic correspondence designed for expressing the heterogeneities between schematic representation of superabstracts and between their extensional domain. The correspondences are bidirectional. In other words, the same correspondence can be used to generate one mapping to populate the target with extent of the source, as well as a mapping to populate the source with extent of the target.

**Definition** A schematic correspondence

$$sc = (sacr|slcr)$$

where **sacr** is a superabstract correspondence and **slcr** is a superlexical correspondence.
CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

Definition  A superabstract correspondence

\[ sacr = (o2osacr|m2osacr|o2msacr|m2nsacr) \]

where \( o2osacr \) denotes one-to-one superabstract correspondence, \( m2osacr \) denotes many-to-one superabstract correspondence, \( o2msacr \) denotes one-to-many superabstract correspondence, and \( m2nsacr \) denotes many-to-many superabstract correspondence.

Definition  A superlexical correspondence

\[ slcr = (o2oslcr|m2oslcr|o2mslcr|m2nslcr) \]

where \( o2oslcr \) denotes one-to-one superlexical correspondence, \( m2oslcr \) denotes many-to-one superlexical correspondence, \( o2mslcr \) denotes one-to-many superlexical correspondence, and \( m2nsacr \) denotes many-to-many superlexical correspondence.

Mappings generated from such schematic correspondence model are denoted as relational algebra queries. The schematic correspondence model is exact, since it generates the mappings that contain operators from the following set:

- **Projection** (\( \pi \)): two sets of conceptually equivalent superabstracts, \( SA_1 \) and \( SA_2 \), may have different sets of superlexical. Thus, some superlexicals in \( SA_1 \) may be missing in \( SA_2 \). These superlexicals should be projected out when populating \( SA_2 \) with tuples of \( SA_1 \).

- **Selection** (\( \sigma \)): the selection operator is used to reconcile the conflicts in the extensional domains of two or two sets of superabstracts. For example, only tuples denoting students studying Maths in a superabstract representing students of all subjects can be populated into the other superabstract representing only Maths students.

- **Join** (\( \Join \)): if one schema applies vertical partitioning method to represent one concept, tuples of the partitioning superabstracts should be joined to populate conceptually equivalent superabstracts of the other schema. Projections or selections may then applied on the joined tuples.
• **Union (∪):** if one schema applies horizontal partitioning method to represent one concept, tuples of the partitioning superabstracts should be unioned before applying projection or selection to populate conceptually equivalent superabstracts of the other schema.

Other operators are omitted, since we focused on reconciling Kim’s schematic heterogeneities with extension to vertical and horizontal partitioning.

In the next two sections, we will define the model of superlexical and superabstract correspondences, as well as their semantics in terms of relational algebra.

### 3.2.2 Superlexical Correspondences

This family of correspondences associates superlexicals of superabstracts that belong to a source schema with superlexicals that represent the same attribute in a target schema. A superlexical correspondences can be classified into four kinds depending on the number of superlexicals involved from the source and target schema.

**Definition** A **one-to-one superlexical correspondence** associates one superlexical in the source schema with one superlexical that represents the same attribute of a concept in the target schema:

\[
< s_l_1, s_l_2, f_{1→2}, f_{2→1}, d >
\]

where

- \( s_l_1 \) and \( s_l_2 \) are two superlexicals in two different schemas and are referenced using the format, \( saName.slName \), where \( saName \) is the name of the superabstract to which the superlexical named \( slName \) belongs.

- \( f_{1→2} \) specifies how data values of \( s_l_2 \) can be computed using data values of \( s_l_1 \), while \( f_{2→1} \) specifies how data values of \( s_l_1 \) can be computed using data values of \( s_l_2 \). For the rest of the thesis and other type of superlexical correspondences, \( IDF \) denotes the application of an identity function for assigning the same value of the source as the value of the target.

- A **quantified uncertainty**, denoted as \( d \in [0, 1] \), measuring the level of confidence that \( s_l_1 \) and \( s_l_2 \) are conceptually equivalent. The derivation of \( d \) is discussed in Chapter 5.
For example, in Figure 3.1, assume that the *name* superlexical in the global schema is equivalent to *name* in *Source1*, and string values of the *name* in the Global Schema are in uppercase, whereas *name* in *Source1* has its string values in lowercase. The following superlexical correspondence is specified to associate the superlexicals:

\[
\langle \text{Student.name, UniStudent.name, toLowercase(Student.name) } \rightarrow \text{(UniStudent.name), toUppercase(UniStudent.name) } \rightarrow \text{(Student.name), 0.7 } >
\]

The function `toLowercase` maps lowercase letters into uppercase, whereas the function `toUppercase` maps uppercase into lowercase.

**Definition** A **one-to-many superlexical correspondence** associates a superlexical, \( sl_1 \), in a source schema to a set of superlexicals, \( SL_2 \), in a target schema:

\[
\langle sl_1, SL_2, f_{1 \rightarrow 2}, f_{2 \rightarrow 1}, d \rangle
\]

where

- \( f_{1 \rightarrow 2} \) is a function specifying how to compute values of superlexicals in \( SL_2 \) using values of superlexical in \( sl_1 \), while \( f_{2 \rightarrow 1} \) is a function specifying how to compute values of superlexicals in \( SL_2 \) using values of superlexical in \( sl_1 \). The range of \( f_{1 \rightarrow 2} \) must match the domains of superlexicals in \( SL_2 \), and the range of \( f_{2 \rightarrow 1} \) must match the domain of \( sl_1 \).

- A **quantified uncertainty**, denoted as \( d \in [0, 1] \), measuring the level of confidence that the superlexicals, \( sl_1 \) and \( sl'_2 \) derived as the combination of superlexicals in \( SL_2 \), are conceptually equivalent.

For example, in Figure 3.1, a superlexical correspondence can be specified to associate the superlexical *name* in *Source1* to *firstname* and *surname* in *Source2*:

\[
\langle \text{Student.name, } \{\text{University_Student.firstname, University_Student.surname}\},
\]


CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

DECOMPOSE(' ', Student.name)  
-> (University_Student.firstname, University_Student.surname)  
CONCATENATE(University_Student.firstname,' ',University_Student.surname)  
-> (Student.name)  
0.2

For Student.name, we assume that the firstname and surname of students are concatenated with a white-space character in between. The function DECOMPOSE is used for splitting string values in Student.name based on a white-space character, and assigns the derived strings to firstname and surname of University_Student. Meanwhile, the function CONCATENATION concatenates firstname and surname with a blank space in the middle to form values of Student.name.

**Definition** A many-to-one superlexical correspondence associates a set of superlexicals, SL\(_1\), in a source schema to one superlexical, sl\(_2\), in a target schema:

\[
< SL_1, sl_2, f_{1\rightarrow 2}, f_{2\rightarrow 1}, d >
\]

where

- \( f_{1\rightarrow 2} \) specifies how to compute values of superlexical in sl\(_2\) using values of the superlexicals in SL\(_1\), while \( f_{2\rightarrow 1} \) specifies how to compute values of superlexicals in SL\(_1\) using values of the superlexical in sl\(_2\).

- A quantified uncertainty, denoted as \( d \in [0, 1] \), measuring the level of confidence that the superlexicals, sl\(_1'\) derived as the combination of superlexicals in SL\(_1\) and sl\(_2\), are conceptually equivalent.

Due the the similarity in the definition of many-to-one and one-to-many superlexical correspondences, the resolution of many-to-one superlexical heterogeneities is analogous to that for one-to-many superlexical heterogeneities.

**Definition** A many-to-many superlexical correspondence associates a set of superlexicals, SL\(_1\), in a source schema to a set of superlexicals, SL\(_2\), in a target schema:

\[
< SL_1, SL_2, f_{1\rightarrow 2}, f_{2\rightarrow 1}, d >
\]

where
CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

• \( f_{1 \rightarrow 2} \) specifies how to compute values of superlexicals in \( SL_1 \) using values of superlexicals in \( SL_2 \), while \( f_{2 \rightarrow 1} \) specifies how to compute values of superlexicals in \( SL_2 \) using values of superlexicals in \( SL_1 \).

• A quantified uncertainty, denoted as \( d \in [0,1] \), measuring the level of confidence that the superlexicals, \( sl'_1 \) and \( sl'_2 \), derived as the combination of superlexicals in \( SL_1 \) and \( SL_2 \) respectively, are conceptually equivalent.

With information provided in a superlexical correspondence, we can map between superlexicals with the projection operator of the relational algebra, as well as superlexical functions specified in the correspondence:

\[
\pi_{f_{1 \rightarrow 2}}(sa_1.sl_1, ..., sa_1.sl_n)(sa_1) \rightarrow sa_2(sa_2.sl_1, ..., sa_2.sl_m)
\]

We assume that superlexicals \( sa_1.sl_1, ..., sa_1.sl_n \) are associated to \( sa_2.sl_1, ..., sa_2.sl_m \) by a many-to-many superlexical correspondence, and the function \( f_{1 \rightarrow 2} \) computes values of \( sa_2.sl_1, ..., sa_2.sl_m \) of \( sa_2 \) based on values of \( sa_1.sl_1, ..., sa_1.sl_n \) of \( sa_1 \).

With a one-to-one superlexical correspondence, we can also derive a relational algebra query to map between a target and source superlexical:

\[
\pi_{f_{1 \rightarrow 2}}(sa_1.sl_1)(sa_1) \rightarrow sa_2(sa_2.sl_1)
\]

We assume that \( sa_1.sl_1 \) is associated to \( sa_2.sl_1 \). With a similar method, we can also derive mappings based on one-to-many and many-to-one superlexical correspondences.

The use of superlexical correspondences for deriving mappings as relational algebra queries is implemented in the algorithm in Figure 3.4. The algorithm iterates through each superlexical of a target superabstract, and accordingly incorporates the functions given in each superlexical correspondences to specify the projection operation.

In the following subsection, we will define superabstract correspondences and their semantics based on relational algebra.

3.2.3 Superabstract Correspondences

We distinguish between four kinds of superabstract correspondences depending on the number of superabstracts used to represent a given concept in the source and
target schemas, namely one-to-one superabstract correspondences, many-to-many superabstract correspondences, many-to-one superabstract correspondences and one-to-many superabstract correspondences.

**Definition** A one-to-one superabstract correspondence associates one superabstract in the source schema with one superabstract that represents the same concept in the target schema:

\[
< \text{sa}_1, \text{sa}_2, \text{sp}_1, \text{sp}_2, \text{LC}, d >
\]

where

- \( \text{sa}_1 \) is a superabstract in the source schema, \( \text{sa}_2 \) is a superabstract in the target schema.
- \( \text{LC} \) is a set of superlexical correspondences that associates superlexicals of \( \text{sa}_1 \) and \( \text{sa}_2 \).
- Two selection predicates, \( \text{sp}_1 \) and \( \text{sp}_2 \), in which each is in the form:

\[
< \text{target\_superabstract}, (\text{selection\_condition}|*) >
\]

where \( \text{target\_superabstract} \) is the superabstract to be populated based on the \( \text{selection\_condition} \) or all tuples are selected (denoted by *). The \( \text{selection\_condition} \) is a conjunction, a disjunction or a mixture of conjunction and disjunction of comparison operations either between a superlexical and a constant or between two superlexicals. A selection predicate \( \text{sp} \) specifies that a subset \( T \) of all tuples represented by a superabstract \( \text{sa}_i \) can be used to populate another superabstract \( \text{sj} \). \( T \) is subsumed by all tuples that can potentially be represented by \( \text{sj} \). The purpose of including \( \text{target\_superabstract} \) in a selection predicate will be discussed later, when selection predicate is used for reconciling many-to-many superabstract heterogeneities.

- A quantified uncertainty, denoted as \( d \in [0, 1] \), which measures the level of confidence that \( \text{sa}_1 \) and \( \text{sa}_2 \) are conceptually equivalent (i.e., represent the same concept). The derivation of \( d \) is discussed in Chapter 5.

In Figure 3.1, an example of a one-to-one superabstract correspondence, \textit{o2osacr}, between the Global Schema and \textit{Source1} is:
CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

\[ o_{2osacr.sa1} = \text{Student} \]
\[ o_{2osacr.sa2} = \text{UniStudent} \]
\[ o_{2osacr.sp1} = <\text{UniStudent}, \text{UniStudent.age} > 28 \& \text{UniStudent.country='UK'}> \]
\[ o_{2osacr.sp2} = <\text{Student}, * > \]
\[ o_{2osacr.d} = 0.5 \]
\[ o_{2osacr.LC} = \{<\text{Student.sid}, \text{UniStudent.sid}, \text{IDF, IDF, 0.5}>, <\text{Student.name}, \text{UniStudent.name}, \text{IDF, IDF, 0.5}>, <\text{Student.address}, \text{UniStudent.address}, \text{IDF, IDF, 0.5}>, <\text{Student.country}, \text{UniStudent.country}, \text{IDF, IDF, 0.5}> \} \]

where \( sp_1 \) specifies that tuples of \text{Student} can be used to populate \text{UniStudent}, if the selection condition on superlexicals, \text{UniStudent.age} and \text{UniStudent.country}, is satisfied. \( sp_2 \) specifies that all tuples of \text{UniStudent} can be used to populate \text{Student}. As mentioned in the definition of one-to-one superlexical correspondence, \( \text{IDF} \) denotes the application of an identity function for assigning the same value of the source as the value of the target.

With a one-to-one superabstract correspondence, we can derive a relational algebra query to map data between a source superabstract and a target superabstract:

\[ \pi_{f_1(sa_1.sl_1), f_2(sa_1.sl_2)}(...) (\sigma_{sp_1}(sa_1)) \rightarrow sa_2 \]

The projection operator operates on an ordered set of superlexicals of \( sa_1 \) (mapped by functions in superlexical correspondences) or the null value (when a superlexical is missing) in corresponding to the set of superlexicals of \( sa_2 \). If a superlexical in \( sa_1 \) is missing in \( sa_2 \) if and only if it is not associated by any superlexical correspondences in \( LC \). The selection operator is operated on the selection predicate given in the superabstract correspondence that associate \( sa_1 \) to \( sa_2 \).

**Definition** A one-to-many superabstract correspondence associates one superabstract in one schema to a set of superabstracts in another schema:

\[ < sa_1, SA_2, pt_2, JP_2, SP_1, LC, d > \]

where

- \( sa_1 \) is a superabstract in the source schema, \( SA_2 \) is a set of superabstracts in the target schema.
• $pt_2$ indicates the type of partitioning used in the target, where two values can be assigned to $pt_2 = (’VP’ | ’HP’)$. ’VP’ denotes that there is a vertical partitioning, while ’HP’ denotes a horizontal partitioning.

• $JP_2$ is a set of join predicates. Each join predicate, $jp$, indicates the condition to join tuples of two superabstracts:

$$< sa_{2i}.sl_i, sa_{2j}.sl_i, \theta >$$

where $sa_{2i}.sl_i$ and $sa_{2j}.sl_i$ are any two superlexicals in two different superabstracts in $SA_2$, and $\theta$ is a binary relation in the set $\{<, \leq, =, \geq, >\}$. For every pair of tuples of $sa_{2i}$ and $sa_{2j}$, can contribute to the result of the mapping, if and only if values of the superlexicals $sa_{2i}.sl_i$ and $sa_{2j}.sl_i$ satisfy the relation $\theta$.

• $SP_1$ is a set of selection predicates indicate, in the case of horizontal partitioning, how to select tuples of $sa_1$ to populate superabstracts in $SA_2$. Each selection predicate is defined in the same form as it is defined for a one-to-one superabstract correspondence. $target\_superabstract$ in the definition of selection predicate is used to indicate which superabstract in $SA_2$ is the target of the selection.

• $LC$ is a set of superlexical correspondences that associate superlexicals of $sa_1$ and superabstracts in $SA_2$.

• A quantified uncertainty, denoted as $d \in [0, 1]$, measuring the level of confidence that $sa_1'$ and $sa_2'$, derived from combining superabstracts of $SA_1$ and $SA_2$ respectively, are conceptually equivalent.

In Figure 3.1, an example of an one-to-many superabstract correspondence between the Global Schema and Source4 is:

```
o2msacr.SA1 = Student
o2msacr.SA2 = {Student, Accommodation}
o2msacr.pt2 = ’VP’
o2msacr.JP2 = {<Student.aid, Accommodation.aid, = >}
o2msacr.SP1 = {}
o2msacr.d = 0.5
```
CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

\[ o2msacr.LC = \{\langle \text{Student}.sid, \text{Student}.sid, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.name, \text{Student}.name, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.country, \text{Student}.country, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.address, \text{Accommodation}.address, \text{IDF}, \text{IDF}, 0.5\rangle \} \]

\[ o2msacr.JP_2 \] indicates that tuples of \textit{Student} and \textit{Accommodation} are joined based on the conditions that values of \textit{Student.aid} is equivalent to values of \textit{Accommodation.aid}. Meanwhile, there is no selection predicate in \( o2msacr.SP_1 \), since as defined that selection predicates are added if the target uses the horizontal partitioning method. Another example of one-to-many superabstract correspondences is between the Global Schema and \textit{Source}3 in Figure 3.1:

\[ o2msacr.SA_1 = \text{Student} \]
\[ o2msacr.SA_2 = \{\text{LocalStudent}, \text{OverseasStudent}\} \]
\[ o2msacr.pt_2 = 'HP' \]
\[ o2msacr.JP_2 = {} \]
\[ o2msacr.SP_1 = \{\langle \text{LocalStudent}, \text{Student}.country='UK'\rangle, \]
\[ \langle \text{OverseasStudent}, \text{Student}.country!='UK'\rangle\} \]
\[ o2msacr.d = 0.5 \]
\[ sacr.LC = \{\langle \text{Student}.sid, \text{LocalStudent}.sid, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.sid, \text{OverseasStudent}.sid, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.name, \text{LocalStudent}.name, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.name, \text{OverseasStudent}.name, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.country, \text{OverseasStudent}.country, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.address, \text{LocalStudent}.address, \text{IDF}, \text{IDF}, 0.5\rangle, \]
\[ \langle \text{Student}.address, \text{OverseasStudent}.address, \text{IDF}, \text{IDF}, 0.5\rangle \} \]

\[ o2msacr.SP_1 \] contains two selection predicates that specify how to select tuples of \textit{Student} to populate \textit{LocalStudent} and \textit{OverseasStudent}. In this case, \( o2msacr.JP_2 \) is empty, since \( SA_2 \) is not vertically partitioned.

With an one-to-many superabstract correspondences whose partitioning type is vertical partitioning. we can derive a relational algebra query to map tuples in the target superabstracts to the source superabstract:

\[ \pi_{f_1(s_{21}, sl_1), f_2(s_{21}, sl_2), \ldots, f_2(s_{22}, sl_1), \ldots}(s_{21} \land s_{21}.sl_1 \land \land_{s_{21}.sl_1, \ldots, s_{22}.sl_1}) \rightarrow s_{11} \]
Based on the join conditions specified in the join predicates, we apply join to combine tuples of superabstracts in the target. In particular, we use outer-join instead of natural join in combining the superabstracts in the target to avoid any loss of information, since an instance of a concept represented by one of the join operands may not be referred to by any instances represented by the other join operand, in which case it will not be returned by the natural join operator. For example, if a student does not live in any university-provided accommodation, he or she will not be returned by the natural join operator. On the other hand, with outer join, if a tuple, \( t_i \), of one operand failed the join condition to match any other tuples in the other operand that satisfy the join condition, we create a tuple in the resulting superabstract firstly with fields corresponding to, and assigned with data values of, the fields of \( t_i \), and secondly with fields corresponding to distinctive fields of the other superabstract and in this case the null value is assigned.

With an one-to-many superabstract correspondences whose partitioning type is horizontal partitioning, we can derive a relational algebra query to map tuples in the target superabstracts to the source superabstract:

\[
\pi_{f_1(s_{a21}.s_{l1}), f_2(s_{a21}.s_{l2}), \ldots, f_2(s_{a22}.s_{l1}) \ldots} (s_{a21} \bigcup s_{a22} \bigcup \ldots) \rightarrow s_{a1}
\]

Instead of union, tuples of superabstracts of the target is combined using outer union operator [Cod79, EN00] which calculates the union of two operands that are partially union compatible. The two operand superabstracts are union compatible if and only if they have the same set of superlexicals, and each pair of corresponding superlexicals must have the same domain. For example, if one superabstract \( s_{a1} \) has superlexicals \( \{s_{l1}, s_{l2}\} \), the other superabstract, \( s_{a2} \), has superlexicals \( \{s_{l1}, s_{l3}\} \), and \( s_{a3} \) has superlexicals \( \{s_{l1}, s_{l2}\} \), then \( s_{a1} \) and \( s_{a2} \) are partially compatible, while \( s_{a1} \) and \( s_{a3} \) are union compatible. The result of the outer union on two partially compatible superabstracts will include one representation for pair of corresponding superlexicals that has the same domain, and one representation for each superlexical has no corresponding superlexical in the other operand that shares the same domain. Tuples of the two operand superabstracts are matched based on having the same combination of values of the shared superlexicals, for example, \( s_{l1} \) of \( s_{a1} \) and \( s_{a2} \). If the values of the shared superlexical of the operand superabstracts match, value of the superlexical \( s_{l2} \) of \( s_{a1} \) and value of the superlexical \( s_{l3} \) of \( s_{a2} \) will be copied to
the resulting tuple; otherwise, one of the superlexical in the resulting tuple will have the null value. For example, if \(< sl_1 = v_1, sl_2 = v_2 >\) is a tuple of \(sa_1\), and there is no other tuples of \(sa_2\) has the same value matches \(sl_1 = v_1\) of \(sa_1\), then we create a tuple, \(< sl_1 = v_1, sl_2 = v_2, sl_3 = null >\), to the result. If there is another tuple \(< sl_1 = v_1, sl_3 = v_3 >\) in \(sa_2\), then we create a tuple, \(< sl_1 = v_1, sl_2 = v_2, sl_3 = v_3 >\), to the result. For the rest of the thesis, we denote outer union as \(\bigcup\).

Given the same correspondence that expresses a horizontal partitioning heterogeneities, we can create a query to map data from the source to the target:

\[
\pi_{f_1(sa_1, sl_1), f_2(sa_1, sl_2)} (\sigma_{sp_i}(sa_1)) \rightarrow sa_{2i}
\]

where \(sa_{2i}\) is one of the superabstract in \(SA_2\), and \(sp_i\) is one of the selection predicates in \(SP_i\) where the selection target superabstract is one of the horizontal partitioning superabstract \(sa_{2i}\).

A many-to-one superabstract correspondences is defined with similar form and semantics as one-to-many superabstract correspondences, except that the target side of one is the source side of the other.

**Definition** A many-to-many superabstract correspondence associates a set of superabstracts in one schema to a set of superabstracts in another schema:

\[
< SA_1, SA_2, pt_1, pt_2, JP_1, JP_2, SP_1, SP_2, LC, d >
\]

where

- \(SA_1\) and \(SA_2\) are two sets of superabstracts.
- \(pt_1\) and \(pt_2\) indicates the type of partitioning used in the source and the target, respectively.
- \(JP_1\) and \(JP_2\) are two sets of join predicates to specify how to join tuples of the source and the target respectively in the case where the corresponding source or target is vertically partitioned.
- \(SP_1\) and \(SP_2\) are two sets of selection predicates which are used in the case that the corresponding side is horizontally partitioned.
- \(LC\) is a set of superlexical correspondences associate superlexicals of \(SA_1\) and superlexicals in \(SA_2\).
A quantified uncertainty, denoted as $d \in [0, 1]$, measuring the level of confidence that $sa'_1$ and $sa'_2$, derived from combining superabstracts of $SA_1$ and $SA_2$ respectively, are conceptually equivalent.

In Figure 3.1, an example of a many-to-many superabstract correspondence between Source4 and Source3 is:

$m2msacr.SA1 = \{\text{Student, Accommodation}\}$
$m2msacr.SA2 = \{\text{LocalStudent, OverseasStudent}\}$
$m2msacr.pt1 = 'VP'$
$m2msacr.pt2 = 'HP'$
$m2msacr.JP2 = \{<\text{Student.aid, Accommodation.aid, = }>\}$
$m2msacr.SP1 = \{<\text{LocalStudent, Student.country='UK'>,}$
$\quad <\text{OverseasStudent, Student.country!='UK'>}\}$
$m2msacr.d = 0.5$
$m2msacr.LC = \{<\text{Student.sid, LocalStudent.sid, IDF, IDF, 0.5>},$
$\quad <\text{Student.sid, OverseasStudent.sid, IDF, IDF, 0.5>},$
$\quad <\text{Student.name, LocalStudent.name, IDF, IDF, 0.5>},$
$\quad <\text{Student.name, OverseasStudent.name, IDF, IDF, 0.5>},$
$\quad <\text{Student.country, OverseasStudent.country, IDF, IDF, 0.5>},$
$\quad <\text{Accommodation.address, LocalStudent.address, IDF, IDF, 0.5>},$
$\quad <\text{Accommodation.address, OverseasStudent.address, IDF, IDF, 0.5}>\}$

With a many-to-many vertical partitioning to horizontal partitioning superabstract correspondence, we can create the following map to map the data from the source to one of the superabstract in the target:

$$
\pi_{f_2(sa_{21}.sl_1), f_2(sa_{22}.sl_2), ..., f_2(sa_{22}.sl_i)}(\sigma_{sp_1}(sa_{21} \bowtie sa_{21}.sl_1 \bowtie sa_{22}.sl_j, sa_{22}...)) \rightarrow sa_{1i}
$$

Here, we join the superabstracts in $SA_2$ with the join predicates given in $JP_2$, since the source side is in vertical partitioning. Meanwhile, we apply a selection predicate (on the result of the join operation) in $SP_2$ whose selection target superabstract is the target superabstract to be mapped to, since the target side is in horizontal partitioning. With the same correspondence, we can also derive a mapping that maps the target to one of the superabstract in the source:

$$
\pi_{f_1(sa_{11}.sl_1), f_1(sa_{12}.sl_2), ..., f_2(sa_{12}.sl_i)}(\sigma_{sp_1}(sa_{11} \cup sa_{12} \cup ...)) \rightarrow sa_{2i}
$$
Since the source is in vertical partitioning, we select the superlexicals in the projection set based on superlexical correspondences given in the superabstract correspondence. Meanwhile, we apply union to combine the superabstract in the target that are in horizontal partitioning.

In this section, we have presented the model of schematic correspondences and their semantics. Our goal is to leverage the information expressed in the schematic correspondence to generate mappings that reconcile schematic heterogeneities for query translation.

As mentioned at the start of this section, the schematic correspondence model is exact. With the definition of each type of schematic correspondence, we can summarise that mappings generated from the schematic correspondences are in one of the following combination of selection, projection, join and union operators:

- A projection only query: this is generated in the situation when we have a one-to-one superabstract correspondence that contain one or more superlexical correspondences, but with no selection predicates.

- A projection and selection query: this is generated when we have a one-to-one superabstract correspondence with selection predicates, and one or more superlexical correspondences. Likewise, we have a one-to-many, many-to-one or many-to-many superabstract correspondence with horizontal partitioning.

- A project and join with or without selection query: this is generated when we have a one-to-many, many-to-one or many-to-many superabstract correspondence with vertical partitioning. Selection operator is included if selection predicates are given in the correspondence.

- A project and union query: this is generated when we have a one-to-many, many-to-one or many-to-many superabstract correspondence with horizontal partitioning.

In the following section, we discuss the algorithmic steps needed to translate schematic correspondences to relational algebra.

As for superrelationship, we do not derive data mappings between superrelationships, since superrelationships associate and describe relationships between superabstractions based on referring to the associated superabstractions or to their superlexicals. For example, a superrelationship \( sl \) associates superabstractions, \( sa_1 \),
and $sa_2$, in a schema $S$, while $sl'$ associates superabstractions, $sa'_1$ and $sa'_2$, in another schema $S'$. We can infer that $sr$ is equivalent to $sr'$ if and only if $sa_1$ is equivalent to $sa'_1$, and $sa_2$ is equivalent to $sa'_2$. If there is a schematic correspondence associate $sa_1$ to $sa'_1$, and another schematic correspondence associates $sa_2$ to $sa'_2$, we can derive mappings to map extent between the associated superabstractions, instead of mapping the superrelationships that only describe relationships between superabstractions, and do not have extent. The algorithm to be presented in the next section derives mapping between superabstractions that are associated by superrelationships.

### 3.3 Schema Mapping Generation

Given the model for schematic correspondences presented in the previous section, we can construct algorithms for automatically generating *global-as-view* schema mappings.

A GAV mapping associates a superabstract of a target schema to a view over one or more superabstractions in a source schema. We represent a GAV mapping as the tuple:

$$< q_1, sa_2 >$$

where $q_1$ is a query on the source schema that define a view superabstract with the same set of superlexicals (and the same arity) and represents the same concept as $sa_2$ which is a superabstract in the target schema. For example, a mapping over the superabstract *UniStudent* of *Source*1 in Figure 3.1 for the superabstract *LocalStudent* of *Source*3 in Figure 3.1 can be defined as:

$$< \pi_{sid, name, address, country}(\sigma_{country=\text{UK}}(UniStudent)), LocalStudent >$$

The query defines a view with projection over superlexicals of *LocalStudent*, $sid$, $name$, $address$, and $country$ (the superlexical $age$ is omitted, since it is missing in *LocalStudent*) and selection based on the condition that only tuple of UK students can be populated into *LocalStudent*.

Furthermore, the mapping

$$< \pi_{Student, sid, name, addr}(Student \bowtie_{sid=sid} Address), Student >$$
specified how to join tuples of the superabstracts \textit{Student} and \textit{Address} to populate the \textit{Student} superabstract of the global schema. The mapping

\[
< \text{OverseasStudent} \sqcup \text{LocalStudent}, \text{Student} >
\]

specifies that \textit{Student} in the global schema is populated by the union of tuples in \textit{OverseasStudent} and \textit{LocalStudent} of \textit{Source3}. Since the superlexical \textit{country} is distinct to \textit{OverseasStudent}, as explained before the outer union will concatenate tuples of \textit{LocalStudent} and \textit{OverseasStudent}, and derive a superabstract has superlexicals, \textit{sid}, \textit{name}, \textit{address}, as well as \textit{country}. Given a tuple of \textit{LocalStudent}, the outer union will create a copy of the tuple and assign \textit{null} to the field \textit{country}.

For GAV mappings, queries posed on the target schema can be translated using \textit{query unfolding} [GMPQ+97a]. For example, in Figure 3.2, a query posed on \textit{sa2} (assume that \textit{sa2} is the target superabstract): \( \pi_{sl_1,...,sl_n} sa_2 \) can be translated by unfolding \textit{sa2} with \textit{q1} (i.e. the translated query is: \( \pi_{sl_1,...,sl_n} q_1 \)), since \textit{q1} specifies \textit{sa2} as a view over source superabstracts.

Apart from notations for mappings, we define other notations to be used by the algorithms.

- Given a superabstract \textit{sa}, \textit{sa.name} denotes the name of \textit{sa}, and \textit{sa.superlexicals} the set of superlexicals of \textit{sa}.
- Given a set of schematic correspondences \textit{CR} and a construct \textit{c}, the function
CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

getNextCorrespondence(CR, c) returns a schematic correspondence with which c is associated. Since schematic correspondences can be inferred based on associations elicited by schema matching algorithms [HBM+10] which may generate a set of plausible associations between two or two sets of constructs in different schemas, a superabstract in the target schema may be associated zero or more times with superabstractions in the source schema. Likewise, a superlexical of a superabstract may be associated zero or more times with superlexicals in another schema. Such conflict is caused by uncertainty in schema matching. Given two correspondences cr1 and cr2, where cr1 associates a set of constructs C1i of schema S, and cr2 associates a set of constructs C2i of S, cr1 and cr2 are in conflict if and only if C1i ∩ C2i ≠ Ø. In this research, we assume that the set of schematic correspondences taken as input to the operators, Mapping Generation, Merge, Compose and Diff, does not contain conflicting correspondences.

- Given a set of superlexical correspondences, LC, getSLCorrespondence(LC, sa.sl_i) returns the correspondence in which the superlexical sa.sl_i occurs if such a correspondence exists, and returns null otherwise.

- Given a superabstract correspondence cr, cr.SA1 denotes the set of superabstractions in the source schema that are associated with cr (the size of cr.SA1 is one if cr is a one-to-one correspondence), cr.SA2 denotes the set of superabstractions in the target schema that are involved in cr, and cr.feature_name denotes other features in the correspondence.

- Given a schematic correspondence cr (either a superabstract or a superlexical correspondence), getCorrCardinality(cr) is a function that specifies whether cr is a one-to-one, many-to-many, one-to-many or many-to-one correspondence.

- Given a set of selection predicates SP the function getSelectionPredicate(SP, sa) returns the selection predicate in which the superabstract sa is the target superabstract of the selection predicate, and returns null otherwise.

- The list projectColumn is the set of superlexicals to be projected by a project operator.

The mapping generation algorithm takes as input a target schema S2, a source schema S1, and a set of schematic correspondences. It generates a set of GAV
Algorithm GenerateMappings

inputs:

$S_i$: source schema; $S_j$: target schema; $C$: a set of schematic correspondences

outputs:

$G$: a set of GAV mappings

begin

1 \hspace{1em} G = \emptyset

2 \hspace{1em} \textbf{FOR EACH } sa \in S_i, \text{superabstracts DO}

3 \hspace{1em} cr = \text{getCorrespondence}(C, sa)

4 \hspace{1em} \textbf{IF } cr \text{ is null } \textbf{THEN}

5 \hspace{1em} \text{signal}(\text{"No mappings are generated for the superabstract" } + sa)

6 \hspace{1em} \text{continue to the next superabstract in } S_i

7 \hspace{1em} \textbf{ELSE}

8 \hspace{1em} gav = \text{generateGAVMapping}(sa, cr)

9 \hspace{1em} G = G \cup gav

10 \hspace{1em} \textbf{RETURN } G

end

Figure 3.3: Algorithm for generating mappings
mappings (Figure 3.3, lines 3-11). The algorithm iterates over superabstracts of the target schema (Figure 3.3, line 2). At each target superabstract, the algorithm retrieves one superabstract correspondence that associates the current target superabstract to the source (Figure 3.3, line 3). If getCorrespondence returns null, meaning that no correspondence associates the current target superabstract, the algorithm will not generate a GAV mapping for that target superabstract (Figure 3.3, lines 4-6). Otherwise, the algorithm invokes the subroutine generateGAVMapping presented in Figure 3.4 to obtain a GAV mapping for the target superabstract. The subroutine algorithm operates in two main phases. Firstly, it specifies the superabstracts taking part in the view query and the way they are to be combined (Figure 3.4, lines 2-15). This phase contains two sub-phases. If the input correspondence \( cr \) is a one-to-one or one-to-many superabstract correspondence then the view query is assigned the source superabstract associated by \( cr \) (Figure 3.4, lines 2-4). If \( cr \) is a many-to-many or many-to-one superabstract correspondence then:

- If \( cr \) indicates that the source superabstracts are in vertical partitioning, the view query is assigned with the superabstract obtained by applying outer join to the source superabstracts conditioned on \( cr.JP1 \) (Figure 3.4, line 7). It then retrieves selection predicates from \( cr.SP1 \) with the function getSelectionPredicate in the case when the target superabstract is in horizontal partitioning. A selection operator is applied to select tuples in the result of the join query (Figure 3.4, lines 8-10) to be populated to the target superabstract.

- If \( cr \) indicates that the source superabstracts are in horizontal partitioning, the view query is assigned with the superabstract obtained by applying outer union to the source superabstracts (Figure 3.4, line 12). It then applies a selection operator on the result of the outer union operation with available selection predicates (Figure 3.4, lines 13-15).

Secondly, the subroutine constructs the list of superlexicals to be projected by the view query. To do this, it iterates over superlexicals of the target superabstract \( sa2 \), retrieving, for each superlexical \( sl \), the associated superlexical correspondences \( slcr \) from the set of superlexical correspondences specified in \( cr \). The subroutine then branches into one of the following three blocks:
Algorithm GenerateGAVMapping  

inputs:  
\( s_a \): a superabstract in a source or target schema; \( sacr \): a superabstract correspondence that involves \( s_a \),  

outputs:  
\( m \): a mapping for populating \( s_a \),  

begin  
\( q = new\ Query() \)  
\( IF\ \) getCorrCardinality(\( cr \)) \in \{ \text{'one-to-one'}, \text{'one-to-many'} \) \( THEN\)  
\( \quad s_a = sacr.SA, \)  
\( \quad q \leftarrow \sigma_{sacr.SP}(s_a) \)  
\( ELSE\ IF getCorrCardinality(\( cr \)) \in \{ \text{'many-to-many'}, \text{'many-to-one'} \) \( THEN\)  
\( \quad IF\ \) sacr.pt, is \text{'VP'} \( THEN\)  
\( \quad \quad q \leftarrow \bigtimes_{\text{cr} \in \text{sacr.sm} cr.SA}, \)  
\( \quad \quad sp = \text{getSelectionPredicate}(sacr.SP, s_a) \)  
\( \quad \quad IF\ sp \text{ is not null}\ THEN\)  
\( \quad \quad \quad q \leftarrow \sigma_{sp} q \)  
\( \quad IF\ sacr.pt, is \text{'HP'} \) \( THEN\)  
\( \quad \quad q \leftarrow \bigcup \text{sacr.SA}, \)  
\( \quad \quad sp = \text{getSelectionPredicate}(sacr.SP, s_a) \)  
\( \quad \quad IF\ sp \text{ is not null}\ THEN\)  
\( \quad \quad \quad q \leftarrow \sigma_{sp} q \)  
\( \) projectColumn = new Set()  
\( \) FOR EACH \( sl \in \text{sa\_superlexicals}\ \) DO  
\( \quad slcr = \text{getSLCorrespondence(cr.LC, sl)} \)  
\( \quad IF\ slcr\ is\ null\ THEN\ //this superlexical is a missing superlexical\)  
\( \quad \quad projectColumn \leftarrow \text{projectColumn} \cup \text{'null AS' + sl.name} \)  
\( \quad IF \) getCorrCardinality(\( slcr \)) = \text{'one-to-one'} \( THEN\)  
\( \quad \quad projectColumn \leftarrow \text{projectColumn} \cup \text{slcr.f1-2+('slcr.SL,+') \rightarrow ' + sl} \)  
\( \quad IF \) getCorrCardinality(\( slcr \)) = \text{'many-to-many'} \( THEN\)  
\( \quad \quad SL = slcr.SL, \)  
\( \quad \quad /// let \ SL = \{sl', ... sl'\} \)  
\( \quad \quad \) projectColumn = \text{projectColumn} \cup slcr.f1-2+('f1+sl'+ ... + sl'+) \rightarrow ' + sl  
\( \quad q \leftarrow [ \text{projectColumn} q \)  
\( \) m = q, sa\_  
\( \) RETURN m  
end
If \( sl \) has no correspondences associating it to superlexicals in the source superabstract (associated by \( cr \)), then \( sl \) is considered a missing superlexical. The projection operator assigns null values to the corresponding field of source tuples that populate the target superabstract \( sa_2 \) (Figure 3.4, line 19-20).

If \( sl \) participates in a one-to-one superlexical correspondence, the corresponding source superlexical is appended to the set of projected columns (Figure 3.4, line 21-22).

If \( sl \) participates in a many-to-many superlexical correspondence, a function provided in the superlexical correspondence \( slcr \) will be applied on source superlexicals as input, and is appended to the set of projected columns of the view query (Figure 3.4, line 23-26).

Finally, a GAV mapping of the form \(< q, sa_2 >\) is returned.

**Implementation**

The algorithm was implemented in Java, and was integrated into the DSToolKit dataspace Management System [HBM+12] as one of the operators for schema mapping generation. The implemented algorithm takes as input two schema objects and a collection of schematic correspondence objects which are modelled as defined in Section 3.2. Based on the type of the schematic correspondences, the algorithm outputs corresponding view objects as discussed in Section 3.2.2 and 3.2.3. The derived queries are then taken as input by other generic operators such as the query evaluator. The class diagram for representing a schema and its constructs is presented and discussed in [HBM+12]. The schema and schematic correspondences are stored in repositories, and are parsed into Java objects before being used by operators. Constructs of schematic correspondences, i.e. selection predicates, join predicates, superlexical functions, and mappings are represented by different classes referenced by the mapping generation algorithm. Classes are created for representing schemas, superabstracts, superlexicals and superrelationships. Each of the classes contain fields corresponding to elements of the schematic constructs presented in Section 3.2.1. In the next section, we demonstrate the use of the algorithm to automatically generate mappings between heterogeneous data sources.
CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

3.4 Scenario Study

In this section, we demonstrate the generation of mappings with the algorithm in section 3.3. For this purpose, we proposed a wide range of heterogeneities (including vertical partitioning or horizontal partitioning of superabstarcets in either the target or the source schema, missing superlexicals, and many-to-many superlexical correspondences) between a pair of schemas, and show that schema mapping can be automatically generated if schematic correspondences of the form presented in this chapter are used as input.

The following scenarios are designed for evaluating schema mapping generation:

- **Scenario 1**: one-to-one superababstract correspondence with one-to-one and many-to-many superlexical correspondences. A source schema contains one superababstract that represents the same concept as a superababstract in a target schema. Some superlexicals in either of the schema are missing in the other schema. There exists a selection of one-to-one and many-to-many superlexical correspondences.

- **Scenario 2**: one-to-VP superababstract correspondence with one-to-one and many-to-many superlexical correspondences. A source schema contains one superababstract that represents the same concept as the superababstract derived as the join of more than one superababstracts in the target schema. There exists a mix of one-to-one and many-to-many superlexical correspondences.

- **Scenario 3**: VP-to-HP superababstract correspondence with one-to-one and many-to-many superlexical correspondences. A source schema contains a set of superababstracts. The superababstract, derived as the join of the source superababstracts, represents the same concept as the superababstract derived as the union of a set of superababstracts in the target schema. There exists a mix of one-to-one and many-to-many superlexical correspondences.

For each scenario listed above, we firstly introduce the schemas. We then present manually created schematic correspondences between the source and target schema. Finally, we specify the expected mappings that should be derived from the algorithm, and walk through the algorithm to elaborate on the generation process.
In Scenario 1, the following source schema is provided:

\[ \text{Student}(\text{sid}, \text{firstname}, \text{surname}, \text{age}, \text{grade}) \]

and the following target schema is provided:

\[ \text{UniStudent}(\text{sno}, \text{sname}, \text{address}, \text{grade}) \]

Between the source and target schema, a one-to-one superabstract correspondences is specified:

\[
\begin{align*}
o2osacr.sa1 &= \text{Student} \\
o2osacr.sa2 &= \text{UniStudent} \\
o2osacr.sp1 &= \langle \text{UniStudent}, \text{Student}.age \rangle \\
o2osacr.sp2 &= \langle \text{Student}, * \rangle \\
o2osacr.d &= 0.5 \\
o2osacr.LC &= \{ \langle \text{Student}.sid, \text{UniStudent}.sno, \text{IDF}, \text{IDF}, 0.5 \rangle, \\
&\quad \langle \{\text{Student}.firstname, \text{Student}.surname\}, \\
&\quad \text{UniStudent}.sname, \\
&\quad \text{CONCATENATE}(\text{Student}.firstname, ', ', \text{Student}.surname) \\
&\quad \rightarrow \text{UniStudent}.sname, \\
&\quad \text{DECOMPOSE}(' ', \text{UniStudent}.sname) \\
&\quad \rightarrow (\text{Student}.firstname, \text{Student}.surname), 0.5> \\
&\quad \langle \text{Student}.grade, \text{UniStudent}.grade, \text{IDF}, \text{IDF}, 0.5 \rangle \}
\end{align*}
\]

The subroutine \textit{generateGAVMappings} (Figure 3.4) is invoked by \textit{GenerateMapping} (Figure 3.3) after encountering the target superabstract \textit{UniStudent}. Since \textit{o2osacr} is a one-to-one superabstract correspondence, \textit{generateGAVMappings} creates a selection query with the predicate specified in \textit{cr.sp1}:

\[
\langle \pi_{\text{sno}, \text{DECOMPOSE}(' ', \text{sname}), \text{null}, \text{grade}}(\sigma_{\text{Student}.age > 28}(\text{UniStudent}), \text{Student}) \rangle
\]

It then iterates over superlexicals of \textit{UniStudent} (Figure 3.4, line 17-26) for constructing a list of projected columns. The superlexical \textit{UniStudent.sno} is appended to the set, since it is associated to \textit{Student.sid} (Figure 3.4, line 21-22). The superabstract \textit{Student.name} is derived with the function \textit{CONCATENATE}
which takes \textit{Student.firstname} and \textit{Student.surname} as its input (Figure 3.4, line 24-27). \textit{UniStudent.address} is considered a missing superlexical (Figure 3.4, line 19-20). The eventual form of the view query and the GAV mapping is:

\[
<\pi_{sid,\text{CONCATENATE}(firstname,' ',surname)},\text{null,grade}(\sigma_{age>28}\text{Student}),\text{UniStudent}>
\]

In Scenario 2, the follow source schema is provided:

\textit{Student} \((sid, surname, age, address, grade)\)

and the following target schema is provided:

\textit{UniStudent} \((sno, surname, grade)\)

\textit{Address} \((aid, sno, address)\)

Between the source and target schema, a one-to-many superabstract correspondence is specified:

\begin{verbatim}
o2msacr.sa1 = Student
o2msacr.SA2 = {UniStudent, Address}
o2msacr.pt2 = 'VP'
o2msacr.JP2 = {<UniStudent.sno, Address.sno, = >}
o2msacr.SP1 = {}
o2msacr.d = 0.5
o2msacr.LC = {<Student.sid, UniStudent.sno, IDF, IDF, 0.5>,
               <Student.sid, Address.sno, IDF, IDF, 0.5>,
               <Student.surname, UniStudent.surname, IDF, IDF, 0.5>,
               <Student.address, UniStudent.address, IDF, IDF, 0.5>,
               <Student.grade, UniStudent.grade, IDF, IDF, 0.5>
            }
\end{verbatim}

To generate mappings from the source to the target, the \textit{GenerateMapping} algorithm generates a mapping (Figure 3.3, line 3-9) that specifies how to populate \textit{UniStudent} in the target schema with tuples from \textit{Student}, and generates another mapping that specifies how to populate \textit{Address} in the target schema with tuples from \textit{Student}. The subroutine algorithm \textit{generateGAVMapping} is
invoked in which lines 2-4 and 17-27 are performed for both cases.

\[
\langle \pi_{\text{sid}, \text{sname}, \text{grade}} \text{Student}, \text{UniStudent} \rangle
\]

\[
\langle \pi_{\text{null, sid, address}} \text{Student}, \text{Address} \rangle
\]

To generate mappings from target to source, \textit{generateGAVMapping} creates a join operation on \textit{UniStudent} and \textit{Address} (Figure 3.4, lines 6-10), and lines 17-27 is executed for creating a list of projected columns. The mapping is derived as

\[
\langle \pi_{\text{sno, sname, address, grade}} (\text{UniStudent} \bowtie \| \bowtie \text{sno} = \text{sno} \text{ Address}), \text{Student} \rangle
\]

In Scenario 3, we assume that the following two superabstracts are in the source schema:

\[
\text{PGStudent}(\text{sid, sname, address, area})
\]

\[
\text{UGStudent}(\text{sid, sname, address, grade})
\]

and the following target schema is provided:

\[
\text{UniStudent}(\text{sno, name, grade, type})
\]

\[
\text{Address}(\text{aid, sno, address})
\]

The source schema partitions the concept of student into undergraduate student and postgraduate student, whilst the target schema uses a separate superabstract to represent the address of student.

Between the source and target schema, a many-to-many superabstract correspondence is given:

\[
\text{m2msacr.SA1} = \{\text{PGStudent, UGStudent}\}
\]

\[
\text{m2msacr.SA2} = \{\text{UniStudent, Address}\}
\]

\[
\text{m2msacr.pt1} = \text{’HP’}
\]

\[
\text{m2msacr.pt2} = \text{’VP’}
\]

\[
\text{m2msacr.JP1} = \{\}
\]

\[
\text{m2msacr.JP2} = \{<\text{UniStudent.sno, Address.sno, = }>)\}
\]

\[
\text{m2msacr.SP1} = \{\}
\]

\[
\text{m2msacr.SP2} = \{<\text{PGStudent, UniStudent.type=’pg’}>,}
\]
CHAPTER 3. VIEW GENERATION OVER SCHEMATIC CORRESPONDENCES

\[
\begin{align*}
<\text{UGStudent}, \text{UniStudent.type='ug'}> \\
\text{cr.d} &= 0.5 \\
\text{cr.LC} &= \{<\text{PGStudent.sid, UniStudent.sno, IDF, IDF, 0.5}>, \\
&\quad <\text{UGStudent.sid, UniStudent.sno, IDF, IDF, 0.5}>, \\
&\quad <\text{PGStudent.sid, Address.sno, IDF, IDF, 0.5}>, \\
&\quad <\text{UGStudent.sid, Address.sno, IDF, IDF, 0.5}>, \\
&\quad <\text{UGStudent.address, Address.address, IDF, IDF, 0.5}>, \\
&\quad <\text{PGStudent.address, Address.address, IDF, IDF, 0.5}>, \\
&\quad <\text{UGStudent.sname, UniStudent.name, IDF, IDF, 0.5}>, \\
&\quad <\text{PGStudent.sname, UniStudent.name, IDF, IDF, 0.5}>, \\
&\quad <\text{UGStudent.grade, UniStudent.grade, IDF, IDF, 0.5}>\}
\end{align*}
\]

The algorithm GenerateMapping generates two GAV mappings for UniStudent and for Address over the union of PGStudent and UGStudent. Since the target schema applies horizontal partitioning, GenerateGAVMappings uses outer union to combine PGStudent and UGStudent (Figure 3.4, lines 11-12), and creates the corresponding set of columns for projection (Figure 3.4, lines 16-27). The derived GAV mappings are:

\[
\begin{align*}
&\pi_{\text{sid,sname,grade,}\text{null}}(\text{PGStudent} \cup \text{UGStudent}), \text{UniStudent} > \\
&\pi_{\text{null,sid,address}}(\text{PGStudent} \cup \text{UGStudent}), \text{Address} >
\end{align*}
\]

Note that null is assigned to UniStudent.type, since there is no associated superlexical in the source superabstracts represent student type.

To derive mappings from target to source, GenerateMapping invokes GenerateGAVMappings which applies outer join on UniStudent and Address (Figure 3.4, line 6-7). The result of the outer join operation is then applied with corresponding selection predicates provided in the correspondence to specify how to populate PGStudent, and how to populate UGStudent (Figure 3.4, lines 8-10). The final phase is to derive projected columns (Figure 3.4, lines 16-26). The derived mappings are:

\[
\begin{align*}
&\pi_{\text{sno,name,address,null}}(\sigma_{\text{type}='pg'}(\text{UniStudent} \bowtie \subseteq \text{sno} = \text{sno} \text{ Address})), \text{PGStudent} > \\
&\pi_{\text{sno,name,address,null}}(\sigma_{\text{type}='ug'}(\text{UniStudent} \bowtie \subseteq \text{sno} = \text{sno} \text{ Address})), \text{UGStudent} >
\end{align*}
\]
3.5 Discussion: Query Result Ranking with Quantified Uncertainty

As mentioned in Section 3.2, each (superabstract or superlexical) schematic correspondence is assigned with a quantified degree of uncertainty that measures the level of confidence that the associated constructs by the schematic correspondence are conceptually equivalent. The strategy for deriving and updating such degrees of uncertainty is discussed in Chapter 5. Although it is not in the scope of this dissertation, mechanisms can be designed for ranking query results retrieved by the generated views based on the quantified uncertainty. For example, a superabstract $sa_1$ from a schema $S_1$ is associated by more than one views which specify how to populate $sa_1$ with tuples in superabstracts of schema $S_2$. Queries posed on $sa_1$ will be answered by tuples from $S_2$ retrieved by the views. The question is that how to rank the retrieved tuples. Dong et al. [DHY09a] proposed a query result ranking mechanism based on the degree of uncertainty assigned to superabstract correspondences. However, they did not provide strategies for deriving and updating the quantified uncertainty.

3.6 Conclusion

In this section, we have defined and illustrated the application of an algorithm for the ViewGen operator which automates the generation of schema mappings on high-level modelled schematic correspondences. Based on the assumption that every construct can be associated by at most one correspondence, we can derive at most one relational algebra query for mapping data to each superabstract in a target schema over superabstracts in a source schema. As discussed at the end of Section 3.2, the derived relational algebra query may contain a specific combination of selection, projection, outer join and outer union operators, and we do not generate mappings between superrelationships, since superrelationships associate and describe relationships between superabstracts based on the associated superabstracts, and mappings that translate extent of superabstracts are derived with the algorithm GenerateGAVMapping. This work was published in [MBPF09] and integrated into the DSToolKit [HBM+12]. Based on the modelling of schematic correspondences, the next chapter focuses on algorithms designed for operators of model management over schematic correspondences that can be
adopt to serve dataspaces [GBM08, HBM+10].
Chapter 4

Model Management on Schematic Correspondences

Schematic correspondences can provide rich information on schematic heterogeneities. In the previous chapter, we have shown how automated mapping generation can be realised over schematic correspondences that capture the schematic heterogeneities in [KCGS95]. Mapping generation was identified in [HBM+10] as one of the generic algebraic operators that are needed to underpin the bootstrapping of dataspaces. Other operators, such as merge, compose and diff [HBM+10], were proposed in the Model Management literature [Ber03], and manipulate associations directly produced by schema matching algorithms.

The most important service that dataspaces must offer is to answer user queries against integrated schemas, but using data stored in autonomous sources. It is crucial, for this purpose that model management operators receive, manipulate and pass on information on schematic heterogeneities if the automated generation of mappings that reconcile schematic heterogeneities is to be supported. In this chapter, we address the definition of model management operators over expressive schematic correspondences.

As we have seen in Chapter 1, the bootstrapping and maintenance of dataspaces can be specified as sequences of generic operator invocations. In Section 4.1, we use some scenarios to show how model management operators can underpin the specification of basic tasks in dataspace management, for example, deriving a global schema, and propagating changes made in source schemas to the global one. Then from Sections 4.3 to 4.5, we present algorithms for merging schemas, composing correspondences and eliciting the difference between two schemas.
4.1 Dataspaces on Model Management

In the bootstrapping phase [HBF+09], dataspaces need to obtain schematic correspondences between pairs of heterogeneous data sources. Given such correspondences, mappings can be generated manually with tool support [FHH+09, ATV08] or automatically using algorithms such as the one presented in Chapter 3 [MBPF09]. Given such mappings, user queries and query results can be translated between the sources and the dataspace. This sequence of tasks needs to be performed for every source schema and global schema. It can be specified as a sequence of invocations of model management operators. For deriving mappings between a pair of schemas $S_1$ and $S_2$ the following sequence of operator invocation can be used:

1. Use match to identify a set of matches indicating the similarity between constructs of $S_1$ and $S_2$:

   \[
   \text{match}(S_1, S_2) \rightarrow <\text{MATCH}_{S_1-S_2} >
   \]

2. Use inferCorrespondences to derive schematic correspondences from the identified matches.

   \[
   \text{inferCorrespondences}(\text{MATCH}_{S_1-S_2}) \rightarrow <\text{CR}_{S_1-S_2} >
   \]

   The correspondence inference strategy proposed by [HBM+10] is represented by the operator inferCorrespondences. In addition to such strategy, mechanisms can be developed for soliciting feedback from users to incrementally refine inferred schematic correspondences [BPF+11].

3. Given the set of schematic correspondences $\text{CR}_{S_1-S_2}$, use viewGen to derive schema mappings that enable translation of queries between the related schemas

   \[
   \text{viewGen}(\text{CR}_{S_1-S_2}) \rightarrow <\text{V}_{S_1-S_2} >
   \]

   For instance, assume that $S_1$ and $S_2$ describe the same concept about student in different ways as follows:

   $S_1 : \text{Student}(\text{sid, name, age})$
S_2 : Student(sid, name)

The match operator would likely associate the attributes sid, as well as the attributes name, in each of the schemas. Based on the elicited matches, a one-to-one schematic correspondence o2osacr12 might be derived:

o2osacr12.sa1 = S1.Student
o2osacr12.sa2 = S2.Student
o2osacr12.sp1 = <S2.Student, S1.Student.age>28>
o2osacr12.sp2 = <S1.Student, *>
o2osacr12.d = 0.5
o2osacr12.LC = {<S1.Student.sid, S2.Student.sid, IDF, IDF, 0.5>,
 <S1.Student.name, S2.Student.name, IDF, IDF, 0.5>}

o2osacr12.sa1 and o2osacr12.sa2 represent the associated relations. o2osacr12.LC is the set of attribute correspondences, which indicates that age attribute of S1.Student is considered to be missing in S2.Student. Based on comparing tuples of the two superabstracts, an instance comparison matcher may conclude that S2.Student only stores students whose age is above 28, while S1.Student store tuples of students of all ages. To inform such heterogeneities, the predicate o2osacr12.sp1 is specified.

Mappings can be generated with the information provided in o2osacr12

v_{S1→S2} := <π_{sid,name}(σ_{age>28}S1.Student), S2.Student >

v_{S2→S1} := <S1.Student, π_{sid,name,null}S2.Student >

v_{S1→S2} specifies which tuples of S1.Student can populate the relation S2.Student, whilst v_{S2→S1} specifies which tuples of S2.Student can populate the relation S1.Student.

In addition to deriving mappings from correspondences, dataspaces need to support the construction of unified global schemas. In some cases, global schemas may need to include a unique representation for each concept described in the source schema, and schema mappings need to be generated between each source schema and the global schema for query translation. Given a set of source schemas S = {S_1,...,S_n}, the derivation of a global schema and annotated mappings over a set of source schemas can also be generalised as a sequence of model management
operator invocations. To illustrate the procedure, we apply it to derive a global schema over the following three source schemas:

\[ S_1 : \text{Student}(sid, name, age) \]

\[ S_2 : \text{Student}(sid, name) \]

\[ S_3 : \text{Student}(sid, name, age, address) \]

and the following correspondences, \( o2osacr12 \) and \( o2osacr23 \), are specified between the source schemas:

\[ o2osacr12.sa1 = S1.\text{Student} \]
\[ o2osacr12.sa2 = S2.\text{Student} \]
\[ o2osacr12.sp1 = \langle S2.\text{Student}, * \rangle \]
\[ o2osacr12.sp2 = \langle S1.\text{Student}, * \rangle \]
\[ o2osacr12.d = 0.5 \]
\[ o2osacr12.LC = \{ \langle S1.\text{Student}.sid, S2.\text{Student}.sid, IDF, IDF, 0.5 \rangle, \langle S1.\text{Student}.name, S2.\text{Student}.name, IDF, IDF, 0.5 \rangle \} \]

\[ o2osacr23.sa1 = S2.\text{Student} \]
\[ o2osacr23.sa2 = S3.\text{Student} \]
\[ o2osacr23.sp1 = \langle S3.\text{Student}, * \rangle \]
\[ o2osacr23.sp2 = \langle S2.\text{Student}, * \rangle \]
\[ o2osacr23.d = 0.5 \]
\[ o2osacr23.LC = \{ \langle S2.\text{Student}.sid, S3.\text{Student}.sid, IDF, IDF, 0.5 \rangle, \langle S3.\text{Student}.name, S3.\text{Student}.name, IDF, IDF, 0.5 \rangle \} \]

Assume that the attribute \textit{age} of \textit{Student} in \( S_3 \) is the same as the attribute \textit{age} in \( S_1 \).

1. Use the \textit{match} and \textit{inferCorrespondences} operators to derive schematic correspondences \( CR_{S_1-S_2} \) between \( S_1 \) and \( S_2 \).

2. Use the \textit{merge} operator [Ber03] to derive a schema \( S_G \) that contains a copy of all the constructs in \( S_1 \) and a copy of constructs in \( S_2 \) that are not associated to constructs in \( S_1 \). As well as deriving \( S_G \), \textit{merge} also derives correspondences that associate constructs in \( S_G \) to constructs in the input...
CHAPTER 4. MODEL MANAGEMENT ON SCHEMATIC CORRESPONDENCES

schemas.

\[ \text{merge}(S_1, S_2, CR_{S_1 - S_2}) \rightarrow < S_G, CR_{S_1 - S_G}, CR_{S_2 - S_G} > \]

Refer to the example, after this step, a global schema, \( S_G \), with correspondences over \( S_1 \) and \( S_2 \) are derived.

\[ S_G : \text{Student}(sid, name, age) \]

\[
\begin{align*}
o2osacr1G.sa1 &= S1.Student \\
o2osacr1G.sa2 &= SG.Student \\
o2osacr1G.sp1 &= <SG.Student, * > \\
o2osacr1G.sp2 &= <S1.Student, * > \\
o2osacr1G.d &= x \\
o2osacr1G.LC &= \{ <S1.Student.sid, SG.Student.sid, IDF, IDF, 0.5>, <S1.Student.name, SG.Student.name, IDF, IDF, 0.5>, <S1.Student.age, SG.Student.age, IDF, IDF, 0.5> \}
\end{align*}
\]

\[
\begin{align*}
o2osacr2G.sa1 &= S2.Student \\
o2osacr2G.sa2 &= SG.Student \\
o2osacr2G.sp1 &= <SG.Student, * > \\
o2osacr2G.sp2 &= <S2.Student, * > \\
o2osacr2G.d &= x' \\
o2osacr2G.LC &= \{ <S2.Student.sid, SG.Student.sid, IDF, IDF, 0.5>, <S2.Student.name, SG.Student.name, IDF, IDF, 0.5> \}
\end{align*}
\]

The \textit{age} attribute in the relation \textit{Student} of \( S_G \) is copied from \( S_1 \), since it is missing in \( S_2 \). The derivation of \( d \) will be discussed in Chapter 5.

3. For a single invocation, the \textit{merge} operator only derives a unified schema over two input schemas. Because the eventual global schema needs to have one representation for every concept described in the set of source schema, thus the derived schema has to include constructs for describing concepts that are unique to other source schema in \( S \). To elicit such constructs, we can firstly elicit, for every other source schema \( S_i \) (where \( i > 2 \)), the set of constructs that represent concepts in \( S_G \), and secondly elicit constructs in \( S_i \) are considered as missing in \( S_G \). In order to identify equivalent constructs
in $S_i$ and $S_G$, we can reuse the *match* operator. However, since the derived schema $S_G$ from *merge* is a schema that does not hold any tuples, and some matchers rely on instance comparison that is required to be supplied with data instances stored in both schema to be compared, therefore eliciting conceptually equivalent constructs in $S_i$ and $S_G$ with instance comparison is not always effective. Alternatively, the *compose* operator can be applied

$$\text{compose}(CR_{S_2-S_G}, CR_{S_2-S_i}) \rightarrow <CR_{S_G-S_i}>$$

The *compose* operator takes as input the correspondences between $S_G$ and $S_2$, and the correspondences between $S_2$ and $S_i$, and outputs a set of correspondences $CR_{S_G-S_i}$ between $S_G$ and $S_i$. Two sets of constructs $C_{12}$ and $C_i$ (respectively in $S_G$ and $S_i$) are associated by *compose* if and only if there exist a set of constructs $C_2$ in $S_2$, a correspondence $c_1$ in $CR_{S_2-S_G}$ and a correspondence $c_2$ in $CR_{S_2-S_i}$ such that $C_{12}$ and $C_2$ are associated by $c_1$, and $C_2$ and $C_i$ is associated by $c_2$.

Refer to the example, given $o2osacr2G$ and $o2osacr23$,

\[
\begin{align*}
o2osacr23.\text{sa1} &= \text{S2.Student} \\
o2osacr23.\text{sa2} &= \text{S3.Student} \\
o2osacr23.\text{sp1} &= <\text{S3.Student}, * > \\
o2osacr23.\text{sp2} &= <\text{S2.Student}, * > \\
o2osacr23.\text{d} &= 0.5 \\
o2osacr23.\text{LC} &= \{<\text{S2.Student.sid}, \text{S3.Student.sid}, \text{IDF}, \text{IDF}, 0.5>, \\
&\quad <\text{S3.Student.name}, \text{S3.Student.name}, \text{IDF}, \text{IDF}, 0.5>\}
\end{align*}
\]

we can apply the *compose* operator to derive a correspondence between $S_3$ and $S_G$ (step 3):

\[
\begin{align*}
o2osacr3G.\text{sa1} &= \text{S3.Student} \\
o2osacr3G.\text{sa2} &= \text{S.G.Student} \\
o2osacr3G.\text{sp1} &= <\text{S.G.Student}, * > \\
o2osacr3G.\text{sp2} &= <\text{S3.Student}, * > \\
o2osacr3G.\text{d} &= x'' \\
o2osacr3G.\text{LC} &= \{<\text{S3.Student.sid}, \text{S.G.Student.sid}, \text{IDF}, \text{IDF}, 0.5>, \\
&\quad <\text{S3.Student.name}, \text{S.G.Student.name}, \text{IDF}, \text{IDF}, 0.5>\}
\end{align*}
\]
4. Although the \textit{compose} operator does not need the support of structural or instance comparison mechanisms or comparison resources (e.g. dictionaries) to elicit associations, it may miss out potential associations between constructs in \( SG \) (copied from \( S_1 \)) that are missing in \( S_2 \) and constructs in \( S_i \) that are missing in \( S_2 \). Thus, the model management operator \textit{diff} needs to be invoked to identify missing constructs.

\[
diff(S_G, S_i, CR_{SG-S_i}) \rightarrow <MC_{SG}, MC_{Si}>
\]

\( MC_{SG} \) denotes the set of constructs in \( S_G \) that are missing in \( MC_{Si} \), whilst \( MC_{Si} \) denotes the set of constructs in \( S_i \) that are missing in \( S_G \).

5. Given the set of constructs of \( S_i \) missing in \( S_G \), we need to elicit a set of correspondences \( CR'_{SG-S_i} \) between \( MC_{Si} \) and \( S_G \) by using \textit{match} and \textit{inferCorrespondences}, and add \( CR'_{SG-S_i} \) to \( CR_{SG-S_i} \).

Refer to the example, the \textit{diff} operator (step 4) should derive \( MC_3 = \{age, address\} \). Since \textit{age} in \( S_3 \) is equivalent to \textit{age} in \( S_G \), an attribute correspondence is created and added to \textit{o2osacr3G} at step 5.

6. Some constructs in \( MC_{Si} \) has been associated to \( S_G \) in the previous step. Since the global schema needs to have at most one representation for every construct in the source schema, each construct in \( MC_{Si} \) that has no association to constructs in \( S_G \), needs to be copied into \( S_G \) and be associated to the original source constructs with correspondences. To elicit the missing construct, \textit{diff} is invoked again:

\[
diff(S_G, S_i, CR_{SG-S_i}) \rightarrow <MC_{SG}, MC_{Si}>
\]

\( MC_{Si} \) contains constructs that are not associated to any construct in \( S_G \) after invoking \textit{match} and \textit{inferCorrespondence} on \( MC_{Si} \) and \( S_G \) in the previous step, and is copied to \( S_G \):

\[
copy(S_G, MC_{Si}) \rightarrow <S_G, CR'_{SG-S_i}>
\]

Note that \( CR'_{SG-S_i} \) contains correspondences associating constructs of \( S_i \) in \( MC_{Si} \) to constructs in \( S_G \) that are copied from \( MC_{Si} \), and needs to be added into \( CR_{SG-S_i} \). Refer to the example, since the attribute \textit{address} in
$S_3$ is missing in $S_G$, the final step is to copy $address$ into $S_G$ and derive a correspondence $o2osacr3G$. Therefore, the final form of $S_G$ and $o2osacr3G$ is:

\[ S_G : \text{Student}(\text{sid, name, age, address}) \]

\[
o2osacr3G.sa1 = S3.\text{Student} \\
o2osacr3G.sa2 = S_G.\text{Student} \\
o2osacr3G.sp1 = <S_G.\text{Student}, * > \\
o2osacr3G.sp2 = <S3.\text{Student}, * > \\
o2osacr3G.d = x'
\]

\[
o2osacr3G.LC = \{<S3.\text{Student}.\text{sid}, S_G.\text{Student}.\text{sid}, \text{IDF}, \text{IDF}, 0.5>, <S3.\text{Student}.\text{name}, S_G.\text{Student}.\text{name}, \text{IDF}, \text{IDF}, 0.5>, <S3.\text{Student}.\text{address}, S_G.\text{Student}.\text{address}, \text{IDF}, \text{IDF}, 0.5>\}
\]

We have demonstrated the flexibility of model management operators for supporting the derivation of mappings and global schemas for dataspaces. Alternatively, these operators can support the update of mappings in the case of schema evolution. This was shown by Gubanov et al. [GBM08]. For example, once the mappings and global schema are derived, whenever a source schema is changed by, e.g., deleting or adding constructs, such a change needs to be propagated to the global schema, and to all the mappings between the global schema and other source schemas, in order to maintain the consistency across the dataspace.

Given a global schema $S_G$ and a source schema $S_i$, together with the set of correspondences, $CR_{S_G-S_i}$, between them, once $S_i$ is changed to $S_i'$, we can update $CR_{S_G-S_i}$ and correspondingly update $S_G$ with model management operators as follows:

1. $S_G$ needs to be extended with constructs that were added to the source schema, and with correspondences to the new schema. To achieve this, we can firstly derive a set of correspondences, $CR_{S_i-S_i'}$, that associates $S_i$ and $S_i'$ with the match and inferCorrespondence operators.

2. We then apply, the $\text{diff}$ operator to identify, the constructs that have been added to $S_i$:

\[ \text{diff}(S_i, S_i', CR_{S_i-S_i'}) \rightarrow <MC_{S_i}, MC_{S_i'}> \]
3. To create correspondences between the new schema $S_i$ and the global schema $S_G$ the \textit{compose} operator is applied:

$$\text{compose}(CR_{SG-S_i}, CR_{Si-S_i'}) \rightarrow CR_{SG-S_i'}$$

4. The newly added constructs in $MC_{Si'}$ are copied to $S_G$ with correspondences, and correspondences are derived for associating constructs in $MC_{Si'}$ to constructs in $S_G$:

$$\text{copy}(SG, MC_{Si'}, CR_{SG-S_i'}) \rightarrow <SG, CR'_{SG-S_i'}>$$

Lastly, the derived correspondences, $CR'_{SG-S_i'}$, are inserted into $CR_{SG-S_i'}$.

Once the global schema is extended in accordance to the changes in the source schema, correspondences between the global schema and other source schemas need to be updated. This can also be achieved using the procedures presented above.

We have shown that model management operators, complemented by an operator for generating schema mappings, can be used to underpin the functional decomposition of dataspace bootstrapping and maintenance phase. The aim of the reminder of this chapter is to define algorithms for model management operators over schematic correspondences that capture expressive schematic heterogeneities so that they can be taken as input to generate mappings reconciliation.

### 4.2 Schema Merging

The model management \textit{merge} operator derives a reconciled schema representing the union of the information content in the two input schemas without creating duplicate representations. Schema merging is one option for integrating databases. For instance, if two organisations decide to merge, they will need to create a single database to store information from their originally separate database. The organisations may initially want to migrate all the information in the old database losslessly to the new database. The schema used for describing the information stored in the new database can be derived with \textit{merge} from the original schemas. Apart from deriving a schema, \textit{merge} derives correspondences
between the derived schema and the input schemas which can be used to generate executable mappings for supporting data migration. The input schemas may adopt heterogeneous representations for the same information, so experts or automated algorithms need to be applied to derive schematic correspondences between the input schemas before merge can be invoked so that the derived schemas can conform to the requirements that no duplicate representation for the same information is created.

In dataspaces, the derivation of a global schema (as discussed, in the previous section) can be achieved by invoking merge together with other model management operators. Pottinger et al. have analysed what constitutes a useful global schema [PB08]. Deriving the global schema as a union of all the constructs in the source schemas is one such way. However, it can create redundant representations of the same concept. This can potentially increase the complexity for users to understand concepts available, and formulate the right queries for the desired information. Therefore, each concept should have exactly one representation in the global schema with correspondences to each source representation that corresponds to that construct. For example, given three source schemas:

\[
S_1 : \text{Student}(sid, name, age), \text{Lecturer}(lid, name)
\]

\[
S_2 : \text{Student}(sid, name)
\]

\[
S_3 : \text{Student}(sid, name, age, address)
\]

one concept represented in these schemas is student. In this case, one superabstract should be created in the global schema for describing student. In addition, there may be concepts that are unique to one data source and should still be represented in the global schema. In the above example, only \( S_1 \) represents the concept Lecturer. As for superabstracts, if the same attribute is differently represented in different source schemas, one single representation should be used in the global schema, and attributes that are unique to a data source should have exactly one representation in the global schema.

It is inevitable that there are conflicts between the reconciled representation of concepts and the representations used by the sources. For example, \( S_3 \) may apply the horizontal partitioning technique for representing students as undergraduates and postgraduates. If students are represented with one relation in the reconciled schema, then there is an one-to-many conflict between the reconciled schema and
If $merge$ is used for deriving a global schema, it is useful if $merge$ can also derive schematic correspondences between the output reconciled schema and source schemas, so that the schematic correspondences can be input to generate mappings.

Most of existing techniques that implements $merge$ separate the derivation of schematic correspondences from $merge$, e.g. with manual resources or invoke the $match$ operator [SRM08, QK07] which may create upfront cost. Alternatively, the $merge$ operator can infer schematic heterogeneities based on input schematic correspondences. For example, since $S_3$ uses horizontal partitioning for representing students, and if student is represented with one relation in the reconciled schema, thus there is an one-to-many horizontal partitioning conflict between $S_3$ and the reconciled schema. On the other hand, if $S_3$ uses vertical partitioning for representing students, there is a one-to-many vertical partitioning conflict between the reconciled schema and $S_3$. How to combine tuples of relations in $S_3$ for populating $Student$ in $S_2$ can be propagated to the output schematic correspondence.

Finally, superrelationships should be created by the $merge$ operator, to relate corresponding superabstracts in the sources. For example, $Student$ in $S_1$ may be associated to $Lecturer$ with a superrelationship $taughtBy$. Once the superabstracts $Student'$ and $Lecturer'$ are created in the reconciled schema, they should be related by superrelationship $taughtBy$ too.

In the reminder of this section, we firstly specify general requirements for the reconciled schema. Next, we specify the $merge$ algorithm, and show that it can be used to derive reconciled schema over input schemas that carry different kinds of heterogeneities and the correspondences that relate the former to the latter. Finally, we discuss the related work on $merge$ operators.

### 4.2.1 Requirements

The motivation and example shown in the previous section lead to the following requirements that the reconciled schema must satisfy:

1. The reconciled schema must describe the union of the information content of the input schemas. We want to ensure that no information is lost in the reconciled schema. This can be done by ensuring that, for each concept or
attribute of a concept described in a source schema, there exists a correspondence that associates the constructs used to represent such a concept or attribute of a concept in the reconciled schema to constructs used to represent the same concept and attribute of a concept in the source schema. Given such a correspondence, a mapping can be generated for translating queries posed on the reconciled schema to a valid query against the source schema. The concept or the attribute of a concept must be represented by a single superabstract or a single superlexical in the reconciled schema.

2. The reconciled schema must expose one representation for any concept or attribute of a concept that is described in more than one input schema. As with the previous requirement, this can be satisfied by ensuring that there is one correspondence associating a constructs in the reconciled schema to every source construct representing the same concept or attribute.

3. A construct that represents unique concepts and attributes of a concept should have one copy in the reconciled schema. To satisfy this requirement, we need to ensure that there is a one-to-one correspondence associating any superabstract or superlexical in a source schema not associated by any correspondence to a construct in any other schema.

4. For every superrelationship in a source schema associating two superabstracts $sa_{i1}$ and $sa_{1j}$ in a schema $S_1$, there must be at most one superrelationship in the reconciled schema relating superabstracts that are associated to $sa_{i1}$ and $sa_{1j}$ by a superabstract correspondence.

We do not claim that users of the *merge* operator will require the reconciled schema to satisfy all of these criteria in all scenarios. This set of requirements is mostly used for validating the design of schema merging mechanisms in the existing literature [PB08, QK07] and other requirements can be made (as mentioned in the following subsection). In addition to requirements on the reconciled schema, we also specify the requirements for the output schematic correspondences.

The correspondences that are output by *merge* are meant to be used by algorithms for generating schema mappings, among others. Based on this motivation, the output schematic correspondences must satisfy the following requirements:

1. Given a reconciled superabstract $sa_R$, which represents the concept represented by two sets of source superabstracts $SA_1$ and $SA_2$ associated by a
many-to-many superabstract correspondence \( sacr \), two many-to-one superabstract correspondences \( sacr_{1-R} \) and \( sacr_{2-R} \) must be created to associate each set of source superabstracts to the reconciled superabstract. Moreover, if \( sacr \) provides information on the partition method, join predicates (that specify how to join vertically partitioned superabstracts) or selection predicates (that specify which tuples of a superabstract can populate another superabstract, or tuples derived from combining a set of superabstracts, in the other schema), this information must be captured in \( sacr_{1-R} \) and \( sacr_{2-R} \).

2. Given a reconciled superabstract \( sa_R \), which represents the concept represented by two source superabstracts \( sa_1 \) and \( sa_2 \) associated by a one-to-one superabstract correspondence \( sacr \), two one-to-one superabstract correspondences \( sacr_{1-R} \) and \( sacr_{2-R} \) should be created to associate each of the source superabstracts to the reconciled superabstract. Moreover, selection predicates that must be represented in \( sacr \) must be captured in \( sacr_{1-R} \) and \( sacr_{2-R} \).

3. Given a reconciled superabstract \( sa_R \), which represents the concept represented by source superabstracts \( sa_1 \) and \( SA_2 \) associated by a one-to-many superabstract correspondence \( sacr \), a one-to-one superabstract correspondence \( sacr_{1-R} \) and a many-to-one superabstract correspondence \( sacr_{2-R} \) must be created to associate the source superabstracts to the reconciled superabstract. Moreover, the partition method, join predicates and selection predicates that must be represented in \( sacr \) must be captured in \( sacr_{2-R} \), whilst the corresponding selection predicates in \( sacr \) must be captured in \( sacr_{1-R} \). Analogous, requirements apply to an input many-to-one superabstract correspondence.

4. Given a reconciled superlexical \( sl_R \), which represents an attribute of a concept represented by two sets of source superlexicals \( SL_1 \) and \( SL_2 \) associated by a many-to-many superlexical correspondence \( slcr \), two many-to-one superlexical correspondences, \( slcr_{1-R} \) and \( slcr_{2-R} \) must be created to associate each set of source superlexicals to the reconciled superlexical.
5. Given a reconciled superlexical $sl_R$, which represents an attribute of a concept represented by two source superlexicals $sl_1$ and $sl_2$ associated by a one-to-one superlexical correspondence $slc_r$, two one-to-one superlexical correspondences $slc_{r1-R}$ and $slc_{r2-R}$ must be created to associate each of the source superlexicals to the reconciled superlexical.

**Naming Scheme for Reconciled Constructs**

The name of a construct informs the concept and data it represents. Likewise, each derived reconciled construct must have a name to inform what it represents. If the reconciled construct is derived from two conceptually equivalent source constructs that have the same name, the reconciled construct can be named with the name of one of the sources, since the source constructs are equivalent. In the situation when there is a different-name-same-construct conflict, the reconciled construct should be named in a way that expresses the union of the names of the source constructs, since in dataspaces manual intervention and consulting dictionaries may not be available, and we do not know which name is more appropriate to be used to name the reconciled construct. For example, *Student* and *Scholar* are both candidate names for naming the reconciled superabstract that represent students. The reconciled superabstract can be assigned with a name e.g. as the concatenation of the strings *Student* and *Scholar*.

The same problem happen for reconciling one-to-many or many-to-many conflicts. If the superabstract *Student* corresponds to *undergraduate* and *postgraduate*, the reconciled superabstract would represent information about both kinds of students, and the concatenation of the names used by the source superabstacts inform what is represented by the reconciled superabstract. Given this observation, the merge algorithm names a reconciled construct as the concatenation of unique names of source constructs delimited by “|”. Thus, if two constructs in a many-to-many conflict have the same name, only one of the name will be concatenated to the name of the reconciled construct. Two reconciled names are equivalent, if they are concatenated from the same set of distinctive names.

If a reconciled construct is a copy of a source construct, it is assigned with the name of the source construct.

We have presented requirements for merging two input schema to satisfy. In the next section, we discuss the algorithm that implements the operator.
Algorithm MergeSchemas
inputs:
S_1 and S_2: two source schema; CR_{s1}: a set of schematic correspondences between the source schema
outputs:
S_r: a schema; CR_{sr}: a set of schematic correspondences between S_1 and S_2; CR_{sr'}: a set of schematic correspondences between S_1 and S_2.

begin
1. S_r = new Schema()
2. CR_{sr} = new Set()
3. FOR EACH superabstract correspondence sacr ∈ CR_{s1} DO
   4. <sa_r, sacr_{s1}, sacr'_{s1}> ← DeriveReconciledSuperabstract(sacr)
   5. S_r, SA ← S_1, SA ∪ sa_r, CR_{sr} ← CR_{s1} ∪ sacr_{s1} ∪ CR_{sr}' ∪ sacr'_{s1}
   6. FOR EACH superabstract sa_r ∈ S_r is not associated to S_1 DO
      7. <sa_r, sacr'_{s1}, sacr''_{s1}> ← CopySuperabstract(sa_r)
      8. S_r, SA ← S_1, SA ∪ sa_r, CR_{sr} ← CR_{sr} ∪ sacr''_{s1}
      9. FOR EACH superabstract sa_r ∈ S_r is not associated to S_1 DO
         10. <sa_r, sacr'_{s1}, sacr''_{s1}> ← CopySuperabstract(sa_r)
         11. S_r, SA ← S_1, SA ∪ sa_r, CR_{sr} ← CR_{sr} ∪ sacr''_{s1}
      12. FOR EACH superrelationship sr_r ∈ S_r DO
         13. sr_r ← DeriveSuperrelationship(sr_r, S_r, CR_{sr})
      14. S_r ← S_r, SR ∪ sr_r
      15. FOR EACH superrelationship sr_r ∈ S_r DO
         16. sr_r ← DeriveSuperrelationship(sr_r, S_r, CR_{sr})
      17. S_r ← S_r, SR ∪ sr_r
   18. RETURN S_r, CR_{sr}, CR_{sr'}

Figure 4.1: Algorithm for deriving a reconciled schema

4.2.2 Generate a Reconciled Schema

In this section, we present the algorithm (Figure 4.1-4.5) for deriving a reconciled schema over schematic correspondences. The algorithm is designed in accordance to the requirements stated in the previous section. At the general level, it firstly derives a reconciled superabstract for describing every concept that is described by superabstracts from both of the input schemas (Figure 4.1 line 3-5 and Figure 4.2 and 4.3). This task is performed based on iterating through each superabstract correspondence given in the input, and is further divided into

1. A subroutine for deriving a reconciled superabstract with a one-to-one or
Algorithm DeriveReconciledSuperabstract
inputs:
  sacr: a superabstract correspondence
outputs:
  sa_r: a reconciled superabstract, sacr_r: a superabstract correspondence between source
constructs of sacr to sa_r, sacr_r: a superabstract correspondence between target constructs of
sacr to sa_r
begin
1  sa_r = new superabstract(), Initializing sacr_r and sacr_r
2  IF getCorrCardinality(sacr) ∈ {'one-to-one', 'one-to-many', 'many-to-one') THEN
3      sa_r.name = concatenate(sacr.SA_name, sacr.SA_2.name)
4      IF getCorrCardinality(sacr) ∈ {'one-to-one') THEN
5          //creating two one-to-one correspondences
6          sacr_r ← <sacr.sa_r, sa_r,<sa_r>,<sacr.sa_r>,LC>
7          sacr_r ← <sacr.sa_r, sa_r,<sa_r>,<sacr.sa_r>,LC>
8      IF getCorrCardinality(sacr) ∈ {'many-to-one') THEN
9          //creating a many-to-one and an one-to-one correspondences
10         sacr_r ← <sacr.SA_r, sa_r,<sacr.pt_r, sacr.SP_r,LC>
11         sacr_r ← <sacr.sa_r, sa_r,<sa_r>,<sacr.sa_r>,LC>
12      IF getCorrCardinality(sacr) ∈ {'one-to-many') THEN
13          //creating an one-to-one and a many-to-one correspondences
14         sacr_r ← <sacr.sa_r, sa_r,<sa_r>,<sacr.sa_r>,LC>
15         sacr_r ← <sacr.SA_r, sa_r,<sacr.pt_r, sacr.SP_r,LC>
16 FOR EACH superlexical correspondence slc_r ∈ sacr.LC DO
17     FOR EACH superlexical sl_r ∈ slc_r
18         & sl_r is not associated to any superlexicals in sa_r DO
19          sl_r = new superlexical(sl_r.name, sl_r.domain, sl_r.isKey)
20          sa_r.SL ← sa_r.SL ∪ sl_r
21          // creating new correspondence with IDF’s, since sl_r is a copy of sl_r
22          slc_r ← <slc_r, sl_r, IDF, IDF>
23          slc_r = <slc_r.SL_r, sl_r, slc_r, slc_r.idf_r, slc_r.idf_r, LC_sacr_r, LC_sacr_r, LC_sacr_r, LC_sacr_r>
24 FOR EACH superlexical sl in sacr.SA_r
25     or in sacr.SA_r, not associated by any correspondences DO
26          sl_r = new superlexical(sl.name, sl.domain, sl.isKey)
27          sa_r.SL ← sa_r.SL ∪ sl_r
28          slc_r ← <slc_r, sl_r, IDF, IDF>
29          IF sl in sacr.SA_r THEN sacr_r ← sacr_r, LC ← sacr_r, LC ∪ slc_r
30          ELSE sacr_r ← sacr_r, LC ← sacr_r, LC ∪ slc_r
31  IF getCorrCardinality(sacr) ∈ {'many-to-many') THEN
32      <sa_r, sacr_r, sacr_r> ← ReconcilingM2MSuperabstractCorrespondence(sacr)
33 RETURN sa_r, sacr_r, sacr_r

Figure 4.2: Subroutine for deriving a reconciled superabstract
Algorithm ReconcilingM2MSuperabstractCorrespondence

inputs:
- \( \text{sacr} \): a many-to-many superabstract correspondence

outputs:
- \( \text{sa}_a \): a reconciled superabstract; \( \text{sacr}_{sa} \): a superabstract correspondences between source constructs of \( \text{sacr} \) to \( \text{sa}_a \); \( \text{sacr}_{st} \): a superabstract correspondence between target constructs of \( \text{sacr} \) to \( \text{sa}_b \);

begin
1. \( \text{sa}_a = \text{new superabstract}() \), initialising \( \text{sacr}_{sa} \) and \( \text{sacr}_{st} \)
2. IF \( \text{sacr}.p_i = \text{"VP"} \) THEN
3. \( \text{sa}_a = \bigcup \text{sacr}.SA_i \)
4. IF \( \text{sacr}.p_i = \text{"HP"} \) THEN
5. \( \text{sa}_a = \bigcup \text{sacr}.SA_i \)
6. // creating two many-to-one superabstract correspondences
7. \( \text{sacr}_{sa} = <\text{sacr}.SA_p, \text{sa}_a, \text{sacr}.p_i, \text{sacr}.JP_i, \text{sacr}.SP_p, \text{LC}> \)
8. \( \text{sacr}_{st} = <\text{sacr}.SA_p, \text{sa}_a, \text{sacr}.p_i, \text{sacr}.JP_i, \text{sacr}.SP_p, \text{LC}> \)
9. FOR EACH superlexical \( \text{sl}_i \in \text{sa}_a \) DO
10. FOR EACH superlexical \( \text{sl}_i \in \text{sacr}.SA \) and \( \text{sl}_i = \text{sl}_i \) DO
11. \( \text{slc}_r_{sl} = <\text{sl}_i, \text{sl}_i', \text{IDF}, \text{IDF}> \) // creating new correspondence
12. \( \text{sacr}_{sa}, \text{LC} \leftarrow \text{sacr}_{sa}, \text{LC} \cup \text{slc}_r_{sl} \)
13. FOR EACH superlexical correspondence \( \text{slc}_r \in \text{sacr}.LC \) DO
14. IF \( \text{sl}_i \) is associated by \( \text{slc}_r \) THEN
15. \( \text{slc}_r_{sl} = <\text{slc}_r, \text{sl}_i, \text{sl}_i', \text{IDF}, \text{IDF}> \)
16. \( \text{sacr}_{sa}, \text{LC} \leftarrow \text{sacr}_{sa}, \text{LC} \cup \text{slc}_r_{sl} \)
17. FOR EACH superlexical \( \text{sl}_i \) in \( \text{sacr}.SA \), not associated by any correspondences DO
18. \( \text{sl}_i = \text{new superlexical}(\text{sl}_i, \text{name}, \text{sl}_i, \text{domain}, \text{sl}_i, \text{isKey}) \)
19. \( \text{sa}_a, \text{SL} \leftarrow \text{sa}_a, \text{SL} \cup \text{sl}_i \)
20. \( \text{slc}_r = <\text{sl}_i, \text{IDF}, \text{IDF}> \)
21. IF \( \text{sl}_i \) in \( \text{sacr}.SA \), THEN \( \text{sacr}_{sa}, \text{LC} \leftarrow \text{sacr}_{sa}, \text{LC} \cup \text{slc}_r \)
22. ELSE \( \text{sacr}_{sa}, \text{LC} \leftarrow \text{sacr}_{sa}, \text{LC} \cup \text{slc}_r \)
23. RETURN \( \text{sa}_a, \text{sacr}_{sa}, \text{sacr}_{st} \)

Figure 4.3: Subroutine for reconciling many-to-many superabstract correspondence
one(many)-to-many(one) superabstract correspondences (Figure 4.2). It creates a reconciled superabstract $s_{AR}$ to be added to the reconciled schema $S_R$. Following from this, if $sacr$ is a one-to-one superabstract correspondence, the subroutine creates two one-to-one superabstract correspondences to associate $s_{AR}$ to each of the source superabstract (Figure 4.2, line 4-7). If $sacr$ is either an one-to-many correspondence, the subroutine creates a one-to-one superabstract correspondence to associate the one side of the associated source superabstracts to $s_{AR}$, and creates an many-to-one superabstract correspondence to associate the many side of the associated source superabstracts to $s_{AR}$ (Figure 4.2, line 12-15). Meanwhile, it derives join predicates or selection predicates for the new many-to-one superabstract correspondence based on information given in the superabstract associating source superabstracts (Figure 4.2, line 10 and line 15). For example, if $sacr$ associates $sa_{11}$ and $sa_{12}$ in $S_1$ to $sa_{21}$ in $S_2$, where the former uses vertical partitioning to represent the same concept as the latter. Thus, $sacr$ may provide join predicates that specify how to join $sa_{11}, sa_{12}$. If $s_{AR}$ is the reconciled superabstract created by the algorithm for the source superabstracts, it should create a many-to-one superabstract correspondence to associate $sa_{11}, sa_{12}$ and $s_{AR}$, contain the same join predicates as specified by $sacr$.

The subroutine then iterates over all the superlexical correspondences to create superlexicals to represent every attribute of the concept described by the associated superabstracts (Figure 4.2, line 16-24). It copies superlexicals of one side into the reconciled schema, and constructs superlexical correspondences for associating each source superlexical to its copied superlexical in the reconciled schema. Because it associates a source superlexical to its copy, the created superlexical correspondence uses identity functions to map their data values. Furthermore, the superlexical functions given in the input correspondences can be propagated to the derived correspondences between the reconciled superlexicals and source superlexicals. For example, assume that the function $f_{s2\rightarrow 1}$ specifies how to compute data values of $sl_1$ by data values of $sl_2$. If $sr_1$ is copied into the reconciled schema (for representing data values of $sl_1$ and $sl_2$) as $sl_{R}$, then the same function can be used for specifying how to compute data values of $sl_{R}$ by data values of $sl_2$. Finally, the subroutine creates one copy of every source superlexical that is
Algorithm CopySuperabstract
inputs:
    sa: superabstract
outputs:
    sa′: a reconciled superabstract; sacr a superabstract correspondence
begin
1. \( SL = new \text{Set}() \), \( sa_{\text{r}} = new \text{superabstract}(sa.\text{name}, SL) \) // initialising \( sa \) and \( SL \), a set of superlexicals
2. \( sacr = <sa, sa_{\mu},<sa_{\nu}>,<sa_{\nu}^*,LC> \) // initialising \( sacr \)
3. FOR EACH superlexical \( sl \) in \( sa \) DO
4. \( sl_{\mu} = new \text{superlexical}(sl.\text{name}, sl.\text{domain}, sl.\text{isKey}) \)
5. \( sa_{\text{r}}.SL \leftarrow sa_{\text{r}}.SL \cup sl_{\text{r}} \)
6. // creating a new superlexical correspondence
7. \( slcr = <sl, sl_{\mu}.\text{IDF},IDF> \)
8. \( sacr.LC \leftarrow sacr.LC \cup slcr \)
9. RETURN \( sa_{\text{r}}, sacr \)

Figure 4.4: Subroutine for copying a superabstract to the reconciled schema

not associated by any superlexical correspondences (Figure 4.3, line 26-32).

2. A subroutine for deriving a reconciled superabstract with a many-to-many superabstract correspondence (Figure 4.3). In this situation, the reconciled superabstract is derived as the combination of source superabstractions \( (sacr.SA_{1}) \) associated by the correspondence either by \emph{join} or \emph{union} depending on the partitioning method indicated in the many-to-many superabstract correspondence (Figure 4.3, line 2-5). As in the previous case, the superabstract correspondences created for associating the reconciled superabstract and source superabstractions indicate how to combine source constructs to populate the reconciled superabstract (Figure 4.3, line 8 and line 11) based on information given in the input many-to-many superabstract correspondence. Finally, It uses the same procedures for reconciling superlexicals as in the previous case.

Followed from deriving reconciled superabstractions, the algorithm invokes the subroutine \emph{copySuperabstract} (Figure 4.1, line 7 and line 10) to create a representation for every superabstract, \( sa \), that is unique to each of the source schema (Figure 4.4). A superabstract \( sa' \) is a copy of another superabstract \( sa \), if and only if they have the same name and the same set of superlexicals. A superlexical
Algorithm DeriveSuperrelationship

inputs:
  sr: a source superrelationship; S_r: the reconciled schema; CR: a set of schemematic
correspondences between the source schema of sr and the reconciled schema

outputs:
  sr_r: a superrelationship

begin

1. initialising sa_m, sa_e
2. FOR EACH sacr \in CR \do
3.   IF sr.sourceSuperabstract \in sacr.sourceSuperabstract THEN
4.     sa_m = sacr.targetSuperabstract
5.   FOR EACH sacr \in CR \do
6.     IF sr.targetSuperabstract \in sacr.sourceSuperabstract THEN
7.     sa_e = sacr.targetSuperabstract
8.   exist \leftarrow false
9.   FOR EACH superrelationship sr \in S_r \do
10.    IF sr.relationshipType = sr_r.relationshipType
11.       exist \leftarrow true
12.   IF exist \& sa_m \models sa_e THEN
13.     sr = new superrelationship(sr_r.name, sr_r.relationshipType)
14.     sr.sourceSuperabstract = sa_m
15.     sr.targetSuperabstract = sa_e
16. RETURN sa_e

Figure 4.5: Subroutine for deriving reconciled superrelationship

sl' is a copy of another superlexical sl, if and only if they have the same name
and domain. This algorithm will create a superabstract in the reconciled schema
with the same name and the same set of superlexicals as sa, but will not copy
tuples of sa to the created superabstract. When copying a superlexical sl of sa
to the derived superabstract, the algorithm iterates through superlexicals of sa,
and creates a copy of each one in the created superabstract (Figure 4.4, line 3-7).

Finally, the algorithm derives superrelationships for the reconciled schema
(Figure 4.1, line 12-14 and 15-17). It iterates over every superrelationship, sr,
in the source schema. If two of the related superabstracts, sa_1 and sa_2, have
associated superabstracts, sa_{R1} and sa_{R2}, in the reconciled schema which are not
CHAPTER 4. MODEL MANAGEMENT ON SCHEMATIC CORRESPONDENCES

associated by a superrelationship of the same type as \( sr \), it creates a superrelationship, \( sr_R \), of the same type as \( sr \) to associate \( sa_{R1} \) and \( sa_{R2} \) in the reconciled schema. The subroutine firstly iterates through each superabstract correspondence between the source and the reconciled schema to identify the reconciled superabstract, \( sa_{R1} \), that is associated to \( sa_1 \) (Figure 4.5, line 2-4). It secondly identify \( sa_{R2} \) with the second for-loop (Figure 4.5, line 5-7). It then iterates through the reconciled schema and checks if there is already a superrelationship associating \( sa_{R1} \) and \( sa_{R2} \) with the same type as \( sl \) in the reconciled schema (Figure 4.5, line 8-11). If the same type of superrelationship does not exist in the reconciled schema, it creates one in the reconciled schema (Figure 4.5, line 12-15) to associate \( sa_{R1} \) and \( sa_{R2} \).

Summary

The reconciled schema derived by the merge algorithm can satisfy the requirements presented in Section 4.3.1. Firstly, it creates one representation for every concept (Figure 4.1, line 3-5, and Figure 4.2 and 4.3), every attribute of a concept (Figure 4.2, line 17-25 and Figure 4.3, line 13-16) and relationships relate concepts (Figure 4.1, line 15-17) that are described by constructs in both input schema. Secondly, it creates one copy for every construct that is unique to one of the input schemas (Figure 4.1 line 6-11, Figure 4.2 line 26-32 and Figure 4.1, line 15-17). In the following section, we use scenarios to demonstrate the algorithms for merging two schemas that contain different kinds of schematic heterogeneities.

Correspondences derived by the algorithm can satisfy requirements presented in section 4.3.1. It derives one-to-many superabstract correspondences for associating the reconciled superabstract to superabstracts associated by every input many-to-many superabstract correspondences (Figure 4.4) and derives information on schematic heterogeneities based on the input correspondences (Figure 4.4, lines 2-12). It derives one-to-one superabstract correspondences or many-to-one superabstract correspondences for every one-to-one or many(one)-to-one(many) superabstract correspondence in the input correspondence set (Figure 4.3). Finally, it derives many-to-one or one-to-one superlexical correspondences for the input superlexical correspondences, and includes superlexical functions derived based on the input correspondences (Figure 4.3 17-25, and Figure 4.4 line 13-20).

The complexity of the algorithm depends on the number of constructs in the input schema and schematic correspondences. It iterates over each superabstract
correspondence, \( sacr \), in \( CR_{1-2} \) (Figure 4.1, line 3), and when it is creating the reconciled superabstract (Figure 4.2 and Figure 4.3), it iterates only once over each superlexical correspondence given in \( sacr.LC \) that associates superlexicals of the associated source superabstracts. In other words, it does not iterate over all superabstract or superlexical correspondences in \( CR_{1-2} \) again, when a new superabstract is encountered (Figure 4.1, line 3). Thus, the algorithm iterates over every correspondence in \( CR_{1-2} \) once for deriving reconciled constructs in \( S_R \).

Meanwhile, when creating representations of unique source superabstracts, the algorithm iterates over every superabstract once in both source schemas, \( S_1 \) and \( S_2 \), to identify all the unique superabstracts (Figure 4.1, line 6 and 9). Once it finds a unique superabstract, it then iterates its superlexicals to create a copy of the unique superabstract in the reconciled schema. Finally, it iterates over every superrelationship of \( S_1 \) and \( S_2 \) once to create reconciled superrelationships in the reconciled schema (Figure 4.1, line 12 and 15).

Therefore, let \( A \) be the total number of superabstracts of \( S_1 \) and \( S_2 \), let \( L \) be the total number of superlexicals of \( S_1 \) and \( S_2 \), let \( R \) be the total number of superrelationships of \( S_1 \) and \( S_2 \), and let \( C \) be the number schematic correspondences, the running time of the algorithm is \( O(A \times L + R + C) \).

### 4.2.3 Scenario Study

In Figure 4.6, two source schemas need to be merged to form a single schema. \( S_1 \) describes information about students using two superabstracts, \( Student \) and \( Address \). \( Address \) holds contact information of each student, and refers to each student tuple with the foreign key \( sid \). \( take \) is a superrelationship that associates \( Student \) to \( Course \). In \( S_2 \), information on student is represented with two superabstracts \( ugStudent \) and \( pgStudent \). Both superabstracts are associated to \( Course \) with superrelationship \( ugTake \) and \( pgTake \). \( S_2 \) furthermore describes information about the lecturers of each course with the superasbtract \( Lecturer \) which is associated to \( Course \) with the superrelationship \( teach \).

The follow schematic correspondence is given:

\[
\begin{align*}
\text{m2msacr1.SA1} &= \{\text{Student, Address}\} \\
\text{m2msacr1.SA2} &= \{\text{ugStudent, pgStudent}\} \\
\text{m2msacr1.pt1} &= 'VP'
\end{align*}
\]
Figure 4.6: Two source schema

$$m2msacr1.pt2 = 'HP'$$
$$m2msacr1.JP1 = \{<\text{Student}.sid, \text{Address}.sid, =>\}$$
$$m2msacr1.SP1 = \{<\text{pgStudent}, \text{Student}.type='pg'>, \text{ugStudent}, \text{Student}.type='ug'>\}$$
$$m2msacr1.d = 0.5$$
$$m2msacr1.LC = \{<\text{Student}.sid, pgStudent.sid, IDF, IDF, 0.5>, <\text{Student}.sid, ugStudent.sid, IDF, IDF, 0.5>, <\text{Student}.sname, pgStudent.sname, IDF, IDF, 0.5>, <\text{Student}.sname, ugStudent.sname, IDF, IDF, 0.5>\}$$

$$o2osacr2.SA1 = \text{Course}$$
$$o2osacr2.SA2 = \text{Course}$$
$$o2osacr.sp1 = <\text{Course}, *>$$
$$o2osacr.sp2 = <\text{Course}, *>$$
$$o2osacr.d = 0.5$$
$$FS.LC = \{<\text{Course}.cid, Course.cid, IDF, IDF, 0.5>, <\text{Course}.cname, Course.cname, \}$$
Given the many-to-many superabstract correspondence $m2msacr1$, the algorithm should create a representation of student in the reconciled schema with name that is the concatenation of names of associated constructs by $m2msacr1$ to indicate where the reconciled construct is derived from. It then combines superabstractions in the set $m2msacr1.SA_1$ of $S_1$ by joining them with given join predicates $m2msacr1.JP_1$ to form a superabstract containing superlexicals \{sid, sname, aid, contact\} (Figure 4.3 line 3). It then creates two (many-to-one) superabstract correspondences to associate Student and address in $S_1$ to the reconciled superabstract, and associate ugStudent and pgStudent to the reconciled superabstract (Figure 4.3 lines 6-8). The subsequent task is to create superlexical correspondences and create a copy for superlexicals that are unique to the source superabstract (such as age to ugStudent) to the reconciled superabstract (Figure 4.3 lines 9-22). The derived many-to-one schematic correspondence between the reconciled schema and $S_1$ is:

sacr1R.SA1 = \{Student, Address\}
sacr1R.SA2 = Student|Address|ugStudent|pgStudent
sacr1R.pt1 = 'VP'
sacr1R.JP1 = \{<Student.sid, Address.sid, =>}\}
sacr1R.SP2 = {}
sacr1R.d = x'

LC = \{<Student.sid, Student|Address|ugStudent|pgStudent.sid, IDF, IDF, 0.5>,
       <Address.aid, Student|Address|ugStudent|pgStudent.aid, IDF, IDF, 0.5>,
       <Address.contact, Student|Address|ugStudent|pgStudent.contact, IDF, IDF, 0.5>,
       <Student.sname, Student|Address|ugStudent|pgStudent.sname, IDF, IDF, 0.5>,
       <Student.type, Student|Address|ugStudent|pgStudent.type, IDF, IDF, 0.5>\}

The derived schematic correspondence between the reconciled schema and $S_2$ is:
CHAPTER 4. MODEL MANAGEMENT ON SCHEMATIC CORRESPONDENCES

sacr2R.SA1 = {ugStudent, pgStudent}
sacr2R.SA2 = Student|Address|ugStudent|pgStudent
sacr2R.pt1 = 'HP'
sacr2R.JP1 = {}
sacr2R.SP2 = {<pgStudent, type='pg'>,
               <ugStudent, type='ug'>}
sacr2R.d = x''
sacr2R.LC = {<ugStudent.sid, Student|Address|ugStudent|pgStudent.sid,
               IDF, IDF, 0.5>,
               <pgStudent.sid, Student|Address|ugStudent|pgStudent.sid,
               IDF, IDF, 0.5>,
               <pgStudent.sname, Student|Address|ugStudent|pgStudent.sname,
               IDF, IDF, 0.5>,
               <ugStudent.sname, Student|Address|ugStudent|pgStudent.sname,
               IDF, IDF, 0.5>,
               <ugStudent.age, Student|Address|ugStudent|pgStudent.age,
               IDF, IDF, 0.5>,
               <pgStudent.age, Student|Address|ugStudent|pgStudent.age,
               IDF, IDF, 0.5>}

Given the next superabstract correspondence o2osacr2, the algorithm creates
a superabstract describing courses in the reconciled schema, and creates cor-responding schematic correspondences to associate the course superabstracts in the
source to the reconciled superabstract (Figure 4.2). The algorithm iterates over
superlexicals of Course of S1, and copy them into the reconciled superabstract
(Figure 4.2, lines 16-24). For the superlexical cname, the given functions in the
input will be reused in the derived superlexical correspondence that associates
the reconciled superlexical to cname in S2, since the reconciled superelexical is a
copy of cname in S1. The derived correspondence between the reconciled schema
S’ and S2 is:

sacr2R.SA1 = Course
sacr2R.SA2 = Course
sacr2R.sp1 = <Course, *>
sacr2R.sp2 = <Course, *>
sacr2R.d = x'''
Figure 4.7: The derived reconciled schema

\[ \text{sacr2R.LC} = \{ \langle \text{Course.cid, Course.cid, IDF, IDF, 0.5} \rangle, \]
\[ \langle \text{Course.cname, Course.cname, toUPPERCASE(Course.cname) -> Course.cname, toLOWERCASE(Course.cname) -> Course.cname, 0.5} \rangle \} \]

The derived correspondence between the reconciled schema \( S' \) and \( S_1 \) is:

\[ \text{sacr1R.SA1} = \text{Course} \]
\[ \text{sacr1R.SA2} = \text{Course} \]
\[ \text{sacr1R.sp1} = \langle \text{Course, *} \rangle \]
\[ \text{sacr1R.sp2} = \langle \text{Course, *} \rangle \]
\[ \text{sacr1R.d} = 'x''', \]
\[ \text{sacr1R.LC} = \{ \langle \text{Course.cid, Course.cid, IDF, IDF, 0.5} \rangle, \]
\[ \langle \text{Course.cname, Course.cname, IDF, IDF, 0.5} \rangle \} \]

Since \( \text{cname} \) in \( S' \) is a copy of \( \text{cname} \) in \( S_1 \), there are no functions assigned in the corresponding superlexical correspondence.

After creating a representation for concepts that are described in both of the input schemas, it then creates a copy for superababstracts describe concepts
that are unique to one of the source schema. In this example, it is the Lecture superabstract in \( S_2 \) (Figure 4.1 and 4.4).

The final task is to derive superrelationships in the reconciled schema. It iterates through every superrelationship in the source schema (Figure 4.1 lines 12-17). For the superrelationship \( \text{take} \) in \( S_1 \), since it is an association relationship relating \( \text{Student} \) to \( \text{Course} \), and both superabstracts have a corresponding reconciled superabstract in the reconciled schema, the algorithm creates a superrelationship of the type \( \text{association} \) between the reconcile superabstracts \( \text{Student} \) and \( \text{Course} \) (Figure 4.5). The algorithm next encounters the superrelationship \( \text{ugTake} \) that relates \( \text{ugStudent} \) to \( \text{Course} \) in \( S_2 \). If two superrelationships of the same type represent the same information, if all of their associated superabstracts are associated by schematic correspondences to each other. In this case, the algorithm will not create a superrelationship in the reconciled schema, since there already exist a superrelationship that relate the reconciled superabstracts are equivalent to \( \text{ugStudent} \) and \( \text{Course} \) in \( S_2 \) (previously created with respect to \( \text{take} \) in \( S_1 \)) that relates the reconciled superabstracts of \( \text{ugStudent} \) and \( \text{Course} \) (Figure 4.5 line 12). The same action will be applied to the superrelationship \( \text{pgTake} \) in \( S_2 \).

Finally, the algorithm creates a superrelationship to relate the reconciled superabstracts of \( \text{Course} \) and \( \text{Lecturer} \) in respond to the superrelationship \( \text{teach} \) in \( S_2 \).

The eventual form of the reconciled schema derived by the algorithm is depicted in Figure 4.7.

### 4.2.4 Related Work

We have defined an algorithm for the model management \( \text{merge} \) operator over schematic correspondences. We have shown that the algorithm can satisfy the general requirements on the derived reconciled schema in problems of information system management where schema merging is an optional solution. Furthermore, the algorithm derives schematic correspondences for associating constructs of the reconciled schema and constructs of input schema. For example, it provides information on how to combine source superabstracts to populate the corresponding reconciled superabstract. Such information is derived based on information given in the input correspondences that are either one-to-one or one-to-many and many-to-many, which is crucial for schema mappings generation algorithms (as the one proposed in the previous chapter) to generate mappings that reconcile
schematic heterogeneities, and specify how to derive tuples for queries posed on
the reconciled schema based on tuples described by the source schema.

There has been proposals for characterising *merge* on schematic correspon-
dences. The Rondo model management system [MRB03] defined a schema merging
algorithm on one-to-one correspondences between superabstracts and super-
lexicals. The derive schematic correspondences from Rondo do not capture
many-to-many heterogeneities between schema constructs. The GeRoMe Suite
[KQCJ07, QK07] defined a merge algorithm on one-to-one and many-to-many
correspondences that inform a wide range of heterogeneities. Furthermore, the
AutoMed system [SRM08] implemented a merge algorithm on one-to-one corres-
spondences. Both GeRoMe and AutoMed do not derive schematic correspond-
dences for associating constructs of the reconciled schema to constructs in the
source schema. Thus, a separate invocation of the *match* operator needs to be
performed between the reconciled schema and source schema. Recall that in
dataspaces there may exist either one-to-one heterogeneities, as well as one-to-
many and many-to-many heterogeneities. Nowadays, it is increasingly common
for organisations to partition their database so that it can be managed and main-
tained distributively [O’B08]. Thus, the inference [Guo11, HBM+10], modelling
and reconciliation of many-to-many schematic heterogeneities play crucial roles
for supporting query translations to comply with the demand for integrating a
large set of data sources.

The initial merge algorithm proposed by Pottinger et al. [PB03] was de-
defined on simple one-to-one correspondences. The algorithm outputs a reconciled
schema which is associated to the input schema with derived schematic correspon-
dences. Based on this work, they studied the commutativity and associativity of
the *merge* operator [PB09]. Mapping generated over their correspondence model
can reconcile a limited set of heterogeneities in comparison to the kinds classi-
fied by Kim. The authors later proposed an algorithm for deriving a reconciled
schema over two relational schema [MH03]. The algorithm is driven by map-
pings in the form of query expressions, which associate superabstracts represent
the same concepts. The algorithm derives schema mappings based on the input.
In comparison to our work, our algorithm operates on a generic model (over
[ACG07b]) which include modelling capabilities for describing relationships (such
as generalisation) between concepts, and it derives superrelationships between the
derived superabstracts in the reconciled schema.
4.3 Correspondence Composition

As in other problems of information system management which involve the manipulation of schema and correspondences, the derivation of a set of correspondences can be done with match. However, match may need to consult external resources (such as dictionaries), or compare both schematic representations and data instances as pieces of evidence with which to derive a certain level of confidence in judging the equivalence of two constructs. The performance of match is reduced in the absence of supporting resources or of access to data instances. For example, in a situation where we want to derive a mapping between a global schema $S_G$ and a source schema $S_i$ but the global schema is not materialised, data instances are not available. The alternative approach to match is to take existing correspondences and compose them into a new set of correspondences. Thus, we can firstly apply match between $S_i$ and another source schema $S_j$ which is already matched to $S_G$. In this case, match is likely to be supplied with data instances for comparison. Secondly, based on correspondences between $S_G$ and $S_j$ and the correspondences between $S_j$ and $S_i$ just derived, we can compose them to derive correspondences between $S_G$ and $S_i$.

A schematic correspondence associates two or two sets of conceptually equivalent superabstracts or superlexicals, if they intentionally describe the same concept or attribute of a concept. If one (or one set) of the associated constructs is then associated to another set of constructs by another schematic correspondence, then by transitivity, all three intentionally describe the same concept or attribute of the same concept. As well as deriving associations between constructs, a compose operator needs to derive schematic heterogeneities between the constructs to be associated. For example, assume a superabstract correspondence $sacr_1$ associates $sa_{11}$ and $sa_{12}$ in schema $S_1$ to $sa_{21}$ in schema $S_2$. $sacr_1$ indicates that $sa_{11}$ and $sa_{12}$ can be construed as a vertical partitioning of $sa_{21}$, and provide selection predicates specify which tuples (derived from combining $sa_{11}$ and $sa_{12}$) can populate $sa_{21}$. Furthermore, assume that there is another superabstract correspondence $sacr_2$ which associates $sa_{21}$ to $sa_{31}$ in schema $S_3$, and provides selection predicates that specify which tuple of $sa_{21}$ can populate $sa_{31}$. When $sacr_1$ and $sacr_2$ are composed, the derived schematic correspondence should capture heterogeneities between $sa_{11}$ and $sa_{12}$ to $sa_{31}$ as well as, information on how to reconcile these heterogeneities.
For the rest of this section, we present the algorithm for composing correspondences. The algorithm firstly composes each encountered superabstract correspondence, and then composes superlexical correspondences. Next, we discuss their use in scenarios when composing different kinds of schematic correspondences is useful. Finally, we discuss related work.

4.3.1 Composing Superabstract Correspondences

Schematic correspondences associate superabstracts representing the same concept and provide information on schematic heterogeneities. When composing two superabstract correspondences, the algorithm firstly has to identify a pair of superabstract correspondences such that one or more of their associated superabstracts are the same. For example, if one correspondence associates superabstract \(sa_1\) in one schema to \(sa_2\) and \(sa_3\) in the other schema, then it can be composed to another correspondence which associates \(sa_2\) and \(sa_3\) to superabstracts in another schema.

Furthermore, information on the schematic heterogeneities in the new correspondence should also be derived so that it can inform schema mapping generation. A selection predicate \(sp_1\), given in a one-to-one superabstract correspondence, specifies which tuples of a superabstract \(sa_1\) can populate another superabstract \(sa_2\). \(sp_1\) includes a selection condition which is defined as a conjunction or disjunction of binary comparisons between two superlexicals, \(sl_{i1}\) and \(sl_{i2}\), of \(sa_1\), or between a superlexical of \(sa_1\) to a data value (in the domain of \(sa_1\)). If there is another selection predicate for specifying how to populate \(sa_3\) with tuples of \(sa_2\), in some cases we can derive a new selection predicate that specifies how to populate \(sa_3\) with tuples of \(sa_1\). For example, assume that a superabstract \(sa_1\) describing the concept student is defined as

\[
\text{Student}(\text{sid}, \text{sname}, \text{type})
\]

Also assume another superabstract \(sa_2\) that describes students taking an undergraduate degree is defined as:

\[
\text{ugStudent}(\text{sid}, \text{sname}, \text{subject})
\]
Algorithm ComposingCorrespondences

inputs:
    CR₁ and CR₂: two sets of schematic correspondences

outputs:
    CR₃: a set of schematic correspondences

begin
1   CRᵢ = new Set()
2   FOR EACH superabstract correspondence sacrᵢ ∈ CRᵢ DO
3       FOR EACH superabstract correspondence sacrᵢ ∈ CRᵢ DO
4           sacrᵢ = composingSuperabstractCorrespondence(sacrᵢ, sacrᵢ)
5           IF sacrᵢ != null THEN
6               CRᵢ ← CRᵢ ∪ sacrᵢ
7   RETURN CRᵢ

Figure 4.8: The algorithm for composing two sets of schematic correspondences

where the subject superlexical describes the subject that a student takes. Furthermore, a correspondence sacr₁ that associates them includes the following selection predicate:

\[ < ugStudent, Student.type = 'ug' > \]

Another correspondence sacr₂ that associates sa₂ to the superabstract sa₃, and also describes undergraduate students as follows:

\[ undergraduateStudent(sid, sname) \]

To associate Student to undergraduateStudent, the selection predicate given in sacr₁ should be used to specify which tuples of Student can populate undergraduateStudent—Student, since sacr₂ indicates that all tuples represented by ugStudent can populate undergraduateStudent.

However, selection predicates for the output correspondence cannot be derived from the selection predicates given in the input correspondences, if the superlexicals refered in the selection condition are missing, or have different domains. For example, assume that sacr₃ associates sa₂ to the superabstrct sa₄ that describes
CHAPTER 4. MODEL MANAGEMENT ON SCHEMATIC CORRESPONDENCES

Algorithm ComposeSuperabstractCorrespondences

inputs:
  sacr, and sacr_j: two superabstract correspondences associate respectively superabstractions of
  schema $S_i$ to superabstractions of $S_j$, and superabstracts of $S_i$ to superabstracts of $S_j$

outputs:
  sacr_j: a superabstract correspondence associate superabstracts of schema $S_i$ to
  superabstracts of $S_j$

begin
1  // if the set of superabstracts of $S_j$ by sacr, and sacr_j are not the same
2  IF sacr_j.SA_j ≠ sacr.SA_i THEN
3      signal("the correspondences are not composable")
4  RETURN
5  sacr_j = new SuperabstractCorrespondence()
6  IF getCorrCardinality(sacr_j) ∈ {"many-to-one", "many-to-many") & sacr pt_i = "VP" THEN
7      sacr_j.pt_i = "VP" // assigning partition type
8      sacr_j.JP_i ← sacr_j.JP_i ∪ sacr_j.JP_i
9  IF getCorrCardinality(sacr_j) ∈ {"one-to-many", "many-to-many") & sacr pt_i = "VP" THEN
10     sacr_j.pt_i = "VP"
11     sacr_j.JP_i ← sacr_j.JP_i ∪ sacr_j.JP_i
12  // sacr_j.SP_i specifies how to populate superabstracts of $S_j$ specified in sacr_j.SA_i
13  FOR EACH selection predicates sp in sacr_j.SP_i DO
14      sacr_j.SP_i ← sacr_j.SP_i ∪ sp
15  // sacr_j.SP_i specifies how to populate superabstracts of $S_j$ specified in sacr_j.SA_i
16  FOR EACH selection predicates sp in sacr_j.SP_i DO
17      IF the superlexical $s_l_i$ referred by sp is associated to the superlexical $s_l_j$ in $S_j$
18          & $s_l_i$.domain = $s_l_j$.domain THEN
19          sacr_j.SP_i ← sacr_j.SP_i ∪ sp
20  // sacr_j.SP_i specifies how to populate superabstracts of $S_j$ specified in sacr_j.SA_i
21  FOR EACH selection predicates sp in sacr_j.SP_i DO
22      sacr_j.SP_i ← sacr_j.SP_i ∪ sp
23  // sacr_j.SP_i specifies how to populate superabstracts of $S_j$ specified in sacr_j.SA_i
24  FOR EACH selection predicates sp in sacr_j.SP_i DO
25      IF the superlexical $s_l_i$ referred by sp is associated to the superlexical $s_l_j$ in $S_j$
26          & $s_l_i$.domain = $s_l_j$.domain THEN
27          sacr_j.SP_i ← sacr_j.SP_i ∪ sp

Figure 4.9: The algorithm for composing two superabstract correspondences
Algorithm ComposeSuperabstractCorrespondences (Continued)

inputs:
sacr, and sacr, two superabstract correspondences associate respectively
superabstracts of schema S, to superabstracts of S, and superabstracts of S, to
superabstracts of S,

outputs:
sacr, a superabstract correspondence associate superabstracts of schema S, to
superabstracts of S

begin
32 FOR EACH superlexical correspondence slcr ∈ sacr.LC DO
33 FOR EACH superlexical correspondence slcr, in sacr.LC DO
34 slcr = composeSuperlexicalCorrespondences(slcr, slcr)
35 IF slcr, ⊑ null THEN
36 CR, ← CR, ∪ slcr
37 RETURN sacr

Figure 4.10: The algorithm for composing two superabstract correspondences
(Continued)

undergraduate students taking maths:

\[ \text{mathStudent} (\text{sid}, \text{sname}, \text{grade}) \]

with a selection predicate defined as:

\[ < \text{mathStudent}, \text{ugStudent.subject} = ' \text{maths}' > \]

When composing sacr, and sacr, we should specify that all the tuple of student
in sa, math can populate mathStudent. However, the superlexical subject in
ugStudent is construed as missing in Student, thus the selection condition can
not be reformulated for selecting tuples of Student. Likewise, even if Student has
the equivalent superlexical subject, it can be defined with different domain (e.g.
varchar) to subject (e.g. integer) in ugStudent. Since our main aim is to provide
dataspaces with a foundation on model management, a best effort solution for
this problem is to output the correspondence without propagating the selection
predicate, ugStudent.subject = ' maths', and let it be refined later by soliciting
user feedback.

When composing many-to-many or one(many)-to-many(one) superabstract
correspondences, partition methods and join predicates should be derived in the output correspondences. For example, given the following two superabstract correspondences (some fields of the correspondence is omitted):

\[
\begin{align*}
\text{sacr1.SA1} &= \{\text{S1.Student, S1.Address}\} \\
\text{sacr1.SA2} &= \text{S2.Student} \\
\text{sacr1.pt1} &= 'VP' \\
\ldots \\
\text{sacr1.JP1} &= \{<\text{Student.sid, Address.sid, =>}\} \\
\ldots \\
\text{sacr2.SA1} &= \text{S2.Student} \\
\text{sacr2.SA2} &= \text{S3.Student} \\
\ldots
\end{align*}
\]

Since both correspondences associate \text{S2.Student}, we can derive a correspondence associates \text{S1.Student} and \text{S1.Address} to \text{S3.Student}. Since \text{S1.Student} and \text{S1.Address} is a vertical partitioning, the derive correspondence should capture the join predicates for combining them, and this has been given in \text{sacr1}. For horizontal partitioning, selection predicates should be derived. For example, given the following two superabstract correspondences:

\[
\begin{align*}
\text{sacr1.SA1} &= \text{S1.Student} \\
\text{sacr1.SA2} &= \text{S2.Student} \\
\ldots \\
\text{sacr2.SA1} &= \text{S2.Student} \\
\text{sacr2.SA2} &= \{\text{S3.ugStudent, S3.pgStudent}\} \\
\text{sacr2.pt2} &= 'HP' \\
\ldots \\
\text{sacr2.SP2} &= \{<\text{S3.ugStudent, S2.Student.type='ug'>}, \\
&\quad <\text{S3.pgStudent, S2.Student.type='pg'>}\}
\ldots
\end{align*}
\]

We can derive a correspondence to associate \text{S1.Student} to \text{S3.ugStudent} and \text{S3.pgStudent}. The selection predicates given in \text{sacr2} can be propagated to the derived correspondence providing that \text{S1.Student} has the equivalent superlexical
type. If $S_1.\text{Student}$ does not have the equivalent superlexical type, then the selection predicates cannot be propagated to the derived correspondence.

Notice that the algorithm does not compose one superabstract correspondence to a set of superabstract correspondences, since we may not know how to combine superabstracts associated by different correspondences. For example, we have an one-to-many superabstract correspondence associates $sa_{11}$ to $\{sa_{21}, sa_{22}\}$ and specifies that $sa_{21}$ and $sa_{22}$ should be combined using join. Meanwhile, there are two one-to-one superabstract correspondences associate $sa_{21}$ to $sa_{31}$ and $sa_{22}$ to $sa_{32}$. We can not create an one-to-many correspondence that associates $sa_{11}$ to $\{sa_{31}, sa_{32}\}$, since we do not know how to join $sa_{31}$ and $sa_{32}$ to populate $sa_{11}$.

The algorithm for composing superabstract correspondences is shown in Figures 4.8, 4.9 and 4.10.

### 4.3.2 Composing Superlexical Correspondences

In Figure 4.10, the subroutine `composeSuperabstractCorrespondences` invokes `composeSuperlexicalCorrespondences` (Figure 4.11). As for composing superabstract correspondences, composing superlexical correspondences should derive information for reconciling schematic heterogeneities. Superlexical correspondences include functions that specify how to compute the value of a superlexical from the value of one or more superlexicals. Superlexical functions in the output correspondences might be derived based on functions given in the composing correspondences.

For example, the value of superlexical `name` in schema $S_2$ is derived by the value of `firstname` and `surname` in schema $S_1$ with the function `concatenation`. Assume that `name` is also associated to the superlexical `fullname` in the schema $S_3$ with a function `toUppercase` that transform strings to uppercase. Assume that all the superlexicals are in the type string. This implies that `fullname` should be computed by firstly applying the function `concatenation` to compute the concatenation of value in `firstname` and `surname`, and then applying the `toUppercase` function to transform concatenated strings into uppercase.

The algorithm for composing superlexical correspondences is shown in Figure 4.11.
Algorithm ComposeSuperlexicalCorrespondences

inputs:
   
   \( \text{slcr}_{1} \) and \( \text{slcr}_{2} \): two superlexical correspondences associate respectively superlexical of schema \( S_{j} \) to superlexical of \( S_{i} \) and superlexical of \( S_{j} \) to superlexical of \( S_{i} \)

outputs:
   
   \( \text{slcr}_{1} \): a superlexical correspondence associate superlexical of schema \( S_{j} \) to superlexical of \( S_{i} \)

begin

1 // if the set of superlexical of \( S_{j} \) by \( \text{slcr}_{1} \) and \( \text{slcr}_{2} \) are not the same

2 \[ \text{IF}\ \text{slcr}_{1}, S_{j} \neq \text{slcr}_{2}, S_{i} \text{ THEN} \]

3 \[ \text{signal(\"the correspondences are not composable\")} \]

4 \[ \text{RETURN} \]

5 \[ \text{slcr}_{1} = \langle \text{slcr}_{1}, S_{j}, \text{slcr}_{1}, S_{i}, \text{slcr}_{1, i}, \text{slcr}_{1, i} \rangle \]

6 \[ \text{IF } \text{slcr}_{1, i} \neq \text{null} \text{ and } \text{slcr}_{1, i} \neq \text{null} \text{ THEN} \]

7 \[ \text{IF } \text{slcr}_{1, i} \text{.range } = \text{slcr}_{2, i} \text{.domain} \text{ THEN} \]

8 \[ \text{slcr}_{1, i} = \text{slcr}_{2, i} \circ \text{slcr}_{1, i} \]

9 \[ \text{IF } \text{slcr}_{1, i} \neq \text{null} \text{ and } \text{slcr}_{1, i} \neq \text{null} \text{ THEN} \]

10 \[ \text{IF } \text{slcr}_{1, i} \text{.range } = \text{slcr}_{2, i} \text{.domain} \text{ THEN} \]

11 \[ \text{slcr}_{1, i} = \text{slcr}_{2, i} \circ \text{slcr}_{1, i} \]

12 \[ \text{RETURN} \text{slcr}_{1} \]

Figure 4.11: The algorithm for composing two superlexical correspondences
4.3.3 Scenario Study

In this section, we discuss the use of the algorithm for composing schematic correspondences that specify different kinds of heterogeneities.

The following scenario contains three schemas describing the concept student using different representations. In schema $S_1$, two superabstracts are defined:

$$\text{Student}(\text{sid}, \text{firstname}, \text{surname}, \text{type})$$

$$\text{Address}(\text{aid}, \text{sid}, \text{contact})$$

Contact information about students is described in the superabstract Address. In schema $S_2$, students are classified into undergraduates and postgraduates:

$$\text{ugStudent}(\text{sid}, \text{name}, \text{contact})$$

$$\text{pgStudent}(\text{sid}, \text{name}, \text{contact})$$

and in schema $S_3$, one superabstract is defined for describing both undergraduates and postgraduates and their contact information.

$$\text{Student}(\text{sid}, \text{name}, \text{contact}, \text{type})$$

Assume that a many-to-many superabstract correspondence $m2msacr1$ is derived that associates the superabstracts in $S_1$ and $S_2$ and captures information on the partition methods, join predicates and selection predicates:

$m2msacr1.SA1=\{\text{Student, Address}\}$

$m2msacr1.SA2=\{\text{ugStudent, pgStudent}\}$

$m2msacr1.pt1='VP'$

$m2msacr1.pt2='HP'$

$m2msacr1.JP1=\{<\text{Student.sid, Address.sid, =>}>\}$

$m2msacr1.JP2=\{}$

$m2msacr1.SP1=\{<\text{ugStudent, Student.type='ug'>, <pgStudent, Student.type='pg'>}\}$

$m2msacr1.JP2=\{}$

$m2msacr1.d = 0.5$

$m2msacr1.LC={...}$
Assume that a many-to-one superabstract correspondence, \( m_{2osacr}^2 \), is derived that associates superabstracts in \( S_2 \) to \( S_3 \) also and captures information on how schematic heterogeneities can be reconciled:

\[
\begin{align*}
\text{m2osacr2.SA1} &= \{ \text{ugStudent, pgStudent} \} \\
\text{m2osacr2.SA2} &= \text{Student} \\
\text{m2osacr2.pt1} &= \text{’HP’} \\
\text{m2osacr2.JP1} &= \{} \\
\text{m2osacr2.SP1} &= \{ <\text{ugStudent}, \text{Student.type} = \text{’ug’}>, <\text{pgStudent}, \text{Student.type} = \text{’pg’}> \} \\
\text{m2osacr2.d} &= 0.5 \\
\text{m2osacr2.LC} &= \{ \ldots \}
\end{align*}
\]

When composing \( m_{2msacr}^1 \) with \( m_{2osacr}^2 \), the algorithm (Figure 4.9) copies \( m_{2msacr}^1.SA_1 \) and \( m_{2osacr}^2.SA_2 \) to \( m_{2osacr}^3 \) (Figure 4.9, line 5). Since \( m_{2msacr}^1.SA_1 \) uses vertical partitioning, the algorithm specifies ‘VP’ as the partition method, \( m_{2msacr}.pt1 \), in \( m_{2osacr}^3 \) (Figure 4.9, line 6-8), and copies the join predicates specified in \( m_{2msacr}^1 \) to \( m_{2osacr}^3 \) (Figure 4.9, line 9-10). Thus, \( m_{2osacr}^3 \) is derived as:

\[
\begin{align*}
\text{m2osacr3.SA1} &= \{ \text{Student, Address} \} \\
\text{m2osacr3.SA2} &= \text{Student} \\
\text{m2osacr3.pt1} &= \text{’VP’} \\
\text{m2osacr3.JP1} &= \{ <\text{Student.sid, Address.sid}, => \} \\
\text{m2osacr3.SP2} &= \{} \\
\text{m2osacr3.d} &= 0.5 \\
\text{m2osacr3.LC} &= \{ \ldots \}
\end{align*}
\]

We now invert the correspondence \( sacr^3 \) (i.e. swap \( m_{2osacr}^3.SA_1 \) with \( m_{2osacr}^3.SA_2 \), assign \( m_{2osacr}^3.pt1 \) to \( m_{2osacr}^3.pt2 \), and assign \( m_{2osacr}^3.JP_1 \) to \( m_{2osacr}^3.JP_2 \)) to form \( o_{2msacr}^3 \):

\[
\begin{align*}
\text{o2msacr3’.SA1} &= \text{Student} \\
\text{o2msacr3’.SA2} &= \{ \text{Student, Address} \} \\
\text{o2msacr3’.pt2} &= \text{’VP’} \\
\text{o2msacr3’.JP2} &= \{ <\text{Student.sid, Address.sid}, => \} \\
\text{o2msacr3’.SP1} &= \{} \\
\text{o2msacr3’.d} &= x \\
\text{o2msacr3’.LC} &= \{ \ldots \}
\end{align*}
\]
and compose $o_2msacr'\text{3}'$ to $m_2msacr1$. We name the output correspondence as $o_2msacr'\text{2}'$. In this case, the algorithm copies $\{\text{Student}\}$ to $o_2msacr'\text{2}'.SA_1$, and copies $\{\text{ugStudent, pgStudent}\}$ to $o_2msacr'\text{2}'.SA_2$. If we assume that the superlexical type in $\text{Student}$ of $S_3$ is associated to type in $\text{Student}$ of $S_1$, the algorithm assigns the selection predicates in $m_2msacr2$ to $o_2msacr'\text{2}'$ (Figure 4.9, line 20-23). Finally, $o_2msacr'\text{2}'$ is derived as

$$
o_2msacr'\text{2}'.SA_1=\text{Student}$$
$$o_2msacr'\text{2}'.SA_2=\{\text{ugStudent, pgStudent}\}$$
$$o_2msacr'\text{2}'.pt='\text{HP}'$$
$$o_2msacr'\text{2}'.SP_1=\{<\text{ugStudent, Student.type='ug'>}, '$\text{pgStudent, Student.type='pg'>}$$
$$o_2msacr'\text{2}'.JP_2=\{}$$
$$o_2msacr'\text{2}'.d = x''$$
$$o_2msacr'\text{2}'.LC=\{\ldots\}$$

The invert of $o_2msacr'\text{2}'$ specifies the same information on schematic heterogeneities between $S_2$ and $S_3$ as in $m_2osacr2$.

To demonstrate the composition of superlexical correspondences, the following superlexical correspondences are given:

$$slcr1.SL_1=\{\text{Student.firstname, Student.surname}\}$$
$$slcr1.SL_2=\text{ugStudent.name}$$
$$\text{COMPOSE(\text{Student.firstname, Student.surname}) -> ugStudent.name}$$
$$\text{DECOMPOSE(ugStudent.name) -> (\text{Student.firstname, Student.surname})}$$
$$slcr1.d = 0.5$$

$slcr1$ associates superlexicals in $S_1$ to superlexicals in $S_2$. Meanwhile, $slcr2$ associates superlexicals in $S_2$ to superlexicals in $S_3$ with functions for converting strings into upper and lower cases:

$$slcr2.SL_1=\text{ugStudent.name}$$
$$slcr2.SL_2=\{\text{Student.fullname}\}$$
$$\text{toUPPERCASE(ugStudent.name) -> Student.fullname}$$
$$\text{toLOWERCASE(Student.fullname) -> ugStudent.name}$$
$$slcr2.d = 0.5$$

To derive $slcr3$, that associates superlexicals of $S_1$ to superlexicals of $S_3$, by composing $slcr1$ to $slcr2$, the algorithm (Figure 4.11) copies $SL_1$ of $slcr1$ to
SL1 of slcr3, and copies SL2 of slcr2 to SL2 of slcr3 (Figure 4.11, line 5). As specified by the superlexical function toUPPERCASE in slcr2, the string of names in Student.fullname of S2 are stored in the form upper case. The function CONCATENATE in slcr1 populates ugStudent.name of S2 by concatenating data values of Student.firstname and Student.surname of S1. To populate Student.fullname with values of Student.firstname and Student.surname, CONCATENATE is applied first, where the output will be then converted into upper cases with the function toUPPERCASE (Figure 4.11, line 20-23). Meanwhile, the functions DECOMPOSE is composed with toLOWERCASE for populating Student.firstname and Student.surname with values of Student.fullname. Therefore, sacr3 is derived as:

\[
\begin{align*}
\text{slcr3.SL1=}&\{\text{Student.firstname, Student.surname}\} \\
\text{slcr3.SL2=}&\text{Student.name} \\
\text{toUPPERCASE:}&(\text{COMPOSE(\text{Student.firstname, ' ', Student.surname})}) \\
&\rightarrow (\text{Student.fullname}) \\
\text{DECOMPOSE:}&(' ', \text{toLOWERCASE:}(\text{Student.name})) \\
&\rightarrow (\text{Student.firstname, Student.surname})
\end{align*}
\]

\text{slcr3.d = x}

We have presented the algorithm that implements the compose operator. The running time of the algorithm is proportional to the size of the input sets of correspondences. The algorithm uses nested loops to identify correspondences that are composable from the input sets of correspondences. Given that the number of correspondences in the two sets of input is N and M, the running time of the algorithm is \(O(N \times M)\).

### 4.3.4 Related Work

Several proposals for correspondence composition operator have been made [MH03, FKPT04, KQ+09]. Bernstein et al. [BGMN08] proposed a best-effort algorithm for composing mappings expressed as relational algebraic queries. Relational algebra is implemented in most of relational database systems and tools. It is familiar to developers of database applications. In our work, we implemented the compose operator to derive schematic correspondences which can be taken as input to generate schema mappings with algorithms such as [MBPF09] or
CHAPTER 4. MODEL MANAGEMENT ON SCHEMATIC CORRESPONDENCES

120
to other model management operators implemented over schematic correspondences. The algorithm presented in [BGMN08] iterates over one set of relational algebra queries, and search through the other set of relational algebra views to apply the view unfolding technique to replace relations of $S_2$, referred by the views, by relations of $S_3$ or $S_1$. However, the algorithm does not provide mechanisms for propagating selection predicates and superlexical functions which are important elements for reconciling heterogeneities. The algorithm presented in this section provides best effort mechanisms for deriving information on Kim et al.'s schematic heterogeneities so that mapping generated (e.g. with the algorithm presented in the previous chapter) over the output correspondences can reconcile such schematic heterogeneities as the vertical partitioning to horizontal partitioning (many-to-many) heterogeneity.

4.4 Difference in Schemas

The $diff$ operator aims to output the sets of constructs that are missing between two input schemas [Mel04]. It can be construed as the complement of $match$. Schema evolution and round-trip engineering problem is an area where $diff$ is mostly needed [Ber03]. In dataspaces, a source schema may change in accordance to evolving business objectives. The most likely actions to be taken in such events are deleting or adding constructs. These changes can also cause the need for an update to the global schema and to mappings from the latter if consistency is to be maintained. The $diff$ operator is defined as follows:

$$diff(S_1, S_2, CR_{S_1\rightarrow S_2}) \rightarrow < MC_{S_1}, MC_{S_2} >$$

where $CR_{S_1\rightarrow S_2}$ is a set of schematic correspondences associating superababstracts or superlexicals of schema $S_1$ and $S_2$ that are conceptually equivalent, $MC_{S_1}$ is a set of constructs of $S_1$ that are missing in $S_2$, and $MC_{S_2}$ is the set of constructs of $S_2$ that are missing in $S_1$. Notice that the $diff$ operator aims not to elicit the difference in the extent of construct, i.e. a set of tuples of one construct is not in the extent of the other construct. This thesis adopts the definition of the $diff$ operator defined by Melnik [Mel04] for eliciting difference in schematic representations.

A superababstract or superlexical of $S_1$ is assigned to $MC_{S_1}$ if it is not associated
CHAPTER 4. MODEL MANAGEMENT ON SCHEMATIC CORRESPONDENCES

Figure 4.12: The algorithm for eliciting missing constructs
to any superabstractions or superlexicals in $S_2$ by correspondences in $CR_{S_1-S_2}$. The $diff$ algorithm is presented in Figure 4.12.

In source schema updates, one action likely to be taken is to add some constructs to the source schema. If the added constructs do not match any constructs in the global schema (which can be checked with the $match$ operator), the change should affect the global schema accordingly, since the global schema should hold a representation of the added source construct. For this purpose, we define an operator $copy$ for making copies of a set of input constructs to a schema, and output the updated schema with a set of schematic correspondences:

$$copy(S_1, CR_{S_1-S_2}, MC_{S_2}) \rightarrow < S_1', CR_{S_1-S_2}' >$$

$CR_{S_1-S_2}$ associates constructs of $S_1$ to constructs of $S_2$, and $MC_{S_2}$ is a set of constructs in $S_2$ missing in $S_1$. For each construct $c$ in $MC_{S_2}$, we firstly need to trace the context where it should be copied to. For example, if $c$ is a superlexical, it should be copied to the superabstract in $S_1$ which is associated to the superabstract of $c$. On the other hand, if $c$ is superabstract, its superlexicals are all copied to $S_1'$ and are associated to the copied superlexical. The algorithm for $copy$ is presented in Figure 4.13.

The algorithm firstly copies $S_1$ to $S_1'$ and then copies $CR_{S_1-S_2}$ to $CR_{S_1-S_2}'$ (Figure 4.13, line 1-2). It then copies every superabstract in $MC_{S_2}$ to $S_1'$, every
Algorithm CopyConstructs

inputs:
   \( S_i \) : a schema; \( CR_{\text{MC}_{\text{MC}}} \) : a set of correspondences; \( MC_{\text{MC}} \) : a set of constructs

outputs:
   \( S'_i \) : a schema contains constructs of \( S_1 \) and constructs copied from \( MC_{\text{MC}} \); \( CR_{\text{S} \text{MC}} \) : a set of correspondence associating constructs of \( S'_i \) to constructs of \( S_i \)

begin
   1. \( S'_i \leftarrow S_i \)
   2. \( CR_{\text{S} \text{MC}} \leftarrow CR_{\text{MC} \text{MC}} \)
   3. FOR EACH superabstract \( sa \in \text{MC}_{\text{MC}} \) DO
      4. \( sa' = \text{new superabstract}(sa., \text{name}) \)
      5. append(\( S'_i \), \( sa' \))
      6. \( sacr = <sa., sa'> \)
      7. append(\( CR_{\text{S} \text{MC}} \), \( sacr \))
   8. FOR EACH \( sl \in sa., \text{superlexical} \) DO
      9. \( sl' = \text{new superlexical}(sl., \text{name}, sl., \text{domain}, sl., \text{isKey}) \)
     10. append(\( sa'. \text{superlexicals} \), \( sl' \))
     11. \( sclr = <sl', sl' > \)
     12. append(\( CR_{\text{S} \text{MC}} \), \( sclr \))
   13. FOR EACH \( sl \in \text{MC}_{\text{MC}} \) DO
      14. IF there exist a superabstract correspondence \( sacr' \) in \( CR_{\text{S} \text{MC}} \) THEN
         15. such that the superabstract \( sa \) of \( sl \) is associated by \( sacr' \)
      16. \( sl' = \text{new superlexical}(sl., \text{name}, sl., \text{domain}, sl., \text{isKey}) \)
      17. append(\( sacr'. \text{superlexicals} \), \( sl' \))
      18. \( sclr = <sl', sl' > \)
      19. append(\( CR_{\text{S} \text{MC}} \), \( sclr \))
   20. RETURN \( S'_i \), \( CR_{\text{S} \text{MC}} \)

Figure 4.13: The algorithm for copying constructs to a schema
superlexical of the copied superabstract to the new superabstract and creates correspondences to associate the copied constructs to the new constructs (Figure 4.13, line 3-12). It finally copies all the missing superlexicals in $MC_{S_2}$ to $S'_1$ (Figure 4.13, line 3-19).

The copy algorithm can be applied for updating the global schema, after constructs have been added to a source schema. The other problem is when constructs are deleted from a source schema. In this case, the corresponding constructs in the global schema should not be deleted immediately following the change in the source, since such constructs may be associated to other constructs in other source schema. If they are not, they can be deleted from the global schema, as well as any correspondences that associate such constructs.

### 4.4.1 Related Work

To the best of our knowledge, most existing model management systems do not explicitly provide the \texttt{diff} operator [SRM08, KQCJ07, ABBG09c]. Eliciting the difference between schema is important for supporting schema evolution in the maintenance of dataspaces. \texttt{diff} is implemented in the Rondo system [MRB03], but the elicited missing constructs are represented in a schema. Any superlexical of one schema that is missing in the other schema is copied to the output schema together with the superabstract under which the superlexicals are defined. The added superabstract is called \textit{support construct} [Ber03] with the purpose of making the derived schema well-formed. Correspondences are also created to associate constructs in the output schema, except all the support constructs, and constructs of the input schema. The output correspondences are used to distinguish constructs that are actually missing to all the support constructs. This implies an overhead for storing and managing the output correspondences and schema. In our algorithm, to remove such overhead (caused by the generation of support constructs new schematic correspondences), we derive direct representations of the missing constructs as a set of references to the constructs.

### 4.5 Implementation

As the mapping generation algorithm, the \textit{merge}, \textit{compose} and \textit{diff} algorithms were implemented and integrated into the dataspaces ToolKit [HBM+12]. The
CHAPTER 4. MODEL MANAGEMENT ON SCHEMATIC CORRESPONDENCES

implementation was written in Java. The class diagram that describes class attributes of the operator classes and relationships between the operator classes and classes of schema and schematic correspondences are presented and discussed in [HBM+12]. Each operator has its own class which contains methods that implements its sub-procedures. The \textit{merge} operator takes as input objects of classes that represent schema constructs and schematic correspondences. The schematic correspondence objects taken as input by the algorithm are derived from the \textit{inferCorrespondences} operator, while the schematic correspondence objects derived by the \textit{merge} operator are then taken as input by \textit{viewGen} to generate mappings, or by \textit{compose} to derive new schematic correspondence objects.

4.6 Conclusion

In this chapter, we have presented and discussed the characterisation of model management operators over the model schematic correspondence defined in Chapter 3. We have discussed the proposed algorithms that can automate model management operations that can be frequently applied in the bootstrapping and maintenance of dataspace systems. The \textit{merge} operator derives a reconcile schema that represents concepts and relationships represented by source schemas with no duplication, along with schematic correspondences that associate constructs of the reconciled schema and constructs of source schemas, so that the derived schematic correspondences can then be used by mapping generator discussed in the previous chapter to derive mappings for query evaluations. The \textit{compose} operator infers new schematic correspondences based on existing schematic correspondences. Finally, the \textit{diff} operator elicits difference between schematic representations without creating the overhead created by existing solutions. The implementation of the algorithms defined in this chapter were integrated as part of the family of operators in the DSToolKit dataspace management system [HBM+12].

In the subsequent chapter, we will discuss the proposed strategy for quantifying uncertainty in associating equivalent constructs.
Chapter 5
Quantification and Update of Uncertainty

In Chapters 3 and 4, we presented an algorithm for generating schema mappings from schematic correspondences, as well as algorithms for operating on schema and schematic correspondences. We have illustrated how the model of schematic correspondences presented in Chapter 3 provides rich information on schematic heterogeneities to the extent of enabling the fully automated generation of schema mappings. Then, in Chapter 4 we described how model management operators can be characterised over such schematic correspondences thereby providing generic mechanisms for tasks in the bootstrapping and maintenance phases of the dataspace life cycle. All the proposed algorithms need to operate on schematic correspondences and propagate rich information on schematic heterogeneities between the associated constructs across repeated applications.

Building on such foundations, the vision of dataspaces aims at removing the need for manual effort in the bootstrapping phase, relying instead on user feedback to incrementally improve the query service [HFM06, BPE+10, CVDN09]. However, this automated approach implies the need to manage the uncertainty caused by the limitations in automated schema matching algorithms [DHY09b, HFM06].

In Chapter 2, we identified a class of issues associated with uncertainty management that need to be addressed in data integration systems. Many research proposals have studied query evaluation over schema mappings derived with uncertainty [DHY09b]. There have also been proposals that studied the characterisation of schema merging algorithms over correspondences with uncertainty.
CHAPTER 5. QUANTIFICATION AND UPDATE OF UNCERTAINTY

[SDH08, MRMM05]. Most of these works rely on the assumption that uncertainty has already been quantified (i.e., there is no attempt to explain in detail how a particular degree of uncertainty is derived). Uncertainty has typically been represented as objective probability. The representation of uncertainty as probability is adopted in many other domains, for example, in probabilistic databases [Koc08, RDS07, DS07a, TCY92, DS07b] and in reasoning with uncertainty [Sha07, GP92].

To complement existing approaches to query evaluation and schema merging with uncertainty, this dissertation contributes approaches for quantifying uncertainty on the equivalence of a pair of schema constructs, with similarity scores and user feedback being taken as sources of evidence.

With a view to judging the equivalence of two schema constructs, similarity comparisons provide evidence in the form of similarity scores [RB01]. Given a pair of schema constructs, we may be informed by different similarity scores from different schema matching algorithms or combination thereof. For example, two constructs with the same name may describe different concepts [KCGS95]. In this case, a name string comparison algorithm may return a relatively high score while an instance level comparison algorithm may return a low score. The combination of the two algorithms may lead to yield another score. The judgment on how certain we are that the constructs are equivalent should depend on the trustworthiness of the score, in other words, on the difference between how likely the score is to be obtained from matching equivalent constructs and how likely the score is to be obtained from matching non-equivalent constructs. For example, if for a given method, we obtain the score 0.8 more often in matching constructs that are known to be equivalent than matching non-equivalent constructs, this supports an increment in the degree of belief on the equivalence of two constructs from a prior belief.

Furthermore, another form of evidence that can give rise to a quantification of the degree of uncertainty on the equivalence of schema constructs is user feedback. In dataspaces, users can provide feedback on different kinds of objects [BPF+11]. As studied in [BPE+10], the quality of schema mappings can be quantified in the form of precision and recall by soliciting feedback on result tuples (as retrieved through the use of schema mappings). For example, assume that a set of tuples is retrieved from the relation Student as answers to a query posed on the relation UndergraduateStudent. Users may indicate, by examining the result that certain
tuples are in the result but should not be (for example, tuples that represent postgraduate students), while others tuples, that are not in the result, should be. The more tuples that are accepted by the user as correct, the higher the likelihood that the mapping (that retrieved the tuples) associate conceptually equivalence relations.

We construe uncertainty as a subjective degree of belief, which is a classical approach used in judging a particular hypothesis by reasoning under insufficient knowledge [Sha07, FH89, J92]. The more evidence we acquire that supports the hypothesis, the more confidence we have that the hypothesis is true (and correspondingly for evidence supporting the negation of the hypothesis). We start with a degree of belief based on the assimilated evidence so far, and then, as more evidence is acquired, the degree of belief is updated and may converge toward an objective belief [FH89].

In this dissertation, we contribute a framework for incrementally updating degrees of belief in the equivalence of a pair of schema constructs with similarity scores and user feedback as kinds of evidence. In dataspaces, information that can influence judgments on the equivalence of constructs can become available in varying orders at different points in the dataspace's life-cycle [HBM+10]. For instance, at the bootstrapping stage, there may be a limited number of schema matching algorithms to elicit similarity scores over UndergraduateStudent and Student. As more and more users provide feedback on tuples retrieved from UndergraduateStudent, the level of uncertainty in the equivalence of UndergraduateStudent and Student must be updated accordingly. Thus, a dynamic and incremental framework is very desirable in the context of dataspaces. In the next section, we briefly review the theoretical foundations of our contributed approach, viz., Bayes's theorem [J92].

As well as schema matching, some model management operators such as Merge and Compose also derive associations between schema constructs, but based on existing correspondences. In the presence of uncertainty in the input correspondences, such operators must infer the degree of belief in the output correspondences (i.e., in the equivalence of associated schema constructs) based on the degree of belief in the input correspondences. The propagated degree of belief can then be updated with evidence such as user feedback with the proposed framework. In this research, we have proposed heuristic functions for Merge and Compose to derive the degree of belief in the output correspondences from the
degree of belief in the input correspondences.

In the reminder of this chapter, we firstly review Bayes’s theorem. We then introduce the proposed strategies for quantifying degree of belief from similarity scores and user feedback. We then demonstrate the use of Bayes’s theorem to incrementally update the degree of belief from evidence in the form of user feedback and similarity scores. Finally, we discuss the propagation of degrees of belief by model management operators.

5.1 Bayes’s Theorem

Bayes’s theorem is widely recognised as a well-formed basis for updating degrees of belief in the presence of accumulating evidence. It states that the degree of belief that hypothesis $H$ is true given where an evidence $E$ is observed depends on acquired experience from situations that $E$ is observed when we are given that $H$ is true. The theorem is mathematically represented as:

$$D(H \mid E) = \frac{D(E \mid H)D(H)}{D(E)}$$

(5.1)

where $D(H \mid E)$ (sometimes called the posterior belief) denotes the degree of belief that the hypothesis $H$ is true given the evidence $E$; $D(E \mid H)$ denotes the degree of belief that the evidence is true given the hypothesis; $D(H)$ denotes the prior degree of belief in $H$; and $D(E)$ is the degree of belief in the evidence itself and acts as a normalising constant. After applying Bayes’s theorem for updating belief with $E$, the posterior belief is then used as a prior belief to be updated with the next evidence. This chaining models the incremental and dynamic acquisition of evidence. The question now is how to acquire probability on the evidence $D(E \mid H)$ and $D(E)$, when, in our cases, the evidence are user feedback and similarity scores.

Following the axioms of probability [FH89], the degree of belief is a continuous value in the range $[0, 1]$. $D(\neg H)$ is the complement of $D(H)$:

$$D(\neg H) = 1 - D(H)$$

(5.2)

Since $D(E)$ is equivalent to:

$$D(E \text{ and } H) + D(E \text{ and } \neg H)$$

(5.3)
and $D(E \text{ and } H)$ is equivalent to

$$D(E | H)D(H)$$

(5.4)

where $D(E | H)$ can be construed as the degree of belief in that the $E$ is observed given that $H$ is true. Also, $D(E \text{ and } \neg H)$ is equivalent to

$$D(E | \neg H)D(\neg H)$$

(5.5)

where $D(E | \neg H)$ can be construed as the degree of belief in that the $E$ is observed given that $H$ is false. Thus, based on (5.4) and (5.5), $D(E)$ can be derived as:

$$D(E | H)D(H) + D(E | \neg H)D(\neg H)$$

(5.6)

Thus, given the derivation of $D(E)$, Bayes’s theorem can be rewritten as [Sha07]:

$$D(H | E) = \frac{D(E | H)D(H)}{D(E | H)D(H) + D(E | \neg H)D(\neg H)}$$

(5.7)

Given Bayes’s theorem, Earman [J92] has shown that two people that were to start with vastly different judgments on the belief in the same hypothesis, as evidence for (or against) the hypothesis accumulates, their initially distinct degrees of belief will tend to converge to the same value.

It can be proved that the difference between $D(H)$ and $D(H | E)$ (in other words, the extent to which the posterior belief in $H$ is changed with respect to the prior belief in $H$ once $E$ is observed) directly depends on the difference between $D(E | H)$ and $D(E | \neg H)$ [Sha07]:

- If $D(E | H) > D(E | \neg H)$, then $D(H | E) > D(H)$ (in other words, the posterior belief in $H$ is increased with respect to the prior).
- If $D(E | H) < D(E | \neg H)$, then $D(H | E) < D(H)$.
- The larger the difference between $D(E | H)$ and $D(E | \neg H)$, the larger the difference between $D(H)$ and $D(H | E)$.

Bayes’s theorem can underpin the incremental update of belief in a hypothesis from an initial prior belief as different kinds of evidence are obtained. For each piece of evidence, Bayes’s theorem can be used to derive a rational update if the trustworthiness of that evidence has been ascertained empirically through a study.
of the methods and procedure used to obtain it [Sha07]. For example, we can learn empirically whether a certain piece of evidence \( E \) is more often observed when the hypothesis is true than when it is false. Given this information, the rational behaviour on observing \( E \) is to have a larger posterior belief than the prior. On the other hand, if the evidence of a statement provided by a human (such as feedback given by users or domain experts) that indicates his or her belief on the hypothesis, it can be directly considered.

In the following sections, we present the framework for quantifying similarity scores and user feedback into degree of belief. Once this framework is in place, we can incrementally and dynamically assimilate scores and feedback for judging the equivalence of schema constructs with Bayes’s theorem.

### 5.2 Overview of the Framework

One of the aims of this thesis is to incorporate Bayes’ theorem into dataspaces for quantifying and updating uncertainty. In order to apply Bayes’ theorem, we firstly need to acquire degrees of belief in different kinds of evidence (i.e. \( D(E \mid H) \) and \( D(E \mid \neg H) \)) for judging the hypothesis. In this thesis, we study the updates of degrees of belief in similarity scores given by schema matching tools, and feedback solicited from users, which will be used to support the use of Bayes’ theorem for deriving posterior degree of belief on the hypothesis. This section outlines how our approach can be deployed in dataspace systems.

Schema matching benchmarks, such as the MatchBench [CGF11], have been developed to assess the performance of schema matching algorithms in judging the equivalence of constructs. The MatchBench can also be used to conduct empirical studies to derive the probability that a similarity score is associated with actual equivalence. Given a substantial number of pairs of constructs that are known to be equivalent or non-equivalent (in other words, the ground truth), we can obtain distributions of similarity scores by applying schema matching algorithms to the ground truth. This can be a continuous process throughout the life-cycle of dataspaces. At the bootstrapping stage, we may obtain biased or erroneous similarity score distributions, since there may be only a small amount of the ground truth available. As a dataspace is a pay-as-you-go integration system, the ground truth may become incrementally available during the usage, improvement or maintenance stages, for example through soliciting feedback. The
A complementary approach for judging the equivalence of two constructs is through soliciting feedback on tuples of one construct that are used to populate constructs. The higher the ratio of the number of tuples that correctly populate to the other construct to the total number of tuples that should be populated, the higher the degree of belief that the constructs are equivalent. In this chapter, we discuss a function for converting feedback into degree of belief, which is to be enhanced by the empirical study on how the degree of belief varies against the amount of feedback provided by users. This work can be implemented and integrated as part of the process of improving dataspaces using user feedback.

The deployment of such framework for updating degree of belief into the dataspace life-cycle is illustrated in Figure 5.1. The framework continuously accepts pieces of evidence at different points of the bootstrapping and improvement stages of the dataspace life-cycle [HBM+10]. At the bootstrapping stage, zero or more schema matching algorithms may be invoked to provide similarity scores. Given a score, we firstly convert it into degree of belief, and then apply the Bayes’ theorem to update the prior belief. At the improvement stage, user feedback, as well as more schema matching algorithms, may provide pieces of evidence. The framework is not restricted to updating the belief only with similarity scores and user feedback: it can be adapted to other kinds of evidence as long as strategies are provided for converting a piece of evidence into a degree of belief.

At the start of the dataspace life-cycle, no evidence has yet become available, therefore we need to stimulate some prior belief that is subsequently updated as evidence becomes available. This initial prior belief can be derived from a probability distribution on all the possible outcomes of the hypothesis. Approaches for deriving an initial prior belief in this fashion are not in scope of this dissertation. The other approach is to adopt the principle of indifference [FH89, J92], which states that, when no prior knowledge or information is available for making a hypothetical judgment, the uniform distribution is used. Thus, if there are \( n \) possible outcomes of a given hypothesis, then the probability that one of the outcomes is obtained is \( 1/n \). In this situation, we adopt the interpretation of probability as degree of belief and there are two possible states for a boolean hypothesis that asserts the equivalence of a pair of constructs, by applying the principle indifference, and the axioms that the sum of the probabilities of all possible states must be 1, the states, \( D(c_1 \equiv c_2) \) and \( D(c_1 \not\equiv c_2) \), should have
Figure 5.1: Incremental update of degree of belief
a probability of 0.5. We anticipate that the probability distribution may vary with the level of knowledge that specific dataspaces will have about the states of the hypothesis. For example, if we are informed that most pairs of constructs in the current dataspace are equivalent, then the probability that two constructs are equivalent could be set higher than 0.5. Likewise, if the probability is interpreted as likelihood, there can be more than two possible outcomes for a given random variable. Nevertheless, the proposed framework can be adapted to different initial prior beliefs by simply performing the update again with all the available evidence. This will be demonstrated in the scenario studies of later sections (Sections 5.3.5, 5.4.1 and 5.5).

In the following sections, we present the framework for converting similarity scores and user feedback into degrees of belief.

### 5.3 Similarity Scores to Degree of Belief

We have observed that Bayes’s theorem can underpin the dynamic update of degrees of belief in response to newly acquired evidence. In this chapter, we present a framework for converting similarity scores produced by schema matching algorithms into degrees of belief in the equivalence of a pair of schema constructs.

We firstly assume that a similarity score, $s$, is derived by a schema matching algorithm, $ss$. Given $ss$, we need to examine the likelihood of obtaining $s$ when $ss$ matches constructs that are known to be equivalent, and the likelihood of obtaining $s$ when $ss$ matches constructs that are known not to be equivalent. The likelihood of $s$ offers the basis for rationally updating a prior belief in the equivalence of two constructs. If the likelihood that $ss$ outputs $s$ when matching equivalent constructs is higher than the likelihood that $s$ is returned when matching non-equivalent constructs, it is rational to increase the degree of belief from a given prior belief.

The problem is to derive the posterior belief that two constructs, $c_1$ and $c_2$, are equivalent given that $c_1$ and $c_2$ are matched by $ss$ with score $s$, i.e. $D(c_1 \equiv c_2 \mid ss(c_1, c_2) = s)$. This can be expressed using Bayes’s theorem as:

$$
D(ss(c_1, c_2) = s \mid c_1 \equiv c_2) = \frac{D(c_1 \equiv c_2)D(ss(c_1, c_2) = s \mid c_1 \equiv c_2)}{D(ss(c_1, c_2) = s \mid c_1 \equiv c_2)D(c_1 \equiv c_2) + D(ss(c_1, c_2) = s \mid c_1 \not\equiv c_2)D(c_1 \not\equiv c_2)}
$$

(5.8)

where $D(c_1 \equiv c_2)$ is the prior belief, i.e. the belief in the hypothesis before the
evidence is observed. $D(c_1 \equiv c_2)$ may be arrived at using a sequence of updates using Bayes’s theorem. $D(ss(c_1, c_2) = s|c_1 \equiv c_2)$ is the degree of belief that $ss(c_1, c_2) = s$ is obtained given that $c_1$ and $c_2$ are known to be equivalent. Correspondingly, $D(ss(c_1, c_2) = s|c_1 \not\equiv c_2)$ is the degree of belief in that $ss(c_1, c_2) = s$ is obtained given that $c_1$ and $c_2$ are known to be not equivalent. Finally, $D(c_1 \not\equiv c_2)$ is the complement of $D(c_1 \equiv c_2)$:

$$D(c_1 \not\equiv c_2) = 1 - D(c_1 \equiv c_2)$$ (5.9)

To obtain $D(ss(c_1, c_2) = s|c_1 \equiv c_2)$ and $D(ss(c_1, c_2) = s|c_1 \not\equiv c_2)$, we can empirically study the matcher (i.e. $ss$) when matching pairs of constructs that are known to be equivalent and non-equivalent, so as to obtain two score distributions: one on matching equivalent constructs and the other on matching non-equivalent constructs). We can then derive the trustworthiness (or the likelihood) of specific scores based on the score distribution.

### 5.3.1 Schema Matcher Benchmark

In conducting such an empirical study, we have built upon the MatchBench benchmark [CGF11]. MatchBench consists of systematic test cases aimed at evaluating the performance of schema matchers. The test cases are classified into positive and negative cases. In positive cases, MatchBench offers scenarios that systematically remove similarities in schematic representations (such as construct names) or data instances between a pair of constructs that are known to be equivalent. Conversely, in negative cases, MatchBench offers scenarios that systematically inject similarities into constructs that are known not to be equivalent. We first divide the continuous range $[0, 1]$ of scores in a discrete number of continuous intervals. We refer to an interval as a bucket and denote buckets by $b$.

By classifying the returned matches into true positives and false positives, $D(ss(c_1, c_2) = s|c_1 \equiv c_2)$ can then be computed as:

$$\frac{|TP|_b}{|TP|}$$ (5.10)

where $|TP|_b$ is the count of pairs of constructs that are known to be equivalent and that are matched in MatchBench experiments with scores in the bucket $b$, and $|TP|$ denotes the count of pairs of constructs that are known to be equivalent.
Likewise, $D(ss(c_1, c_2) = s|c_1 \neq c_2)$ can be computed as:

$$\frac{|FP|_b}{|FP|}$$

where $|FP|_b$ is the count of pairs of constructs that are known to be not equivalent and that are matched with scores in the bucket $b$, and $|FP|$ denotes the count of pairs of constructs that are known to be not equivalent.

Based on what we learnt from the MatchBench experiment on schema matchers [CGF11], we expect that it should be more likely to obtain high values of the similarity score than obtaining low values of the score if we know that $c_1$ and $c_2$ are equivalent. Thus, the degree of belief in the equivalence of two constructs should be influenced by the value of similarity score. Meanwhile, the degree of belief in the non-equivalence of two constructs should be inversely proportional to the value of similarity score. The more of the ground truth is available, the more accurate the estimation on the rate of increase or decrease against similarity score we can derive.

Given equations (5.10) and (5.11), a score $s$ in the bucket $b$, a prior degree of belief that $c_1$ is equivalent to $c_2$, and training from MatchBench, $\frac{|TP|_b}{|FP|} > \frac{|FP|_b}{|FP|}$ indicates that the likelihood of obtaining true positive matches with scores in $b$ is higher than that of obtaining false positive matches in the same bucket. These data (from the MatchBench experiments) should support an increase in the belief that $c_1$ is equivalent to $c_2$ with respect to the prior belief. Otherwise, the degree of belief should be decreased from the prior belief if $\frac{|TP|_b}{|FP|} < \frac{|FP|_b}{|FP|}$.

In the following section, we discuss the procedure for classifying matches into true positives and false positives, and illustrate the derivation of trustworthiness of scores from the classified matches with a simple example.

### 5.3.2 The Classification of Matches into TP and FP

True positive matches and false positive matches are classified as the ground truth $G$, passed to a MatchBench experiment, where $G = \{E, NE\}$:

- $E = \{e_1, e_2, ... e_n\}$ is a set of pairs of constructs that are known to be equivalent and each $e_i$ is a pair $< leftConst, rightConst >$ where $leftConst$ and $rightConst$ denote the left and right constructs respectively.

- $NE = \{ne_1, ne_2, ... ne_m\}$ is a set of pairs of constructs that are known not
Algorithm ClassifyingMatchesIntoTPandFP

inputs:

G: the ground truth; M: a set of matches

outputs:

TP: the set of true positive matches; FP: the set of false positive matches

begin

1 \( TP, FP = \text{new set}() \)

2 FOR EACH \( m \in M \) DO

3 \hspace{1em} FOR EACH \( e \in E \) DO

4 \hspace{2em} IF \( (e\text{.leftConst}==m\text{.leftConst} & e\text{.rightConst}==m\text{.rightConst}) \)

5 \hspace{3em} | \( (e\text{.leftConst}==m\text{.rightConst} & e\text{.rightConst}==m\text{.leftConst}) \) \) THEN

6 \hspace{4em} insert \( m \) into \( TP \)

7 \hspace{1em} FOR EACH \( ne \in NE \) DO

8 \hspace{2em} IF \( (ne\text{.leftConst}==m\text{.leftConst} & ne\text{.rightConst}==m\text{.rightConst}) \)

9 \hspace{3em} | \( (ne\text{.leftConst}==m\text{.rightConst} & ne\text{.rightConst}==m\text{.leftConst}) \) \) THEN

10 \hspace{4em} insert \( m \) into \( FP \)

11 RETURN \( TP, FP \)

Figure 5.2: Procedure for classifying matches into true positives and false positives.

to be equivalent and each \( ne_i \) is a pair \( <|leftConst, rightConst|> \) where \( leftConst \) and \( rightConst \) denote the left and right constructs, respectively.

A matcher is run on pairs of constructs in \( G \) to yield a set of matches \( \{m_1, ..., m_k\} \) where \( m_i \) is a pair \( <|leftConst, rightConst|> \) where \( leftConst \) and \( rightConst \) denote the left and right constructs, respectively.

The classification procedure is shown in Figure 5.2. It iterates through every match, \( m \in M \) to check whether \( m \) matched constructs that are known to be equivalent (i.e., belong to \( E \)) and to check whether \( m \) matched constructs that are known to be not equivalent (i.e., belong to \( NE \)).

Given the returned matches and the ground truth, the classified matches provide information on the distribution of scores returned by the matchers used when matching equivalent and non-equivalent constructs. This is the first step
towards knowing the likelihood of scores. For example, if a score $s$ is never observed in true positive matches, then this is evidence to support a reduction in the belief in the equivalence of construct.

The classification of matches depends on an empirical experience driven by some ground truth that reflects knowledge of the constructs provided by external sources (such as administrators or users). However, we should anticipate that the ground truth may contain false information. Likewise, there may be potentially a large set of equivalent (or not equivalent) constructs that are not made available to a MatchBench experiment. Nevertheless, the derived score distribution is an indication of the limitation of the matching algorithms available.

### 5.3.3 The Derivation of Score Distribution

A score is a value in the continuous interval $[0, 1]$. We need to partition the range into a set of discrete but disjoint buckets, and then count the matches for each of the buckets. We assume that there is no best choice for the size of a bucket, therefore we partition the range into buckets of equal width.

The procedure for acquiring the score distribution is shown in Figure 5.3. It takes as input a set of true positive matches and a set of false positive matches, and derives a score distribution $D$ for either a set of intervals $B$ given as input or a number $t$ that specifies the number of disjoint uniform intervals in $[0, 1]$. If it receives a set of intervals $B$, it iterates through each interval and counts the number of true positive and false positive matches. Otherwise, the algorithm firstly derives a set of disjoint uniform intervals based on the number of intervals specified in $t$ (Figure 5.3, line 8-10), and then count the number of true positive and false positive matches for each interval. Thus, the algorithm can derive score distribution for either uniform or non-uniform distribution.

### 5.3.4 Trustworthiness of Similarity Score

We now discuss the application of the proposal with an example.

Given a score $s$, if $s$ is more likely to be obtained in true positive matches (i.e. in associating equivalent constructs) than being obtained in false positive matches, then $s$ should have strong association to equivalence rather than to non-equivalence. For example, if we are informed that $|TP|_{[0,8,1,0]} = 10$, $|TP| = 11$, $|FP|_{[0,8,1,0]} = 2$ and $|FP| = 11$, in other words, more than 90 percent of true
Algorithm DerivingScoreDistribution

inputs:
  TP: the set of true positive matches; FP: the set of false positive matches;
  B: a set of disjoint and discrete intervals in [0, 1], where each interval is called bucket denoted by b.
  t: the number of uniform intervals.

outputs:
  D: a set of tuples, <b, count\textsubscript{tp}, count\textsubscript{fp}>, where count\textsubscript{tp} and count\textsubscript{fp} is the counts of true positive and false positive matches respectively with scores in a bucket b.

begin
1  D = new set()
2  IF B = null THEN
3      B = new set()
4  IF t = 0 THEN
5      b = [0, 1]
6      append(B, b)
7  ELSE
8      FOR i = 0 to t-1 DO
9        B = [i/t, (i+1)/t]
10       append(B, b)
11  FOR EACH bucket b ∈ B DO
12     count\textsubscript{tp}, count\textsubscript{fp} = 0
13     FOR EACH m in TP DO
14       IF the score s of m is in b THEN
15          count\textsubscript{tp} ++
16     FOR EACH m in FP DO
17       IF the score s of m is in b THEN
18          count\textsubscript{fp} ++
19     append(D, <b, count\textsubscript{tp}, count\textsubscript{fp}>)
20  RETURN D

Figure 5.3: Deriving score distributions for true positive matches and false positive matches.
positive matches are returned with scores in the interval $[0.8, 1.0]$, whilst such scores are rare in false positive matches, a score in the range $[0.8, 1.0]$ should support an increase in the belief in equivalence from a given prior belief. The derived score distributions reveal the likelihood that $s$ is obtained in true positive matches, and the likelihood that $s$ is obtained in false positive matches (i.e., $D(ss(c_1, c_2) = s|c_1 \equiv c_2)$ and $D(ss(c_1, c_2) = s|c_1 \not\equiv c_2)$ respectively).

In the following, we illustrate with an example how to derive degree of update with Bayes’s theorem by acquiring the likelihood of similarity scores from score distributions. Assume that there are two schemas $S = \{c_1, c_2, c_3\}$ where $c_i$ denotes a schema construct, and $T = \{c'_1, c'_2, c'_3\}$. The ground truth indicates that $< S.c_2, T.c'_2 >$ and $< S.c_3, T.c'_3 >$ are pairs of equivalent constructs, whilst any other pairs of constructs are non-equivalent. Assume that the following matches are returned by a schema matcher:

- $< S.c_1, T.c'_1, 0.1 >$, $< S.c_1, T.c'_2, 0.1 >$, $< S.c_1, T.c'_3, 0.2 >$
- $< S.c_2, T.c'_1, 0.2 >$, $< S.c_2, T.c'_2, 0.5 >$, $< S.c_2, T.c'_3, 0.3 >$
- $< S.c_3, T.c'_1, 0.3 >$, $< S.c_3, T.c'_2, 0.4 >$, $< S.c_3, T.c'_3, 0.8 >$

By classifying the returned matches into the sets of $TPset$ and $FPset$:

$$TPset = \{< S.c_2, T.c'_2, 0.5 >, < S.c_3, T.c'_3, 0.8 >\}$$

$$FPset = \{< S.c_1, T.c'_1, 0.1 >, < S.c_1, T.c'_2, 0.1 >, < S.c_1, T.c'_3, 0.2 >$$
- $< S.c_2, T.c'_1, 0.2 >$, $< S.c_2, T.c'_3, 0.3 >$, $< S.c_3, T.c'_1, 0.3 >$, $< S.c_3, T.c'_2, 0.4 >\}$$

We can then derive the count for matches with score in $[0.4, 0.6]$:

$$|TP|_{[0.4,0.6]} = 1; |FP|_{[0.4,0.6]} = 1$$

Given that the total number of true positive matches is 2 and the total number of false positive matches is 7, we can derive the likelihood for the bucket $[0.4, 0.6]$ as follows:

$$\frac{|TP|_{[0.4,0.6]}}{|TP|} = \frac{1}{2} = 0.5; \frac{|FP|_{[0.4,0.6]}}{|FP|} = \frac{1}{7} = 0.14$$

Assume now that the prior belief is $D(c_1 \equiv c_2) = 0.7$ (thus $D(c_1 \not\equiv c_2) = 0.3$). Using the Bayes’s theorem, the posterior belief in response to a score $s = 0.5$ (in
CHAPTER 5. QUANTIFICATION AND UPDATE OF UNCERTAINTY

\[ D(c_1 \equiv c_2 \mid ss(c_1, c_2) = 0.5) = \frac{0.5 \times 0.7}{0.5 \times 0.7 + 0.14 \times 0.3} = 0.89 \]

The result shows that the posterior belief is higher than the prior belief as the likelihood of obtaining \( s = 0.5 \) in true positive matches is higher than the likelihood of obtaining the same score in false positive matches.

5.3.5 MatchBench Experiment

We have been given experimental result from MatchBench that match 3 pairs of relational schema in 69 positive scenarios. The scenarios systematically remove similarities (this involves, e.g., change or removal of characters from the names of constructs or removal of data instances represented by constructs that are known to be equivalent) from schema constructs to cover different combination of entity types and attribute heterogeneities classified by Kim et al. [KCGS95]. Each pair of schemas contains one pair of entity types that are known to be equivalent, and 35 pairs of entity types that are known not to be equivalent. Each scenario varies the similarity of the schematic representation of the relations and data instances. The experiment evaluated COMA++ [DR02] on matching the relations, and obtained the following score distribution for true positive matches (bucket: count):

\[ [0, 0.2) : 3, [0.2, 0.4) : 49, [0.4, 0.6) : 57, [0.6, 0.8) : 48, [0.8, 1.0] : 50 \]

and the following score distribution for false positive matches:

\[ [0, 0.2) : 4557, [0.2, 0.4) : 2621, [0.4, 0.6) : 67, [0.6, 0.8) : 0, [0.8, 1.0] : 0 \]

From this data, we derived degrees of belief using Bayes’s theorem. The trend in posterior belief is shown in Figure 5.4. From this experiment, we can observe that there are significant differences in the update of belief in a small range of scores from 0.1 to 0.3, and from 0.3 to 0.5, which means that the values for the buckets representing these ranges will give significant errors relative to the raw data from the experiments. Thus, it is more appropriate to fit a curve to the data in the buckets than to use the buckets directly. For this purpose, the range from
Figure 5.4: Degree of belief update with MatchBench data.

Figure 5.5: Score distribution for true positive matches with estimated curve.
CHAPTER 5. QUANTIFICATION AND UPDATE OF UNCERTAINTY

Figure 5.6: Score distribution for false positive matches with estimated curve.

0.0 to 1.0 was partitioned into buckets with smaller widths:

\[ [0, 0.05), [0.05, 0.1), [0.1, 0.15), \ldots [0.9, 0.95), [0.95, 1.0] \]

We once again collect the score distribution for true positive matches, which is shown in Figure 5.5.

Based on the actual data, we performed curve fitting with MATLAB [MAT10] which yield the following function:

\[
|TP|_s = \frac{13}{1 + e^{-50(s-0.18)}}
\]  

(5.12)

We also collected score distributions for false positive matches (shown in Figure 5.6) with the following estimated function:

\[
|FP|_s = \frac{1150e^{-50(s-0.17)}}{1 + e^{-50(s-0.17)}}
\]  

(5.13)

With the estimated functions for true positive scores and false positive scores, we can then compute the total number of matches with the following functions,
respectively:

\[ |TP| = \int_0^1 \frac{13}{1 + e^{-50(s-0.18)}} ds \] (5.14)

\[ |FP| = \int_0^1 \frac{1150e^{-50(s-0.17)}}{1 + e^{-50(s-0.17)}} ds \] (5.15)

The functions are estimated with data from experiments on COMA++ with 207 pairs of entity types that are known to be equivalent (three pairs of schema, each with 1 pair of equivalent constructs, are matched in 69 scenarios), and 7245 pairs are known to be not equivalent (three pairs of schema, each with 35 pairs of non-equivalent constructs, are matched in 69 scenarios). We anticipate that this constitutes only a portion of all potential pairs of equivalent (and non-equivalent) entity types, thus data from the experiment may have missed the precise trend of the distribution if all potential pairs of equivalent (and non-equivalent) entity types had been studied. Moreover, experiments to identify trends in score distribution of other matchers (e.g. similarity flooding or matchers using WordNet) were not conducted, though they would follow the same strategy described here. Nevertheless, the functions have modelled the expected rational relationship between similarity scores and equivalence (and non-equivalence), i.e.
equivalence is more likely to be associated to high scores than non-equivalence.

The estimated function for true positive matches compensated for the fluctuation of actual MatchBench data after a score is greater than 0.25. The function closely models the trend before the scores is about 0.25. This means that the rate of error would increase when Bayes’s theorem is applied for deriving updates of belief when scores are greater than 0.25. In contrast, the estimated function for false positive matches precisely models the trend generated by actual MatchBench data for scores greater than 0.5, although there are slight errors before that.

Given the estimated functions, we can then use them to derive updates of belief using Bayes’s theorem using equation (5.8). Figure 5.7 shows how the result of updating belief with respect to 3 different prior (viz., 0.2, 0.5, and 0.8) for scores in the range [0.12, 0.47]. The degree of belief in the equivalence rises to 1 when scores are relatively low, around 0.47, when prior = 0.2. A score of 0.47 may encode the result of similarity comparisons on different aspects (such as schematic representation or data instances). However, on the basis of the MatchBench experimental data, the difference between the likelihood in obtaining 0.47 from the true positive matches and the likelihood of obtaining such scores from false positive matches is large enough to bring the confidence in the equivalence to the maximum level. Moreover, the experiments were based on positive scenarios. As mentioned previously, the designs of the positive scenarios systematically remove similarities from equivalent constructs. In other words, in one scenario a pair of equivalent constructs may have different names or disjoint data instances, whilst in another scenario the same pair of constructs may have similar names but disjoint data instances. This is the cause that sometimes the matcher returns scores as low as 0.1 on matching equivalent constructs. However, scores of false positive matches stay within the interval [0.0, 0.4] (Figure 5.6). In other words, there are no situations when non-equivalent constructs share very similar or the same schematic representation or data instance.

We should anticipate that the result of such updates will be influenced by the limitations of the MatchBench experiments that underpin them, as follows:

- Not all potential pairs of equivalent or non-equivalent constructs can be used for experiments. As mentioned previously, the experimental results cannot be expected to reflect the precise trend of the score distributions. Currently, MatchBench adopts the TPC-E benchmark [CAA+11] as sample schema.
Wider ranges of sample schemas can be adopted by the experiments.

- The score distribution models the trend (Figure 5.5 and 5.6) for only one matcher, i.e. COMA++, although COMA++ results on an aggregation of many individual matching algorithms. There are many other schema matching tools or algorithms that might have been used, such similarity flooding (a survey was contributed by Rahm et al. [RB01]).

- No systematic variation on the similarity between non-equivalent constructs in contrast to equivalent constructs. In real situations, there may be the case when two non-equivalent constructs use the same names, or representing similar sets of data instances. Thus, scores of false positive matches stay in the interval [0.0, 0.4] (Figure 5.6), whereas scores of true positive matches stay in a wider interval [0.1, 1.0] (Figure 5.5). With such a distribution, Bayes’s theorem rises the posterior belief to 1 from a given prior for scores greater than 0.47 (Figure 5.7).

Nevertheless, the purpose of this experiment is to demonstrate that functions for converting similarity scores to degree of belief underpinned by MatchBench do lead to rational degrees and directions of update on belief in the equivalence of two constructs.

The general degree and direction of update on the belief derived from using the trustworthiness on similarity scores derived by the proposed framework comply to the rational expectation: a score that has higher likelihood of being obtained from true positive matches than from false positive matches leads to an increase on the belief in the equivalences from a given prior belief, otherwise, it leads to a decrease.

5.3.6 Implementation of the Strategy in Dataspaces

As we have shown in Section 5.3.5, to apply the Bayesian strategy for updating degree of belief in the equivalence of constructs in the light of similarity scores, we need two kinds of information, i.e. the initial prior belief and the score distribution for deriving the degree of belief in similarity scores. The prior degree of belief in the equivalence of a pair of constructs can be acquired using different approaches, e.g. based on the principle of indifference [FH89, Ram26] or applying statistical analysis [Sha07]. As we have demonstrated in Figure 5.7, the strategy
can derive rational change to the degree of belief with different starting prior belief. With the procedures defined in Section 5.3.2 and 5.3.3 (as shown in Figure 5.2 and 5.3, respectively), score distributions can be derived for different kinds of schema matching algorithm, given MatchBench and the ground truth (i.e. a set of pairs of constructs known to be equivalent and a set of pairs of constructs known not to be equivalent). During the life cycle of dataspaces, new score distributions can be created whenever new schema matchers are available. Meanwhile, whenever a new ground truth is available, existing score distributions can be updated or recreated.

5.4 User Feedback to Degree of Belief

We have presented a strategy for converting similarity scores into degrees of belief. In this section, we present the strategy for converting user feedback into degrees of belief.

In dataspaces, schema mappings specify how tuples described in source schema constructs can be used to populate constructs of a global schema. Thus, schema mappings play the role of mediating the translation of queries between the global schema and the source schemas. In dataspaces, users may be invited to provide feedback on tuples returned for a query posed against a global schema. Such feedback can be used to evaluate the performance of schema mappings. Belhajjame et al. [BPE+10] studied the effectiveness of feedback for assessing the performance of mappings by simulating user behaviour in annotating which tuples should (and should not) be returned in the result. Such information is then used to derive the precision and recall of the mappings used in answering the query. This information is then used as a basis to select and refine schema mappings incrementally. In this section, we study how to make use of this kind of feedback as evidence for judging the degree of belief that two constructs are equivalent.

In order to learn from users whether two constructs are equivalent, we should focus on users’ feedback on tuples populated by equivalence mappings. An equivalence mapping will select all tuples of one construct to populate the other construct.

We assume that the following types of feedback annotation are provided by users on the result of a query:

- Tuples are annotated as true positive if they should be, and actually are,
in the result. Let $|TP|$ denote the total number of true positive tuples.

- Tuples are annotated as false positive if they should not be, but actually are, in the result. Let $|FP|$ denotes the total number of false positive tuples.

- Tuples are annotated as false negative if they should be, but are not actually, in the result. Let $|FN|$ denotes the total number of false negative tuples.

A query is posed on schema constructs of the global schema which are associated with source constructs by schema mappings. Thus, for equivalence mappings user feedback is direct testimony by users about the equivalence of the associated constructs. It reflects users' knowledge of the conceptual domain of constructs (in the global schema). The more tuples that are annotated as true positive by users, the more likely it is that the associated constructs represent the same set of tuples or values, and thus the more likely that the constructs are equivalent. However, the question arises as to how much we should trust the feedback. For example, this would seem to depend on the ratio of tuples are annotated by the user to the total number of tuples. Users are not expected to give feedback on all tuples returned to them. This would be against the aim of a pay-as-you-go approach.

For example, assume that a relation in the global schema is expected to represent information about 100 undergraduate students. Assume also that user provide feedback on 10 tuples and that all are annotated as true positives. However, the source relation may represent both postgraduate and undergraduate students. Furthermore, the majority of students may be undergraduate and therefore the returned tuples may only constitute a small portion of all the tuples represented by the source relation. Therefore, feedback provided on just a small set of result tuples may misrepresent users' judgments.

Notice that true negative tuples are not solicited, which are tuples that should not be, and are not actually, in the result, or that the two constructs do not represent. An alternative way to gain evidence on whether two constructs are equivalent is to study what are the tuples they do not represent, since if two constructs are equivalent, the sets of tuples they do not represent are the same. However, these sets of tuples are infinite. Thus, we focus on what are the tuples they mutually represent.

Given all the tuples of the query result evaluated based on the equivalence mapping, we count the number of tuples annotated as true positive ($|TP|$), the
number of tuples annotated as false positive ($|FP|$) and the number of tuple annotated as false negative ($|FN|$). With these counts of tuples, we can then derive the following ratio $K$, that is the ratio of tuples that are mutually represented by both constructs to all the constructs represented by both constructs, which should influence the degree of belief in the equivalence of both constructs:

$$K = \frac{|TP|}{|TP| + |FP| + |FN|}$$ (5.16)

For example, with a high value of $K = 0.9$ (in other words, a large ratio of $|TP|$ to $|FP|$ and $|FN|$), and the prior belief is 0.1, it is plausible to increase the degree of belief from the prior belief. The degree of increment should depend on how likely we obtain $K = 0.9$ in the situation that two constructs are equivalent.

We have mentioned that the trustworthiness of the feedback depends on the amount of tuples that are annotated. The ratio between total number of tuples annotated with feedback and the total number of tuples in $c_1$ and $c_2$ is referred to as support and defined as follows:

$$supp = \frac{|TP| + |FP| + |FN|}{|c_1| + |c_2|}$$ (5.17)

where $|c_1|$ and $|c_2|$ denote the total number of tuples stored in the constructs $c_1$ and $c_2$. A high value for support indicates that users have provided feedback on a high proportion of result tuples with respect to $c_1$ and $c_2$ and should increase the degree of belief in $K$ for judging the equivalence of $c_1$ and $c_2$. In other words, if support is low, we should hold the belief, whilst if $K = 1$ and $supp = 0.1$, we should increase the degree of belief but only slightly despite $K$ being at the maximum level.

Given $K$ and $supp$ as evidence, following equation (5.7), we can derive the posterior belief, $D(c_1 \equiv c_2 \mid K = v_1 \land supp = v_2)$, of a pair of constructs being equivalent using Bayes’s theorem:

$$\frac{D(\psi \mid c_1 \equiv c_2)D(c_1 \equiv c_2)}{D(\psi \mid c_1 \equiv c_2)D(c_1 \equiv c_2) + D(\psi \mid c_1 \not\equiv c_2)D(c_1 \not\equiv c_2)}$$ (5.18)

where $\psi \equiv K = v_1 \land supp = v_2$.

$K$ should be proportional to $D(K = v_1 \land supp = v_2 \mid c_1 \equiv c_2)$, since the higher the value of $K$, the higher the likelihood that $c_1$ and $c_2$ represent the same set of
tuples or values. Thus, if we know \( c_1 \) is equivalent to \( c_2 \), we should expect a high value of \( K \). Based on the postulated relationship between \( K \) and the expected degree and direction update of degree of belief, we define

\[
D(K = v_1 \land supp = v_2 \mid c_1 \equiv c_2) = 0.5 + (K - 0.5)f(supp)
\]  
(5.19)

and

\[
D(K = v_1 \land supp = v_2 \mid c_1 \not\equiv c_2) = 0.5 + (0.5 - K)f(supp)
\]  
(5.20)

The value of \( K \) is in the interval \([0, 1]\). To decide whether the value of \( K \) is high or low, we choose 0.5 as a fair point to judge \( K \). In other words, \( K \) is defined to be high, if \( K \) is greater than 0.5. Otherwise, it is low.

\( f(supp) \) is a function of support returning a value in \([0, 1]\). Thus, if \( f(supp) = 0 \), this leads to \( D(K = v_1 \land supp = v_2 \mid c_1 \equiv c_2) = D(K = v_1 \land supp = v_2 \mid c_1 \equiv c_2) \), which leads in turn to the posterior belief being the same as the prior belief. The higher \( f(supp) \), the greater the difference between \( D(K = v_1 \land supp = v_2 \mid c_1 \equiv c_2) \) and \( D(K = v_1 \land supp = v_2 \mid c_1 \equiv c_2) \). This is a fair and adaptive function for incorporating factors that affect the trustworthiness of \( K \). For example, \( f(supp) \) can be replaced by another function, \( f' \), that derive trustworthiness of \( K \) based on other factors. In the next section, we present the approach for estimating the trustworthiness of \( K \) based on the experimental setup of Belhajjame et al. [BPE⁺10].

### 5.4.1 The Trustworthiness of User Feedback

We have presented an approach for converting user feedback into degrees of belief. The higher the value of \( K \), the higher the likelihood that two constructs represent the same set of tuples, and the higher the degree of belief that the constructs are equivalent. Thus, we can derive a degree of belief as to whether two constructs are equivalent given \( K \) as evidence. However, we cannot assume that users will provide feedback on all the tuples returned to them. For example, a user may specify feedback on 10 out of 1000 tuples returned to him. Thus, there is a difference between the ground truth (i.e. the value of \( K \) that would be derived from feedback posed on all tuples returned to users) and \( K \) (derived from feedback posed on one subset actually returned to users). We have introduced the notion
of support for measuring the amount of feedback that users have contributed. To demonstrate the use of the strategy for quantifying uncertainty, we conducted an experimental study of the relationship between support and error in $K$. The error in $K$ is defined as the difference between $K$ and the ground truth.

We adapt the experiments used in [BPE+10] to estimate a function for modelling the relationship between support and error in $K$. Belhajjame et al. [BPE+10] used the Mondial data integration benchmark [May99a]. An integration relation was created, as well as mappings between the integration relation and the relations in the Mondial database (as data sources). To specify the mappings, IBM Infosphere Data Architect was used. Three candidate mappings were created for populating the integration relation. A set of tuples was created by randomly selecting a subset of tuples returned from the mappings, which were then used to populate the integration relation. Those tuples serve two purposes: (1) the automatic generation of synthetic user feedback (i.e., the randomly selected tuples represent tuples that should be returned, which are compared to tuples returned by the candidate mappings to derive feedback instances), and (2) they allow the computation of a gold standard $K'$ (i.e., computed based on counting the number of true positive, false positive and false negative feedback that should be specified on all tuples returned by the mappings in comparison to the tuples should be returned) to be compared with $K$ computed from feedback instances. For each run of the experiment, 10 feedback instances are given on randomly selected subsets of 3000 tuples (thus this simulates ten users being asked to provide feedback on a subset of the 3000 tuples). Based on the result of the simulated feedback, we calculated the average error in $K$:

1. For each of the 10 feedback instances, we derive its $K$.
2. $K$ is then subtracted from $K'$ (derived from giving the correct feedback on all the 3000 tuples).
3. The absolute value of this subtraction is the error for that feedback instance.
4. Finally, the average is calculated on the errors derived from the 10 feedback instances.
5. Repeat 1-4 for 100 times on different subsets of the 3000 tuples with different sizes (from 10 tuples to 1000 tuples), i.e. to simulate different supports on

[^0]: [http://www-01.ibm.com/software/data/studio/data-architect]
specifying the feedback.

6. By applying curve-fitting provided in MATLAB, we estimate a function that maps support to error in K.

Figure 5.8 shows the derived average error in K against support together with the estimated curve, which is:

\[
error_{InK} = -\frac{\ln(supp)}{7500 \times supp} + 0.025 
\]  

Although there are errors in the estimated function for \( supp < 0.025 \), the function models well the general trend of the actual experimental data. From the experimental result, we can observe that the average error falls to less than 0.1 after \( supp \approx 0.01 \), where 0.01 is about 30 tuples out of 3000, in other words a relatively small proportion. Based on the estimated function in equation (5.19), \( f(supp) \) is defined as:

\[
f(supp) = 1 - error_{InK} = \frac{\ln(supp)}{7500 \times supp} + 0.975 
\]  

since the higher the average error, the lower the degree of belief in K. However, since the curve will never touch the vertical axis, and support equal to zero means that no feedback is given and the degree of belief in K with \( supp = 0 \) is zero. \( f(supp) \) is revised to

\[
f(supp) = \begin{cases} 
0, & \text{if } supp = 0 \\
\frac{\ln(supp)}{7500 \times supp} + 0.975, & \text{otherwise}
\end{cases} 
\]  

We have empirically studied one factor that can influence the trustworthiness of K. There are potentially other factors can affect the trustworthiness of K, such as the quality of feedback, i.e. users may make mistake when providing feedback or their knowledge on the conceptual domain of the constructs are limited.

Given the estimated function of support, we now demonstrate the use of the proposed functions (for converting feedback into belief) in updating degree of belief in the equivalence with Bayes’s theorem.

Figure 5.9 shows the first set of results derived from updating belief with respect to different prior belief when support for different values of K is high. The values of K give high influence on the degree of update in the context of a
Figure 5.8: Average errors in $K$ with respect to support.
Figure 5.9: Degree of belief update for different priors for a fixed, high level of support.

Figure 5.10: Degree of belief update for a fixed, high prior for different levels of support.
high support, and the higher the value of $K$, the stronger the influence. Figure 5.10 shows the results of belief update with different levels of support and a fixed prior belief. The degree of update is almost identical from $support = 0.8$ to $support = 0.1$. When support falls from 0.1 to 0.11, the degree of update is smaller, a reflection of the average error in $K$ (as shown in Figure 5.8), which drops to a low level (i.e. 0.1) in that case. Both experiments show that the proposed functions for converting feedback into degree of belief can deliver, using Bayes’s theorem, rational values for the degree and direction of one belief update in response to $K$ and support. In the following section, we demonstrate the use of the proposed functions for incrementally assimilating evidences into degree of belief in the hypothesis using Bayes’s theorem.

### 5.5 Scenario Study: Incremental Update of Degree of Belief

We have presented the functions for converting evidence in the form of similarity scores and of user feedback into degrees of belief. We have shown now the degree of belief in the equivalence of schema constructs can be updated using Bayes’s theorem. In dataspaces, evidence for updating degrees of belief becomes available in a undefined order throughout the dataspace life-cycle. Therefore, flexibility of the frameworks for updating belief is a necessity in dataspaces. In this section, we consider different scenarios as to when evidence becomes available to be assimilated by the framework.

We assume that in a dataspace system we have derived similarity score distributions with MatchBench as shown in Figure 5.4 and 5.5. Furthermore, we have the functions for converting user feedback into degree of belief, which were discussed in Section 5.4. Each scenario represents the situation when several pieces of evidence are given at different points of the life cycle of the dataspace system. As illustrated in Section 5.2 (and in Figure 5.1), each pieces of evidence is assimilated into the existing belief in the equivalence of pairs of constructs with three different staring prior belief (i.e., 0.1, 0.5 and 0.9). For this purpose, we adopt MS Office Excel to simulate the derivation of corresponding posterior belief from a given prior belief, and the derived posterior belief is passed onto the next update round with the next evidence. We will show the derived posterior belief from each update of degree of belief using the Bayes’ theorem, and demonstrate the
rationality and flexibility of using the proposed framework for updating degree of belief.

**Scenario 1** Assume that, in the bootstrapping stage, we are given two similarity scores from different matchers:

\[ <\text{MATCHER}_1, 0.2>, <\text{MATCHER}_2, 0.3> \]

where \( \text{MATCHER}_1 \) and \( \text{MATCHER}_2 \) denote two matchers. At the improvement stage, we are given the result of feedback from a user and a similarity score from another matcher:

\[ <\text{FEEDBACK}, <K, 0.5>, <\text{supp}, 0.9>>, <\text{MATCHER}_3, 0.2> \]

With these pieces of evidence, we firstly apply equations 5.12 and 5.14 to derive \( D(ss(c_1, c_2) = 0.2 \mid c_1 \equiv c_2) \), while apply equations 5.13 and 5.15 to derive \( D(ss(c_1, c_2) = 0.2 \mid c_1 \not\equiv c_2) \). With the Bayes’ theorem (equation 5.8), we derive the posterior belief with three different values of prior belief, i.e. 0.1, 0.5 and 0.9. As shown in Figure 5.11, the posterior belief drops after encountering the first similarity score, 0.2, as evidence, because MatchBench data (Figure 5.5 and 5.6) indicates that the likelihood of obtaining a score of 0.2 in matching equivalent constructs is lower than that of obtaining that score in matching non-equivalent constructs. At the third round, given \( K = 0.5 \) and \( \text{supp} = 0.9 \), we apply equations 5.19 and 5.22 to derive \( D(K = v_1 \land \text{supp} = v_2 \mid c_1 \equiv c_2) \), while apply equation 5.20 and 5.22 to derive \( D(K = v_1 \land \text{supp} = v_2 \mid c_1 \not\equiv c_2) \). With Bayes’ theorem (equation 5.8), and different values of prior belief, we derive the posterior belief. We observed from the degree of belief update that the update of belief not only depends on evidence, but also on the prior belief at each round on which the belief is updated.

**Scenario 2** In this case, assume that the similarity score given by \( \text{MATCHER}_2 \) arrives after \( \text{MATCHER}_3 \), at the improvement stage:

\[ <\text{MATCHER}_1, 0.2>, <\text{FEEDBACK}, <K, 0.5>, <\text{supp}, 0.9>>, <\text{MATCHER}_3, 0.2>, <\text{MATCHER}_2, 0.3> \]

Figure 5.12 shows 4 rounds of belief update. The posterior belief is derived by applying the same procedures as in Scenario 1, i.e. using equations 5.12 to 5.15
CHAPTER 5. QUANTIFICATION AND UPDATE OF UNCERTAINTY

for deriving degree of belief in similarity score and equations 5.19 to 5.22 to derive degree of belief in $K$. The largest rates of increment (from different prior belief) are at the second round of update due to user feedback, which is then brought down by the next piece of evidence, i.e. similarity score of 0.2. Nevertheless, Figure 5.12 reveals that the belief update (with different starting prior belief) eventually arrives at the same final posterior belief as in Scenario 1 even though the evidence arrives in a different order.

**Scenario 3** In this case, we start with a low similarity score, and then two high values of $K$ and of support are given:

$$< \text{MATCHER}_1, 0.2 >, < \text{FEEDBACK}_1, < \text{K}, 0.9 >, < \text{supp}, 0.2 >>, \quad < \text{FEEDBACK}_2, < \text{K}, 0.98 >, < \text{supp}, 0.2 >>, \quad < \text{MATCHER}_2, 0.9 >$$

Figure 5.13 shows that with different starting prior belief, the degree of belief were updated to a value close to 1 after three rounds of update due to high values of $K$ and similarity score. At the third update round, given the same value of $K$, it is rational that the lower the prior belief, the lower the posterior belief.

As in previous scenarios, Figure 5.13 reveals that the update of degree of belief depends on the proposed functions for converting evidence as well as on given prior belief at each round of the update. These scenarios indicate that the proposed functions lead to the derivation of rational updates with Bayes’s theorem.

With the deployment of this framework and a substantial amount of training for MatchBench (on pair of constructs known to be equivalent and non-equivalent), as well as the procedures discussed in Section 5.3 and 5.4 for converting similarity scores and user feedback into degrees of belief, the degree of belief in the equivalence of constructs is always ready to be incrementally updated whenever a new piece of evidence is available, which complies to the pay-as-you-go style of data integration brought by dataspace systems.
Figure 5.11: Incrementally assimilating evidence into degree of belief, Scenario 1.

Figure 5.12: Incrementally assimilating evidence into a degree of belief, Scenario 2.
Figure 5.13: Incrementally assimilating evidence into a degree of belief, Scenario 3.

5.6 Propagation of Uncertainty by Model Management

The *match* operator postulates a degree of similarity between schema constructs and outputs a set of associations between schema constructs. *inferCorrespondences*, *merge*, *compose* and *copy* also output schematic correspondences between newly created and/or existing schema constructs, but based on manipulating existing associations or schematic correspondences. The derived schematic correspondences are then used for generating schema mappings, or deriving new correspondences by other *merge* and *compose* operations. Since the degree of belief in the equivalence of associated constructs is important in query evaluation [DHY09a, GMSS09], it would be useful if model management operators could propagate uncertainty in the equivalence of input constructs to output schematic correspondences. For example, the *copy* operator creates a copy of existing constructs, thus the degree of belief in the equivalence of the original construct and its copy is 1. The *inferCorrespondences* operator derives schematic correspondences based on the result of a set of schema matching algorithms. Thus, it
can adopt the strategy proposed in Section 5.3 to derive degree of belief in the
equivalence of constructs associated by the output correspondences. Other model
management operators, i.e. ViewGen and diff, take schematic correspondences
as input, but do not derive new associations between constructs. The merge
operator derives a reconciled superabstract based on a pair of associated super-
abstracts postulated to be equivalent. The question is what is the degree of belief
that the reconciled superabstract is equivalent to each of the source constructs in
terms of the degree of belief in (or evidence that may be assimilated to support)
the equivalence of the source superabstracts. In this section, we propose heuristic
functions for propagating uncertainty through merge and compose.

There has been research on schema merging with correspondences that as-
signed a degree of uncertainty. Rather than deriving a single reconciled schema,
Magnani et al. [MRMM05] proposed a strategy for deriving a set of plausible
reconciled schemas to avoid the generation of duplicated representations of the
same concept. For example, assume that a superabstract Student in one schema
is associated to superabstracts ugStudent and pgStudent in the sources via two
one-to-one correspondences and one one-to-many correspondence:

\[< \text{Student}, \text{ugStudent}, 0.7>, < \text{Student}, \text{pgStudent}, 0.5>,\]

\[< \text{Student}, \text{ugStudent}, \text{pgStudent}, 0.9>\]

Each correspondence is assigned a degree of belief derived by combining a set of
manually given degrees of belief in the equivalence of associated constructs with
the Dempster’s combination rule [MRMM05]. The output schema should not
contain three duplicate representations for the same set of tuples on students.
To solve this problem, the mechanism proposed by Magnani et al. derives one
schema for each of the reconciled superabstracts that are generated by each of
the correspondences. However, it does not derive correspondences and degrees
of belief for the reconciled superabstracts. In Chapter 4, we presented an al-
gorithm for merge, which generates correspondences between constructs in the
reconciled schema and constructs of source schemas based on the input schematic
correspondences. We aim in this section to propose heuristic functions for de-
riving degree of uncertainty for the output correspondences based on degree of
uncertainty given in the input correspondences.
Schema Merge

The algorithm proposed for merge in Chapter 4 takes as input two schemas and a set of schematic correspondences, and derives a reconciled schema that represents the union of the concepts represented by the source schema together with schematic correspondences that associate constructs of the reconciled schema to constructs in the source schemas. The algorithm avoids the derivation of duplicate representations for the same concept (or attributes of the concept) in the reconciled schema. Recall that the subroutine DeriveReconciledSuperabstract (Figure 4.2) creates a reconciled superabstract for every pair of superabstracts that are postulated to be equivalent, which in turn creates reconciled superlexicals. It also creates one copy of every superabstract or superlexical in the source schema that is not associated to any superabstract or superlexical, respectively, in the other source schema. For each created superabstract and superlexical in the reconciled schema, the algorithm derives correspondences to associate them to constructs in the source schema. As mentioned in Section 3.2, each input schematic correspondence may be assigned with the degree of belief in the equivalence of associated constructs. The merge operator should decide how to assign a degree of belief to correspondences that it outputs.

The subjective degree of belief in the equivalence of the schemas to be merged is the result of assimilating different evidence with the strategy discussed in the previous sections. Thus, the problem is to study how evidence support for the equivalence of constructs associated by input correspondences can be used to judge the equivalence of constructs associated in the output correspondences.

A reconciled construct is a newly create construct. Thus, no similarity comparison or user feedback are given for judging the equivalence between the reconciled construct and each of the source constructs. Based on the principle of indiscernibility, the degree of belief should be assigned with 0.5 which indicates that we can not judge the equivalence.

Although we are given degree of belief, $d$, in the equivalence between the source constructs, this can not be used to derive the degree of belief in the equivalence between the reconciled construct and each of the source constructs for the following reasons:

1. $d$ could be derived based on assimilating results of similarity comparisons on the source constructs. However, any similarity comparison applied on the reconciled construct and the source constructs can yield a complete
different score if the same comparison method is applied on comparing the
source constructs. For example, the reconciled construct does not have
data instances to be used for performing instance level comparison. Thus,
any instance comparison method applied on comparing one of the source
construct with the reconciled construct will return \textit{null}. Moreover, the
dataspace may not be able to assign a name for the reconciled construct
that reconciles the name conflict between source constructs. For instance,
two superabstracts are named as \textit{undergraduate} and \textit{postgraduate} can lead
to the generation of a reconciled superabstract for representing all students,
but the name for the reconciled construct need to be selected with the
assistance of dictionaries or domain experts, and dataspaces may not have
such resources to use.

2. \(d\) could also be derived based on assimilating a set of user feedback inst-
ances. Recall that, users provide feedback on tuples of one of the source
constructs \(c_1\) (or on tuples of the other construct \(c_2\)) used for populating \(c_2\)
(or, respectively, for populating \(c_1\)). The feedback are given based on his
or her knowledge on one of the source constructs, but not on the reconciled
construct.

Meanwhile, the \textit{merge} operator copies any constructs that are unique to the
source schemas into the reconciled schema. It also generates correspondences for
associating the original construct to its copy. In this case, the degree of belief in
the equivalence between the source construct with its copied representation is 1.

\textbf{Correspondence Composition}

The algorithm presented in Section 4.3 implements the \textit{compose} operator which
postulates equivalence between schema constructs, and derives superabstract
(Figure 4.9) or superlexical (Figure 4.11) correspondences. The question is whether
we can use the degree of belief in (in other words, the evidence that supports)
the equivalence of constructs associated by input correspondences to judge the
equivalence of constructs associated by the output correspondence.

Let \(d_1\) be the degree of belief \(D(c_1 \equiv c_2)\) and \(d_2\) be \(D(c_2 \equiv c_3)\), \(D(c_1 \equiv c_3)\)
should be assigned as

\[D(c_1 \equiv c_3) = d_1 \times d_2\] (5.23)

since
1. The equivalence relationship is transitive [Gri04], and the less degree of belief that \( c_1 \) or \( c_3 \) are representing the same concept as \( c_2 \), the less the degree of belief that \( c_1 \) and \( c_3 \) represent the same concept.

2. If \( d_1 = 1 \) (or \( d_2 = 1 \)), \( D(c_1 \equiv c_3) = d_2 \) (or \( D(c_1 \equiv c_3) = d_1 \)), since if \( c_1 \) represents the same concept as \( c_2 \), then \( c_1 \) is another \( c_2 \).

3. If either \( d_1 = 0 \) or \( d_2 = 0 \), \( D(c_1 \equiv c_3) = 0 \), since if \( c_1 \) represents different concepts from \( c_2 \), and \( c_2 \) represents the same concept as \( c_3 \), then \( c_1 \) must not represent the same concept as \( c_3 \).

With equation (5.23), if \( d_1 = 0.8 \) and \( d_2 = 0.8 \), then \( D(c_1 \equiv c_3) = 0.64 \), and if \( d_1 = 1 \) and \( d_2 = 0.3 \), \( D(c_1 \equiv c_3) = 0.3 \). Moreover, if \( d_1 = 0.2 \) and \( d_2 = 0.2 \), \( D(c_1 \equiv c_3) = 0.04 \), and if \( d_1 = 0 \) and \( d_2 = 0.2 \), \( D(c_1 \equiv c_3) = 0 \).

The heuristic function can be inserted between line 15 and 16 of the subroutine \textit{composing Superabstract Correspondences} (Figure 4.9), and between line 5 and 6 of the subroutine \textit{Composing Superlexical Correspondence} (Figure 4.11).

Summary

The heuristic functions were proposed based on reasoning over the subjective degree of belief given in the input correspondences. The derived degree of belief can then be incrementally updated with the Bayesian framework described in Sections 5.2 to 5.4. For example, once a correspondence is derived between two constructs, the user may provide feedback when querying the constructs or they are matched with similarity comparison methods. These evidence can then be used to update the degree of belief derived by the \textit{compose} operator.

5.7 Conclusion

This chapter presented our contribution on the quantification of uncertainty for the judgment of equivalence between schema constructs in the context of dataspaces. We have defined functions for converting similarity scores and user feedback into degrees of belief. The result indicates that with enough ground truth to train MatchBench, we can acquire similarity score distributions that contain less error. With the procedure discussed in Section 5.3, the derivation and refinement of score distributions can be performed through the life-cycle of dataspaces.
Furthermore, scenario studies have demonstrated the flexibility provided by our Bayesian framework making it suitable to for deployment in the context of dataspaces.

We anticipate that there might exist other methods for acquiring, or other factors that can affect, the trustworthiness of evidence (as mentioned in the previous sections). There may also exist other kinds of evidence that can influence the judgment of the equivalence, or other types of hypothesis that can be raised and studied (for example the selection predicate \( \sigma_1 \) is the right way to map tuples of one superabstract to be represented by the other superabstract can be supported by instance level similarity comparisons as evidence). Nevertheless, Bayes’s theorem can support the incremental update of belief as long as the appropriate functions are provided for converting the evidence into degrees of belief.

Although, the work defined in this chapter has yet been implemented in a dataspace management system, it can be part of the future work of developing uncertainty management applications to be integrated into any dataspace system.
Chapter 6

Conclusion

In this chapter, we summarise the contributions that have been reported in this dissertation and comment on future research opportunities.

6.1 Summary of Research Contributions

In Chapter 1, we have the vision of dataspaces, and made two observations that motivated the contributions reported in this dissertation. Firstly, dataspaces trade off greater quality of query service at the start for lower upfront cost based on high automation at the bootstrapping stage. Such low quality is inherent in any attempt to automate correspondence elicitation, and is subsequently propagated to, and impairs, the quality of the outcome of other dataspace operations, such as mapping generation, mediating schema generation and query evaluation [HBF+09]. The shortcomings of correspondence elicitation lead to uncertainty in the output correspondences. Uncertainty management for dataspaces has emerged as an important issue. However, very few research proposals have addressed the issue of how uncertainty can be quantified in the context of dataspaces. None of the existing work has studied the management of uncertainty in (and using) user feedback [BPE+10, CVDN09], which is the centrepiece of the dataspace vision.

Secondly, the kind of schematic correspondences proposed by Hedeler et al. [HBM+10] provide rich information on a wide-range of schematic heterogeneities. Moreover, there have been research proposals [GBM08, BM07, HBM+10] on
studying the functional decomposition of problems in bootstrapping, usage, improvement and maintenance of dataspaces using model management. Model management operators were originally proposed to operate on associations generated using schema matching algorithms. No prior work has studied the characterisation of model management operators over expressive schematic correspondences such as introduced in [KCGS95].

From these observations, this dissertation set out to study the characterisation of model management operations on schematic correspondences, and the quantification and propagation of uncertainty in schematic correspondences for dataspaces founded on model management.

6.1.1 Schema Mapping and Model Management on Expressive Schematic Correspondences

Express schematic correspondences have been proposed in the Manchester Dataspaces Project [HBM+10] as an intermediate knowledge structure between the similarity associations produced by schema matching and the executable expressions produced by schema mapping tools and techniques in the classical data integration design stage. Schematic correspondences contain rich information on many types of schematic heterogeneities originally studied and classified by Kim et al. [KCGS95]. In this dissertation, the first contribution we have made was on the design of algorithms for deriving mappings from such expressive schematic correspondences. The generated mappings, which later become a crucial element for query translation between schemas, reconcile the schematic heterogeneities captured in the input schematic correspondences. The schematic correspondence model we designed can express mappings implemented as relational algebra which contain a collection of selection, projection, join or union operations. Proceeding from this work, we then showed that the generation of mappings from expressive schematic correspondence can be automated. This work was published in [MBPF09].

We then focused on the proposal of incorporating model management to underpin the bootstrapping and maintenance stage of dataspaces [GBM08, BM07, HBM+10]. In this context, we made the second contribution to characterise the merge, compose and diff model management operators over expressive schematic correspondences so that the operators output schematic correspondences which
can then be input to the mapping generation algorithm and thereby automatically generate schema mappings for supporting query evaluation.

All the operator together with the mapping generation have been implemented in Java and integrated into the dataspace ToolKit system [HBM+10].

### 6.1.2 Quantification and Propagation of Uncertainty in Schematic Correspondences

The study of related work in Chapter 2 has revealed that although there have been many research proposals focused on enriching the semantic information conveyed in schematic correspondences [KQCJ07, SRM08, BM07, MBPF09, PB03] and on characterising model management operators over them, very few of them addressed the issue that uncertainty is inherent in the elicitation of correspondences and in schema matching. The study of related work also revealed that existing proposals for uncertainty management in data integration and dataspaces have tended to assume, without much justification, that the degree of uncertainty (typically expressed as probability) is quantified in the output of correspondence elicitation method. Most proposals in probabilistic data management [SDH08, DHY09a, GMSS09] rely on a prior step for quantifying uncertainty but offer no account of the process by which such quantification is obtained. We have therefore proposed a framework for assimilating different kinds of evidence, such as the results of similarity comparison methods and user feedback on query results, into degrees of belief (which we proposed for expressing uncertainty) in the context of dataspaces. We have shown that this framework can flexibly update uncertainty regarding the equivalence of schema constructs with evidence that become available in any order. Finally, we have proposed heuristic functions for assigning a degree of belief to the schematic correspondences that are produced by the *merge* and *compose* operators based on the degrees of belief in the inputs provided.
6.2 Limitations and Future Research

This dissertation has proposed a high-level model of schematic correspondences, that provides information for reconciling schematic heterogeneities for query translation. We have also characterised schema mapping generation and model management operators over such a model. At the time of writing, there has been research work in progress on deriving or inferring schematic correspondences, although some of the research has specified high-level modelling of schematic correspondences [SRM08, KQCJ07, PB03, ABBG09c, ABBG09b]. A parallel research activity has been initiated at the same period of this research in the project [Guo11] on inferring schematic correspondences from the results of schema matching. Nevertheless, we anticipate that the elicitation of information such as selection predicates, join predicates, superlexical functions, which is needed for reconciling Kim’s schematic heterogeneities, are still challenging research problems. Despite the ongoing research activity on inferring schematic correspondences and to provide best effort unified data access services, the current shortcomings could be mitigated with two approaches:

1. The management of uncertainty needs to be in place. In this research, we have studied the hypothetical judgment on the equivalence of schema constructs. There are still potentially other kinds of hypothesis that could be addressed, such as uncertainty in inferring schematic heterogeneities and their resolution methods. Especially, the inference of selection predicates for reconciling horizontal partitioning heterogeneities can become difficult if the partition keys (superlexicals) are not explicitly specified. There are many ways a superabstract can be partitioned and it is hard to infer how the partition was made. For example, information on students can be partitioned into local and overseas students based on the country they are from, and country may be embedded in the string that represents home address. Furthermore, a source may apply nested partitioning technique. For example, one source represents student using a single superabstract Student, while the other source partitions students into Undergraduate, Postgraduate and Address, where Address represents addresses of both undergraduate and postgraduate students. There may be several superabstract correspondences associate Student to superababstracts in the other source. For example, an one-to-many vertical partitioning correspondence
associate Student to Undergraduate and Address, an one-to-many vertical partitioning correspondence associate Student to Postgraduate and Address, and an one-to-many horizontal partitioning correspondence associate Student to Undergraduate and Postgraduate. Thus, there may be different degrees of belief for inferring each of the partitioning methods.

2. We have reviewed research work on soliciting feedback from users to improve schema mapping and other dataspaces operations. Some of this work [BPF+11] can be extended to support feedback on information of schematic heterogeneities. For example, users can be prompted or assisted to provide feedback on superlexical functions or selection predicates.

3. The trustworthiness of similarity scores and feedback was studied based on experiments on a limited set of sample data, viz, the TPC-E benchmark [CAA+11] and Mondial dataset [May99b]. The proposed uncertainty update framework and functions for quantifying uncertainty can be further evaluated with data from different application domains of information integration, such as in bioinformatics [BY06, LMMS+07].
Bibliography


[CVDN09] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In SIGMOD Conference, pages 87–100, 2009.


[JXW+08] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, and Peter J. Haas. Mcdb: a monte carlo


