Map-IT: An advanced multi-strategy and learning approach to schema matching
a learning, flexible & extendible framework for matching schemas based on FlexiMatch

Sander Bosman

Supervisors:
Dr. Ir. Maurice van Keulen
Ir. Arthur van Bunningen
Ir. Ander de Keijzer

Enschede, 23 april 2007
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE OF CONTENTS</td>
<td>2</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>4</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>5</td>
</tr>
<tr>
<td>1.1 SCHEMA MATCHING</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Sync-IT</td>
<td>5</td>
</tr>
<tr>
<td>1.3 FLEXIMATCH</td>
<td>5</td>
</tr>
<tr>
<td>1.4 MAP-IT</td>
<td>6</td>
</tr>
<tr>
<td>1.5 ASSIGNMENT</td>
<td>8</td>
</tr>
<tr>
<td>1.5.1 Assignment formulation</td>
<td>8</td>
</tr>
<tr>
<td>1.5.2 Research questions</td>
<td>8</td>
</tr>
<tr>
<td>1.5.3 Goals</td>
<td>8</td>
</tr>
<tr>
<td>1.5.4 Assignment boundaries</td>
<td>9</td>
</tr>
<tr>
<td>1.6 OVERVIEW</td>
<td>9</td>
</tr>
<tr>
<td>FLEXIMATCH</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Overview of FLEXIMATCH</td>
<td>10</td>
</tr>
<tr>
<td>2.2 ELABORATION OF FLEXIMATCH</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1 Format converter</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2 Global Intermediate Schema (GIS)</td>
<td>11</td>
</tr>
<tr>
<td>2.2.3 Schema combiner</td>
<td>11</td>
</tr>
<tr>
<td>2.2.4 Validators</td>
<td>12</td>
</tr>
<tr>
<td>2.2.5 Similarity cube</td>
<td>13</td>
</tr>
<tr>
<td>2.2.6 Prediction aggregator</td>
<td>13</td>
</tr>
<tr>
<td>2.2.7 Transitive combiner</td>
<td>13</td>
</tr>
<tr>
<td>2.2.8 Suggestion combiner</td>
<td>14</td>
</tr>
<tr>
<td>2.2.9 Mapping generator</td>
<td>14</td>
</tr>
<tr>
<td>2.2.10 User feedback handler</td>
<td>14</td>
</tr>
<tr>
<td>EVALUATION SYSTEM</td>
<td>15</td>
</tr>
<tr>
<td>3.1 CRITERIA FOR AN EVALUATION SYSTEM</td>
<td>15</td>
</tr>
<tr>
<td>3.2 EVALUATION SYSTEM REQUIREMENTS</td>
<td>19</td>
</tr>
<tr>
<td>3.3 DESIGN</td>
<td>21</td>
</tr>
<tr>
<td>3.4 IMPLEMENTATION</td>
<td>24</td>
</tr>
<tr>
<td>3.4.1 Data structure evaluation system</td>
<td>24</td>
</tr>
<tr>
<td>3.4.2 Implementation evaluation system</td>
<td>25</td>
</tr>
<tr>
<td>3.4.3 Evaluation configurator</td>
<td>25</td>
</tr>
<tr>
<td>3.4.4 Configuration handler</td>
<td>26</td>
</tr>
<tr>
<td>3.4.5 Evaluation result</td>
<td>26</td>
</tr>
<tr>
<td>3.4.6 Feedback provider</td>
<td>26</td>
</tr>
<tr>
<td>3.4.7 Evaluation result handler</td>
<td>26</td>
</tr>
<tr>
<td>DUPLICATED SEARCH VALIDATOR</td>
<td>27</td>
</tr>
<tr>
<td>4.1 DESCRIPTION</td>
<td>27</td>
</tr>
<tr>
<td>4.2 DESIGN</td>
<td>29</td>
</tr>
<tr>
<td>4.3 IMPLEMENTATION</td>
<td>34</td>
</tr>
<tr>
<td>META-LEARNER</td>
<td>36</td>
</tr>
<tr>
<td>5.1 DESCRIPTION</td>
<td>36</td>
</tr>
<tr>
<td>5.2 DESIGN</td>
<td>37</td>
</tr>
<tr>
<td>5.3 IMPLEMENTATION</td>
<td>37</td>
</tr>
</tbody>
</table>
Summary

Schema matching is the task of finding semantic correspondences between elements of two schemas. It is needed in many database applications, such as integration of web data sources and data warehouse loading. Automating matching schemas has been under investigation in many areas for already some decades, but matching schemas is still often done manually by domain experts. To reduce the amount of user effort as much as possible, automatic approaches combining several match techniques are required. While such match approaches have found considerable interest recently, the problem of how to best combine different match algorithms still requires further work.

At the start of this report we investigate the schema matcher FlexiMatch. Several recommendations where addressed in the master's thesis from FlexiMatch. Possible feasibility of the recommendations and new devised potential improvements are taken as the starting point of this thesis.

This thesis describes the schema matching framework Map-IT, which is based on FlexiMatch. The schema matcher supports the multi-strategy approach, with each strategy represented as a Validator. Key characteristics of Map-IT are:

- Map-IT and its Validators can learn from previous mappings.
- Validator can easily be added to or selected from the Validator repository, in order to boost future matching performance or to adapt the system to the match task at hand.
- Current Validators exploit different database information aspects.
- Map-IT adapts the weights of the Validators to its environment using the Meta-Learner.

The main perspective of Map-IT is that the elements of schemas relational column and table elements from a certain domain share domain concepts. Map-IT learns these concepts by using the user feedback of previous mappings. Schema elements belonging to a certain domain concept can have various representations. Within Map-IT, concepts are therefore represented by interconnected subconcepts. The subconcepts in this group are derived from the different schema elements representations of a certain domain concept, which Map-IT encountered during previous schema mappings.

These subconcepts and their interrelations are used as an intermediate schema to derive matches between input schema elements. Schema elements are therefore first matched with subconcepts. Schema elements that are matched with similar, interrelated subconcepts are then combined with each other.

The main goal of Map-IT of this thesis is to make an evaluation system to evaluate the schema matcher and the improvements in an efficient and accurate way. Although there are few aspects of Map-IT that could be improved or implement to enhance the performance, the evaluation of Map-IT shows that the main goal is achieved: an improvement schema matcher in contrast to the previous schema matcher FlexiMatch, containing an automated evaluation system.
1 Introduction

This thesis is about improving schema matching. For this thesis the framework/prototype called FlexiMatch will be used. In the following sections Schema Matching and FlexiMatch are introduced. The subsequent two sections elaborate on the context of this thesis, such as who formulated the assignment and what its underlying motivation for improving the framework/prototype. In the assignment formulation the formulation, question, goals and boundaries are described. The introduction concludes with overview section, the paragraphs that will be addressed in this thesis are described.

1.1 Schema Matching

A schema is a set of related elements, such as tables, columns, classes, XML elements or attributes. Schema matching is the process of determining semantic correspondences or matches between elements in two different schemas [Madhavan et al., 2003]. A schema matching result or mapping consist of all possible matches between the elements of both schemas.

Schema matching examples:
A schema matching example is given in figure 1.1, which is taken from [Doan and Halevy, 2005]. Here two matches are depicted: ‘location’ of schema S semantically corresponds to ‘area’ of schema T and ‘price ($)’ semantically corresponds to ‘list-price’ in schema T.

The example above describes a simple match situation. A complex match situation occurs when in schema S the attribute ‘location’ consists out of two attributes instead of one at the moment e.g. attribute province and attribute city.

1.2 Sync-IT

E-System Solutions is a company which was founded in 2001 by Joris Visser and Sander Bosman, both are currently students at the Twente University. The company is specialized in Internet related projects and database-synchronization. One of the products that have been developed by E-System Solutions is Sync-IT.

Sync-IT supports the synchronization of data among databases e.g. Outlook, Outlook Express, MSSQL, Exchange. In the current version of Sync-IT the database schemas must be mapped manually, which is a time consuming process. From out of this prospective the project FlexiMatch was defined to overcome this procedure, to make database schemas mapping semi-automatically.

One of the goals of FlexiMatch project was to integrate the framework/prototype into the Sync-IT. This goal wasn’t achieved during the FlexiMatch project, but will be addressed and solved in this thesis.

1.3 FlexiMatch

FlexiMatch is schema matching framework/prototype based on several combined techniques from LSD [Doan et al., 2003], COMA [Do and Rahm, 2002] and Cupid [Madhavan et al., 2001]. FlexiMatch is a schema match system that supports the multi-strategy approach. Multiple-strategy exploits different aspects of available information i.e. schema information or instance data, using matchers (in FlexiMatch called Validators). The current framework/prototype of FlexiMatch [Appendix A: FlexiMatch framework] contains two Validators, namely exploiting...
schema and instance based information. *FlexiMatch* also learns from previous approved and disapproved mappings. Besides finding matches between schema elements of both source schemas, it can also create (after learning) matches that the system would not have come up with otherwise.

One of the purpose of *FlexiMatch* is that the elements of schemas (relational column and table elements, or XML elements and attributes) from a certain domain, share *domain concepts* for example ‘Name’ and ‘Naam’. *FlexiMatch* learns these concepts from previous mappings. Schema elements belonging to a certain domain concept can have several *subconcepts* (see previous example). Within *FlexiMatch* concepts are represented by interconnected *subconcepts*. The *subconcepts* in this group are derived from the different schema elements of a certain domain concept, which *FlexiMatch* encountered during previous schema mappings.

These *subconcepts* and their interrelations are used as an *intermediate schema* to derive matches between input schema elements. Schema elements are therefore first matched with *subconcepts*. Schema elements that are matched with similar, interrelated *subconcepts* are then combined with each other.

The main goal of the *FlexiMatch* was to make a learning framework/prototype in which the learned knowledge could be used in future match tasks.

*FlexiMatch* is developed by a student, during his master thesis, at the *University of Twente* together with E-System Solutions.

### 1.4 Map-IT

This thesis is about extending and improving *FlexiMatch* framework in such way, that more correct produced mappings are generated. The commercial purpose of *E-System Solutions* is to improve and exploit the product, using the name *Map-IT*. Further on in this thesis the term *Map-IT* will only be used to address this project.

Several recommendations where addressed in the thesis of *FlexiMatch*. *E-System Solutions* saw potential in some of the recommendations and adjusted them to their need. Possible feasibility, of the recommendations and new devised potential improvements, must be investigated.

In the next subsections the recommendations are addressed containing an introduction, which is elaborated later on in the thesis.

#### New Validators

To improve the matching results the following new Validators could be used:

- *Duplicate search* [Bilke and Naurmann, 2005]
- *N-Gram* [Do and Rahm, 2002]

*Duplicate search Validator*

The duplicate search approach performs a horizontal matching: which searches for similar rows (or tuples) in the tables, in effect for detecting duplicates. Once a few duplicates have been discovered, deriving a schema matching is simple (in principle): Same or similar data values among the duplicates imply corresponding attributes of the schemas.

The precision of the result, as described in the paper [Bilke and Naurmann, 2005], goes up to almost 100%. Therefore using this as a possible new Validator for framework/prototype must be investigated. The purpose of *Duplicate search Validator* is to improve the current suggestions and to find new suggestions that framework/prototype didn’t find.
N-Gram Validator
With the N-Gram Validators, strings are compared according to their set of n-grams, i.e., sequences of n characters, leading to different variants of this matcher, e.g., Digram (2), Trigram (3). For example, Cust and Customer are similar as their trigram sets, \{cus, ust\} and \{cus, ust, sto, tom, ome, mer\}, respectively, share two trigrams cust and ust.

The purpose of N-Gram Validator is to improve the current suggestions and to find new suggestions that framework/prototype didn’t find.

Meta-Learner
Several parameters can be specified for the framework/prototype e.g. enabling edit distance Validator (Validator1) or instance based Validator (Validator2), adapting the weight of edit distance Validator and instance based Validator. These parameters are, at the moment, only adaptable manually in the code. Using the Meta-Learner this process can be automated so that the system will adapt the parameters in its specific environment.

The purpose of introducing the Meta-Learner is that Map-IT will (semi)automatically adjust itself according to its environment.

Missing Match problem
The missing match problem is one of the unsolved problems of the FlexiMatch project. This problem will be addressed and clarified in section 6.3.

Data type similarity
Relational databases supports several data types for storing data in the database e.g. smallint, bigint, float, nvarchar, etc. In the framework/prototype schema elements are combined with subconcepts, which are based on schema element names. An implementation that Cupid [Madhavan et al., 2001] addresses is, using a table containing all possible compatible data types.

The purpose of Data type similarity is to reduce the solution space and also allowing only compatible data types with direct combinations and combinations with subconcepts.

Evaluation schema matcher
Several comparisons of schema matching systems have been carried out [Melnik and Rahm, 2002]. Using an automated Evaluation system, the improvements of the Validators and framework/prototype can be validated. The current framework doesn’t support an evaluation system to do efficient evaluation validation. Therefore it is necessary to have an evaluation system. The purpose of automated Evaluation system is to evaluate and test changes to the framework in accurate and efficient way.

Sync-IT Integration
Due to lack of integration of the current framework/prototype into Sync-IT, a part of the research consists of, how to combine the Map-IT into Sync-IT. Several issues need to be addressed before integration can be proceeded e.g. a standalone package for optimal distribution or a shareable project for other projects.

The purpose of Map-IT is being integrated into Sync-IT for better commercial exploitation of Sync-IT.
1.5 Assignment
In this section the assignment is defined. In the next section the assignment formulation is given. In subsequent sections we discuss what questions are answered in this report, what the goals of this thesis are but also what will not be included in this thesis.

1.5.1 Assignment formulation

Improve schema matching using techniques as new validators and learning capabilities. First, to design additional validators which improve the schema matching. Second, to introduce learning capability, which automatically adjusts the schema matching parameters, using user feedback. Finally, to evaluate the schema match improvements.

1.5.2 Research questions
The next questions are answered in subsequent chapters of this report.
- How can new Validators improve the schema matching results?
  - What approach do they use?
  - How do they work?
  - What are the advantages and disadvantages of the approach?
- How can Meta-learner improve the schema matching results?
  - Which approaches are available?
  - How do they work?
  - What are the advantages and disadvantages of these approaches?
- How can the evaluation system evaluate the improvements?
  - What approaches are there available?
  - What are the performance criteria?
  - How do they work?
  - What are the advantages and disadvantages of these approaches?
- How can solution space reduction be accomplished using data type similarity?

1.5.3 Goals
The assignment formulation requires that the following elements are taken into account:
- Design and implement an improved schema matching framework/prototype based on FlexiMatch.
- Map-IT should be able to generate an evaluation overview: how the system performed according to the measurements and calculated metrics.
- Map-IT should be able to learn from its environment e.g. adapting weights of a Validator, so that the parameters automatically are adjusted to its environment to improve the matching results.
- Integrate Map-IT within Sync-IT.
1.5.4 Assignment boundaries
To get a clearer view of what is contained in the assignment, the following elements will not be included:

- Map-IT will not be so intelligent that it is able to generate complex matches. Complex matches can be taken into account after including a schema level structure matcher is added as a base-learner.
- Map-IT will not take care of the actual synchronization or data translation and migration between the databases. These steps are the actual data synchronization process, and this thesis is only intended to support that process.
- Map-IT will be based on the FlexiMatch framework/prototype, no effort will be made to fundamentally redesign the framework architecture.
- Map-IT will only support relation models, because the current schema matcher doesn’t allow XML schemas and it is not a part of this thesis to implement support for XML schemas.

1.6 Overview
This section briefly mentions what is discussed in every subsequent chapter.

In chapter 1 Introduction & Assignment formulation, an introduction of master thesis is done and the assignment formulation. This chapter includes an introduction and what actions are undertaken to solve the optimize mend and improvement of Map-IT. In chapter 2 FlexiMatch describes the current schema matcher. In chapter 3 the Evaluation system addresses the available techniques of existing evaluation system with their advantages and disadvantages, including design and implementation. In chapter 4 the Duplicated search Validator from the DUMAS project is described. The advantages and disadvantages are addressed after that the design specification is made. Based on the design section the implementation is described. In chapter 5 the Meta-Learner addresses existing Meta-Learners with their advantages and disadvantages, including a design and an implementation. In chapter 6 Other schema match improvements address several other possible improvements with their advantages and disadvantages including a design and an implementation. In chapter 7 Evaluation discusses the evaluation of Map-IT, using the evaluation system. Observations that are acquired from evaluation are also addressed. In chapter 8 a Conclusion is discussed derived from this research project. In the last chapter 9 Recommendations are done to enhance future performance.
2 FlexiMatch

In this chapter the framework of FlexiMatch is described. In the next section an overview of the schema matching framework is addressed. In subsequent sections we discuss the framework in detail.

2.1 Overview of FlexiMatch

This section describes FlexiMatch’s framework as depicted in Appendix A: FlexiMatch’s framework. This section describes the framework from the left side to the right side.

The framework supports one source schema type, which can be fed into the format converter. As output the schema matcher produce a schema match consisting of mapping combinations between schema 1 and schema 2. The Dataset, which is only supported at the moment as input, is a generic data structure of the Microsoft .Net framework that contains the schema, structural information and instance data of the database. Dataset S1 and S2 are fed to the schema matcher at the left side of the framework. The Format converter component converts the input schemas into the internal representation within FlexiMatch, namely graphs.

The internal representation is passed on to the Schema combiner, which generates combinations between schema elements and the Global Intermediate Schema (GIS). GIS is an internal graph and contains subconcepts of the real life concepts. The combinations that are generated consist of three types: between schema 1 elements and subconcepts, schema 2 elements and subconcepts and between schema 1 and schema 2 elements.

The generated combinations are passed on to the Validators. The Validators assign similarity values to these combinations, indicating to what extent both elements in each combination are alike, according to the information cue (e.g. schema names) each Validator exploits. There are three similarity cubes depicted in FlexiMatch’s framework, one for each combination type. After validation, each validated combination is put in the Similarity cube corresponding to its type.

Per combination, the Prediction aggregator combines all the different similarity values, assigned by the different Validators, into a single aggregated similarity value. Elements that were combined with subconcepts (GIS) are transitively combined with each other by the Transitive combiner.

The Suggestion combiner then merges the transitively derived combinations with the combinations coming from directly combining both source schemas. From this resulting set of combinations, the Mapping generator produces match suggestions for the user, including the corresponding similarity values. In one or more iterations, the user gives feedback on the suggested matches and adds matches FlexiMatch did not come up with. If the total mapping is finally accepted, the GIS component and the ‘intelligent’ Validators (i.e. Validators that do something with the approved, disapproved and added matches of the accepted mapping) learn from it.

2.2 Elaboration of FlexiMatch

This section describes FlexiMatch’s framework as depicted in Appendix A: FlexiMatch’s framework in further detail. Every subsequent section discusses another framework component including its advantages and disadvantages. The order of framework component sections is the same as the order of components in the framework, from the left to the right. This is also the execution order.
2.2.1 Format converter

COMA [Do and Rahm, 2002] and Cupid [Madhavan et al., 2001] are using graphs as internal representation for the data structure. Several data structure, of database schemas, relational database and XML can be converted into a graph. Therefore FlexiMatch uses also graphs as an internal representation of the data structure. At the moment the Format converter supports one data structure, namely Datasets.

The graph representation consists of three types, namely: Nodes, Edges and Instances. All schema elements are converted to nodes. Additional properties can be stored within every node, depending on the data model to be converted. The edges of the graph represent the relations between schema elements. A relation, in the relational model, is the relation between two tables. For the instance based Validators (see section 2.2.4 Validators later on), instance data is stored with every node derived from an instance data containing column type schema element.

Example schema presentation

To illustrate the function of the Format converter, assume the input table ‘Worker’, with columns ‘Postal code’, ‘Name’ and ‘Number’ a graphical view of the final internal representation within FlexiMatch, including instance data, is depicted in figure 2.1: Example input.

2.2.2 Global Intermediate Schema (GIS)

The Global Intermediate Schema (GIS) of FlexiMatch is based on the mediated schema component of LSD [Doan et al., 2003]. The mediated schema is a schema, containing real life domain concepts. A real life domain concept is Zip Code. In LSD a real life domain concept is Postcode, this can be represented by interconnected subconcepts ‘Postal code’ and ‘ZIP code’ within FlexiMatch (see figure 2.2: Example GIS schema).

The difference between FlexiMatch and LSD is that in FlexiMatch real life domain concepts consist of interconnected subconcepts. Each subconcept keeps a record of its data that is collected from the database (see figure 2.2: Example GIS schema). This data is called Instance data.

Another difference with the LSD system, is that the GIS is not set up manually. This is because the subconcepts and relating interconnections are automatically learned from accepted mappings, which reduces the required manual effort in matching schemas. The interconnection between the subconcepts doesn’t only represent accepted mappings but also a Trust value. The Trust value represents the number of times a match was approved and rejected of the corresponding subconcept interrelation.

2.2.3 Schema combiner

The Schema combiner is the component of FlexiMatch which is responsible for generating the combinations. Based on the names and data types, the Schema combiner generates combinations between GIS, the first source schema and the second source schema. This results in three types of combinations, which are summarized below and discussed in subsequent paragraphs:

- Combinations between subconcepts and source schema 1 elements.
- Combinations between subconcepts and source schema 2 elements.
- Combinations between source schema 1 and source schema 2 elements.
Example schema combination
To illustrate how the schema combiner combines schemas, see figure 2.3: Possible combinations. This figure is explained in subsequent paragraphs in the discussion of the different types of combinations.

Combinations with subconcepts
The dotted lines from figure 2.3 represent examples of element with subconcept combinations. The similarity of the element and the subconcept is only based on the name similarity. FlexiMatch makes use of the knowledge of previously accepted mappings. Those mappings are generated combinations between source schema elements and learned subconcepts, and later transitive combined these combinations over similar subconcepts with the Transitive combiner.

Combinations between schema elements
The continuing line from figure 2.3 represents a direct combination between schema elements. There is no subconcept in the GIS which is similar enough to a certain schema element. The approval of match suggestions, coming from these combinations, will result in the addition of new subconcepts (see 2.2.2 Global Intermediate Schema)

2.2.4 Validators
The idea behind the Validators is based on similar components that are addressed in LSD, COMA and Corpus-Based Knowledge Representation [Madhavan et al., 2003]. Several schema match systems use a different terminology for the term Validators. COMA uses the matchers, LSD and Corpus-Based Knowledge Representation use base learners. Each schema match system addresses multiple Validators e.g. Name Validator, Instance data Validator, Structural Validator and Data type Validator. The strategy behind using multiple Validators is to exploit multiple information in improvement of matching results. In FlexiMatch each Validator is manually assigned a weight during its execution. This weight is later used by the Prediction aggregator (see 2.2.6) to produce a weighted average for each combination. During the schema matching process, the Validators assign a similarity value to each combination generated by the Schema combiner (see 2.2.3). This value is between 0 and 1 and reflects to what extent the elements of each combination are alike according to the information cue the respective Validator is exploiting. The similarity value is stored afterwards in the Similarity cube (see 2.2.5).

In the remainder of this section, we discuss the schema name based edit distance Validator (Validator1) and the instance based Validator (Validator2) in more detail.

Edit distance Validator1
Validator1 computes a similarity for two elements ‘elt1’ and ‘elt2’ based on similarity of schema names. This is accomplished based on the Levenshtein Distance-algorithm [Levenshtein Distance].

Instance based Validator2
Validator2 computes a similarity value between two elements ‘elt1’ and ‘elt2’ based on equality of character averages, using normal distribution [Normal distribution], in the character-sets of their instances. The computation is performed based on the assumption that character averages of both elements ‘elt1’ and ‘elt2’ are normally distributed [Normal distribution].
2.2.5 Similarity cube

The idea behind the Similarity cube is based on COMA Similarity cube. The similarity values computed by all Validators for combinations are stored in Similarity cubes. FlexiMatch has three types of combinations (see 2.1 Overview of FlexiMatch). Therefore there are three Similarity cubes containing combination similarity values. Each combination belongs to a certain Similarity cube. Because of the multiple Validators and different type of combinations each Similarity cube contains multiple dimensions as depicted in figure 2.4: Similarity cube.

2.2.6 Prediction aggregator

The Prediction aggregator component is based on COMA’s aggregation operation on the Similarity cube. The Prediction aggregator receives a Similarity cube with combinations which are validated by all the Validators which were put to use within FlexiMatch. Based on the validated combinations, the Prediction aggregator computes a single similarity value for every combination. In FlexiMatch there are three types of combinations (see 2.1 Overview of FlexiMatch), therefore also three Prediction aggregators.

2.2.7 Transitive combiner

The Transitive combiner principle is based on the transitive nature of the schema level reuse matchers of COMA. The Transitive combiner (transitively) combines schema 1 elements with schema 2 elements when they are combined with similar subconcepts.

The transitive closure function assumes a transitive nature of the similarity relation between elements, i.e. if a is similar to b and b to c, then a is (very likely) also similar to c. The Transitive combiner does the same, considering elements from source schema 1 and source schema 2, and b as the same subconcept or similar subconcepts. An example of a transitive closure is depicted in figure 2.5, the dotted line represent that Home number (Schema1) is mapped on Number (GIS), the dotted line Phone (Schema2) is mapped on Telephone (GIS) and Telephone (GIS) is related to Number (GIS) therefore (solid line) Home number (Schema1) is similar to Phone (Schema2). The same holds for the solid line between Postcode (Schema1) and Zip code (Schema2).

One-step GIS subconcept paths

Transitive combinations are created if both concerning subconcepts are connected in the GIS with a path of at most 1 step (the path is zero steps if both subconcepts of both combinations are the same). We do not consider multi-step paths, because if a match suggestion (see example figure 2.6: Example multiple subconcepts), based on a multi-step path in the GIS, is rejected by the user, it is impossible to tell which step(s) of the chain is responsible for the mismatch e.g. full-name subconcept is combined with last-name subconcept. The path between the subconcepts consist of full-name – name – last-name, the problem is to identify which of the paths is the incorrect one. Weakening all paths between the subconcepts is not an option, because the link between name and last-name can be correct in other match situations. If there is a relation between full-name and last-name a direct connection between the subconcepts will be created.
2.2.8 Suggestion combiner

The Suggestion combiner merges the combinations that were derived from the Transitive combiner (GIS combinations) and Prediction aggregator (direct combinations).

Normalization

The similarity value of a transitive combined suggestion is derived by a.o. multiplying with the trust value of the relation between the concerning subconcepts. Normalization is required because the final similarity value would be degenerated compared to the similarity values of suggestions derived from direct combinations, in the case the trust value is less than 1. This gives a wrong image about how good both suggestions are. To compensate for this effect, the final direct similarity value is retrieved by taking its similarity value to the square.

2.2.9 Mapping generator

The Mapping generator generates match suggestions which are then presented to the user. The suggested matches consist of potentially multiple schema 2 elements match candidates per single source schema 1 element.

2.2.10 User feedback handler

The User feedback handler handles the user feedback at the end of each schema match action. The user can give several user-feedback options. The options that are available are:

- Approve a match suggestion: When one of the suggested match candidates for a single source schema 1 element is correct, the user can approve the match. The GIS and intelligent Validators learn from an accepted mapping.
- Reject a match suggestion: If a suggested match candidate for a given schema 1 element is not correct, the user can reject the match suggestion. When a match suggestion is created by the Schema Combiner with GIS and the match suggestion is rejected the relationship between the subconcepts is decreased.
- Ignore a match suggestion (default); suggested matches that are not approved nor rejected are automatically ignored.
- Add a match; the system didn’t come up with the match suggestion. The user has the possibility to manually add a combination. When added the mapping FlexiMatch learns from the mapping in two ways:
  - GIS is updated (see section 2.2.2); creates subconcepts if necessary and the relationship between the subconcepts (if there is a relationship if not one is created) is strengthened
  - Updating internal administration of the instance based Validator (see section 2.2.4); the internal administration of the character averages is updated.
3 Evaluation system

In this chapter we describe the evaluation system. This section is divided into several sections, summed up below:

- **Criteria for an evaluation system**, which describes the criteria for an evaluation system.
- **Evaluation system requirements**, which describes the requirements based on the criteria.
- **Design**, which describes the design of the evaluation system including the architecture.
- **Implementation**, which describes the implementation of the evaluation system based on the design.

### 3.1 Criteria for an evaluation system

An evaluation can consist of different aspects that can be relevant, i.e. domain, methods, execution time, expected results. Depending on what there is to be achieved, the relevant aspects need to be investigated. The purpose of Evaluation system is to evaluate learning behavior, consistency of matching results and improvements of the framework in an efficient way.

In [Do et al., 2002] it is argued that due to the different ways schema match systems have been evaluated, it is difficult to compare the effectiveness of each single system. The paper describes a set of criteria influencing the effectiveness of a schema matching approach. In this e.g., the chosen test problems, the design of the experiments, the metrics used to quantify the match quality and the amount of saved manual effort. With the criteria, future schema matching evaluations can be better documented. Their results are more reproducible and comparisons between different systems are easier.

Before comparison can take place, several facts need to be addressed e.g. pre-processing. The facts are summarized and elaborated on below:

- **Input**: what kind of input data has been used (schema information, data instances, dictionaries etc.). The simpler the test problems are and the more auxiliary information is used, the more likely the systems can achieve better effectiveness.
- **Output**: what kind of information has been included in the match result (mappings between attributes or whole tables, nodes or paths etc.) and what is the correct result. The less information the systems provide as output, the lower the probability of making errors but the higher the post-processing effort can be.
- **Human effort**: how much human effort is saved and how is this being quantified.
- **Match Quality measures**: what metrics have been chosen to quantify the accuracy and completeness of the match result. This is because the evaluations usually use different metrics, it is necessary to understand their behavior, i.e. how optimistic or pessimistic their quality estimation is.
- **Learning capability**: to prove that the schema matcher performs better with learning capability in contrast to non learning schema matcher.

**Input**

As described above, the simpler the test problems are and the more auxiliary information is used the more likely the systems can achieve better effectiveness. Real world schemas come in all sizes and contain various amounts of auxiliary data.

To cover various aspects of a real world scenario, different test sets can be used:

- **Schema language (relational, XML schemas, etc.)**: Different schema languages can exhibit different facets to be exploited by match algorithms.
- **Number of schemas and match tasks**: With a large number of different match tasks, it is more likely to achieve a realistic match behavior.
- **Schema information**: An important issue is the number of the schema elements for which match candidates are to be determined. The larger the input schemas are, the greater the search space for match candidates will be, which often leads to lower match quality.
• **Schema similarity:** when a match task with schemas of the same size becomes more difficult if the similarity between them drops.

• **Auxiliary information used:** using dictionaries or thesauri, or the constraints that apply to certain match tasks can greatly improve the result quality.

**Output**
The output of a match system is a mapping indicating which elements of the input schemas correspond to each other. To assess and to compare the output of different match systems all the output must be uniformly represented. **Map-IT** represents values between 0 (strong dissimilarity) and 1 (strong similarity) to indicate the plausibility of the correspondences. This is also how other schema match systems create their output. The quality and quantity of the correspondences in a match result still depend on several aspects:

• **Element representation:** Schema matching systems often use, like **Map-IT**, a graph for the internal representation of schemas. **Figure 3.1** shows a simple match problem with two schemas in directed graph representation; a sample match between nodes would be **Contact**↔**ContactPers**. **PO2** contains elements **DeliverTo** and **BillTo** which should be considered independently. Some systems return only one of two match suggestions e.g. **PO1.Contact**↔**PO2.DeliverTo.ContactPers** and forget the **PO1.Contact**↔**PO2.BillTo.ContactPers** match. Therefore this kind of mapping is not allowed to be generated by **Map-IT**, only paths with the length one.

• **Cardinality:** One or more elements of the first schema may be matched with one or more elements of the second schema (local cardinality of 1:1, 1:n/n:1). For example **figure 3.1**, **PO1.Contact** must be matched to both **PO2.DeliverTo.ContactPers** and **PO2.BillTo.ContactPers**. Grouping these two match relationships within a single correspondence, we have 1:n local cardinality. Representing them as two separate correspondences leads to 1:n global and 1:1 local cardinality. Most automatic match approaches, like **Map-IT**, are restricted to 1:1 local cardinality by selecting for a schema element the most similar one from the other schema as the match candidate.

The problems that can occur with **Output** of the schema matcher need to be investigated and validated during the evaluation.

**Human Effort**
Given that the main purpose of automatic schema matching is to reduce the amount of manual work, quantifying the user effort still needed, is a major requirement. To analyze the human effort, one should consider pre-match and post-match effort. Pre-match is the effort that is required before an automatic matcher can run and the post-match effort is to add the **false negatives** and to remove the **false positives** from the final match result. Pre-match effort may include:

• Configuration of the various parameters of the match algorithms, e.g. threshold and weight values.

• Specification of auxiliary information, such as domain synonyms and constraints.

In most schema match systems or approaches the simple measures **Recall** and **Precision** (the explanation of the metrics is done further on in **Match Quality measures**) only partially consider the post-match effort. To estimate the effort to add **false negatives** can be calculated by:

1−Recall, the formula 1−Precision can be regarded as an estimate for the effort to remove **false negatives** contrast. The combined measures F-measure (α) and Overall take both kinds of effort into account. Overall assumes equal effort to remove **false positives** and to **false negatives** although the latter may require manual searching in the input schema. Determining that a match is correct requires extra work not considered in both Overall and F-measure (α).
Finally, the specification of the real match result depends on the individual user perception about correct and false correspondences as well as on the application context. Hence, the match quality can differ from user to user and from application to give the same input schemas. This effect can be limited to some extent by consulting different users to obtain multiple subjective real match results.

Comparison criteria
The criteria that are used for evaluating different schema match systems are from two different areas. The criteria are summarized and elaborated on below:

- **Match Quality measures**: to quantify the accuracy and completeness of the match result. This is because the evaluations usually use different metrics. It is necessary to understand their behavior, i.e. how optimistic or pessimistic their quality estimation is.
- **Learning capability**: what metrics have been chosen to quantify the learning capability of a schema matcher? This is done in such a way that after a few iterations it proves: how fast the schema matcher learns and the match accuracy is improved in contrast to non-learning schema matchers.

Match Quality measures
The usual measures for reporting effectiveness of semantic retrieval systems are **Precision** and **Recall**. These measures are regularly used for schema matching systems [Melnik and Rahm, 2002]. In principle, to acquire the precision and recall measures, the process illustrated in figure 3.2 must be executed.

First a test collection must exist. Usually this is a set of real-world schema matching problems. Next, the system S (see figure 3.2) creates a set of number of schema mappings which rank high $As=\{a_1, \ldots, a_n\}$. The system expects that these mappings are correct. Independently, a human evaluator (see figure 3.2) selects only the semantically correct schema mappings, creating in that way a set of correct answers $Cs=\{a_1, \ldots, aj\}$. The human evaluator inspects the whole search space and selects all and only correct mappings. A way to measure how good a schema match system performs in matching two schemas, is by considering the relation between the amount of real matches found, the amount of derived matches by the system and the total amount of possible matches. To provide a basis for evaluating the quality of automatic match strategies, the match task first has to be manually solved.

In figure 3.3 two circles are presented, the circle A (false negatives) with the continuing line represents the **Real matches** that manually would be created. The dashed circle C (false positives) represents the matches the schema match system created. The overlapping part B (true positives), are the matches the schema match system created and created by user manually. The last symbol D (true negatives), are false matches which were correctly discarded by the system.

Let's assume that $|X|$ stands for the number of elements of set X. Based on the cardinality of these sets, two common measures, **Precision** and **Recall** can be calculated.

**Precision** is the share of real matches among the found ones, and is computed as follows:

$$\text{Precision} = \frac{|B|}{|B| + |C|}$$

**Recall** is the share of real matches that the system came up with, and is computed as follows:

$$\text{Recall} = \frac{|B|}{|A| + |B|}$$
The most ideal case, when there are no false negatives and false positives returned, Precision = 1 and is Recall = 1. However, neither Precision nor Recall alone can accurately assess the match quality. Recall can easily be maximized at the expense of a Precision by returning all possible matches e.g. the cross product of two input schemas. On the other side, a high Precision can be achieved at the expense of the Recall by returning only few (correct) matches.

Hence it is necessary to consider both measures separately or a combined measurement. Several combined measures are proposed and are elaborated on below:

\[
F\text{-}Measure(\alpha) = \frac{|B|}{(1-\alpha)|A| + |B| + \alpha^*|C|} = \frac{\text{Precision} \times \text{Recall}}{(1-\alpha) \times \text{Precision} + \alpha \times \text{Recall}}
\]

\[
E(b) = 1 - \frac{1 + b^2}{\frac{1}{(|A| + |B|)} + \frac{1}{(|B| + |C|)}} \times \frac{\text{Precision} + \text{Recall}}{1 + b^2}
\]

\[
\text{Overall} = 1 - \frac{|A| + |C|}{|A| + |B|} = \frac{|B| - |C|}{|A| + |B|} = \text{Recall} \times \left(2 - \frac{1}{\text{Precision}}\right)
\]

Since Precision and Recall, despite their popularity, are not always the most appropriate measures for evaluating, alternative measures have been proposed over the years. We discuss the metrics described above in more detail in the following sections.

**F-Measure (The Harmonic Mean)**

F-Measure (\(\alpha\)) is the harmonic mean of Recall and Precision. The function F-Measure assumes values between the interval [0,1], where 0 is a strong dissimilarity and 1 strong similarity. Further, the harmonic mean F-Measure assumes a high value only when both recall and precision are high. Therefore, determination of the maximum value for F-Measure can be interpreted as an attempt to find the best possible compromise between precision and recall.

In the paper of Do et al [Do et al., 2002], it is described that the F-Measure metric is more optimistic than Overall metric. The F-Measure is the most common metric used in information retrieval [Motro, 2002]. For the same Precision and Recall values, F-Measure is still much higher than Overall metric. Unlike the other metrics, Overall can have negative values if the number of false positives exceeds the number of the True positives, i.e. Precision < 0.5.

**The E Measure**

As with the harmonic mean, this is also a measure which combines Recall and Precision. The formula was proposed by Van Rijsbergen and is called the E measure [Rijsbergen, 1979].

\(B\) is a user specified parameter that represents the relative importance of Recall and Precision. When \(b = 1\) the measure works as the complement of the harmonic mean function F-Measure (\(\alpha\)). To indicate that Precision is more important than recall, \(b\) is set to greater than 1. When \(b\) is set below 1, Recall will be more important than the Precision. The average setting of \(b = 0.5\), a choice giving equal weight to precision and recall and giving rise to the normalized symmetric difference as a good single number indicator of system performance.

**Overall measure**

This metric was developed specifically in the schema matching context. The main purpose of the Overall metric is to quantify the post-match effort needed for adding missed matches and removing false ones.
The Overall metric is based upon human effort needed to transform a match result obtained automatically into the intended result. We assume a strict notion of matching quality i.e. being close is not good enough. For example, imagine that a matching algorithm comes up with five equally plausible match candidates for a given element, then decides to return only two of them and misses the intended candidate(s).

The Overall metric does not address semi-automatic matching, in which the user iteratively adjusts the result and invokes repeatedly the matching procedure. Thus, the accuracy results we obtain here can be considered ‘pessimistic’, i.e., the matching algorithm may be ‘more useful’ than what the metric predicts. The goal is to estimate how much effort it costs the user to modify the proposed match result \( P = \{(x_1, y_1), \ldots, (x_n, y_n)\} \) into the intended result \( I = \{(a_1, b_1), \ldots, (a_m, b_m)\} \). The user effort can be measured in terms of additions and deletions of map pairs performed on the proposed match result \( P \). This metric was also used in [Melnik, 2002] and [Do and Rahm, 2002].

**Learning capability**

In the paper of FlexiMatch different evaluation experiments were done: if a schema matcher performs better with or without learning capability. The purpose of this evaluation is to determine how fast a schema matcher improves the schema match accuracy after a few iterations using learning capability. With the term improves we define when the overall performance of the schema matcher improves after each iteration. To determine, how fast a schema matcher improves its schema matching results, using the learning capability, could be by doing an experiment whereby a set of fixed parameters is set. To achieve a set of optimal parameters can be done with an Evaluation system which determines this. When the optimal parameters are set the experiments can prove that running multiple schema match iterations improves the schema match results. A metric for determining, if the schema match result constantly improves, could be:

\[
\text{Given } \forall t | \Rightarrow \text{Overall}_t \leq \text{Overall}_{t+1}
\]

The metric defines that after an iteration the overall performance is greater/equal than the previous iteration. The metric allows that after an iteration the performance needs to increase or remain the same.

**3.2 Evaluation system requirements**

In this section the requirements are specified for the evaluation system.

**Input**

To have a realistic match behavior of Map-IT, the purpose is to have schemas of multiple domains. Using different sizes of database schemas e.g. small, medium and large exploits different characteristics of schema matcher.

At the moment Map-IT doesn’t contain any auxiliary information e.g. dictionaries, thesauri or constraints that apply to certain match tasks. A possible solution, that may greatly improve the performance of Map-IT, is to import WordNet [WordNet] into the GIS. WordNet defines relations between words as synonymy, hyponymy, etc. which could be used to match schema elements.

**Output**

Several issues concerning the representation using XML schema languages need to be dealt with. The current version of Map-IT supports only relation databases to be imported. It is possible to convert XML schemas into a relational database structure. The problem that is addressed, in the Output section above relating to XML schema problems, is not a problem with relation database structure. The disadvantage using only relational databases is that a Structural Validator (that is not implemented in Map-IT) does not exploit this kind of information (structural information) in a relation database.
To log the behavior of the schema matcher during an iteration, several components need to log various amounts of information. The information that we can think of are:

- The time the schema matcher needed to create combinations between the two source schemas.
- The time each Validator needed to validated a combination.
- The time the schema matcher needed to process the user feedback.

**Human Effort**
The pre-match effort consists of several things. One of them is, creating sets of schemas containing the correct matches between the test schemas. Creating the sets of true positive will decrease the human effort for evaluating the sets of match results.

Map-IT doesn’t support at the moment auxiliary information which will not be taken into account as human effort for importing this information.

**Match Quality measures**
To make a choice between the described metrics they are first evaluated with their advantages and disadvantages.

**F-Measure (The Harmonic Mean)**
The following describes the advantages and disadvantages of the harmonic mean metric:

**Advantages**
- Combines the Precision and Recall into one measurement, so that the lowest value has the most impact.
- Widely used measurement for information retrieval systems.
- Good metric for average calculation of the observations.
- Is flexible due that \( \alpha \) is adjustable, values of \( \alpha \) greater than one indicates that the user is more interested in Precision than Recall, while value smaller than 1 indicate Recall is more important than Precision.

**Disadvantages**
- When the Precision or Recall are significantly different towards each other, it will have a negative impact on the end result.
- Applicability for schema matching systems is unknown.
- Does not address semi-automatic matching, which the user iteratively adjusts the result and invokes repeatedly the matching procedure. The accuracy results we obtain here can be considered ‘pessimistic’.

**The E Measure**
The following describes the advantages and disadvantages of the E measure metric:

**Advantages**
- Is flexible due that \( b \) is adjustable, values of \( b \) greater than one indicates that the user is more interested in Precision than Recall, while value smaller than 1 indicate Recall is more important than Precision.

Several disadvantages of the E measure, that are addressed below, where defined in Extended Performance Graphs for Cluster Retrieval [Dionysius, Huijsmans and Sebe, 2001].

**Disadvantages**
- Due the several adjustable parameters a problem can occur for choosing the correct parameters for the specific environment.
- If it is applicable also for schema matching systems is unknown.
- It is unknown if the E Measure supports semi-automatic matching.
Overall measure
The following describes the advantages and disadvantages of the overall measure metric:

Advantages
- Developed specifically for the schema matching systems and which quantify the post-match effort needed for adding false negatives and removing false positives.
- Used in other schema matching system [Do and Rahm, 2002].

Disadvantages
- No results when precision drops below 0.5 (metric results will be 0).
- Only address automatic schema matching, no semi-automatic schema matching.

The F-measure metric is used for information retrieval purpose only. This is therefore not a good candidate to use for the evaluating of the schema matcher. The E-measure metric has lots of optimization possibilities, which is therefore also not a good candidate.

Based on the experiences described in the paper of Do and Rahm [Do et al., 2002] the metric that will be implemented in the Evaluation system will be the Overall metric. This metric is specially developed for the schema matching purpose. The metric gives an extra weight on the effort of adding missed combinations.

### 3.3 Design

In this section we discuss the designed evaluation system architecture. First we discuss the design of the evaluation system architecture and elaborated on afterwards.

The framework, as depicted in Appendix A, doesn’t reflect the current framework of the schema matcher. For example in the framework the schema matcher supports multiple iterations but the implementation doesn’t contain an iteration option. Each type of information flow is described in the legend table below the framework.
Overview evaluation

The idea of this Evaluation system is that it evaluates and tests changes to the framework in an accurate and efficient way. In figure 3.4 the design is depicted of the evaluation framework containing also the design of schema matching framework based on the current implementation.

The evaluation of the schema match system consists of different test domains. Using different test domains, different behavior can be analyzed. First the test collection of a domain containing all possible valid combinations and the parameters for the test set are loaded into the Configuration module. After the configuration is loaded the two databases are converted into the correct format DataSet and are fed into Map-IT, whereby Map-IT generates possible combinations from schemas. When Map-IT is done generating all possible combinations, the combinations are evaluated by the Evaluation system. The Evaluation system compares the valid combinations, which already have been loaded, with the mappings that Map-IT generated. The schema matcher uses a multi-strategy approach (described in the introduction) therefore it is important to know how well Validators performed. This information is stored in the Similarity cube per Validator per combination a similarity value is generated. Using this type of information the Evaluation system can validate the performance of each Validator. When the schema matcher is done with generation of possible match suggestions these are evaluated with the Evaluation schema combinations. After the match suggestions are evaluated the feedback is sent to the Feedback provider which mappings are correct and which are incorrect. The result, how the...
schema matcher performed, is sent to the *Evaluation result handler* which can present the results to the User or start a new iteration.

For each requirement described in section 3.2 a choice is made how this could be used for the evaluation system:

- **Input**: Realistic data from different domains will be used. The supported type that is fed into the schema matcher will be the *DataSet*.
- **Output**: The combination, that the current schema match system produces, will be used for the evaluation. The output that is generated consists out of combinations from schema1 and schema2 containing a similarity value.
- **Human effort**: The pre-match effort consists of creating configurations for the different test schemas. The post-match effort consists only out validating the values that are produced by the metrics that are chosen for validating the performance of the schema matcher.
- **Match Quality measures**: The *Overall* metric will be used in the first stadium, for evaluating the performance of the schema match system, as described in section 3.2. When there is a need to use also the other described metrics e.g. comparing this schema matcher with other schema matchers, then the other metrics will also be implemented.
- **Learning capability**: The proposed metric will be used in the evaluation to show if the schema matcher learns from previous schema matchers.

Now we describe the *Map-IT* evaluation framework, as depicted in *figure 3.4*, in further detail. Every subsequent section discusses each new framework component.

**Elaboration of the evaluation framework**

In this section each component that is within the rectangle of *Map-IT* evaluation framework, which is depicted in *figure 3.4*, will be addressed in further detail.

**Evaluation configurator**

The *evaluation configurator* holds a list of various *configurations* that will be used for the evaluation. An additional parameter holds the number of runs for evaluating the schema matcher.

**Configuration**

The *configuration* holds various amounts of parameters such as the database information, correct mappings per database and parameters information, which is used for the evaluation. The correct mappings for each database are manually created by a user and stored in the configuration. The parameters information consist of Database1 & Database2, TopNumber (suggested combinations give only the top X), FromPercentage (select only the combinations that above X threshold), selected Validators, enable *Meta-Learner* and location of the LogFile.

**Configuration handler**

The *configuration handler* loads the *evaluation configurator* file. After the file is loaded the list of configurations is iterated. For each configuration the parameters in the *configuration handler* are set e.g. weight of the Validators, Thresholds, etc. After the parameters are set the schema & data are loaded from the databases into the *DataSet*, which are stored in *configuration handler*. The specified correct mappings by the user are loaded from the configuration file and are stored for the *evaluation result*. A function validates the specified correct mappings if they really exist in the databases.

**Logging**

In each logging module logs, depicted in *figure 3.4*, a specific item of the schema match framework is observed. In the *schema combiner* the performance is measure: how fast the creation of schema match combination is created. In the *Validators* region the performance is measurement: how fast a schema match combination is validated per *Validator*. In the *User feedback handler* the performance is measured: how fast the feedback is processed. The
information that has been logged is finally sent to the Evaluation result handler that processes the logging information.

**Evaluation result**
The generated match suggestions, from the schema matcher, are evaluated by the Evaluation result. The results are compared with the specified correct mappings by the user that is loaded in the Configuration handler. For each combination that the schema matcher created are validated with the mappings that are produced manually by the user. When all schema match suggestions are validated the results are sent to the Feedback provider and to the Evaluation result handler.

**Feedback provider**
The correct and incorrect match suggestions that are provided from the Evaluation result are sent to the User feedback handler. Depending on the parameters that are set in the evaluation configuration e.g. only give the correct or incorrect match suggestions. Because of the adjustable parameters the Feedback provider is more flexible and creates a more realistic behavior of the user feedback.

**Evaluation result handler**
After the schema matcher learned from the correct and incorrect match suggestions the data is received from the Evaluation result: how the schema matcher performed according to the time and effectiveness. The information is stored in a propertied data structure for effective transformation and aggregation statistics. When the evaluation data is processed the Evaluation result handler presents it to the user how the schema matcher performed or initiated another iteration of the schema match evaluation.

### 3.4 Implementation

This section discusses the implementation of Map-IT. The following two sections respectively describe:

- The data structure of the evaluation system.
- The implementation of the evaluation system, using the described data structure.

**Programming environment**
The programming language C# will be used within the .NET 1.1 framework. Microsoft Visual Studio is used as an integrated development environment.

#### 3.4.1 Data structure evaluation system

Two data structures are used for the Evaluation system, these data structures are described in the following subsequent sections.

**Configuration**
The Configuration object is based on the design specification that is specified in the section Design of the Evaluation system.

The Configuration holds various amount of information that is needed for the Evaluation system. The object is stored in a XML data structure. The main purpose for storing the object data to a XML file is:

- Easy manipulation of the data.
- Viewable with every text editor.
- Interchangeable between other applications.
The *Configuration* object holds the following properties:

- **Good combinations.**
  This property contains a list of the good combinations, which is specified by the user.

- **Databases.**
  This property contains the two databases with their database connection information.

- **TopNumber.**
  This property contains the number of how many elements per combinations are possible in the suggested combinations e.g. for each attribute in schema1 three combinations are allowed.

- **FromPercentage.**
  This property contains the number of the threshold, which all the suggested combinations need to be above, to be presented to the *User / Evaluation system*, e.g. for every combinations whereby the similarity that is equal or greater is than 0.55 will be presented to the *User / Evaluation system*.

- **Validators.**
  This property contains the Validators that are selected for the evaluation. They are stored in a list, containing the weight per Validator.

- **LogFile.**
  For storing the evaluation result, a location of the file is specified in this property.

- **Meta-Learner.**
  For enabling/disabling the Meta-Learner, this property is set.

**Logging**

The *Logging* object is based on the design specification that is specified in the section *Design* of the *Evaluation system*.

To evaluate the schema matcher all the needed information is stored in this object. *Logging* object contains the following properties:

- **True positive.**
  This property contains the amount of true positives found by schema matcher, which are also correct in the real world.

- **False positive.**
  This property contains the amount of false positives found by schema matcher, which are mappings that the schema matcher considers to be correct but are in the real world incorrect.

- **False Negative.**
  This property contains the amount of false negatives found by schema matcher, which are mappings the schema matcher didn’t find but are correct in the real world.

- **Timer.**
  This property contains a timer per object e.g. Schema Combiner, such an amount is needed to complete the task creating combination. This information is stored per logged object in the timer.

### 3.4.2 Implementation evaluation system

The following sections discuss important implementation issues of the *Evaluation system* framework.

### 3.4.3 Evaluation configurator

The *Evaluation configurator* holds a list of various configurations files that will be used for evaluation of the schema matcher. With this ability the schema matcher can be evaluated with different configurations, containing different databases, with different settings. To test schema matcher with multiple runs, an additional parameter is introduced which stores the number of runs. To store all this information the object data of this class is transferred into a XML file.
3.4.4 Configuration handler

The Configuration handler handles different aspects for the Evaluation system. First the XML Configuration file is loaded, containing the parameters for the Evaluation system that are set in the Configuration handler. After the parameters are set, the databases are being opened and the schema & data information are transferred into the DataSet. The specified correct mappings from the user are then validated using the DataSet, if they really exist in the database. After the function validated the specified mappings the schema matcher is started for creating element combinations between the two databases.

3.4.5 Evaluation result

The schema matcher generated match suggestions that are evaluated in the Evaluation result. The specified correct mappings, which where loaded in the configuration handler, are used to compare the suggested match suggestions from the schema matcher. Each combination that the schema matcher created is validated towards the mappings that are specified by the user. For each good match suggestion the property of the Logging object True positive is increased. For every bad match suggestion the property of the Logging object False positive is increased. For every match suggestion the schema matcher didn’t came up with the property of the Logging object False negative is increased. When all the match suggestions are validated the results are sent to the Feedback provider and to the Evaluation result handler.

3.4.6 Feedback provider

The correct and incorrect match suggestions that are provided from the Evaluation result are sent to the User feedback handler. Before the data is sent to the User feedback handler the Logging object starts a timer so that the overall time, that is needed to process the user feedback, can be measured. The feedback is adjustable with the parameters that are set in the Evaluation configuration object e.g. only give the correct or incorrect match suggestions. This results in a more realistic behavior of the user feedback.

3.4.7 Evaluation result handler

When the data is sent to the Feedback provider, the Logging object contains various amount of information about the time that several operations needed and effectiveness of the schema matcher. Using the properties from the Logging object several metrics are calculated:

- Precision: truePositives / (truePositives + falsePositives).
- Recall: truePositives / (falseNegatives + truePositives).
- F-Measure: (Precision * Recall) / ( (1-0.5) * Precision + 0.5 * Recall).
- Overall measure: Recall * (2- (1/Precision)).
- E-Measure: 1+(0.5)^2/ ( (0.5)^2/Precision + 1/Recall).
- Overall time creation of schema element combinations.
- Average time a Validator needs for validation of a schema element combination.
- Overall time needed to process the User Feedback.

All this information is stored in a XML document and after each run the new evaluation data is added to the existing XML document. When the evaluation data is processed, the Evaluation result handler presents to the user how the schema matcher performed or initiated another iteration of the schema match evaluation. An example of how the overall performance is presented to the user see figure 3.5. Figure 3.5 contains two configurations which run each configuration five times. The min, max average and the standard deviation is shown right above in the corner of the overall performance.

![Figure 3.5: Overall performance two configurations.](image)
4 Duplicated search Validator

In this chapter we describe a duplicated search Validator for schema matching. In the next section a description of the possible improvements is addressed. In subsequent sections we discuss the design of the improvement in detail. In the last section the implementation is addressed.

4.1 Description

In the introduction we described the duplicated search Validator [Bilke and Naurnmann, 2005] called the DUMAS (Duplicate-based Matching of Schemas). They approached the schema matching problem by designing an instance-based matching algorithm. Usual instance-based approaches analyze attributes of each schema individually, extracting properties about the attributes, such as distribution of characters, average string length, etc. Attributes having similar properties are subsequently matched, i.e., they are assumed to have the same meaning. The approach is called vertical matching, because properties of columns of tables are compared. The DUMAS approach uses horizontal matching, tables are being traversed in search for similar rows (or tuples), in effect detecting duplicates. Once a few duplicates have been discovered, deriving schema matches is simple in principle: Same or similar data values among the duplicates imply matching attributes.

The horizontal approach is to solve two main problems:

- Detecting duplicates among databases with a transparent/unknown schema design.
- Deriving a schema matching suggestion from a set of fuzzy duplicates.

Duplicate detection is the problem of identifying multiple representations of the same real-world object within a set of objects. Such multiple representations are called fuzzy or approximate duplicates, because they might not be exact copies of one another. In the sequel, we use the term duplicates.

Duplicates arise during data creation, where they are unintentionally generated, and during data integration if multiple sources store data about same real-world objects. Finding duplicates among two databases with mapping between the schemas is more difficult than the classical duplicate detection problem. There, duplicates are searched within a single table, so it is already clear which data values among a pair of tuples to compare. Typical approaches use domain-specific rules based on such comparisons to determine whether a pair of tuples represents the same real-world object.

Research on the classical duplicate detection problem is further concerned with efficiently finding all duplicates by reducing the number of tuple comparisons while still maintaining good detection levels. Reduction is usually achieved by partitioning tuples based on application specific criteria and searching for duplicates only within the partitions. The semantics of the attributes are not available therefore partitioning is not possible. Instead a duplicate detection algorithm, it produces a similarity measure for which techniques are known that efficiently produce the top K results. When such duplicates have been found it will be used to drive schema matching.

The basic idea of duplicate search looks simple. Equal attribute values in a duplicate imply equivalent attributes in the schemas. Several subtle problems must be overcome due to misspellings, different formats and other fuzziness, duplicates often do not have equal but merely similar attribute values. Therefore a decision needs to be made: how similar attribute values must be and how many duplicates are needed to confidently derive to a final attribute match. In real-world examples different attributes within a tuple may have the same value. For example, shipping address and billing address often have the same value in a record but have different meaning. Thus, even given some duplicates, finding corresponding attributes is not always trivial.
To simplify the problem, using an example with the relational model: Let R and S be two relations R(A, B, C, D, E) and S(B', F, E', G). There is some intensional overlap. Intensional overlap is when attributes from two schemas are equal e.g. attribute name is similar. The attribute names are chosen in such a way to reflect real-world situations. In the example, corresponding attributes have the same letters as names; Real-world names of corresponding attributes can be widely different. R and S have some yet unknown extensional overlap, i.e., some duplicates. Extensional overlap is equal overlapping tuples from two schemas. This simple scenario is shown in figure 4.1. Close examination of figure 4.1 reveals an extensional overlap: Tuple-pairs (r3; s3), (r4; s4), and (r9; s2) are duplicates, which we can use to perform schema matching. In table 4.1 the correct schema matches between the relations R & S are described.

<table>
<thead>
<tr>
<th>Attr. of R &amp; S</th>
<th>Match / no match</th>
<th>Due which tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>B→B'</td>
<td>Is a perfect match with respect to this duplicate, because B and B' have the same value.</td>
<td>r1→s1, r3→s3, r4→s4, r6→s5</td>
</tr>
<tr>
<td>A→F</td>
<td>Is a perfect match, because the values in A and F are equal</td>
<td>r3→s3, r7→s5</td>
</tr>
<tr>
<td>E→E'</td>
<td>Is an almost perfect match, because of the formatting. D→E' is not a match because the value is also different as the formatting.</td>
<td>r4→s4</td>
</tr>
</tbody>
</table>

Problems that can occur with duplicates are: duplicates are usually only fuzzy, so a similarity function to deduce duplicates is needed. Second, low intensional overlap can mislead duplicate detection into believing that no duplicates exist. Third, low extensional overlap may result in an insufficient number of duplicates to deduce attribute matches with enough certainty. Fourth, same or similar values do not always imply matching attributes; for instance, an author in one publication record could be an editor in another, or first and last names are chosen as user accounts. To overcome all these problems they [Bilke and Naurmann, 2005] described four main problems and looked for each particular problem a solution:

1. Unknown schema alignment: It is unclear which field in one tuple to compare with which field in the other.
2. Unknown attribute semantics: Cannot make use of domain knowledge to formulate an effective comparison measure. Common duplicate detection methods use manually or statistically created rules that are based on the similarity of certain corresponding attributes. Without such matches, meaningful rules cannot be created. Instead, a comparison measure that is independent of the fields’ semantics must be applied.
3. Misleading value similarities: Attribute values of non-corresponding attributes could coincidentally be similar, although their respective tuples are not duplicates. Looking at tuples r7 and s5 in figure 4.1: Both tuples have an attribute value ‘kate’, but are not duplicates. Without knowledge of the correct attribute matches, such a value match can mislead duplicate detection.
4. Partial schema overlap: Not all fields in one tuple necessarily have a matching partner in the other. With only few corresponding attributes, the similarity of two tuples is typically low. Looking at the example from figure 4.1 only two attributes among the relations actually match (B→B’ and E→E’).

The solution for the problems that are above addressed, are solved in section 4.2.
Evaluation

The DUMAS project started with a simple idea of using duplicates to detect corresponding attributes in tables. They identified the major and the subtle obstacles in achieving high-precision and high-recall schema matching. One major obstacle finding duplicates among tables with unmatched and only partially overlapping schemas, has been overcome using a similarity function which successfully identifies duplicates.

The following describes the advantages and disadvantages of the DUMAS project.

Advantages
- The effectiveness, robustness and efficiency of their approach were shown in several experiments on artificial and real-world data.
- It doesn’t require a training period.
- Well documented proposal for solving schema match problem using duplicates.
- The Validator can produce schema match suggestions when schemas are unaligned, in contrast to Validators that need domain knowledge.

Disadvantages
- Implementing the duplicated search is not well documented.
- Test scenarios that are described are not available.
- When no duplicates are available the Validator is useless.
- Requires framework adjust because the current Validators do not support generation of combinations.

Projects towards duplicated search, like Eliminating fuzzy duplicates in data warehouses [Hernandez and Stolfo, 1998] and Real-world data is dirty: Data cleansing and the merge/purge problem [Ananthakrishna, Chaudhuri and Ganti, 2002], only assumes aligned schemas, which is not only the case in Map-IT. Therefore the duplicated search Validator needs to operate with schemas that are aligned and unaligned.

4.2 Design

String comparison methods

To solve problem 1 described in section 4.1, the cosine measure is used to solve the problem. The similarity of two strings is determined with the cosine measure, which is the product of two vectors normalized to unit length, and thus, equal to the cosine of the angle between the two vectors. An advantage of the vector space model over an edit-distance model is its independence of term ordering. Therefore, it is used as the model for comparing tuples in the duplicate detection step.

In the next sections several sections are cited from Bilke and Naurmann paper [Bilke and Naurmann, 2005].

Similarity of unaligned tuples

The cosine measure is used to tokenize the tuples and compare the resulting vector representations. The assignment of weights for the tokens in each tuple is crucial for the effectiveness of the cosine measure: A weight should represent the relative importance of a token within the tuple. The well-known TFIDF weighting scheme calculates the weight as a function of the term frequency (TF), i.e., the number of times the term occurs in the string, and the inverse document frequency (IDF), which is the overall number of strings (tuples) divided by the number of strings in which the given term occurs. We define the weight \( w'(r, t) \) of a term \( t \) in a string \( r \) as

\[
w'(r, t) = \log(tf_{r,t} + 1) \times \log\left(\frac{N}{df_t} + 1\right)
\]
The \( tf_r(t) \) is the term frequency of \( t \) in \( r \), \( N \) is the overall number of tuples, and \( df(t) \) is the number of tuples in which \( t \) appears. These weights are normalized such that their respective vector has unit length. The normalized weight is calculated as:

\[
w(r, t) = \frac{w'(r, t)}{\sqrt{\sum_i w'(r, t)^2}}
\]

Using a TF-IDF based measure has several advantages. First, it is order independent, which is important with respect to Problem 1. Second, by using the inverse document frequency, terms that occur in only few tuples receive a higher weight. Intuitively, the reason is that infrequent terms have a higher identifying power, and thus, should have a higher influence on the similarity score. This behavior can be of help in solving Problem 3.

**Efficiently finding similar tuples**

Duplicate detection is a problem with inherently quadratic complexity: Each tuple in the first table has to be compared with each tuple in the second table. Such exhaustive search is clearly infeasible for larger data sets. Reducing the number of tuple comparisons is very important to make duplicate detection scalable. A semi-naïve algorithm would pick each tuple from the source table and look for tuples in the target database that contain at least one of its tokens. This lookup can be efficiently performed using an inverted index on the target table, which maps terms to tuple identifiers. Compared to exhaustive search, the semi-naïve algorithm achieves a major reduction in the number of tuple comparisons in most scenarios. However, it ignores the TFIDF weighting of tokens, and thus, misses the chance for a larger performance gain. The semi-naïve algorithm computes the similarity even of tuples that share only low-weight terms although those terms have only a small effect on the similarity score. Hence, the semi-naïve algorithm is considered suboptimal because tuples which have only low-weight terms in common are less likely to be contained in the set of top-k tuple pairs.

A more intelligent algorithm would search for the top-k duplicates by finding tuple pairs that have high-weight tokens in common and stop searching when no tuple pairs with a high similarity score can be expected. This idea is realized in an adaptation of the Whirl algorithm for similarity joins in relational databases [William W. Cohen, 1998]. Whirl performs \( A^* \) search in the space of possible tuple pairs. \( A^* \) is a widely known best-first search algorithm that finds a path from a given start state to a goal state with the smallest cost. In each iteration, the algorithm picks the state \( n \) from a list of open states with the smallest assigned cost. If \( n \) is a goal state, then it is presented as the result. A goal state is when \( k \) duplicates are found. If \( n \) is an intermediate state, the graph is traversed further and new states are added to the list of open states. The cost \( f(n) \) of a state \( n \) is calculated as \( f(n) = g(n) + h(n) \), where \( g(n) \) is the actual cost of the path from the source to state \( n \) and \( h(n) \) is the estimated cost of the path from \( n \) to the closest goal state. \( A^* \) is optimal if \( h(n) \) is an admissible heuristic, i.e., it never overestimates the cost to reach a goal. In most cases \( h(n) \) is defined to be zero if \( n \) is a goal state. In the duplicate detection implementation each state is a four-tuple \( <r, s, b, e> \), where \( r \) represents a source tuple, \( s \) represents a target tuple, \( b \) is the current bound and \( e \) is the exclusion list. Both \( r \) and \( s \) can be either unbound (denoted as \( \perp \)) or bound to a tuple. Based on the values of \( r \) and \( s \), three state types are distinguished: (i) A state is a start state when both \( r \) and \( s \) are unbound, (ii) a state is an intermediate state when only \( r \) is bound and (iii) a state is a goal state if both \( r \) and \( s \) are bound. The exclusion list \( e \) is a list of tokens which may not be contained in target tuples - the intention of this list is made clear in the description of the algorithm below. The bound \( b \) is the maximum similarity of two tuples that can be reached from the given state. Note that the goal is to maximize similarity as opposed to minimize cost. Thus, \( b \) must be an overestimate instead of an underestimate. The bound function \( B(r, s) \) is defined as:

\[
B(r, s) = \begin{cases} 
\infty & \text{if } r = \perp \land s = \perp \\
B(r) & \text{if } r \neq \perp \land s = \perp \\
tupsim(r, s) & \text{if } r \neq \perp \land s \neq \perp 
\end{cases}
\]
The function $B(r)$ determines the bound for intermediate states. It is computed as:

$$B(r) = \sum_{t \notin e} w(r, t) \cdot \max \text{weight}(t)$$

The tuple similarity of two tuples $r$ and $s$ is then calculated as:

$$\text{tupsim}(r, s) = \sum_{t \in r \cap s} w(r, t) \cdot w(s, t)$$

where $t$ is a term that does not appear in the exclusion list $e$ and $\maxweight(t)$ is the maximum weight of term $t$ in the target relation. The maximum weight of a term is stored as additional information in the inverted index, and thus, can be efficiently retrieved. The $A^*$ search graph is not entirely kept in memory, only the list of open states, which is called OPEN. For each iteration the state with the smallest cost (largest bound) is calculated using formula $B(r)$ or $\text{tupsim}(r, s)$ depending $r$ and $s$. Using a priority queue, which handles insertion and remove of a single state with the largest bound in $O(\log n)$ time.

**Algorithm 1: $A^*$ search by Peter Hart, Nils Nilsson, and Bertram Raphael, 1968**

```plaintext
function $A^*(\text{start}, \text{goal})$
1 result := {};
2 OPEN := make_queue(path(start))
3 while OPEN ≠ Ø do
4 p := remove_first(OPEN);
5 x := the last node of p;
6 OPEN := OPEN – {s'};
7 if x = goal then
8 return p;
9 else
10 result := result U {x};
11 foreach y in neighbors(p)
12 enqueue(OPEN, y)
13 end
14 end
15 return failure
```

**Algorithm 2: $A^*$ search for top-k duplicates (adapted from [William W. Cohen, 1998])**

Output: Set of states representing the k most similar tuple pairs

```plaintext
1 result := {};
2 OPEN := {S0};
3 while OPEN ≠ Ø and |result| < k do
4 s := argmax$_{s \in OPEN} B(s)$;
5 OPEN := OPEN – {s'};
6 if goalState(s) then
7 result := result U {s};
8 else
9 OPEN := OPEN U children(s);
10 end
11 end
12 return result
```

**Explanation Algorithm 1**

A-star search algorithm is a graph search algorithm that finds a path from a given initial node to a given goal node or one passing a given goal test. It is a search algorithm that makes use of a heuristic to evaluate the un-searched portions of the graph. It visits the nodes in order of this heuristic estimate. The $A^*$ algorithm is therefore an example of best-first search. $A^*$ incrementally builds all routes leading from the starting point until it finds one that reaches the goal. It also takes the distance already traveled into account. This makes $A^*$ complete and optimal. However, it is not guaranteed to perform better than simpler search algorithms. $A^*$ maintains paths through the
graph starting at the start node, stored in a priority queue. A Star Search Algorithm finds that the priority assigned to a path \( x \) is determined by the function \( f(x) = g(x) + h(x) \). Here, \( g(x) \) is the cost of the path so far. \( h(x) \) is the heuristic estimate of the minimal cost to reach the goal from \( x \). When \( A^* \) ends its search, it by definition has found a path whose actual cost is less than the estimated cost of any path through any open node. But since those estimates are optimistic, \( A^* \) can safely ignore those nodes. \( A^* \) is considered to be computationally optimal in terms of the number of nodes it considers. The \( A^* \) algorithm is often used on road maps and maps in computer games. These maps are often planar and the heuristic used by \( A^* \) does not take benefit out of this. These maps are also spatially coherent and \( A^* \) does not take advantage of this as well.

**Explanation Algorithm 2**

The duplicate detection algorithm is depicted in Algorithm 2. Variables are initialized at the beginning: The variable result set is set to an empty set (line 1), while the list of open states OPEN contains the start state \( s_0 \) (line 2). Combinations of the type \( <r, \perp, b, \varnothing> \) are added to the OPEN list for each source tuple \( r_1...r_m \), a new state is created where \( r \) is bound to the source tuple, \( s \) is unbound, \( e \) is empty, and \( b \) is computed using the function \( B(r) \) defined above. The following loop is executed until (i) the list of open states is empty or (ii) \( k \) goal states have been found. At the beginning of the loop, the current state \( s \) becomes the state with the largest bound (line 4), which is also removed from the OPEN list (line 5). If the extracted state is a goal state, then it is added to the result set result (line 7). Otherwise, child states are created for state \( s \) and added to the list of open states (line 9) see further one Children. The creation of child states is described below. The result set is returned after the loop has terminated (line 12).

**Children:** An intermediate state that has been extracted from the OPEN list is constrained by “creating” its child states and adding them to the open list. The child states have the same source tuple \( r \), but either a bound target tuple \( s \) or an extended exclusion list \( e \). They are created as follows: A term \( t \) that appears in the source tuple \( r \), but not in the exclusion list of the state, is picked, and target tuples containing \( t \) are extracted using the inverted index on the target relation. From the extracted tuples only the \( l \) tuples which do not contain any term of the exclusion list are used to create \( l + 1 \) new states: \( l \) goal states in which the target tuple is bound, and an intermediate state in which the target tuple remains unbound, but the term \( t \) is added to the exclusion list. Because a new term has been added to the exclusion list, the bound of the new intermediate state is lower than the bound of its parent. In order to reduce the number of tuple comparisons, a term \( t \) should be chosen such that the bound of intermediate states quickly decreases. Thus, we pick a term \( t \) that maximizes \( w(r; t) \ast \text{maxweight}(t) \), because its insertion into the exclusion list has the largest effect on the bound of the derived intermediate state (see equation \( B(r) \) defined above). Beside computation of the bound of intermediate states, the intention of the exclusion list is also to avoid creating the same tuple combination twice: If a term \( t_i \) appears in an intermediate state for source tuple \( r \), it implies that for all target tuples \( s \) containing \( t_i \), there already is or has been a goal state in the OPEN list with \( r \) as source tuple and \( s \) as target tuple. If such an intermediate state is constrained using term \( t_j \), then no goal states for target tuples containing \( t_i \) are created.

**Difference between Algorithm 1 & Algorithm 2**

The difference between the Algorithm 1 towards Algorithm 2 is: at line 4 in Algorithm 1 retrieves the first element that added to the OPEN list algorithm 2 retrieves the element that has the largest bound. If the element in algorithm 1 is retrieved from the OPEN list and its no GOAL state the neighbors of the element are iterated and added to the OPEN list for further exploration. In algorithm 2 the all possible combinations are created using the children functions and added to the OPEN list for further exploration. When the element that is retrieved from the OPEN list (line 4) is a goal state element Algorithm 1 stops in contrast to algorithm 2 the element is added to the result list and when \( k \) duplicates are found the algorithm stops.
To explain the algorithm in more detail using this example:

<table>
<thead>
<tr>
<th>Table R</th>
<th>Table S</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>s1</td>
</tr>
<tr>
<td>r2</td>
<td>s2</td>
</tr>
<tr>
<td>r3</td>
<td>s3</td>
</tr>
<tr>
<td>r4</td>
<td>s4</td>
</tr>
<tr>
<td>r5</td>
<td>s5</td>
</tr>
</tbody>
</table>

Start state: The OPEN list contains all the tuples from table R that are bound with a computed `b` using the function `B(r)` described above.

OPEN = \{<r1, \bot, b1, ø>, <r2, \bot, b2, ø>, <r3, \bot, b3, ø>, <r4, \bot, b4, ø>, <r5, \bot, b5, ø>\}

After the OPEN list is created the algorithm starts and selects from the OPEN list the picking the highest bound, in this case tuple r3 has the highest bound, and remove the state from the OPEN list. After the state is removed from the OPEN list the terms are analyzed using the formula \(w(r,t)\) described above. From tuple r3 term a2 is picked and looked up in the inverted index of table S and found a matching term in tuple s1 and s4. A new state is created for tuples s1, s4 and for tuple r3, using the children functions, and added to the OPEN list: <r3, s1, b6, ø>, <r3, s4, b7, ø> and <r3, \bot, b8, {a2}>. When a state confirms to a goal state it is added to the results set. After enough duplicates are detected the algorithm stops.

Given a duplicate pair \(r = (a_1 \ldots a_m)\) and \(s = (b_1 \ldots b_n)\), this step produces a \(m \times n\) matrix that stores the similarity fieldsim of each pair of field values \(a_i\) and \(b_j\) in the tuples. For comparing tuple fields we use the SoftTFIDF measure [W. W. Cohen, P. Ravikumar, and S. E. Fienberg, 2003], a variation of the previously described TFIDF measure that also considers similar terms (as opposed to equal terms). The SoftTFIDF is defined as:

\[
\text{SoftTFIDF}(r,s) = \sum_{t \in \text{CLOSE}(\theta, r,s)} w(r,t) \cdot w(s,t) \cdot D(s, t)
\]

Where by the \(\theta=0.9\) defined. We generate such a matrix \(M_k\) for each of the \(K\) duplicates and combine all matrices to produce the overall average similarity matrix \(M\) as input to the next step:

\[
M = \frac{1}{K} \sum_{k=1}^{K} M_k
\]

In summary, \(M\) stores average similarity scores that are accumulated over the field-similarities of the \(K\) most similar duplicates. The field-similarity matrix in table 4.2 is based on the figure 4.1 tuples r3 and s3. Problem 2 and problem 3 are solved using a \(K=3\) with a max 5 duplicates, because the computed SoftTFIDF is normalize over multiple combinations.

| Table 4.2: Field-similarity matrix for a duplicate pair |
|-----------|-----------|------------------|------------------|
| Suzy      | Klein     | f                | (358)2436231     | (358)2436321     |
| Klein     | 0         | 1                | 0                | 0                |
| Suzy      | 1.0       | 0                | 0                | 0                |
| 358-2436321 | 0         | 0                | 0.67             | 0.67             |
| UNIX      | 0         | 0.2              | 0                | 0                |

Using the formula of \(M\), which is described above, computes the similarity matrix depicted in table 4.3. The computation is partially based on table 4.2 this table represents only one tuple mapped on a tuple from a different table. Table 4.3 shows matrix \(M\) for the duplicate pairs (r3, s3), (r4, s4), and (r9, s2). The values that are above a certain threshold e.g. 0.60 will be produced as a match all the values below the threshold will be set to zero.
Table 4.3: Accumulated similarity matrix $M$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>B'</td>
<td>0.22</td>
<td>0.92</td>
<td>0.07</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0.60</td>
<td>0.60</td>
<td>0.07</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E'</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.58</td>
<td>0.64</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0.07</td>
<td>0</td>
<td>0.07</td>
<td>0.02</td>
</tr>
</tbody>
</table>

When using this schema matching algorithm one has to determine a meaningful number for $K$. To avoid false positives the proposed value for $K$ is a small one. The algorithm will automatically determine if the number was sufficient by computing the certainty of the resulting correspondences. More duplicates will be detected by iterating back to the duplicate detection step.

4.3 Implementation

The duplicated search Validator is partly based on the design specification that is specified in the section 4.2 Design.

During implementation it became clear that only this Validator can be used with direct combinations, because it searches for duplicates in both database schema elements and not in GIS – subconcepts.

To be able to find fast duplicates an appropriate data structure is needed. A good data structure to store various amount of data is a DataTable. A DataTable object is the representation of a table, with rows and columns. A DataTable allows to fast query rows based on a filter criteria.

Therefore the choice has been made for this data structure to store the inverted index. The inverted index consists of three columns per database: “Keyword”, “Attribute”, “Rowindex”. The inverted index implementation is created in another project “Finding non-trivial semantic mappings between database Schemas” by J.T. Visser.

In the next part each calculation that has been done on the DataTable, term freq., inverse document freq. and bound calculation are addressed.

**Term frequency**
Each term is counted, per row, how often the term appears. The result of the term frequency function is stored in an extra column in the inverted index.

**Inverse document frequency**
Each term, per row, is counted and divided by the number of rows. The result of the inverse document frequency function is stored in an extra column in the inverted index.

**Bound calculation**
The appropriate calculations are done (term freq and inv. doc. freq.) to calculate the bound function. The result of the bound function is stored in an extra column in the inverted index.

To use the best keywords from Database1 to find duplicates in Database2 the inverted index of Database1 is sorted on the column of the bound calculation. After the sorting is done the algorithm searches for top K duplicates in Database2. Each duplicated candidate that is found, in Database2, is stored in a new DataTable structure containing: keyword, rowIndexDB1, columnNameDB1, rowIndexDB2, columnNameDB2. After a new row is added to the new DataTable the keyword that was used, is added to the exclusion list so this keyword cannot and will not be used anymore. This prevents that the same keyword will be used over and over again.

The top K duplicates are now found and a selection is done, of which are good duplicates and false duplicates. To remove the false duplicates from the new DataTable the duplicate keywords are compared with each other using the vector space model called in DUMAS paper SoftTDIDF. The vector is called between the found duplicates. When vector angle is too wide the duplicates...
don’t belong to each other. The advantage, of using the vector space model, is that the order of the terms is irrelevant. A disadvantage is that the algorithm is expensive when the string size is large.

Table 4.4: Example of a container, containing duplicates for attribute ‘address’.

<table>
<thead>
<tr>
<th>Duplicate instances</th>
<th>Number of duplicates found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sand Ambachtstraat 40</td>
<td>2</td>
</tr>
<tr>
<td>Hoofdstraat 262</td>
<td>1</td>
</tr>
<tr>
<td>Sonderholm 164</td>
<td>1</td>
</tr>
<tr>
<td>Fazantenstraat 71p</td>
<td>2</td>
</tr>
<tr>
<td>Albert Soncklaan 2</td>
<td>1</td>
</tr>
<tr>
<td>Helderseweg 52</td>
<td>3</td>
</tr>
</tbody>
</table>

After the false duplicates have been deleted from the DataTable the final similarity score is calculated for the remaining duplicates. The finale similarity calculation is done by the following criteria:

1. Per attribute all the found duplicates are stored in a container (see table 4.4).
2. Each duplicate is iterated and the similarity score is calculated using the following metric:

   \[
   \text{similarity} = \frac{\text{initial value} + \sum_{d \in \text{DataTable}} \text{correction}(d) \times \text{numberDuplicates}(d)}{1 + \sum_{d \in \text{DataTable}} \text{numberDuplicates}(d)}
   \]

   \[
   \text{correction}(d) = \begin{cases} 
   1.0 & \text{if numberDuplicates}(d) = 1 \\ 
   0.80 & \text{if numberDuplicates}(d) > 1 
   \end{cases}
   \]

   a. Initial value of similarity is 0.4, this is because finding one duplicated result in a low similarity. The more duplicates there are found the similarity increases.
   
   b. Number of duplicates greater than one (first row duplicates of table 4.4): When the duplicated instance is greater than one then the similarity is multiplied with the current amount of already calculated similarities (in this case 1 initialize value of 0.4) plus 0.8 times number of duplicates found (in this case 2). The similarity score, after one duplicate, will be: \(0.4 \times 1 + 0.8 \times 2 = 2\).
   
   To calculate the intermediate similarity can be done using the following metric:
   \[
   \frac{\text{sumCurrentSim}}{\text{numberOfCurrentDuplicates}} = \frac{2}{1+2} = 0.66.
   \]

   c. Number of duplicates is one (second row duplicates of table 4.4): When the duplicated instances are one, then the similarity is increased by one plus the current sum similarity two. The similarity score, after two duplicates, will be: \(2 + 1 \times 1 = 3\). The intermediate similarity is: \(3 / (3+1) = 0.75\).

   d. This process continues until all duplicates are iterated from table 4.4, which results then in a final similarity value.

Using this technique, for calculating the similarity, only relevant duplicates (more then one) will result in a high similarity score. After the similarities have been calculated for each duplicate the similarity is assigned to the internal representation of the combinations namely \texttt{elementCombinations}. 

Map-IT: An advanced multi-strategy and learning approach to schema matching
5 Meta-Learner

In this chapter we describe a *Meta-Learner* for schema matching. In the next section a description of the possible improvements are addressed. In subsequent sections we discuss the design of the improvement in detail. In the last section the implementation is addressed.

5.1 Description

For matching schemas, LSD [Doan et al., 2003] uses multiple machine learning techniques, called *base learners*. A other machine learning method called *stacking* [Wolpert, 1992][Ting and Witten, 1999] is to semi-automatically find matches between the source database schema and the mediated schema. The results of the individual learners are combined via a weighted average by the *Meta-Learner*. The weight consists of the relative importance of the base learner for a specific combination of source schema element and mediated schema element.

The *Meta-Learner* requires a training period for the matchers in order to predict match candidates and for the Meta-Learner to make accurate weighted averages. The result of the *Meta-Learner* is fed to the prediction converter and subsequently to the constraint handler. The final result consists of matches with their respective similarity values.

The *Meta-Learner* is a *Prediction aggregator* alike component. The weight for every mediated schema element (GIS, in Map-IT) – Learner combination is assigned by the *Meta-Learner*, and is a reflection of how much the *Meta-Learner* trusts the prediction of a specific *Learner* for a combination with a mediated schema element. For this trust value the *Meta-Learner* uses the *least-squares linear regression* algorithm. This is based on how many times the *Learner* falsely predicted a low or a high value in previous accepted schema combinations, for combinations with the mediated schema element. Using and training the *Meta-Learner* has a significant improvement towards the schema match system of LSD. They report on experimental results of applying LSD to five sources in the real-estate domain. In figure 5.1 the experiment runs are shown from the schema matcher LSD, containing runs without the *Meta-Learner* and with the *Meta-Learner*. In the figure it is clearly seen that the average matching accuracy improved substantially by 5 - 22%.

In Map-IT there are at the moment several parameters to tune. Using the *Meta-Learner* to automatically tune the parameters e.g. threshold (Transitive Combiner), after each iteration, can significantly improve the performance of the schema matcher.

Evaluation

In Map-IT there is already a component like the *Meta-Learner* namely the *Prediction aggregator*. Only in Map-IT the parameters that are adjustable are not automatically adjusted to its environment. Map-IT doesn’t predict the trust values, because it doesn’t use the *least-squares linear regression* algorithm. Instead it uses the *weighted strategy approach* described in section 6.1 Combination strategies, when the *Meta-Learner* is used the Weights are automatically adjusted.

The following describes the advantages and disadvantages of the *Meta-Learner*.

Advantages
- Automatic parameters adjustment in working domain, after each iteration.
- Promising experimental results using the *Meta-Learner*.
5.2 Design

The Prediction aggregator is the place for the implementation of the Meta-Learner. When the user gives the feedback on the suggested matches, the Meta-Learner evaluates how the schema matcher (Validators, parameters, etc) performed. This reflection is done by evaluating the Similarity cube, which holds the score per Validator of the suggested matches. A scenario can occur when a Validator validates a mapping which it gives a similarity of 0.85, but the user reviewed the mapping, which finds it incorrect. To overcome this problem, to not occur again, the weight of the Validator is adjusted. A different situation can occur when the suggested matches that are produced, using the GIS, are most of the time accurate and correct. The weight of combination, that is created using the GIS, could therefore be adjusted.

The parameters are not adjustable at the moment. To realize this, a default configuration needs to be used, when there is no custom configuration available. A custom configuration consists of the parameters that are adjustable such as the weight of the Validators, thresholds, etc. When there is no custom configuration available the default parameters are used.

In figure 5.2 a part of the Map-IT framework is depicted including the Meta-learner. The dotted line represents the old information flow, which has not been changed. The continuing lines represent the introduced information flow. The Meta-learner is located in the Predication Aggregator. The scenario that above described, when parameters need to be adjusted, is handled by the Predication Aggregator.

5.3 Implementation

In the class MainMapper the good, bad and missed combinations are received from the user or the Evaluation system. For each category a separate function is created in the prediction aggregator. This is to evaluate how many good & bad prediction each Validator has calculated.

The function that handles the good combinations predictions receives the list of combinations. For each combination, whereby the property of user feedback is set true, the similarity score of each Validator is stored in a proprietyed data structure, namely an ArrayList. An ArrayList is a data structure which can grow and shrink instead of a normal Array. After the function stored all the similarity scores per Validator in the ArrayList it is returned to the MainMapper class. The same procedure is done when the property of user feedback is set false in a combination. Each Validator similarity is stored again in the ArrayList and returned to the MainMapper class.

The similarities for each good & bad combination are stored in the ArrayList. For the good & bad combination the similarity per Validator is iterated calculating the average given similarity per Validator. When the average similarity score is calculated per Validator a comparison is done of
how many similarity scores are above the average and then are counted. This average calculation is done because the similarity scores given for each Validator can differ a lot. Using this approach the similarity score is normalized.

After the counting is done, how many good & bad combinations are above the average similarity per Validator, the Validators are iterated and per Validator the good counter subtract from bad counter result in a positive or a negative score. If the score is positive the Validator weight is increased. When the score holds a negative score the weight of the Validator is decreased. In Table 5.1 is shown the criteria for adjusting the weight of the Validators. The total variable holds the score x good counter subtract from bad counter.

Table 5.1: criteria for adjustment weight Validators.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Weight adjustment for the Validator</th>
</tr>
</thead>
<tbody>
<tr>
<td>total &gt;= 3 &amp;&amp; total &lt;= 5</td>
<td>Current weight * 1.025</td>
</tr>
<tr>
<td>total &gt; 5 &amp;&amp; total &lt; 10</td>
<td>Current weight * 1.05</td>
</tr>
<tr>
<td>total &gt;= 10</td>
<td>Current weight * 1.1</td>
</tr>
<tr>
<td>total &lt;= -3 &amp;&amp; total &gt;= -5</td>
<td>Current weight * 0.975</td>
</tr>
<tr>
<td>total &lt; -5 &amp;&amp; total &gt; -10</td>
<td>Current weight * 0.95</td>
</tr>
<tr>
<td>total &lt;= -10</td>
<td>Current weight * 0.9</td>
</tr>
</tbody>
</table>

The multiplication is done in percentages, because the initial weights of the Validators are different. To prevent that the Validators scores are above or below a certain score e.g. 0 or 5 a weight can be set for the minWeight and a maxWeight a Validator can reach. These parameters are default set.

To prevent that the Validators use the default weight at a new iteration the weights are stored in a XML document. When a new iteration is started the weights that have been stored, at the previous iteration, will be used. A property in MainMapper holds when the Meta-Learner is enable/disable.
6 Other schema match improvements

In this chapter we describe the remaining improvements for schema matching. In the next section an overview of the possible improvements are addressed. In subsequent sections we discuss the improvements in detail.

In this section several improvement are discussed that can improve the schema match results. The improvements are summed up below and elaborated on afterwards:

- Combination strategies: Due to the flexibility of Map-IT which can improve schema matching the items aggregation, direction and selection are further addressed.
- Data type similarity: Several schema matchers use data type similarity to reduce the search space.
- Missing Match problem: To improve the learning capability of the schema matcher the Missing Match problem needs to be investigated.

6.1 Combination strategies

In this section we address the Combination strategies.

6.1.1 Description

Due to the flexibility of Map-IT in configuring match strategies and matchers, an exhaustive evaluation to investigate the effectiveness of all possible configurations is virtually impossible. It is difficult to investigate all match parameters at the same time. To investigate different aspects of the matching process in Map-IT, the following item is evaluated:

- Combination strategies: This evaluation aimed at identifying the best strategies for similarity combination, i.e., aggregation, direction and selection.

Evaluating different aspects of the matching process it is helpful for improving individual components of the framework. The goal of improving individual components is to improve the schema match suggestions.

Aggregation

The Weighted strategy determines a weighted sum of similarity values of the individual Validators. The Weighted strategy needs relative weights which should correspond to the expected importance of the matchers. With $\text{sim}(s_1, s_2, v)$, we denote the similarity between $s_1$ and $s_2$ computed by the Validator $v$ in $V$.

The following formula is currently used in Map-IT:

$$\text{WeightedSim}(s_1, s_2) = \sum_{v \in V} w_v \times \text{sim}(s_1, s_2, v) \text{ with } \sum_{v \in V} W_v = 1$$

FlexiMatch made the choice for the Weighted strategy based on quote: “intuitively expect that this formula will perform better than the Average or Max strategy discussed below”.

The Max strategy returns the maximal similarity value of any matcher. It is optimistic, particular in case of contradicting similarity values. Furthermore, matchers can maximally complement each other. The formula for the Max strategy is:

$$\text{MaxSim}(s_1, s_2) = \max_{v \in V}(s_1, s_2, v)$$

The Average strategy represents a special case of Weighted and returns the average similarity over all matchers, i.e., considers them equally important. The formula for the Average strategy is:

$$\text{Average}(s_1, s_2) = \frac{1}{|V|} \sum_{v \in V} \text{sim}(s_1, s_2, v)$$
Design

The current schema matcher supports only the Weighted similarity metric. To evaluate also the other metrics e.g. Max similarity and the Average similarity new parameters need to be introduced. The default selected metric will be the Weighted similarity metric and its optional to select the other metrics. An extra parameter needs to be introduced in the Prediction Aggregator, for selection of a different metric. Two extra methods need to be introduced, in the Prediction aggregator, for the implementation of the Max similarity and the Average similarity.

Direction

COMA supports determination of directional and un-directional match results. To select match candidates for one element from one database schema, all elements from the other database schema are ranked in descending order of their similarity value. The following directions can be performed:

- LargeSmall: In this directional approach the larger database schema S1 is mapped against the smaller target S2, i.e., elements from S1 are ranked with respect to each S2 element.
- SmallLarge: As opposed to LargeSmall, match candidate selection is performed based on ranking S2 elements for each S1 element.

Design

To test if the Direction has an impact on the schema match results, the schemas need to be analyzed which schema is the largest or smallest. After the schemas have been analyzed which is smaller, depending on the strategy that has been chosen e.g. SmallLarge will map the smallest on the larger schema. In Schema combiner the direction selection takes place. A new parameter needs to be introduced in the Schema combiner which direction is selected (LargeSmall or SmallLarge).

Selection

Given a ranking, for example, of all S1 elements for a particular S2 element, one of the following strategies can be used for selecting the match candidates:

- topN(number): The n S1 elements with maximal similarity are selected as match candidates. n=1, i.e., top1, represents the natural choice for 1:1 correspondences. Generally, n>1 is useful in interactive mode to allow the user to select among several match candidates.
- Threshold (called in the schema matcher “FromPercentage”): All S1 elements showing a similarity exceeding a given threshold value t are selected.
- MaxDelta: The S1 element with the maximal similarity Max is determined as match candidate plus all S1 elements with a similarity differing at most by a tolerance value d, which can be specified either as an absolute or relative value. In particular, the tolerance range is defined as [Max-d, Max] and [Max-Max*d, Max] for the absolute and relative case, respectively. The idea is to return multiple match candidates when there are several S1 elements with the same or almost the same similarity value.
- Threshold (Transitive Combiner): There are three values that play a role in the derivation of the final similarity value for the transitive combination. These values are:
  - simval1, the similarity value of combination schema 1 element – subconcept 1.
  - simval2, the similarity value of combination schema 2 element – subconcept 2.
  - linkCost, the cost of the link between subconcept 1 and subconcept 2.

The threshold of the transitive combiner restricts the result of:

\[
\text{transSimval} = \left( \frac{(\text{simval1} + \text{simval2})}{2} \right) \times \text{linkCost}
\]

Design

To make a selection of the suggested combination, different strategies are possible. The topN and Threshold (FromPercentage) are already available in the schema matcher. The variable topN runs from 1 to nxm and Threshold (FromPercentage) runs from 0.00 to 1.00.
In case of the MaxDelta whereby suggested match results, with a same or almost the same similarity value, are suggested. In the current schema matcher the match suggestions are reduced in Mapping generator (depending on the threshold or the topN). The adjustment, to make the MaxDelta possible, only needs an implementation adjustment and not an adjustment to the framework. The appropriated place to implement the MaxDelta is in the Mapping generator next to the other functions. The MaxDelta can be in adjusted from 0.01 to 0.05 whereby more or less match suggestions are presented to the user. The Threshold (Transitive Combiner) parameter is already available in the current schema matcher but not adjustable. This parameter is default to a specific value set (default=0.7), which can be altered in configuration before the schema matching takes place. In altering the Threshold downwards will result in more match suggestions that are created with the GIS.

6.1.2 Implementation
Though it was the intention to include the combination strategies option in Map-IT, it is not yet included in the current version. It was decided that this feature has a low priority. See section 9.2.2 Combination strategies of the recommendations how this option could be implemented.

6.2 Data type similarity
In this section we address the Data Type similarity feature.

6.2.1 Description
The data type validation is disregarded in the current implementation, when schema element combinations are created by Schema Combiner. A combination that never will be approved is e.g. datetime data type combined with a float data type. The instance data values cannot be converted from a datetime element to a float element and vice versa. Cupid [Madhavan et al., 2001] and COMA [Do and Rahm, 2002] also addressed this problem and created a solution using a table containing all possible compatible data types. For each data type a list of possible compatible data types is described. When only compatible data types are allowed the solution space will be reduced.

The following describes the advantages and disadvantages of the Data type similarity.

**Advantages**
- Only compatible mappings are created.
- Simple to implement.
- Solution space is reduced.

**Disadvantages**
- When new data types are added to a database system, the table of compatible data types needs to be updated.

6.2.2 Design
In the Schema combiner the schema match combinations are created. Therefore the place where the data type similarity needs to be dealt with is in the Schema combiner. A compatible list needs to be created in the Schema combiner, whereby the compatible data types are defined e.g. numeric data type is compatible with a float data type. The data types are named differently for every database vendor therefore a generic data type definition will be used from the .Net framework OLE DB (Object Linking and Embedding for Databases) data types. When the schema, of the database, is loaded the data types from the database vendor are transformed automatically into the OLE DB data types. Therefore the problem is solved that every database vendor uses a different appellation for the data types.
After the compatible data type list is created, a function needs to be introduced in the Schema combiner, whereby two schema elements are validated. The function returns true when schema elements are compatible, else a false.

### 6.2.3 Implementation

The data type similarity is based on the design specification that is specified in the design section of the data type similarity. In the Schema combiner an extra function is introduced to handle the different data types. First a compatible list is specified which data types are compatible with each other, see e.g. table 6.1

<table>
<thead>
<tr>
<th>Data type1</th>
<th>Data type2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>Integer</td>
</tr>
<tr>
<td>Decimal</td>
<td>Double</td>
</tr>
<tr>
<td>Decimal</td>
<td>Single</td>
</tr>
</tbody>
</table>

All the different data types are transferred into a generic data type of the .Net environment OLE DB (discussed in the design section in 6.2). After the compatible data type list is created, the function handles the data types that are compatible, this will return as a true. When schema elements are incompatible the function returns as a false. This function doesn’t only handle schema elements data types but also GIS - subconcepts data type compatible. In this way only compatible combination with GIS – subconcepts and schema elements are created.

### 6.3 Missing Match problem

#### 6.3.1 Description

The Schema combiner creates a set of combinations. This set of combinations consists of combinations with GIS subconcepts and direct combinations. A complicated situation occurs when schema1 element is combined with a GIS subconcept and schema2 element is also combined with a GIS subconcept, but the path length between the subconcepts is greater than one.

Another situation which can occur when schema1 element doesn’t have a subconcept in the GIS and schema2 element does have a subconcept in the GIS. This situation can occur because the schema matcher does not have seen this element before. In this case the GIS cannot help creating a combination between the subconcepts, because it misses a subconcept for schema1.

**Missing match problem example**

A missing match problem is given in figure 6.1. In the figure schema1 ‘full-name’ is mapped on GIS subconcept ‘full-name’ and from schema2 ‘name-last’ is mapped on the GIS subconcept ‘name-last’. The path between the subconcepts ‘full-name’ and ‘name-last’ is greater than one, namely three. Because the path length is greater than one they will never be transitive combined.

![Figure 6.1: Missing match problem path length greater than one example.](image)
Possible solution missing match problem path length greater than one
When combinations are created using GIS subconcepts, as in the example of figure 6.1 is depicted, the Validators validate the relation between the schema element and the GIS subconcept. The problem using this method is when instead of using GIS combination, a direct combination with the belonging similarity is more reliable than the similarity result that is calculated using GIS combination. This can occur when the instance data between the schemas isn’t validated and the combination that is created, using the GIS, is invalid. To solve this problem, when a path between subconcepts is greater than one, a direct combination between the schema elements is also created. If this approach is used the schema matcher results will contain two the same schema combination but the similarity between the schema match suggestions will be different e.g. table 6.2.

Table 6.2: Combination using the GIS and direct Combination between the schema elements.

<table>
<thead>
<tr>
<th>Schema 1 – element</th>
<th>Schema 2 – element</th>
<th>Similarity</th>
<th>GIS / Directly Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘full-name’</td>
<td>‘first-name’</td>
<td>0.56</td>
<td>Using GIS</td>
</tr>
<tr>
<td>‘full-name’</td>
<td>‘first-name’</td>
<td>0.38</td>
<td>Directly Combined</td>
</tr>
</tbody>
</table>

The similarity values in the table 6.2 are not relevant for explanation of the problem, which one to choose is relevant. A situation that now occurs is that ‘first-name’ is a part of relation with ‘full-name’ therefore using the GIS to create a combination results in a similarity value of 0.58. When the direct combination is fed into the schema matcher it looks at the different aspects of the schema elements towards each other and therefore results into a similarity value of 0.38. This similarity values occurs, because the instance data is not similar enough to each other. The introduction of creating a direct combination when a path between subconcepts is greater than one, created also another problem namely: Which of the schema match combinations is the correct one? An approach could always choose the highest similarity (0.58) or the lowest similarity (0.38), because choosing always the highest similarity can result in incorrect schema match suggestions.

Possible solution missing match problem one of the elements has a subconcept
To explain the problem, figure 6.1 will be used. When an element of schema1 ‘full-name’ has been combined with subconcept ‘fullname’ and schema2 element ‘complete name’ cannot not be combined with a subconcept, because the schema matcher hasn’t seen schema2 element before. Therefore schema1 element and schema2 element cannot be transitively combined with each other.

A possible solution for this problem is when one of the two schema elements is not available. In this case schema2 element, the schema1 elements iterates all his neighbors if there is a schema2 element available that matches. In the case that there is a match, a combination is created using the GIS else a direct combination is created.

The following describes the advantages and disadvantages of solving the Missing Match problem path length greater than one.

Advantages
This recommendation saves a lot of user interaction:
- The match suggestions are now suggested to the user using direct combination or a combination using the GIS instead of manually add the match suggestion in the current situation.
- The user is also relieved from the responsibility to select relevant relating subconcepts (if available) for the elements of every manual added match.
Disadvantages

- The subconcepts that are out of each others reach must be validated before they are fed into the Validators, because of the problem depicted in figure 6.1. Therefore the Schema Combiner needs to be more intelligent.

The following describes the advantages and disadvantages of solving the Missing Match problem when one of the elements has a subconcept.

Advantages

This recommendation saves a lot of user interaction:

- The match suggestions are now suggested to the user using direct combination, instead of, to manually add the match suggestion in the current situation.

Disadvantages

- The Schema Combiner needs to be more intelligent, because for each neighbor, of the subconcept, a possible candidate needs to be validated for the other schema elements. The Schema Combiner will also be more time-consuming, because of the solution.

6.3.2 Design

In the Schema combiner combinations are created with the GIS. The problem is solved for the ‘missing match problem path length greater than one’ and ‘missing match problem one of the elements has a subconcept’ using the above described technique in section 6.2.1. For the first problem path between the neighbors is greater than one in that case a direct combination is created. For the second problem, when there is no neighbor available, a direct combination is also created.

6.3.3 Implementation

The missing match problem is based on the design specification that is specified in the design section of the 6.3 Missing Match problem.

The combinations that are created in the Schema combiner are created with the GIS - subconcepts. Implementing the design features, specified in section 6.2.2, solves the problem when there is no edge available between the subconcepts and when the path length of the subconcepts is greater than one. The problem is solved using the knowledge, that each subconcept has a list of neighbors. For each neighbor subconcept is iterated and looked up in the other Graph of Database2 if the schema element exists. When a schema element is found, a validation takes place if the name and data type are similar enough. After the validation is done and the schema element is similar enough with the GIS – subconcept a combination is created. A different situation can occur when all the neighbors are iterated of the GIS – subconcept and no match is found in the Graph of Database2. In this case only direct combination between Graph of Database1 and Graph of Database2 can be created for this schema element.

6.4 Instance based validator2

Instance based Validator (Validator2) computes a similarity value between two elements ‘elt1’ and ‘elt2’ based on equality of character averages in the character-sets of their instances. The computation is performed based on the assumption that character averages of both elements ‘elt1’ and ‘elt2’ are normally distributed. In practice, when a large schema was used, the time performance of Validator2 was time consuming (several minutes). Therefore it was not acceptable.

With this motivation a new algorithm was implemented for Validator2, whereby only the distribution between the characters, numbers and the rest symbols is analyzed. The new metric is based on the following metric:
When the deviance exceeds 20%, the result of the metric will automatically result in a zero as output. Based on this a simplified version is created for calculating the distribution between the characters, numbers and the rest symbols:

\[
\text{CharacterSimilarity} = 1 - \min(1, 10* \frac{|\text{avgchar}_1 - \text{avgchar}_2|}{\text{avgchar}_1 + \text{avgchar}_2})
\]

\[
\text{NumberSimilarity} = 1 - \min(1, 10* \frac{|\text{avgnum}_1 - \text{avgnum}_2|}{\text{avgnum}_1 + \text{avgnum}_2})
\]

\[
\text{RestSymbolsSimilarity} = 1 - \min(1, 10* \frac{|\text{avgrest}_1 - \text{avgrest}_2|}{\text{avgrest}_1 + \text{avgrest}_2})
\]

Determining the totalSimilarity between the different metrics is done with the following metric:

\[
\text{TotalSimilarity} = \min(\text{CharacterSimilarity}, \text{NumberSimilarity}, \text{RestSimilarity})
\]

This metric determines the weakest similarity between the different metrics and has therefore the most influence. In this way only similar schema elements will result in a similarity score and the rest will receive a score of zero.
7 Evaluation
This section discusses the various aspects of the evaluation of the current Map-IT system. The section ends with a conclusion.

7.1 Evaluation goal
The aim of the evaluation is to verify whether:
- The evaluation system works correctly, i.e.
  - If it can provide an overview how the schema matcher performed according to the measurements and calculated metrics.
- Map-IT learns from its environment e.g. adapting weights of a Validator, so that the parameters automatically are adjusted to improve the matching results.
- Map-IT schema matching quality is improved as resulting the improvements, i.e.
  - The new duplicated search Validator improves the schema matching result.
  - The solving of the Data type problem reduces the solution space and improves the schema matching result.
  - The Meta-Learner improves the schema matching result.

Benchmark
Several schema matchers like [Do and Rahm, 2002] and Cupid [Madhavan et al., 2001] are using the same benchmark collection, which are in XML. Map-IT doesn’t support XML as an input. Therefore, to benchmark the schema matcher, existing available relational databases will be used.

7.2 Theoretical foundation of the evaluation
In section 3.1 it is argued that due to the different ways the current schema match systems have been evaluated, it is difficult to compare the effectiveness of each single system. These criteria input, output, match quality measures, human effort and learning capability, which are discussed in section 3.1, are used for evaluating the schema matcher.

Map-IT is based on the framework of FlexiMatch, therefore it is reasonable that the evaluation results of FlexiMatch will be used to make a comparison with the improved schema matcher. After a series of evaluating the schema matcher, it became clear that the evaluation result of FlexiMatch were not reliable, because none of the reported values correspond with results of the Evaluation system. Therefore the evaluation results of FlexiMatch will not be used for evaluating the improvement of the schema matcher, but instead several runs will be held whereby the improvement are enabled & disabled.

7.3 Evaluation setup
In this section the actual evaluation is described. It is divided into several sections, summed up below and elaborated on afterwards.
- Test schemas; this describes the input of the test.
- Schema matcher configuration; this describes the parameter values of Map-IT during the evaluations.
- Evaluation configuration; this describes the parameter values of evaluation for the schema matcher.
- Evaluation; this describes the actual evaluation.

7.3.1 Test schemas (input)
For this evaluation, we use relational databases created in Microsoft Access. The schemas which are used are described below.
Two domains will be used for evaluation namely:
- Person domain.
- Products domain.

The following databases are used for the person domain:
- Address Book of Thunderbird (e-mail program of Mozilla).
- Address Book of a Sony PDA.
- Database SQL containing a various amount of information e.g. person information.

The following databases are used for the products domain:
- Goodyear pricelist of the customer Techno Benelux.
- Database SQL containing a various amount of information e.g. product information.

The database schemas can be found in Appendix B.

**Schema combinations**

For the person domain 3 input schemas are used. We can create 3 different schema combinations to do the evaluation with.

For testing data type a product domain is introduced containing 2 input schemas are used. We can create 1 different schema combination to do the evaluation with.

**Good schema combinations**

In order to do performance tests, it should be clear which combinations are good and which are not. See appendix C (output), this shows what combinations are considered good. For the less trivial correct combinations, explanations are given in appendix D.

**Observation schemas**

The available databases mostly consist of one table except the customer database. The ability for testing a scenario whereby two large databases can be evaluated will not be done in this evaluation.

### 7.3.2 Schema matcher configuration

Map-IT contains quite some parameters that can influence the performance of Map-IT. We have tuned these parameters using a few pre-evaluation tests resulting in the parameter values shown in table 7.1. It is certain that these values are not the optimal values, because for every schema a different parameter setting is possible.

The list of parameters influencing the performance of Map-IT is given below, together with the values used during the evaluation. The parameters which require further explanation are elaborated on afterwards.

Table 7.1: Default evaluation parameters for the schema matcher whereby all the Validators are enabled.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Class variable of</th>
<th>Set in class</th>
</tr>
</thead>
<tbody>
<tr>
<td>_TOPNumber</td>
<td>10</td>
<td>MainMapper</td>
<td>Evaluation</td>
</tr>
<tr>
<td>_FromPercentage</td>
<td>0.55</td>
<td>MainMapper</td>
<td>Evaluation</td>
</tr>
<tr>
<td>InstanceDataUpdatable</td>
<td>true</td>
<td>MainMapper</td>
<td>MainMapper</td>
</tr>
<tr>
<td>_NameAndDatatypeSim</td>
<td>0.9</td>
<td>SchemaCombiner</td>
<td>SchemaCombiner</td>
</tr>
<tr>
<td>_NbrInstancePerConcept</td>
<td>200</td>
<td>GIS</td>
<td>GIS</td>
</tr>
<tr>
<td>_NbrInstanceToUpdate</td>
<td>12</td>
<td>GIS</td>
<td>GIS</td>
</tr>
<tr>
<td>_CombineThreshold</td>
<td>0.7</td>
<td>TransitiveCombiner</td>
<td>TransitiveCombiner</td>
</tr>
<tr>
<td>Weight</td>
<td>0.5</td>
<td>Validator1</td>
<td>Evaluation</td>
</tr>
<tr>
<td>Weight</td>
<td>1.5</td>
<td>Validator2</td>
<td>Evaluation</td>
</tr>
<tr>
<td>Weight</td>
<td>1.5</td>
<td>Validator3</td>
<td>Evaluation</td>
</tr>
<tr>
<td>InstanceCntr</td>
<td>1000</td>
<td>Validator2</td>
<td>Validator2</td>
</tr>
</tbody>
</table>
This value is set to 0.55. Only combinations that have a higher similarity score than 0.55 will be presented to the user or the Evaluation system.

For creating combinations with GIS subconcepts a validation is done based on name and data type. If the name and data type correspond it will result in a combination with the corresponding GIS subconcept, else a direct combination is created.

This parameter denotes the number of instances that are updated per subconcept. This takes place when the GIS learns from an accepted schema combination.

The Weight parameter for each Validator object denotes its respective initial weight to be used in the Prediction aggregator (see section 2.2.6), to derive final similarity values for every combination.

This parameter denotes the amount of instance data used in the computation of the instance based similarity value. This occurs within the instance based Validator (Validator2) for both elements of the combination.

7.3.3 Evaluation configuration

To evaluate the improvements using the Evaluation system several configurations need to be created for each improvement. The improvements that will be evaluated in this case are:

- Duplicated search Validator.
- Meta-Learner.
- Data type similarity.
- Combination of all the improvements.

To evaluate the improvements several tests need to be done with and without the improvements. The databases that are specified above will be used to validate the improvements using different configurations. Evaluating different aspects of the schema matcher can not be done with one test run. Therefore multiple test runs will be held using the Evaluation system. The metric that will be used for the match quality measure is the overall performance, described in section 3.1. In the conclusion the variables that were needed for calculating the overall performance measure are manually compared with the results of the evaluation system.

7.3.4 Evaluation

In this section the actual evaluation is discussed based on the evaluation configuration.

Duplicated search Validator

To test the duplicated search efficiently the Meta-Learner is disabled, because the Meta-Learner can influence the result. The test consists out of two states disabled the improvement and when the improvement is enabled (each configuration holds two databases):

- Duplicated search Validator disabled:
  - One configuration with three test runs.
  - Three configurations with three test runs.
- Duplicated search Validator enabled:
  - One configuration with three test runs.
  - Three configurations with three test runs.
Map-IT: An advanced multi-strategy and learning approach to schema matching

One configuration

Table 7.2: Overall performance duplicated search disabled.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Run1</th>
<th>Run2</th>
<th>Run3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address book Thunderbird – Customer Database SQL</td>
<td>0.23</td>
<td>0.38</td>
<td>0.45</td>
</tr>
<tr>
<td>PDA address – Customer Database SQL</td>
<td>0.11</td>
<td>0.70</td>
<td>0.65</td>
</tr>
<tr>
<td>PDA address – Address book thunderbird</td>
<td>0.29</td>
<td>0.54</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Table 7.3: Overall performance duplicated search enabled.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Run1</th>
<th>Run2</th>
<th>Run3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address book Thunderbird - Customer Database SQL</td>
<td>0.42</td>
<td>0.66</td>
<td>0.69</td>
</tr>
<tr>
<td>PDA address – Customer Database SQL</td>
<td>0.16</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>PDA address – Address book thunderbird</td>
<td>0.34</td>
<td>0.58</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 7.4: Found correct combinations (True positives) for PDA address – Address book thunderbird (total correct combinations 24).

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Run1</th>
<th>Run2</th>
<th>Run3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicated search disabled</td>
<td>6</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Duplicated search enabled</td>
<td>8</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

**Observation with one configuration**

In figure 7.2 it is explained how many direct and GIS combinations are created per run. This is to explain evaluation results of the duplicated search Validator. In the first run only direct combinations are created because the GIS is empty. In the second run the system learned from the approved and disapproved mapping. This results that almost every attribute is combined with GIS. In figure 7.1 it is clearly visible that in the first run, the first bar (duplicated search disabled) is 6% lower than the second bar (duplicated search enabled) which is visible in table 7.4 in column Run1, two more correct combinations are found. In the second and third run there is almost no difference in the overall performance, because there are only a few direct combinations, namely four.
Conclusion
The duplicated search **Validator** improves the overall performance of the schema matcher in the first run with 6%, which results in two correct mappings and one false mapping. After the first run the schema matcher learned from the approved and the disapproved mappings which results in a few direct combinations see figure 7.2. The overall performance doesn’t increase after the first iteration, because of the few direct combinations.

Meta-Learner
To test the **Meta-Learner** efficiently the duplicated search **Validator** is disabled, because this can influence the result. The test consists out of two states:

- **Meta-Learner** disabled:
  - One configuration with fifty test runs.
  - Three configurations with fifty test runs.
- **Meta-Learner** enabled:
  - One configuration with fifty test runs.
  - Three configurations with fifty test runs.

Table 7.5: Found correct combinations (True positives) for PDA address – Address book thunderbird (total correct combinations 24).

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average over 50 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaLearner disabled</td>
<td>26</td>
</tr>
<tr>
<td>MetaLearner enabled</td>
<td>29</td>
</tr>
</tbody>
</table>

Observation with one configuration
To evaluate the **Meta-Learner** a set of fifty runs are performed with a configuration that holds two databases. In figure 7.4 it is clearly visible (right corner) that the average is 5% higher in contrast to figure 7.3. The maximum similarity of 0.73 in figure 7.4 occurs only twice, which is statistically not relevant. After the first run the schema matcher learned from the approved and disapproved...
mappings. This is clearly visible in both images overall performance improves with 50%. The overall performance in figure 7.3 fluctuates a lot per run in contrast to in figure 7.4. The fluctuation occurs because of the problem with the ambiguous attributes names in the GIS.
From run 48 until 50 in figure 7.3 the overall performance decreases. This occurs because the true positives and the false negatives are stable, but the false positives are increased.

**Conclusion**
The Meta-Learner improves the mapping results and saves time for the user on average (in our test set) three mappings. When the Meta-Learner is enabled the overall performance is more stable, because the false positives are reduced. This is not the case when the Meta-Learner is disabled.

### Three configurations

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Meta-Learner disabled overall performance</th>
<th>Meta-Learner enabled overall performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address book Thunderbird - Customer Database SQL</td>
<td>0.44</td>
<td>0.47</td>
</tr>
<tr>
<td>PDA address – Customer Database SQL</td>
<td>0.49</td>
<td>0.47</td>
</tr>
<tr>
<td>PDA address – Address book thunderbird</td>
<td>0.59</td>
<td>0.65</td>
</tr>
</tbody>
</table>

### Three configuration

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average over 50 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetaLearner disabled</td>
<td>26</td>
</tr>
<tr>
<td>MetaLearner enabled</td>
<td>28</td>
</tr>
</tbody>
</table>

**Observation with three configurations**
To evaluate the Meta-Learner a set of fifty runs are performed with four configurations that hold two databases per configuration. In figure 7.6 it is clearly visible that the average is 3% higher in contrast to figure 7.5. The minimum, maximum and deviation are constant in both figures. After
the first run the schema matcher learned from the approved and disapproved mappings. This is clearly visible in both images. The overall performance in figure 7.4 fluctuates a lot per run in contrast to in figure 7.6. In table 7.6 almost all the overall performances are better when the Meta-Learner is enabled, except for the configuration PDA address – Customer Database SQL. The explanation for this is that the weights of the Validators are adjusted in an incorrect way, because of the ambiguous attributes in the Customer Database SQL database. The weights of the Validators are already adjusted in the run PDA address – Address book thunderbird. This is visible in the improved overall performance in table 7.6.

Conclusion
The Meta-Learner improves the mapping results with 3%. This saves the user (in our test set) two mappings on average per run, see table 7.6. When the Meta-Learner is enabled the overall performance doesn’t fluctuate a lot anymore per run, in contrast to when the Meta-Learner is disabled.

Data type similarity
To test the Data type similarity efficient the duplicated search Validator and the Meta-Learner is disabled, because this can influence the result. The test consists out of two states:

- **Data type similarity enabled**:
  - One configuration with fifty test runs.
  - Four configurations with fifty test runs.

- **Data type similarity disabled**:
  - One configuration with fifty test runs.
  - Four configurations with fifty test runs.

![Figure 7.7: Data type similarity disabled.](image1)

![Figure 7.8: Data type similarity enabled.](image2)
Table 7.8: Overall performance per configuration.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Data type similarity disabled overall performance</th>
<th>Data type similarity enabled overall performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address book Thunderbird - Customer Database SQL</td>
<td>0.42</td>
<td>0.41</td>
</tr>
<tr>
<td>Goodyear – Customer Database SQL</td>
<td>0.19</td>
<td>0.22</td>
</tr>
<tr>
<td>PDA address – Customer Database SQL</td>
<td>0.47</td>
<td>0.47</td>
</tr>
<tr>
<td>PDA address – Address book thunderbird</td>
<td>0.59</td>
<td>0.59</td>
</tr>
</tbody>
</table>

**Observation with four configurations**

To evaluate the data type validation a set of fifty runs are performed with four configurations that hold two databases per configuration. Observing the figure 7.7 and figure 7.8 it is visible that there are no improvements in the overall performance or in any other metric values for this configuration. In table 7.8 almost all the overall performances are similar when the data type similarity is enabled except for the configuration Goodyear – Customer Database SQL which is better. The explanation for this is that the Goodyear database contains numeric fields in contrast to the other databases.

**Conclusion**

The database of Customer Database SQL consists of various amounts of data types. When a configuration contains two databases that both consist out of string attributes, the data type validation function has no effect on the overall performance. In a configuration which contains databases that have numeric fields in both databases the overall performance is improved using the function.

**One configuration**

Figure 7.9: Data type similarity disabled.

Figure 7.10: Data type similarity enabled.
Table 7.9: Average not found combinations (false negatives) for PDA address – Address book thunderbird.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average over 50 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type similarity disabled</td>
<td>8</td>
</tr>
<tr>
<td>Data type similarity enabled</td>
<td>7</td>
</tr>
</tbody>
</table>

**Observation with one configuration**

To evaluate the data type validation a set of fifty runs are performed with two configurations that hold two databases. Observing the figure 7.9 and figure 7.10 it is visible that there is 3% improvement in the overall performance. This occurs because *false negatives* are increased from run 45 until 50 in figure 7.9. In figure 7.10 from run 38 until 50 the overall performance is stable but does not perform less when data type similarity is disabled in figure 7.9. This occurs because the *false positives* are increased (see table 7.9).

**Conclusion**

For this configuration the data type validation improves the overall performance with 3%. When a configuration contains two databases that have numeric fields in both databases, the overall performance is improved using the function.

**Combination of all the improvements**

The final evaluation consists of enabling the improvements together (duplicated search *Validator*, *Meta-Learner* and data type validation). The test consists out of two states:

- All improvements enabled:
  - One configuration with fifty test runs.
  - Four configurations with fifty test runs.
- All improvements disabled:
  - One configuration with fifty test runs.
  - Four configurations with fifty test runs.

**One configuration**

![Figure 7.11: All improvements disabled.](image)

![Figure 7.12: All improvements enabled.](image)
Table 7.10: Found correct combinations (True positives) for PDA address – Customer Database SQL (total correct combinations 39).

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average over 50 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>All improvements disabled</td>
<td>27</td>
</tr>
<tr>
<td>All improvements enabled</td>
<td>30</td>
</tr>
</tbody>
</table>

Observation with one configuration
To evaluate all the improvements a set of fifty runs are performed with a configuration that holds two databases. In figure 7.12 it is visible that the average is 4% higher in contrast to figure 7.11. The maximum similarity of 0.77, in figure 7.12, difference only 2% from the maximum of figure 7.11. This is statistically not relevant. After the first run the schema matcher learned from the approved and disapproved mappings. This is clearly visible in both images. In the first run, in figure 7.12, the duplicated search Validator improves the overall performance with 2% in contrast to figure 7.11.

Conclusion
When all the improvements are enabled the mapping results are improved with 4%, which saves the user on average three mappings (see table 7.10). Because the ambiguous attributes in the Customer Database SQL database, the overall performance fluctuates in both images.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>All improvements disabled overall performance</th>
<th>All improvements enabled overall performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address book Thunderbird - Customer Database SQL</td>
<td>0.43</td>
<td>0.48</td>
</tr>
<tr>
<td>PDA address – Customer Database SQL</td>
<td>0.48</td>
<td>0.49</td>
</tr>
<tr>
<td>PDA address – Address book thunderbird</td>
<td>0.59</td>
<td>0.61</td>
</tr>
</tbody>
</table>

Figure 7.13: All improvements disabled.

Figure 7.14: All improvements enabled.

Table 7.11: Overall performance per configuration.
Table 7.12: Found correct combinations (True positives).

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Average over 50 runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>All improvements disabled</td>
<td>26</td>
</tr>
<tr>
<td>All improvements enabled</td>
<td>29</td>
</tr>
</tbody>
</table>

Observation with four configurations
To evaluate all the improvements a set of fifty runs are performed with four configurations that hold two databases per configuration. In figure 7.14 it is visible that the average is 3% higher in contrast to figure 7.13. The overall performance fluctuates a lot per configuration the minimum, maximum and deviation is constant in both figures. After the first run the schema matcher learned from the approved and disapproved mappings, which is clearly visible in both images. In table 7.11 almost all the overall performance are better when all the improvements are enabled.

Conclusion
When all the improvements are enabled the mapping results are improved with 3%, which saves the user on average three mappings. The Goodyear database consists out of attribute that contain a consistent structure of instance data, but in the matching database the attribute doesn’t have a consistent structure. Therefore the instance based Validator scores not high, which has a high weight. The duplicated search Validator could found duplicates, if a snapshot was not taken from the large database (Customer Database SQL). The time that is needed to create an inverted index, for the duplicated search Validator, from a large database is not acceptable in practice, therefore snapshots are taken.

7.4 Evaluation conclusion
This section discusses the following:
• The test results of the evaluation, by looking at the evaluation goals mentioned in Evaluation goals at the start of this chapter.

For clarity reasons, the evaluation goals are repeated with the bullets below. It is aimed to verify whether:
• The evaluation system works correctly.
  This is verified based on:
    § The calculated overall metric.
    § GIS is adapted correctly.
• The Map-IT learns from its environment using the Meta-Learner.
• The Duplicated search Validator enhances overall performance.
• Validating compatible data types reduces the solutions space and improves the schema matching result.
• All the improvements enabled, improves the overall performance.

These bullets are elaborated on in subsequent sections.

7.4.1 Evaluation system works correctly
Map-IT Evaluation system works correctly, which is concluded based on two checks:
• Calculated overall metric conforms to the manual calculated overall performance.
  To calculate the overall performance the following variables are used true positives, false negatives and false positives. Comparing the manual specified correct mappings it can be concluded that the produced numbers, from the Evaluation system, are correct.
• Map-IT adapts the GIS correctly.
  GIS subconcepts are created from the feedback of the Evaluation system. Relations between subconcepts are created, strengthened and weakened correctly based on specified correct mappings in the configuration object (see Appendix C: Good mappings).
7.4.2 Duplicated search improves the overall performance

In this section we will discuss whether including the Duplicated search Validator enhances the overall performance. This is done by comparing the results of the test Duplicated search Validator disabled/enabled. In the first run the schema matcher didn’t learn from the previous mappings, which is visible in the overall performance figure 7.1. After the first run the schema matcher learned from the previous mappings. Most of the combinations are GIS combinations and a few direct combinations, which is visible in figure 7.2. When the Duplicated search Validator is enabled the overall performance is increased by 6% in comparison with when the Duplicated search Validator is disabled (see figure 7.1). More runs do not have any effect, on the performance, of the duplicated search, because most of the combinations are GIS combinations. Duplicated search Validator doesn’t support GIS combinations, which cannot increase the overall performance. Therefore we can conclude that the Duplicated search Validator in this evaluation improves the overall performance.

7.4.3 Meta-Learner learns from its environment

In this section we will discuss whether including the Meta-Learner enhances the overall performance. This is done by comparing the results of the test Meta-Learner disabled/enabled. The correct and incorrect mappings fluctuate, because the schema matcher doesn’t adapt the weights of the Validator. The ambiguous fields have a great impact on the overall performance, which is also an explanation why the overall performance fluctuates. The correct and incorrect mappings are more stable, because the weights of the Validators are adjusted due to its environment. The true positives are greater, false positives are smaller and more stable when the Meta-Learner is disabled. Therefore we can conclude that the Meta-Learner does in fact, in this evaluation, enhances the overall performance of the schema matcher.

7.4.4 Data type similarity increases overall performance

In this section we will discuss whether including the Data type similarity enhances the overall performance. This is done by comparing the results of the test Data type similarity enabled. In the situation when one configuration is used, the overall performance of the system increased by 3%. The explanation for this is both databases contain numeric data types. In the state two, when the data type similarity is enabled, the overall performance is only increased when both databases contain numeric attributes. This is the case again for configuration Goodyear – Customer Database SQL. Therefore we can conclude that in the data type similarity in some cases do in fact, in this evaluation, enhances the overall performance of the schema matcher.

7.4.5 All improvements enabled increases overall performance

In this last section we will discuss whether enabling all the improvements enhances the overall performance. This is done by comparing the results of the test All improvements disabled/enabled. When all the improvements are disabled the user missed on average per run 3 mappings, which is on average 3%. Observing table 7.11 all the configuration showing an increased average overall performance when the improvements are enabled. This shows that the improvements, for the schema matcher, increase the overall performance.
8 Conclusions

Map-IT is a schema match system that supports the multi-strategy approach. The strategies are implemented as Validators, and the current version of Map-IT contains three Validators, respectively exploiting schema, instance based information and tuple based information. Map-IT also learns from previous mappings and from its environment. Besides finding matches between schema elements of both source schemas, it now also can indirectly find schema match, the system would not have come up with otherwise. In every subsequent subsection another conclusion from the thesis is discussed.

Map-IT is an implementation of advanced schema matching solution

Based on FlexiMatch, which uses the multi-strategy learning approach, the schema matching framework Map-IT is designed and implemented. From the various evaluation experiments, it can be concluded that the system is improved and Evaluation system works: suggested matches between schema elements of both schemas are validated and when it is necessary corrected by the Evaluation system.

Map-IT exploits tuple based information

Based in the DUMAS technique the duplicated search Validator exploits tuple based information to find semantic overlap in both databases. The duplicates that are found are evaluated using the SoftTFIDF technique which results in a few remaining accurate duplicates. The remaining duplicates are aggregated to derive a final similarity for each attributed, which is finally stored in the similarity cube.

Map-IT learns from its environment

The Weight parameter for each Validator object denotes its respective initial weight to be used in the Prediction aggregator to derive final similarity values for every combination. Using the knowledge of the user feedback and the similarity score for each Validator, stored in the similarity cube, the Validators can be evaluated and adjusted. Using this information the schema matcher automatically adjust the weight of the Validators to its environment e.g. person information domain. This increases the correct and decreases the missed combinations, which result in saving time to find the missed combinations.

Implementation within Sync-IT & N-Gram Validator

Two design goals have not been accomplished. These are discussed below.

Implementation within Sync-IT

One of the design goals was to implement the Map-IT within the tool Sync-IT.

Implementation of the N-Gram Validator

One of the design goals was to implement the N-Gram Validator into the schema matcher.
9 Recommendations

This recommendation section consists of three types of recommendations, namely:
- Recommendations arising from the evaluation observations.
- Recommendations which could not be implemented due to time constraints.
- Recommendations for improvements which were beyond the scope of this thesis in the first place.

These types of recommendations are discussed in the following three sections.

9.1 Evaluation recommendations

The remainder of this section discusses several recommendations based on observations made during the evaluation.

9.1.1 Ambiguous GIS concept problem

This recommendation is relevant in case that a database contains various ambiguous attributes names.

Current implementation

In the current implementation of Map-IT, each subconcept in the GIS is unique using the name in combination with the data type as an identifier. The schema combiner generates combinations based on schema attributes with GIS subconcepts. When a database contains ambiguous attributes names e.g. Database1 contains a table ‘products’ which contains an attribute ‘name’ and Database2 contains a table ‘persons’ which contains an attribute ‘name’ both the attributes have the same data-type, namely string. In GIS a subconcept exists with the identifier ‘name’. The schema combiner creates a combination based on the attributes names of the database, which generates two combinations namely:
- Attribute ‘name’ from the table ‘products’ with GIS subconcept ‘name’.
- Attribute ‘name’ from the table ‘persons’ with GIS subconcept ‘name’.

The Validators validated the combinations and stores the similarity, in the similarity cube. The results are presented to the user / Evaluation system. The user / Evaluation system gives the feedback on the suggested combinations. The instance data of the GIS subconcept ‘name’ is updated using both attributes fields from the two tables ‘products’ and ‘persons’. In this situation the instance data is mixed from two different domains. This influences strongly the similarity calculation of the instance based Validator in the following iterations. This Validator has also a high influence on the finale similarity calculations due to its initial weight.

9.2 Designed functionality recommendations

Some designed components are not included in the current version of Map-IT. The subsequent sections discuss these components and describes how they still could be implemented.

9.2.1 Meta-Learner extensibility

The current implementation of Map-IT only focuses on the automatically weight adjustment of the Validators. In the current Map-IT framework various parameters could be adjusted by the Meta-Learner namely:
- TOPNumber: This property contains the number of how many schema elements per combination are allowed as suggested combinations e.g. for each attribute in schema1 three combinations are allowed.
- FromPercentage: This property contains the number of the threshold, which all the suggested combinations need to be above to be presented to the user / Evaluation
system, e.g. for every combinations whereby the similarity that is equal or greater is then 0.55 will be presented to the user / Evaluation system.

- NbrInstanceToUpdate: This parameter denotes the number of instances that are updated per subconcept when the GIS learns from an accepted schema combination.
- InstanceCnt: This parameter denotes the amount of instance data used in the computation of the instance based similarity value within Validator2 for both elements of the combination.

**Advantage**
The advantage of making more parameters adjustable is:
- Improves overall performance of the schema matcher, because the parameters are adjusted due to its environment.
- Less user interaction for adjusting the parameters in the Map-IT framework.

**Disadvantage**
When the schema matcher operates in various domains the parameters are constantly adjusted, which can influence the overall performance of the schema matcher.

### 9.2.2 Combination strategies

Due the lack of time the combination strategies described in section 6.1 could not be implemented. Several options are interested:
- Aggregation.
- Direction.
- Selection.

**Aggregation**
Techniques for combination strategies are discussed in the section 6.1; *Average*, *Max*, and *Weighted* strategy. The current schema matcher supports only the *Weighted* similarity metric. To evaluate also the other metrics e.g. *Max* similarity and the *Average* similarity new parameters need to be introduced in the Prediction Aggregator. Two extra methods need to be introduced, in the Prediction aggregator, for the implementation of the *Max* similarity and the *Average* similarity.

**Direction**
In *Schema combiner* the direction selection takes place. For selecting which database should be used first, a new parameter needs to be introduced in the *Schema combiner*.

**Selection**
The *MaxDelta* parameter allows suggested match results with the same or almost similar similarity values, to be suggested. In the current schema matcher the match suggestions are reduced in *Mapping generator* (depending on the *threshold* or the *topN*). The adjustment, to make the *MaxDelta* possible, only needs an implementation adjustment and not an adjustment to the framework. The *MaxDelta* can be in adjusted from 0.01 to 0.05 whereby more or less match suggestions are presented to the user.

### 9.2.3 N-Gram Validator

The *N-Gram* Validator is an instance based *Validator*. This *Validator* compares strings according to their set of n-grams, i.e., sequences of n characters. This leads to different variants e.g., Bigram (2), Trigram (3). The *Validator* will be a good additional *Validator* that exploits specific instance data strings. This Validator cannot be used for numeric data-types. Next to the already existing Validators the *N-Gram Validator* can increase the overall performance of the schema matcher.
9.2.4 Iteration option

Map-IT was designed to be able to make more iterations before a final schema combination is accepted.

**Advantage**
The advantage of making more iterations possible are:
- To be able to have the computational more expensive Validators be applied on a solution space which was reduced by cheap Validators in earlier iterations.
- To have special (not yet implemented) structural Validators cascade match and mismatch information to related combinations.

**Implementation ability**
To be able to iterate within Map-IT, an intermediate accepted mapping should be integrated with the initial combinations to be able to use the user feedback in the next iteration. All initial combinations are the ElementCombination objects stored in the three SourcesCombinations objects generated by the Schema combiner.

**Integrate possibility**
From the GISCombined property of node representations of schema elements, it can be derived whether a match suggestion was based on a direct or a transitive combination. In case of a transitive combination, the initial combinations with subconcepts can be traced back via the ViaGIS property of the ElementCombination object. This implies that we can retrieve all the information we need to integrate the intermediate accepted mapping with the initial combinations, hence, that the iteration option is supported.

9.3 Future enhancement recommendations

Some components are implemented in an ad-hoc fashion, because putting much effort in it was outside the scope of the thesis. This section includes recommendations to replace, add or adapt components of the current implementation of Map-IT, in order to enhance its future (commercial) performance.

**Complex Mapping**
In COMA [Do and Rahm, 2002] complex mappings are addressed, which makes it possible to generate combinations from the order 1:n, n:1 and n:m. Current implementation only 1:1 combinations can be generated by the schema matcher. A distinction is made between element-level matching and structural-level matching. Element-level matching is used for 1:1, 1:n and n:1 and for the structural-level matching the n:m combinations are used. In the case of structural-level matching several tables could be used to generate the same amount of information as in the other database from one table.

**Structural Validator**
An example of a good structural Validator algorithm is the TreeMatch algorithm used in Cupid [Madhavan et al., 2001]. For the computation of the similarity between two combined schema elements ‘elt1’ and ‘elt2’, the structural algorithm of Cupid checks whether schema elements related to ‘elt1’ and ‘elt2’ are also combined, and if so, what their respective similarity values are.

**Machine Learning Validators**
In the paper of SemInt [Clifton, 2000] it is concluded that Neural Networks perform well in schema matching. Adding a Validator based on Neural Networks would therefore be a good addition to Map-IT. Another good performing and popular Machine Learning algorithm is the Naïve Bayesian implementation. The LSD [Doan et al., 2003] system, has implemented this algorithm.
References


[Dionysius, Huijsmans and Sebe, 2001] Dionysius P. Huijsmans, Nicu Sebe: Extended Performance Graphs for Cluster Retrieval. CVPR (1) 2001:


[Levenshtein Distance] http://www.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm


Appendix A: FlexiMatch framework
Appendix B: Evaluation input schemas

<table>
<thead>
<tr>
<th>Thunderbird</th>
<th>Address Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td></td>
</tr>
<tr>
<td>Last Name</td>
<td></td>
</tr>
<tr>
<td>Display Name</td>
<td></td>
</tr>
<tr>
<td>Nick Name</td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td></td>
</tr>
<tr>
<td>Additional Email</td>
<td></td>
</tr>
<tr>
<td>Screen Name</td>
<td></td>
</tr>
<tr>
<td>Work Number</td>
<td></td>
</tr>
<tr>
<td>Home Number</td>
<td></td>
</tr>
<tr>
<td>Fax Number</td>
<td></td>
</tr>
<tr>
<td>Pager Number</td>
<td></td>
</tr>
<tr>
<td>Mobile Number</td>
<td></td>
</tr>
<tr>
<td>Home Address</td>
<td></td>
</tr>
<tr>
<td>Home City</td>
<td></td>
</tr>
<tr>
<td>Home State</td>
<td></td>
</tr>
<tr>
<td>Province</td>
<td></td>
</tr>
<tr>
<td>Home ZIP</td>
<td></td>
</tr>
<tr>
<td>Home Country</td>
<td></td>
</tr>
<tr>
<td>Web page</td>
<td></td>
</tr>
<tr>
<td>Work Title</td>
<td></td>
</tr>
<tr>
<td>Work Department</td>
<td></td>
</tr>
<tr>
<td>Work Organisation</td>
<td></td>
</tr>
<tr>
<td>Work Address</td>
<td></td>
</tr>
<tr>
<td>Work City</td>
<td></td>
</tr>
<tr>
<td>Work State</td>
<td></td>
</tr>
<tr>
<td>Work Province</td>
<td></td>
</tr>
<tr>
<td>Work ZIP</td>
<td></td>
</tr>
<tr>
<td>Work Country</td>
<td></td>
</tr>
<tr>
<td>Work Web page</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PDA Arjen</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last name</td>
<td></td>
</tr>
<tr>
<td>First name</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td></td>
</tr>
<tr>
<td>Home number</td>
<td></td>
</tr>
<tr>
<td>Fax</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>Email</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>Zip code</td>
<td></td>
</tr>
<tr>
<td>Country</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix C: Good mappings

<table>
<thead>
<tr>
<th>PDA Arjen – Address</th>
<th>Database SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Address],[First Name], [Administrators],[name]</td>
<td></td>
</tr>
<tr>
<td>[Address],[First Name], [dealer_users],[firstname]</td>
<td></td>
</tr>
<tr>
<td>[Address],[First Name], [dealers],[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>[Address],[First Name], [dealers],[contactperson2]</td>
<td></td>
</tr>
<tr>
<td>[Address],[First Name], [supplier_users],[firstname]</td>
<td></td>
</tr>
<tr>
<td>[Address],[First Name], [suppliers],[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>[Address],[First Name], [suppliers],[contactperson2]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Last Name], [Administrators],[name]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Last Name], [dealer_users],[lastname]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Last Name], [dealers],[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Last Name], [dealers],[contactperson2]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Last Name], [supplier_users],[lastname]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Last Name], [suppliers],[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Last Name], [suppliers],[contactperson2]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Email], [Administrators],[email]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Email], [dealer_users],[email]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Email], [dealers],[email]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Email], [supplier_users],[email]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Email], [suppliers],[email]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Email], [fax_email],[email]</td>
<td></td>
</tr>
<tr>
<td>[Address],[City], [dealers],[city]</td>
<td></td>
</tr>
<tr>
<td>[Address],[City], [suppliers],[street_city]</td>
<td></td>
</tr>
<tr>
<td>[Address],[City], [suppliers],[city]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Address], [Administrators],[adres]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Address], [dealers],[deliveradres]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Address], [suppliers],[adres]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Zip code], [Administrators],[zipcode]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Zip code], [dealers],[city_zipcode]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Zip code], [suppliers],[city_zipcode]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Zip code], [suppliers],[city_zipcode]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Home Number], [Administrators],[telephonenumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Home Number], [dealer_users],[telephonenumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Home Number], [dealers],[telephonenumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Home Number], [supplier_users],[telephonenumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Home Number], [suppliers],[telephonenumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Fax], [Administrators],[faxnumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Fax], [dealer_users],[faxnumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Fax], [dealers],[faxnumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Fax], [supplier_users],[faxnumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Fax], [suppliers],[faxnumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Fax], [fax_email],[faxnumber]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Company], [dealers],[name]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Company], [suppliers],[name]</td>
<td></td>
</tr>
<tr>
<td>[Address],[Company], [supplier_users],[name]</td>
<td></td>
</tr>
<tr>
<td>Thunderbird – Address Book</td>
<td>Database SQL</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>• [Address Book].[First Name], [Administrators].[name]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[First Name], [dealers].[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[First Name], [dealers].[contactperson2]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[First Name], [suppliers].[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Last Name], [Administrators].[name]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Last Name], [dealers].[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Last Name], [supplier_users].[lastname]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Last Name], [suppliers].[contactperson2]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Display Name], [Administrators].[name]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Display Name], [dealers].[contactperson1]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Display Name], [supplier_users].[firstname]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Display Name], [suppliers].[contactperson2]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[EMail], [Administrators].[email]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[EMail], [supplier_users].[email]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[EMail], [suppliers].[email]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[EMail], [fax_email].[email]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Additional Email], [Administrators].[email]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Additional Email], [dealers].[email]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Additional Email], [suppliers].[email]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home City], [dealers].[city]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home City], [suppliers].[city]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home City], [suppliers].[street_city]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home City], [suppliers].[street_city]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home Address], [Administrators].[adresse]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home Address], [dealers].[deliveradresse]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home Address], [supplier_users].[adresse]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home ZIP], [Administrators].[zipcode]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home ZIP], [dealers].[city_zipcode]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home ZIP], [suppliers].[city_zipcode]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home ZIP], [supplier_users].[city_zipcode]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Web page], [dealers],[homepage]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Web page], [suppliers],[homepage]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home Number], [Administrators],[telephonenumber]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home Number], [dealer_users],[telephonenumber]</td>
<td></td>
</tr>
<tr>
<td>• [Address Book].[Home Number], [dealers],[telephonenumber]</td>
<td></td>
</tr>
</tbody>
</table>
Map-IT: An advanced multi-strategy and learning approach to schema matching

- [Address Book].[Home Number], [suppliers].[telephonenumber]
- [Address Book].[Fax Number], [Administrators].[faxnumber]
- [Address Book].[Fax Number], [dealer_users].[faxnumber]
- [Address Book].[Fax Number], [dealers].[faxnumber]
- [Address Book].[Fax Number], [supplier_users].[faxnumber]
- [Address Book].[Fax Number], [suppliers].[faxnumber]
- [Address Book].[Fax Number], [fax_email].[faxnumber]
- [Address Book].[Work Address], [Administrators].[adress]
- [Address Book].[Work Address], [dealers].[deliveradress]
- [Address Book].[Work Address], [suppliers].[street_adres]
- [Address Book].[Work Address], [supplier_users].[adress]
- [Address Book].[Work Address], [fax_email].[faxnumber]
- [Address Book].[Work Address], [Administrators].[adress]
- [Address Book].[Work City], [dealers].[city]
- [Address Book].[Work City], [suppliers].[city]
- [Address Book].[Work City], [suppliers].[street_city]
- [Address Book].[Work ZIP], [Administrators].[zipcode]
- [Address Book].[Work ZIP], [dealers].[city_zipcode]
- [Address Book].[Work ZIP], [dealers].[del_zipcode]
- [Address Book].[Work ZIP], [suppliers].[city_zipcode]
- [Address Book].[Work ZIP], [supplier_users].[street_zipcode]
- [Address Book].[Work ZIP], [Administrators].[homepage]
- [Address Book].[Work Web page], [dealers].[homepage]
- [Address Book].[Work Web page], [suppliers].[homepage]
- [Address Book].[Work Title], [company_functions].[name]
- [Address Book].[Work Number], [Administrators].[telephonenumber]
- [Address Book].[Work Number], [dealer_users].[telephonenumber]
- [Address Book].[Work Number], [dealers].[telephonenumber]
- [Address Book].[Work Number], [supplier_users].[telephonenumber]
- [Address Book].[Work Number], [suppliers].[telephonenumber]
- [Address Book].[Fax Number], [Administrators].[faxnumber]
- [Address Book].[Fax Number], [dealers].[faxnumber]
- [Address Book].[Fax Number], [supplier_users].[faxnumber]
- [Address Book].[Fax Number], [suppliers].[faxnumber]
- [Address Book].[Fax Number], [fax_email].[faxnumber]

<table>
<thead>
<tr>
<th>Goodyear - pricelist</th>
<th>Database SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>[producten].[Leveranciers artikel code], [products].[product_code]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Artikelgroep], [productgroups].[name]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Merk], [brands].[name]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Leverancier], [suppliers].[name]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Artikelomschrijving], [products].[name]</td>
<td></td>
</tr>
<tr>
<td>[producten].[size], [products].[tire_size]</td>
<td></td>
</tr>
<tr>
<td>[producten].[type], [products].[tire_type]</td>
<td></td>
</tr>
<tr>
<td>[producten].[lisi], [products].[tire_lisi]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Bruto Prijs], [products].[price]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Korting], [products].[discount]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Bestel eenheid], [products].[packageunit]</td>
<td></td>
</tr>
<tr>
<td>[producten].[Minimale afname], [products].[min_quantity]</td>
<td></td>
</tr>
<tr>
<td>PDA Arjen – Address</td>
<td>Thunderbird – Address Book</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>[Address].[Last Name], [Address Book].[Last Name]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Last Name], [Address Book].[Display Name]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Last Name], [Address Book].[Nick Name]</td>
<td></td>
</tr>
<tr>
<td>[Address].[First Name], [Address Book].[First Name]</td>
<td></td>
</tr>
<tr>
<td>[Address].[First Name], [Address Book].[Display Name]</td>
<td></td>
</tr>
<tr>
<td>[Address].[First Name], [Address Book].[Nick Name]</td>
<td></td>
</tr>
<tr>
<td>[Address].[First Name], [Address Book].[Screen Name]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Title], [Address Book].[Work Title]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Company], [Address Book].[Work Department]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Home Number], [Address Book].[Home Number]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Fax], [Address Book].[Fax Number]</td>
<td></td>
</tr>
<tr>
<td>[Address].[E-Mail], [Address Book].[Email]</td>
<td></td>
</tr>
<tr>
<td>[Address].[E-Mail], [Address Book].[Additional Email]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Address], [Address Book].[Home Address]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Address], [Address Book].[Work Address]</td>
<td></td>
</tr>
<tr>
<td>[Address].[City], [Address Book].[Home City]</td>
<td></td>
</tr>
<tr>
<td>[Address].[City], [Address Book].[Work City]</td>
<td></td>
</tr>
<tr>
<td>[Address].[State], [Address Book].[Home State]</td>
<td></td>
</tr>
<tr>
<td>[Address].[State], [Address Book].[Work State]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Zip code], [Address Book].[Home ZIP]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Zip code], [Address Book].[Work ZIP]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Country], [Address Book].[Home Country]</td>
<td></td>
</tr>
<tr>
<td>[Address].[Country], [Address Book].[Work Country]</td>
<td></td>
</tr>
</tbody>
</table>
Appendix D: Justification good mappings

This section includes the justification for choices which were made for disputable element matches.

**Combinations with Display name, Nickname, and Screen name fields**
In many cases or applications Display name, Nickname and Screen name fields have a default value, namely the value of Name, First name and/or Last Name. Combining them with Name, First Name and/or Last Name therefore make sense.

**Distinction between business and private info**
There are schemas that make distinction between private and business contact info, such as:
- a private and a business Phone number
- a private and a business Address
- a private and a business Country

There are also (smaller) schemas that at most contain a single element for each of the above bulleted cases.

Because a device with a small schema could be used for both private and business aims, the combination between a single schema element of the small schema is approved with its private and business equivalents in bigger schemas.