Topimatch
Schema matching using descriptions

Thesis submitted for the degree
of Master of Science at
the University of Twente

Bart Schapendonk

Enschede, July 20, 2007

Graduation commission
dr. ir. Maurice van Keulen (UT)
Dipl. Wirtsch.-Inf. Robin Aly (UT)
ir. Jens Willems (Topicus)
ir. Maarten Kok (Topicus)

Chair Databases
Department of Electrical Engineering,
Mathematics and Computer Science
University of Twente

topicus

University of Twente
Enschede - The Netherlands
Summary

The problem of matching schemas is a tough, older problem, which is still current. In this report Schema Matching is part of the translation process that is needed when an e-commerce application uses web-services.

Finture an ASP (Application Service Provider) based application, developed by Topicus (an innovative IT service provider) uses the HDN (Mortgage Data Network) web-services to communicate with other financial institutions. These HDN messages are in another format (syntactically and semantically) than used by Finture. A translation between these formats is required, schema matching is part of this process. Currently this is done manually, which is time consuming, there is a need to automated this process.

Web-services are usually well documented, having descriptions on the element-level of the schema. The rationale is to use these descriptions (auxiliary information) as similarity clues. This leads to the following hypothesis: “Element-level descriptions can be used as similarity clues by a matcher, to increase the quality of the matches”

To test this thesis, a baseline comparison between a standard schema matching system and a system including a matcher that uses descriptions must be done. A literature study into the current state of the art regarding schema matching concludes that promising systems exist, but they are not available for testing. Therefore a new minimal functional schema system is designed and implemented, which is used for the baseline comparison.

The new matcher that uses descriptions as similarity clues, is named the Description Matcher. This new matcher is based on the Vector Space Model from the Information Retrieval (IR) research field. This model uses the descriptions as queries and documents. The set of documents can be searched with a query. If a query (description linked to a source schema element) yields as a result a document (description linked to a target schema element), there is a similarity between the source and target scheme elements.

In the Vector Space Model, query and documents are expressed as vectors in a multi-dimensional space were each term has its own axis. Similarity is then expressed as the angle between the query and document vector, the smaller the angle the more similar they are. Another important aspect is Term Weighting which is used to assign a weight of relevance to a term. This to promote unique terms or to demote common terms (e. g., the, and, if, or).

This new matcher is evaluated using well known quality measures: Precision, Recall and Overall (a combined form of Precision and Recall). The results of the test schemas that were used show an increase in these measures, meaning the hypothesis is correct. The remaining question is, how significant this increase is? More research is needed to answer this question, but nevertheless the usage of descriptions as a similarity measure is promising.
Preface

This thesis is the written result of my final project at the University of Twente, conducted externally at Topicus Finance B.V. in Deventer. This final project is my last step to attain the degree of Master of Science.

I started this project in October 2006, I had a bit of a hard time writing the thesis, but it has been a very educative period nevertheless. It was also difficult to find the right balance between the research interests of the University of Twente, and the practical interest of Topicus.

I would like to thank the members of my graduation committee: Maurice van Keulen, Robin Aly, Jens Willems, Maarten Kok, who where of great help, by giving advise, feedback and hints, especially regarding the writing this thesis.

I also would like to thank my colleagues at Topicus: Martin Krans, Laurens Brouwer, Robin Zagers, Johan te Winkel, Sebastiaan van Dijk, Thijs Munsterman and others, for their feedback and help with reviewing.

Bart Schapendonk
Enschede, July 20, 2007
Contents

Summary iii

Preface v

1 Introduction 1

1.1 Motivation 1

1.1.1 Topicus 1

1.1.2 Finture case-study 1

1.1.3 Schema matching 2

1.1.4 Schema matching is hard 4

1.2 Problem statement 5

1.3 Research questions 5

1.4 Demarcation 6

1.5 Overview 6

2 Related work 7

2.1 Schema matching approaches 7

2.1.1 Schema-level matchers 8

2.1.2 Instance-level matchers 9

2.1.3 Auxiliary information 10

2.1.4 Combining matchers 10

2.2 Comparisons of known systems 10

2.2.1 CUPID 10

2.2.2 Similarity Flooding 11

2.2.3 Protoplasm 11

2.2.4 COMA and COMA++ 12

2.2.5 Harmony 12

2.2.6 LSD, iMAP and Corpus-based Matching 12

2.2.7 FlexiMatch and Map-IT 12

2.2.8 Conclusions 12

3 Topimatch 13

3.1 Goals 13

3.2 Architecture 13

3.3 Internal data representation 14

3.4 Matcher interface 15

3.5 User interface 16
4 Description Matcher

4.1 Rationale .......................................................... 19

4.2 Information retrieval ........................................ 19

4.2.1 Vector space model ........................................ 20

4.2.2 Term weighting ............................................. 21

4.3 Implementation .................................................. 22

5 Evaluation .......................................................... 23

5.1 Method ............................................................ 23

5.2 Quality Measure ................................................ 24

5.3 Experiment ....................................................... 25

5.4 Results ........................................................... 26

5.4.1 Precision ..................................................... 26

5.4.2 Recall ......................................................... 26

5.4.3 Overall ....................................................... 27

5.4.4 Conclusion ................................................... 27

6 Conclusions and Recommendations ............................ 29

6.1 Conclusions ...................................................... 29

6.2 Recommendations ............................................. 31

Bibliography .................................................................. 33

A Results evaluation .................................................... 37

B Listings ..................................................................... 39

B.1 Schemas ............................................................ 39

B.2 HDN Datacatalogus AX Message ............................. 42
1 Introduction

In this chapter the importance and problems of schema matching are motivated in scientific context as well as in the context of Topicus. This is followed by the problem statement, the research questions and the approach that is taken to come to a solution.

1.1 Motivation

1.1.1 Topicus

Topicus is an innovative IT service provider located in Deventer, the Netherlands. The main focuses of Topicus are the financial, health-care and educational sectors. They specialize in supply chain management, application service providing and process management in these sectors.

Topicus develops, maintains and deploys ASP, short for application service provider, solutions in those sectors. ASP is a business model where services are provided to a customer over the internet.

Figure 1.1 shows the organigram of Topicus. Topicus uses a cell based structure. The idea behind this is that the core cell focuses on innovation using a small set of resources. If there is a success this core cell is converted into a spinoff. The outer cells are partners which provide the gateway to the market for the spinoffs. For example Topicus Finance develops ASP applications for the financial sector and Finture and Finan are partners which provide the domain specific expertise and an entrance to the market.

1.1.2 Finture case-study

Finture is an ASP application developed by Topicus Finance for mortgage brokers. It provides the brokers with common administrative services and gives them access to a large group of underwriters. Normally a broker would use either no automation, a custom application or an application provided by the underwriter. For the broker Finture has the following advantages;

- Since Finture is an ASP application, the broker has no maintenance costs for hosting and maintaining the application in house, except the costs involved with maintaining his, or her own internet connection and workstations.

- Since Finture gives the broker access to more than one underwriter, this gives the broker a more neutral position.
• For the broker there is no monthly fee to use the application. Finture’s revenue model is based upon a small percentage taken per successful transaction, e.g., sold mortgages.

The schema matching problem in Finture comes around in two forms, first the classical data integration task of importing a legacy database and second, communication with other systems. To give a concrete example of communication with other systems, Finture uses the HDN [16] protocol (HDN is an abbreviation for Hypotheken Data Netwerk which translates into Mortgage Data Network), which is based on web-service technology. HDN works asynchronously in the sense that Finture sends a request (which is the digital equivalent of a handwritten form) and then receives a response later in time. These requests adhere to a different model than the model that is used by Finture, therefore a translation is needed. Schema matching is part of this translation step.

### 1.1.3 Schema matching

To motivate the relevance of schema matching, the question “What is schema matching?” must be answered first. A reasonable answer to those questions is given by Do [8] in the form of the following definitions:

A schema is a structure of meta-data describing how data, i.e., instances, can be stored, accessed, and interpreted by users and applications.

Schema matching is the task of identifying semantic correspondences between these meta-data structures.

Ramez A. Elmasri et al. [26] distinguish three schema architectures:
**1.1 Motivation**

**Internal schema** Describes the physical storage structure of the database.

**Conceptual schema** Describes the semantics of the physical schema.

**View level** Describes a user specific view on the conceptual schema.

The schemas used in this project fall in the conceptual schema category.

Many applications, such as data warehousing, mediating between web-sites and data mining, need to integrate data from multiple sources. This integration process is called data integration and aims to provide a uniform, consistent view of the data; this view is also referred to as the global schema. The integration of a new data source into an existing data source or global schema can be performed in two steps, a matching and a data transformation step. The first step compares the source schema to the target schema to discover similarities and dissimilarities. The second step generates the necessary transformations in the form of, e.g., queries or code to transform the instances from the source schema into the target schema.

Some examples of domains where schema matching is used are:

**Schema Integration** Usually schemas are developed independently. Therefore, they often have a different structure and terminology. For example the schemas from different domains, like health-care and finance. The problem here is to integrate both schemas into one global view. Most of the work on schema matching focuses on this type of problem, going back to the early 1980’s.

**Data warehouses** This is a variation of the schema integration problem. A data warehouse is a combination of aggregated data from multiple sources.

**E-Commerce** This domain is relatively new. In e-commerce, trading partners frequently exchange messages describing business transactions. These messages could be in different formats, use a different schema, or differ from the representation used internally by the systems. Hence, some form of schema matching is needed.

Figure 1.2 illustrates the schema matching step of data integration. There are two schemas, Client and Customer, as shown figure 1.2a. Client is matched against Customer, the result of this is shown in figure 1.2b. The result shows that the element Id from the Client schema semantically corresponds to the CID element of the Customer schema. This can be calculated by using a string similarity algorithm, e.g., the Levenshtein distance [13], which calculates the number of insertions, deletions and substitutions needed to change Id into CID as a similarity measure. Less trivial are similarities like:

- First, Last is similar to Name.
- Home is similar to Address.
- Phone is an example of a mismatch, an element that occurs only in one schema.
1 Introduction

- Client
  - Id
  - First
  - Last
  - Home
  - Phone

- Customer
  - Id
  - Name
  - Address
  - Phone

(a) Two schemas Client and Customer

<table>
<thead>
<tr>
<th>Id</th>
<th>First</th>
<th>Last</th>
<th>Home</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Doe</td>
<td>Example 1</td>
<td>1234</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(b) Matches between Client and Customer

<table>
<thead>
<tr>
<th>CID</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Doe</td>
<td>Example str. 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(c) Instance of Client

(d) Instance of Customer

Figure 1.2: A schema matching example

To discover these similarities, more elaborate algorithms are needed which use more information, i.e., structure, instance data or auxiliary information (e.g., documentation, manuals, dictionaries) than just the names of the elements.

1.1.4 Schema matching is hard

This simple example shows that matching schemas is relatively hard. Even though an extensive amount of research on the problem of schema matching has been done, a real solution is not in sight. It is very hard to discover the semantic correspondences between elements, because this requires a thorough analysis and understanding of the schemas. Since schemas are usually designed by different people in different application domains, this implies that to fully understand the schemas, one must understand the thought processes and the domain knowledge of the designers. This is, if not impossible, very difficult to achieve. Most systems use clues in the schemas and instances to deduce semantic correspondences. These clues, no matter how helpful they sometimes are, could also mislead the system, for example [8]:

- Two elements have the same name, so it is feasible that they have a semantic correspondence. But this is not always the case, e.g., the elements have the name Name, but is this the name of a product, person or something else that can have a name?

- Element names might be encoded or abbreviated, so that only their creators understand their semantic meaning.

- Elements may be modeled in different ways, e.g., from the example in figure 1.2, First and Last in Client are modeled as Name in Customer

- The same information might be encoded in different ways, e.g., gender information (‘Male’ versus ‘M’) and monetary values can be with tax or without tax or represented using different currencies.
• Instances might contain erroneous or corrupted information, e.g., misspelling, missing values or duplicates.

Most of the current research is focused on algorithms trying to solve some of these or similar problems. They are fragile and often need manual tuning, such as setting thresholds, providing auxiliary sources or training using examples. In other words, they are optimized to solve a specific type of schema matching problem in a specific domain.

Bernstein et al. [3] propose an “Industrial-Strength Schema Matching” system which is inspired by the approach taken by the COMA [9] system. Fragility is seen as inherent to the problem. To overcome this, a system is needed that can exploit the best algorithms and is customizable by a user. Customizable in such a way, that the user can select the best algorithms or strategies for a specific schema matching problem and the system can be extended in an easy way by adding new technologies and algorithms. Another important aspect mentioned in the iMAP [7] system is the ability for the system to explain how and why a match was generated.

1.2 Problem statement

Rahm and Bernstein [25] state, that in the e-commerce application domain, schema matching is a relevant problem, e.g., messages between systems need to be translated. Finture fits in the e-commerce application domain: it connects with other services, e.g., HDN. This schema matching task is currently solved manually. So there is a need to optimize this, by using an automatic schema matching system. Since HDN contains element-level descriptions in the schema, these might be used as clues to match schemas.

Hypothesis Element-level descriptions can be used as similarity clues by a matcher, to increase the quality of the matches.

To evaluate this hypothesis, a baseline is needed, to which this new matcher can be compared. Therefore a minimal working schema matching system must be designed and developed.

1.3 Research questions

To test the hypothesis the following research questions have to be answered.

• What is the current state of the art regarding schema matching?
  – Which approaches exist?
  – Which systems exist and how do they compare?

• Schema matching for Topicus.
- What are the minimum requirements for a working schema matching system?
- Which approach or approaches are best suited for this system?

• The description matcher.
- Are there any systems or approaches that use or have thought about using descriptions as a similarity measure?
- How to use descriptions as a similarity measure?

• How to evaluate the system?

1.4 Demarcation

The schema matching research field is very broad and an extensive amount of research has been done. Therefore we limit the scope of this project to the following:

• The scope of the project is limited to the e-commerce application domain of schema matching.

• The scope of the matches is limited to simple matches, i.e., 1 : 1 cardinality.

• The system only considers schema-level matches, i.e., no instance or structural information is used.

• The system does not provide any input or output conversions, e.g., SQL schema input module, or the creation of schema mappings.

1.5 Overview

Chapter 2 focuses on related work (the current state of the art regarding schema matching) and provides the necessary basis for the following chapters.

Chapter 3 motivates the design decisions and the design itself of the Topimatch schema matching system. This system is the bare minimum schema matching system which will be used as a test platform to evaluate the description matcher.

Chapter 4 introduces a new kind of matcher, called description matcher. This matcher uses element-level descriptions in natural language to further improve the quality of the matches.

Chapter 5 motivates the methods used for evaluation, the experiment itself and the results.

Chapter 6 gives the conclusions and recommendation.
2 Related work

In this chapter, an overview of the current state of the art regarding the field of schema matching is given. In the first section, approaches are classified. This is followed by a comparison of existing systems in the last section.

2.1 Schema matching approaches

Rahm and Bernstein \cite{25} give a classification for schema matching approaches, which is shown in figure 2.1. They make two important distinctions. First, there is the realization of individual matchers, each of which computes a mapping based on a single matching criterion. Second, there is the combination of individual matchers, either by using multiple matching criteria within a matcher (“hybrid matcher”) or by the aggregation of the different results produced by individual matchers (“composite matcher”). For the individual matchers they consider the following criteria:

**Instance versus schema** matching approaches can consider instance data (i.e., contents) or only schema-level information.

**Element versus structure** matching can be performed on individual schema-elements such as attributes, or on more complex schema structures which contain these individual schema-elements.

**Language versus constraint** a matcher can use a linguistic based approach such as string similarity or a constraint based approach.

![Classification of schema matching approaches](image)

Figure 2.1: Classification of schema matching approaches
Auxiliary information besides instance and schema information a matcher can also use so-called auxiliary information, such as dictionaries, global schemas, previous match results, synonym lists and user input.

2.1.1 Schema-level matchers

Schema-level matchers only consider schema information. Examples of information that schemas typically contain are; element names, data types (string, integer), constraints (cardinality, uniqueness), relations and structural information.

Element versus structure

As shown in figure 2.1 there are two sub approaches for schema-level matching, element-level and structure-level matching. The difference between those two is best explained using an example. In table 2.1 two schema fragments are shown, S1 and S2. Both contain two tables, Address/AccountOwner and CustomerAddress/Customer. If element-level matching is used, the matcher only considers elements and no structural information, e.g., Address.Street=CustomerAddress.Street. A structural-level matcher considers structure and might discover that Customer.CAddress element in S2 matches the Address table in S1. In this case Customer.CAddress is a combination of the Street, City, State and ZIP elements from the Address table.

Linguistic approaches

Language based approaches are commonly used by schema-level matchers. They use text, words, sentences to find similarities between schema elements. Rahm and Bernstein[25] discuss two types of schema-level matchers, the name matcher and a description matcher.

The name matcher takes the element names of schema elements and determines their similarity. An example of a string similarity algorithm is the Levenshtein Distance[19],

<table>
<thead>
<tr>
<th>S1 elements</th>
<th>S2 elements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Address</strong></td>
<td><strong>CustomerAddress</strong></td>
</tr>
<tr>
<td>Street</td>
<td>Street</td>
</tr>
<tr>
<td>City</td>
<td>City</td>
</tr>
<tr>
<td>State</td>
<td>USState</td>
</tr>
<tr>
<td>ZIP</td>
<td>PostalCode</td>
</tr>
<tr>
<td><strong>AccountOwner</strong></td>
<td><strong>Customer</strong></td>
</tr>
<tr>
<td>Name</td>
<td>CName</td>
</tr>
<tr>
<td>Address</td>
<td>CAddress</td>
</tr>
<tr>
<td>BirthDate</td>
<td>CPhone</td>
</tr>
<tr>
<td>TaxExempt</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Example schema fragments
2.1 Schema matching approaches

<table>
<thead>
<tr>
<th>Cardinality</th>
<th>S1 element</th>
<th>S2 element</th>
<th>Mapping expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 : 1</td>
<td>Price</td>
<td>Cost</td>
<td>(Price = Cost)</td>
</tr>
<tr>
<td>(n : 1)</td>
<td>FirstName,</td>
<td>Name</td>
<td>(\text{Concat}(\text{FirstName}, \text{LastName}) = \text{Name})</td>
</tr>
<tr>
<td>1 : (n)</td>
<td>Name</td>
<td>FirstName,</td>
<td>(\text{Split}(\text{Name}) = \text{FirstName}, \text{LastName})</td>
</tr>
</tbody>
</table>

Table 2.2: Match cardinalities

also known as the edit distance. The idea is to transform a string from one schema into a string of the other schema using copy, delete, insert and substitute operations. All operations have cost 1 except copy which has cost 0. So for example the costs to transform \(\text{Cat}\) into \(\text{Dog}\) is 3, since three characters are substituted.

The description matcher looks at element descriptions in natural language and uses linguistic analysis as a similarity measure, for example keyword extraction. In chapter 4 a new matcher is introduced, that uses these descriptions as similarity clues.

**Constraint**

Schema’s often contain constraints. Constraints define, i.e., data types, value ranges, uniqueness, optionality, relationships. If both the source and target schema contain constraints, these can be used for either candidate elimination (e.g., a string containing alphanumerical values cannot be converted into an integer), or to boost certain matches (e.g., an integer to integer match is more probable to be correct than an integer to string match).

**Match cardinality**

One or more elements in the source schema may be matches with one or more elements in the target schema. This is called local cardinality and comes in the form of \(1 : 1\), \(1 : n\) or \(n : 1\) as shown in table 2.2. For example a cardinality of \(1 : 1\) means, that one element of the source schema is matched against one element of the target schema. And in the case of \(1 : n\) that one element of the source schema is matched against zero or more elements of the target schema.

2.1.2 Instance-level matchers

Instance-level matchers use the instance data, i.e., the data itself, as a reference to discover clues for similarity between elements. Instance-level matching is very powerful, especially if the useful schema information is limited. Most of the approaches used for schema-level matching can also be used for instance-level matching.

For example, by extracting keywords from the instance data, it might be discovered that the column \(\text{Name}\) from the source contains a lot of rows containing the string \(\text{John Doe}\) and that there is a column in the target, that also contains a lot of rows with the string \(\text{John Doe}\). Hence there might be a similarity between those columns. Another
good example is the recognition of patterns like zip codes (e.g., Dutch zip codes consist of a sequence of four digits and two letters.)

A more advanced matcher could harness the power of machine learning. In this case, the system must be trained first. Training is done with sets of examples both containing correct and incorrect examples.

2.1.3 Auxiliary information

Auxiliary information, is all useful information that is no schema information or instance data. The idea is that all useful information can be used as clues to further help improve the overall quality of the matches. For example, descriptions of elements in natural language; if the similarity between those descriptions can be calculated this can be used as a clue for the similarity between the elements.

2.1.4 Combining matchers

In the sections above, a number of matchers have been discussed. Depending on the type of problem and the available information, i.e., schema, instance, auxiliary, some matchers are more effective than others. Since there is no perfect matcher, matchers are usually combined. There are two strategies to combine matchers; a hybrid approach and a composite approach, also known as a multi-strategy matcher.

The hybrid approach directly combines several matchers or matching approaches. The main advantage of this approach is that it is faster, because matchers are not executed separately. The composite approach on the other hand executes each matcher separately and combines their results in the end. The biggest advantage of this approach is flexibility and ease of development. Hybrid matchers are usually hard-wired combination of particular matchers for a particular types of problems. Using a composite matcher, individual matchers can easily be added to or removed from the aggregation.

2.2 Comparisons of known systems

There are numerous surveys on schema matching approaches in literature \([8, 10, 23, 25]\). This section gives a short summary of the most influential of those systems using the criteria mentioned in the previous section. A quick overview is shown in table 2.3.

2.2.1 CUPID

CUPID \([17]\), by Microsoft Research, is based on a hybrid match approach combining a name and a structural matcher. It is intended to be a generic approach which has been applied to XML and relational schemas. Trees are used for an internal representation of the schemas. The name matcher exploits auxiliary sources for synonyms and abbreviations to obtain the linguistic similarity between element names.

The hybrid matching algorithm has three phases: the first phase, does linguistic element-level matching using element names and data types (making cupid hybrid).
2.2 Comparisons of known systems

<table>
<thead>
<tr>
<th></th>
<th>CUPID</th>
<th>Similarity Flooding</th>
<th>Protoplasm</th>
<th>COMA / COMA++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Types</td>
<td>XSD, SQL</td>
<td>SQL, RDF</td>
<td>XSD, SQL</td>
<td>XSD, SQL, OWL</td>
</tr>
<tr>
<td>Internal Representation</td>
<td>Tree</td>
<td>Graph</td>
<td>Graph</td>
<td>Directed Graph</td>
</tr>
<tr>
<td>Element Criteria</td>
<td>Name, Type</td>
<td>Name</td>
<td>Name</td>
<td>Name, Comment, Type</td>
</tr>
<tr>
<td>Structural Criteria</td>
<td>Containment, Referential</td>
<td>RDF predicate links</td>
<td>Containment, Referential</td>
<td></td>
</tr>
<tr>
<td>Instance Criteria</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auxiliary Criteria</td>
<td>Synonyms</td>
<td></td>
<td></td>
<td>Synonyms, Reuse</td>
</tr>
<tr>
<td>Combination of Matchers</td>
<td>Hybrid</td>
<td>Hybrid</td>
<td>Composite</td>
<td>Composite</td>
</tr>
<tr>
<td>Internal Representation</td>
<td>XSD</td>
<td>DTD, SQL</td>
<td>XML</td>
<td></td>
</tr>
<tr>
<td>Element Criteria</td>
<td>Name</td>
<td>Name</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Structural Criteria</td>
<td>Nesting</td>
<td></td>
<td>Values</td>
<td></td>
</tr>
<tr>
<td>Instance Criteria</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Auxiliary Criteria</td>
<td>Stopword, Stemmer</td>
<td>Synonyms, Constraints</td>
<td>Reuse</td>
<td></td>
</tr>
<tr>
<td>Combination of Matchers</td>
<td>Composite</td>
<td>Composite</td>
<td>Composite</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Comparison of existing approaches

The second phase converts the schema into a tree structure and performs a bottom-up structure matching. The third phase, uses a weighted mean on the mappings.

2.2.2 Similarity Flooding

Sergey Melnik et al. [28] [29] present a graph matching algorithm called Similarity Flooding (SF) and explore its usability for schema matching. The Similarity Flooding match algorithm works as follows.

First the schemas are converted into a labeled graph structure. Then a string similarity matcher is used on the element names to produce an initial mapping. This initial mapping is than used by the structural matcher (SF) which uses clues in the structure of the graph to promote certain results. Finally various filters are used to select the relevant match results.

2.2.3 Protoplasm

Bernstein et al. [3] present an approach that aims at an industrial strength schema matching solution, by providing a flexible and highly customizable platform for combining different match algorithms. One of the biggest problems with most existing systems, is that they focus on a specific problem, and therefore only offer a specialized solution for that problem.

Protoplasm aims to solve this, by providing a high-level architecture for a composite matcher approach. It provides a foundation in the form of a internal data representation, result storage (similarity matrix), and result aggregations. Now only a collection of matchers must be provided that build upon this foundation.

The only thing left to the user, is to select the subset of matchers that provide the best strategy to solve some problem.
2.2.4 COMA and COMA++

The COMA++ [2, 8, 9] (successor of COMA) prototype, which is still under active development, uses the composite matching approach to combine different matchers. It implements a significant amount of matching strategies and provides matchers that use schema, element and auxiliary information.

2.2.5 Harmony

Peter Mork et al. [24] present an approach to integrate schema matching systems into one workbench. In an earlier survey [18] a breakdown of data integration tasks is presented, the survey shows how these tasks are distributed and the amount of effort that is needed. Based upon the breakdown of data integration tasks it is concluded that some tasks are already implemented in commercial tools. Other tasks, i.e., schema matching are not. They propose a common workbench, that interconnects existing tooling and still to be developed tooling, and so combine their strengths to provide one integral solution.

2.2.6 LSD, iMAP and Corpus-based Matching

[7, 11, 20] are all systems, which succeeded each other and focus on the usage of instance data for matching clues. A composite matching approach is used. It has a global domain specific schema against witch the source and target schemas are matches. Machine learning is used for both the individual matchers and the automatic combination of the match results. iMAP also is capable to explain how a match was formed.

2.2.7 FlexiMatch and Map-IT

FlexiMatch [6] and Map-IT [4] which builds upon the Fleximatch framework provide a multi-strategy schema matching approach. Their matching approach is based heavily on machine learning using instance data. Fleximatch provides a basic set of learners which can be trained and have the ability to be refined with relevance feedback. The match results are stored in a central repository for later reuse. Map-IT provides an automated evaluation framework for Fleximatch, which is used to fine-tune the matchers.

2.2.8 Conclusions

Besides the systems listed in this comparison, a more extensive comparison of existing systems was done by Do [8].

None of the above systems are available, because their license excludes commercial usage. So a new system must be developed to function as a basis to evaluate the thesis. This new system must use the composite matching approach to combine matchers, this approach provides the ability to add and remove matchers. This contrasts with hybrid matching approach where all the matcher are integrated into a single matcher.
3 Topimatch

This chapter presents the design and motivates the design decisions of the Topimatch schema matching system.

3.1 Goals

The main goal of the Topimatch schema matching system is to function as a basis for evaluating the description matcher, which is described in chapter 4. Therefore it should provide a minimal functioning schema matching system.

This also implies that a multi-strategy or composite approach has to be used since this gives us the required flexibility to arbitrarily add or remove a certain matcher. Therefore we can evaluate the system with and without the description matcher.

3.2 Architecture

Topimatch follows the architecture of a composite multi strategy approach [8,25]. The overall architecture is shown in figure 3.1. The system takes two schemas as input and produces the derived matches as output. The matches are derived by individual matchers. The matches generate match candidates which are then evaluated. The evaluation results are stored in a similarity cube, which can then be aggregated using functions into a single result. There are three distinct components:

Matcher

The first component is the Matcher. The Matcher consists of four sub-components:

Pre-processor The pre-processor prepares the input for the candidate generator, e.g., splitting names like FirstName into First and Name, converting all elements to lowercase.

Candidate generator The candidate generator generates candidates for evaluation, i.e., element pairs from both schemas, a cross product from both schemas.

Candidate evaluator The candidate evaluator evaluates the candidates generated by the candidate generator, by using a similarity measure. The result is a numerical score which is stored in a matrix, which becomes one slice in the similarity matrix.
Figure 3.1: Architecture of the Topimatch schema matching system

**Post-processor** The post-processor prepares the output to be stored in the Similarity Cube, e. g., it filters out values that exceed the minimum or maximum thresholds. This is then passed on to the similarity cube or displayed on screen.

**Similarity Cube**

The second component is the Similarity Cube. This is the data-structure which stores all the individual results from the Matcher components. The result provided by a Matcher is a two-dimensional matrix containing the similarity values. These are stacked to form a cube, hence the name Similarity Cube.

**Aggregation**

The third component is Aggregation. This component takes the similarity values stored in the Similarity Cube and aggregates the values of each candidate into one single matrix containing the similarity values. The following aggregation functions are available at the time of writing: average, weighted average, normalized weighted average, minimum and maximum. New aggregation functions can easily be added.

In addition to these three components, two more components (input and output) could be identified. But these are not relevant for a minimal functional system, since they provide support to the schema matching system.

### 3.3 Internal data representation

The internal data representation of Topimatch is shown in figure 3.2. All information used or generated by the system is stored in a Workspace. Besides the source and target schema, this Workspace also contains the results in the form of a similarity cube and all auxiliary information such as thesauri and synonym lists. The internal data representation of the schemas is a Directed Acyclic Graph (DAG). In Topimatch, a schema is called a Model and a Model has a collection of Entities (vertices in the graph).
3.4 Matcher interface

The matcher interface provides the Topimatch system with an infrastructure for plug-gable matchers, which is one of the requirements. The interface, shown in figure 3.3, only has one function, `match()`, which should do the pre-processing, candidate generation, candidate evaluation and postprocessing.

The result is a matrix which will become part of the similarity cube. The matcher interface is implemented by the AbstractMatcher and provides basic functionality. New matchers must be derived from AbstractMatcher.

Figure 3.3 gives an overview of the matchers in the system. There are three concrete linguistic matchers that are based upon string similarity: Levenshtein [19, 27], JaroWinkler [27, 30] and SmithWatermanGotoh [21, 27]. Levenshtein, is also known as the edit distance (explained in section 2.1.1). The other two linguistic matchers are variation on this idea [27].

Next to the description matcher, which is elaborated in chapter 4, there are two more concrete matchers: the UserMatcher and the AggregationMatcher. These two matchers are special, since they do not generate candidates or evaluate candidates. The UserMatcher enables the user to make match decisions: Match, No Match, Unknown. These matches, except Unknown, override all other matchers in the aggregation process. The aggregation matcher uses a function, e.g., Minimum, Maximum, Average, Weighted Average, to combine the results of the individual matchers into a single result.
3.5 User interface

The graphical user interface is used to quickly visualize schema matches and to intuitively experiment with parameters for the matchers and aggregation.

Figure 3.4 shows the Matrix View of the user interface. In this view the elements of the source schema are the rows and the elements of the target schema are the columns. Since all matchers use a match generator that produces a cross product of the elements, the result is a matrix, hence the name Matrix View.

Similarity is indicated with a numerical value ranging between 0 (colored red) for minimum similarity and 1 (colored green) for maximum similarity in the cells of the matrix.

Another view is the Tree View shown in figure 3.5. This view shows the structure of the schema in the form of a tree of element names. Matches are shown using lines between candidates with a label with the numerical similarity value. Again the lines and labels are color coded using red for the most unlikely candidates and green for the most likely.

On the left side of the matrix- and tree-view, there is a list containing all the matchers and two sliders. A Matcher can be added to the aggregation results by checking the checkboxes left to the matcher name. Using the weight slider, the weight of the matcher in the aggregated result can be set, in case weighted aggregation is used. The other slider is used to set a minimum and maximum threshold. This affects the matches that are shown on the screen and the aggregation functions.
3.5 User interface

Figure 3.4: The Topimatch schema matcher *Matrix View* mode

Figure 3.5: The Topimatch schema matcher *Tree View* mode
4 Description Matcher

In this chapter a new matcher is introduced, the *Description Matcher*, which bases similarity between elements on the element-level descriptions to further increase the quality of the match result.

4.1 Rationale

As already stated in the first chapter, there are more sources containing information that can be used to improve the match result. Next to the information contained in the schema and the information contained in the data instances, there are more sources. These could be any kind of documentation, e.g., design documentation, dictionaries, specifications.

In the case of Topicus Finance, the focus of schema matching lies in two application domains: The legacy conversion of databases and the conversion of messages used in web-services (e-commerce). Usually in the latter case, those web-services are well documented. For example the HDN specification has a document called “Datacatalogus AX message” (see appendix B.2). This document describes the AX message, which is a request for a quotation on a mortgage, down to the level of an individual description of the fields.

Rahm and Bernstein [25] hinted that descriptions can be used as a similarity measure. They suggest that it can be as simple as keyword extraction or as complex as using natural language understanding techniques. The new matcher that is introduced in this chapter is based upon ideas from the Information Retrieval (IR) field.

4.2 Information retrieval

For thousands of years people have realized the importance of archiving and finding information. With inventions like paper and the printing press, the need to store and retrieve information has become even more important. With the advent of computers, which made it possible to store even larger amounts of information, it became a necessity to support the process of searching for useful information.

The information retrieval (IR) research field was born to facilitate in this need [11]. Over the last forty years, the field has matured considerably. Today many people use information retrieval systems on a daily bases, for example search engines such as Google Search or Yahoo! Search.

There are different models for retrieving information. Early information retrieval systems were boolean systems. These systems measure similarity according to logical
propositions using operations such as AND, OR and NOT, and the answers are TRUE if there is there is a similarity or FALSE if there is none.

This system is still widely used in relational databases. Major drawbacks of the boolean system are that there is no ranking in the results (either there is a match or there is not), and it is very hard for the user to formulate a good search request.

This inability of the boolean model to rank documents is solved by ranked retrieval models. These ranked models give an estimation of the usefulness of a document by giving it a numeric score, usually between zero and one. Now the result can be ranked according to this score. One of the best known and understood models of ranked retrieval is the vector space model. Therefore this model is chosen to be the basis for the description matcher.

4.2.1 Vector space model

In the vector space model, a document is represented by a vector of terms. A term is typically a word. Every word in the vocabulary (the distinct words in the entire collection of documents) is an independent dimension in the vector space model.

If a term belongs to a document, a non-zero score or weight is assigned to that document’s vector along the dimension corresponding to that term. Similarity is measured as the cosine of the angle between the vector representing the query and the vector representing the document.

If the documents are identical, the cosine of the angle is 1. If the vectors are orthogonal, the cosine of the angle is 0. As an alternative, the inner product between two vectors is often used as the similarity measure.

For example figure 4.1 shows a vector space with three dimensions (Name, Address, Phone).
4.2 Information retrieval

There are two vectors: vector $\vec{q}$ representing the query containing the terms Address and Phone and the vector representing the document $\vec{d}$ containing all the terms.

The similarity between query and document equals the cosine of the angle between vectors $\vec{q}$ and $\vec{d}$. In case the angle is $0^\circ$, i.e., the vectors overlap, and the cosines is 1 meaning the query and the document are equal. An alternative is, to use the inner-product [11] (or dot-product) between the two vectors $\vec{q}$ and $\vec{d}$. This is shown in the upper half of equation [4.1] the lower half is used for normalization [12] [14]. Normalization is needed since not all documents have the same length, longer documents tend to have more terms in common with the query and thus score higher.

$$\text{score}(\vec{d}, \vec{q}) = \frac{\sum_{k=1}^{m} d_k \cdot q_k}{\sqrt{\sum_{k=1}^{m} (d_k)^2} \cdot \sqrt{\sum_{k=1}^{m} (q_k)^2}} \quad (4.1)$$

### 4.2.2 Term weighting

All the ranked retrieval models need term weighting. Term weighting algorithms assign a numerical score to a term which indicates how relevant it is. The development of term weighting algorithms is foremost based upon experiments and experience of researches. In fact there have been thousands of experiments with term weighting algorithms [14]. Most term weighting schemes consider two main factors.

First the term frequency: $(t_f)$, the number of occurrences of a term in a document. This is based on the intuition that if a term occurs more frequently in a document, this term is more relevant in relation to the document.

The second is the document frequency: $(d_f)$, the number of documents containing the term. This is based on the intuition that terms that occur often in the entire collection of documents do not have a high relevance value. Usually the inverse document frequency $(idf)$ is used. This is the $\log(N/d_f)$ where $N$ is the number of documents in the corpus. $d_k$ and $q_k$ are the weights assigned to a query or document term.

Gerard Salton et al. [12] experimented with weighting algorithms. They suggested to combine the term frequency with the inverse document frequency. This introduced the so called $tf.idf$ weighting schema that is shown in equation [4.2] Most modern weighting schemes are derived from the $tf.idf$ scheme.

$$d_k = q_k = t_f \cdot \log \frac{N}{d_f} \quad (4.2)$$

An alternative term weighting scheme, called lnc.1tc, is shown in equation [4.3] This scheme has a logarithmic $tf$ component and it was discovered [5] that this scheme outperformed that of a linear $tf$ component.

$$d_k = 1 + \log (t_f)$$

$$q_k = (1 + \log (t_f)) \cdot \log \frac{N+1}{d_f} \quad (4.3)$$
4.3 Implementation

The implementation of the description matcher is straightforward. The `match()` operation first fills the indexes shown in listing 4.1.

```java
private Map<String, Map<String, Double>> termFrequency = new HashMap<String, Map<String, Double>>();
private Map<String, Vector<Integer>> invertedIndex = new HashMap<String, Vector<Integer>>();
```

Listing 4.1: DescriptionMatcher, indexes (Java code)

The `termFrequency` is a map data structure that uses the name of a document (in the case of schema-level descriptions the name of the element is used) as a key and to lookup another map. This map has as key the term, and as value the frequency of this term in the document.

The other index is the `invertedIndex`, this index is a map with a term as key, the value is a list of the documents that contain this term.

Listing 4.2 shows the pseudo algorithm of the description matcher. It consists of three nested loops. The outer loop iterates over all element descriptions from the source schema, these descriptions are used as a query.

```java
for sourceEntity in source.entities { // outer loop
    query = sourceEntity.description
    score = 0
    for targetEntity in target.entities { // middle loop
        document = targetEntity.description
        for term in tokenizeTerms(query) { // inner loop
            termFrequency = termFrequency(document, term)
            documentFrequency = invertedIndex(term).size
            numberOfDocuments = target.size
            documentVector = 1 + log(termFrequency)
            queryVector = (1 + log(termFrequency)) * log((numberOfDocuments + 1) / documentFrequency)
            score += documentVector * queryVector
        }
        normalizeScore
    }
}
```

Listing 4.2: DescriptionMatcher, pseudo algorithm

The middle loop iterates over all the element descriptions of the target schema, which are used as the collection of documents that is being searched in.

The inner loop gets the `termFrequency`, `documentFrequency` and `numberOfDocuments`, which are precalculated. They are then used to calculate the `queryVector` and the `documentVector`, which are used to produce a part of the total score.

At the end of the middle loop the score is normalized, to compensate for different document lengths.
5 Evaluation

In this chapter the performance of the Topimatch schema matching system, and the description matcher in particular, are evaluated. First the method of evaluation and the quality measures are motivated. This is followed by the setup of the experiment and the results.

5.1 Method

Hong Hai Do et al. [15] give a theoretical basis to compare evaluations of schema matching approaches. Their intention is to provide criteria so that future schema matching evaluations, can be documented better, the results are more reproducible, and to standardize the method for comparing different systems. To achieve this, four criteria are considered:

**Input** Which kind of input information is available (e.g., schema information, instance information, auxiliary information)? The rationale here is that if the input is more expressive then it is likely that the system will be more effective. But a more expressive input may require more effort, e.g., in the case that information needs to be manually extracted from a design document.

**Output** What information is included in the match result (mappings)? If less information is provided the probability of errors occurring is lower, but the effort needed for post-processing might be higher.

**Quality Measure** Which metrics were chosen to quantify the accuracy and completeness of the match result? To compare different evaluations, an understanding of these measures is needed.

**Effort** What was the amount of manual effort required? Which kind of effort, pre-match effort (training learner, converting the input, preparing auxiliary information), post-match effort (correction and improvement of the match result).

As only one system is evaluated in our case, the Input, Output and Effort criteria are less important since they are virtually the same (the same input was used and the same format of output was generated, there was a minimal amount of effort needed to add the auxiliary information to the input). The focus is on the Quality Measure criteria, since the interest lies in what the contribution of the description matcher is in terms of accuracy and completeness.
5.2 Quality Measure

A baseline for the quality measure, is provided by a manually solved match task. Comparing the automatically derived matches with the human matches results in the sets shown in figure 5.1. These sets can be used to define a quality measure for schema matching.

There are four sets: Set $A$ (the false negatives are the matches that were needed but not found), $B$ (the true positives are the matches that are human matches and were found), $C$ (the false positives are the incorrect matches that were found) and $D$ (the true negatives are the matches that are not human matches and were not found).

There are two common metrics based on these sets, which originate from the information retrieval field. These metrics are the Precision and Recall. The Precision, shown in equation (5.1) is the ratio of correct (as determined by a human) matches among the automatically derived matches.

$$Precision = \frac{|B|}{|B| + |C|}$$ (5.1)

The Recall, shown in equation (5.2) is the ratio of the found real matches and the total real matches. In the ideal case, the derived result equals the real result, so there are no false negatives and no false positives. In this best case scenario, the Precision and Recall, both have the value 1. In reality, the relation between Precision and Recall is inverse, e.g., an increase in Precision results in a decrease in Recall. Precision and recall must be considered as a combined measure and not as separate ones, because it is, e.g., quite easy to maximize recall (by returning the cross product of the two input schemas, i.e., the complete set of all possible results), or gain a high precision at the expense of a very poor recall (by returning a very small result).

$$Recall = \frac{|B|}{|A| + |B|}$$ (5.2)

It is common to combine both measures into a single measure. A number of these combined measures are discussed by Hong Hai Do et al. [15], including the Overall measure (initially introduced by Sergey Melnik et al. [29] as the Accuracy measure,
Table 5.1: List of source and target schemas

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1 Klant</td>
<td>PartijNAW</td>
</tr>
<tr>
<td>e2 Klant</td>
<td>Customer</td>
</tr>
<tr>
<td>e3 Klant</td>
<td>Thunderbird Address Book</td>
</tr>
<tr>
<td>e4 Customer</td>
<td>Thunderbird Address Book</td>
</tr>
</tbody>
</table>

Table 5.2: List of matchers

<table>
<thead>
<tr>
<th>Matcher(s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>Smith-Waterman-Gotoh</td>
</tr>
<tr>
<td>m2</td>
<td>Jaro Winkler</td>
</tr>
<tr>
<td>m3</td>
<td>Levenshtein</td>
</tr>
<tr>
<td>dm</td>
<td>Description Matcher</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>Aggregation of m1, m2 and m3</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>Aggregation of m1, m2, m3 and dm</td>
</tr>
</tbody>
</table>

5.3 Experiment

The goal of the experiment is, to test the hypothesis “Element-level descriptions can be used as similarity clues by a matcher, to increase the quality of the matches”. For this we need to establish a baseline result without the description matcher, and then compare this with the result of the experiments including the description matcher. If there is an increase in the Overall measure, then the hypothesis is correct.

For the experiment the source and target schemas listed in table 5.1 (see appendix B.1 for the schema listings) were matched against each other using the matcher listed in table 5.2. The results (the number of false negatives, true positives and false positives) were gathered manually and are shown in table A.1 in appendix A. This table also shows the settings of the minimum and maximum thresholds during the experiments.

It is important to note the Customer and Thunderbird Address Book schemas have English element names, but the descriptions of the elements are in Dutch. All other schemas are in Dutch. In the case that descriptions were not available, these were provided by asking a third person to write them down.

The Overall measure, shown in equation 5.3 embodies the idea to quantify the costs for the effort that is needed to post-process the result, e.g., adding false negatives, and removing false positives. Therefor the Overall measure was chosen, this implies that also the Precision and Recall are included.

\[
Overall = 1 - \frac{|A| + |C|}{|A| + |B|} = \frac{|B| - |C|}{|A| + |B|} = \text{Recall} \cdot \left(2 - \frac{1}{\text{Precision}}\right)
\] (5.3)
5 Evaluation

There are two sets of aggregated results: one using a minimum threshold of 0.27 and one with a minimum threshold of 0.48, the maximum threshold is in both cases 1. A lower threshold means that more matches are included in the aggregation. This results in a lower precision. These values were chosen intuitively by experimenting with the setting of the minimum threshold.

5.4 Results

The results are shown in table A.1. These results were used to calculate the Precision, Recall and Overall scores, which are presented in figures 5.2, 5.3 and 5.4.

5.4.1 Precision

Figure 5.2 shows the results for the Precision measure. The first four bars for each experiment set show the Precision of the individual matcher $m_1$, $m_2$, $m_3$ and $m_4$. Going back to the hypotheses in chapter 1.2, “does the addition of the description matcher improve the overall quality of the match result?”. The interesting bars are those of the aggregated results, the aggregation of the normal matcher, $m[1-3]$ and the results of the normal plus the description matcher, $m[1-3]+dm$.

All the experiments except $e2$ show a higher precision for the $m[1-3]+dm$ matcher aggregation than the $m[1-3]$ aggregation.

5.4.2 Recall

The results of the Recall are shown in figure 5.3. Again the relevant results are those of the $m[1-3]$ and $m[1-3]+dm$ matcher aggregations.
In e3, e4 there is in increase in recall visible between the m[1-3] and m[1-3]+dm results. This again confirms the hypothesis. But in e1 there is a decrease and in e2 there is no change in recall.

The reason for this are matches that have both the highest and the same score, since the system can not decide between two options, these results were classified as a False Positive. For example the Klant schema contains elements like TelefoonOverdag, TelefoonAvond, MobileTelefoon of which the descriptions are very similar, so if the search query is Telefoon the results are the same.

5.4.3 Overall

The results of the Overall measure are shown in figure 5.4. Again there is an increase in e1, e2, e3 and e4. Thus supporting the hypothesis. The overall of e3 is negative, this is caused by the low Precision which influences the Overall measure the most. The reason for the low Precision is the fact that one schema has Dutch element names and the other English, because of this, the performance of the string similarity matcher decreases.

5.4.4 Conclusion

In general the addition of the description matcher results in an increase of the Precision and Recall and thus also the Overall quality measure. But in absolute values this increase is quite small, approximately between a 5% and 10% increase. The question that still remains, is to show how significant this increase is, to get a better insight into this more experiments are needed.

But the experiments confirm that for our example schemas, the usage of description as extra similarity clue improves the overall quality of the matches, thus confirming the hypothesis.
Figure 5.4: Overall measure
6 Conclusions and Recommendations

This chapter answers the problem statement questions and gives recommendations.

6.1 Conclusions

**Hypothesis** Element-level descriptions can be used as similarity clues by a matcher, to increase the quality of the matches.

To evaluate the hypothesis, stated above, a number of research questions had to be answered:

**What is the current state of the art regarding schema matching?**
An extensive amount of research, has been, and still is currently done being in the field of schema matching. Systems like COMA++ and FlexiMatch show promising results. Unfortunately, there is no system available that can be used as a base to develop new ideas on, because these systems are research systems that are either not ready or not available for commercial usage.

**Which approaches exist?**
Most approaches focus on the usage of schema or instance information for similarity clues. There are some approaches that use of auxiliary information such as synonyms.

**Which systems exist and how do they compare?**
There is a large number of existing systems and Do [8] has done an extensive comparison. Unfortunately none of the systems is available, so they cannot be used as a baseline reference for the evaluation.

**What are the minimum requirements for a working schema matching system?**
The minimum set of functionality that a schema matching system based on the composite approach needs are:

- A model for the internal data representations, this is commonly a graph structure.
- A common interface for the individual matchers.
A method to aggregate the results of the individual matchers. Commonly a cube structure is used, each slice of the cube contains the results of the individual matchers. This cube can then be aggregated or flattened by using a function, i.e., Minimum, Maximum, Average.

This functionality combined with an initial matcher (e.g., one based on string similarity), forms the basis of a working schema matching system.

Which approach or approaches are best suited for this system?
The composite or multi-strategy approach is best suited for the evaluation system. This approach allows for the addition or removal of individual matchers, in contrast to the hybrid approach, where the individual matchers are integrated into a single hybrid matcher.

Are there any systems or approaches that use or have thought about using descriptions as a similarity measure?
Rahm and Bernstein [25] mention that descriptions can be used as a source for similarity clues, but they do not mention anything specific. Apart from this, there has not been done any research, focusing on the usage of descriptions.

How to use descriptions as a similarity measure?
The rationale is to see descriptions as documents. Using methods from the information retrieval research field, these documents can be searched for. A standard retrieval model was chosen, the vector space model, in combination with term weighting algorithms. This method results in a similarity measure that can give a similarity between source and target documents. This can then be used to say something about the similarity of schema elements.

How to evaluate the system?
Hong Hai Do et al. [15] give a theoretical basis to evaluate schema matching systems. From this basis, a quality measure was taken to evaluate the quality of the matches of the Topimatch schema matching system, with and without the usage of the description matcher. This quality measure uses the Precision and Recall metrics, from the information retrieval research field, and introduces a new combined metric called Overall.

The system is evaluated using four schemas, which are matched against each other in four experiments.

Conclusion
The results from the evaluation of the system confirm the hypothesis. The result show an increase in Precision, Recall and Overall, when the description matcher is added to
6.2 Recommendations

- It was decided not to include reuse of previous matching results. But looking at the schemas that were used for the evaluation it is clear that they have a common overlay, in other words the type of schemas that are used by Topicus often contain information about clients. For this, we conclude that a matcher which reuses previous match results seems a promising addition to the system, especially if combined with relevance feedback.

- Next to the element-level descriptions, there are many other sources of auxiliary information, e.g., project documentation. These can be indexed using a desktop search engine (e.g., Terrier [22], Google Desktop), after which element names or descriptions can be used to query this search engine. If a query from a source element yields a result similar to querying using a target element, then this might be used as a clue for similarity.

- For Topicus itself, the best solution to reduce the amount of effort that is needed for the schema matching task is, a system that provides easy manual matching, using an intuitive user interface and is supported by an automatic system that gives suggestions. But this does not provide a solution for the necessary conversions of the input schemas and the generation of transformation scripts.

- An automated framework to evaluate the system. This will ease the amount of effort that is needed for a manual evaluation, and therefore make it more practical to evaluate the effect of different parameters or new matches. It could also serve as the basis for an automatic tuning framework.
Bibliography


[27] Sam Chapman. SimMetrics. URL http://sourceforge.net/projects/simmetrics/ SimMetrics is a Similarity Metric Library, e.g. from edit distance’s (Levenshtein, Gotoh, Jaro etc) to other metrics, (e.g Soundex, Chapman). Work provided by UK Sheffield University funded by (AKT) an IRC sponsored by EPSRC, grant number GR/N15764/01.


### A Results evaluation

<table>
<thead>
<tr>
<th>Matcher(s)</th>
<th>Weight</th>
<th>Min</th>
<th>Max</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Precision</th>
<th>Recall</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1 m1</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>0.47</td>
<td>0.89</td>
<td>-0.11</td>
</tr>
<tr>
<td>m2</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>3</td>
<td>6</td>
<td>12</td>
<td>0.33</td>
<td>0.67</td>
<td>-0.67</td>
</tr>
<tr>
<td>m3</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>0.75</td>
<td>0.33</td>
<td>0.22</td>
</tr>
<tr>
<td>dm</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>0.75</td>
<td>0.33</td>
<td>0.22</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>0.50</td>
<td>0.89</td>
<td>0.00</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>0.64</td>
<td>0.78</td>
<td>0.33</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>0.53</td>
<td>0.89</td>
<td>0.11</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>0.75</td>
<td>0.67</td>
<td>0.44</td>
</tr>
<tr>
<td>e2 m1</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>0.50</td>
<td>0.23</td>
<td>0.00</td>
</tr>
<tr>
<td>m2</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>0.25</td>
<td>0.17</td>
<td>-0.40</td>
</tr>
<tr>
<td>m3</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>1.00</td>
<td>0.17</td>
<td>0.20</td>
</tr>
<tr>
<td>dm</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>0.70</td>
<td>0.41</td>
<td>0.40</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>1.00</td>
<td>0.23</td>
<td>0.30</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0.75</td>
<td>0.23</td>
<td>0.20</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>1.00</td>
<td>0.17</td>
<td>0.20</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>8</td>
<td>2</td>
<td>0</td>
<td>1.00</td>
<td>0.17</td>
<td>0.20</td>
</tr>
<tr>
<td>e3 m1</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>m2</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>0.33</td>
<td>0.14</td>
<td>-0.14</td>
</tr>
<tr>
<td>m3</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>dm</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0.67</td>
<td>0.29</td>
<td>0.14</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>5</td>
<td>2</td>
<td>16</td>
<td>0.11</td>
<td>0.29</td>
<td>-2.00</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>4</td>
<td>3</td>
<td>14</td>
<td>0.18</td>
<td>0.43</td>
<td>-1.57</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.14</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0.67</td>
<td>0.29</td>
<td>0.14</td>
</tr>
<tr>
<td>e4 m1</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>1.00</td>
<td>0.33</td>
<td>0.33</td>
</tr>
<tr>
<td>m2</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>0.80</td>
<td>0.33</td>
<td>0.25</td>
</tr>
<tr>
<td>m3</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>1.00</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>dm</td>
<td>1.00</td>
<td>0.60</td>
<td>1.00</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>0.75</td>
<td>0.25</td>
<td>0.17</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>0.39</td>
<td>0.58</td>
<td>-0.33</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.27</td>
<td>1.00</td>
<td>3</td>
<td>9</td>
<td>10</td>
<td>0.47</td>
<td>0.75</td>
<td>-0.80</td>
</tr>
<tr>
<td>m[1-3]</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>0.44</td>
<td>0.67</td>
<td>-0.17</td>
</tr>
<tr>
<td>m[1-3]+dm</td>
<td>1.00</td>
<td>0.48</td>
<td>1.00</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>0.50</td>
<td>0.75</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table A.1: Raw results from the experiments e1, e2, e3 and e4
B Listings

B.1 Schemas

Listing B.1: Finture Klant Object
<hibernate-mapping>
  <class table="PartijNAWData">
    <property name="AchterNaam" description="De achternaam zonder tussenvoegsel."
      />
    <property name="VoorNaam" description="Dit de betreft volledige eerste voornaam van de aanvrager"/>
    <property name="TussenVoegsels" description="De tussenvoegsels behorende bij de achternaam."/>
    <property name="GebAchterNaam" description="De geboortenaam van de partij."/>
    <property name="GebTussenVoegsels" description="De tussenvoegsels behorende bij de geboortenaam."/>
    <property name="VoorLetters" description="Voorletter(s)"/>
    <property name="Geslacht" description="Geslacht"/>
    <property name="StraatNaam" description="Straatnaam, indien sprake is van een correspondentie adres kan ook de aanduiding postbus gebruikt worden."/>
    <property name="StraatNaamToev" description="Straatnaam toevoeging."/>
    <property name="HuisNr" description="Het huisnummer, indien sprake is van een correspondentie adres kan dit ook een postbusnummer zijn."/>
    <property name="Postcode" description="De postcode, behorend bij ingevulde straatnaam / huisnummer of bij ingevuld postbusnummer. Voor Nederland 6 posities lang, zonder spatie ertussen."/>
    <property name="LocatieAanduiding" description="De locatieaanduiding. Wordt veelal gebruikt bij bedrijfscomplexen (bijv. toren II) Aanduidingen als Huis etc. bij Huisnummertoevoeging."/>
    <property name="PlaatsNaam" description="De plaatsnaam."/>
    <property name="Land" description="ISO landcode (ISO-3166)"/>
    <property name="RekeningNr" description="Het rekeningnummer van de partij."/>
    <property name="LandRekening" description="Het land van het rekeningnummer."/>
    <property name="VoorTitel" description="Voortitel (b.v. ing)"/>
  </class>
</hibernate-mapping>

Listing B.2: HDN ParijNAWData schema fragment

<hibernate-mapping>
  <class table="tbl_Customers">
    <property name="TxtTitle" description=""/>
    <property name="TxtFirstName" description="Voornaam"/>
    <property name="TxtMiddleName" description="Tussenvoegsel"/>
    <property name="TxtLastName" description="Achternaam"/>
    <property name="TxtEmail" description="Emailadres"/>
    <property name="TxtJobTitle" description="Beroep (vervalt, gaat via tabel tbldtl_CustomerIncomes)"/>
    <property name="TxtCompany" description="Bedrijfsnaam"/>
    <property name="TxtBusinessPhone" description="Telefoonnummer zakelijk"/>
    <property name="TxtHomePhone" description="Telefoonnummer thuis"/>
    <property name="TxtMobilePhone" description="Telefoonnummer mobiel"/>
    <property name="TxtFax" description="Faxnummer"/>
  </class>
</hibernate-mapping>
<hibernate-mapping>
  <class table="AddressBook">
    <property name="FirstName" description="Voornaam"/>
    <property name="LastName" description="Achternaam"/>
    <property name="DisplayName" description="Naam die wordt gebruikt voor publieke publicatie"/>
    <property name="NickName" description="Roepnaam"/>
    <property name="Email" description="Email adres"/>
    <property name="AdditionalEmail" description="Tweede email adres"/>
    <property name="ScreenName" description="Gebruikersnaam"/>
    <property name="WorkNumber" description="Telefoon nummer op het werk"/>
    <property name="HomeNumber" description="Telefoon nummer thuis"/>
    <property name="FaxNumber" description="Fax nummer"/>
    <property name="PagerNumber" description="Nummer van je pieper telefoon"/>
    <property name="MobileNumber" description="Nummer van je mobiele telefoon"/>
    <property name="HomeAddress" description="Adres"/>
    <property name="HomeCity" description="Plaats"/>
    <property name="HomeState" description="Provincie"/>
    <property name="HomeZIP" description="Postcode"/>
    <property name="HomeCountry" description="Land"/>
    <property name="Webpage" description="Webpagina"/>
    <property name="WorkTitle" description="Werk beschrijving"/>
    <property name="WorkDepartment" description="Afdeling waar je werkt"/>
    <property name="WorkOrganisation" description="Bedrijf waar je werkt"/>
    <property name="WordAddress" description="Adres van het bedrijf"/>
    <property name="WorkCity" description="Plaats van het bedrijf"/>
    <property name="WorkState" description="Provincie van het bedrijf"/>
    <property name="WorkZIP" description="Postcode van het bedrijf"/>
    <property name="WorkCountry" description="Land waar het bedrijf is gevestigd"/>
    <property name="WorkWebpage" description="Webpagina van het bedrijf"/>
  </class>
</hibernate-mapping>

Listing B.4: Thunderbird Address Book label
## B.2 HDN Datacatalogus AX Message

<table>
<thead>
<tr>
<th>PartijNAWData/overtreknummer</th>
<th>Naam</th>
<th>Data type</th>
<th>Omschrijving</th>
<th>Beschrijving</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Eidsnaam</td>
<td>PD3</td>
<td>O</td>
<td>Naam van de lidmaatschap zoals op de ambachtelijke inschrijving.</td>
</tr>
<tr>
<td>-</td>
<td>Eidsnaam/Tev</td>
<td>KD1</td>
<td>O</td>
<td>Naam van de lidmaatschap zoals op de ambachtelijke inschrijving.</td>
</tr>
<tr>
<td>-</td>
<td>Naam</td>
<td>K8</td>
<td>O</td>
<td>Naam van de lidmaatschap zoals op de ambachtelijke inschrijving.</td>
</tr>
<tr>
<td>-</td>
<td>Naam/Tev</td>
<td>KD1</td>
<td>O</td>
<td>Naam van de lidmaatschap zoals op de ambachtelijke inschrijving.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Makelaar/ordeovername</th>
<th>Naam</th>
<th>Data type</th>
<th>Omschrijving</th>
<th>Beschrijving</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Eidsnaam/Feitelidnummer</td>
<td>PD4</td>
<td>O</td>
<td>Naam van de lidmaatschap zoals op de ambachtelijke inschrijving.</td>
</tr>
</tbody>
</table>

Figure B.1: HDN Datacatalogus AX Message