



## Thesis report

### *A Data Transformation Walkthrough*

#### **Final research project, Universiteit Twente**

<b>name</b>	R.J.G. van Bloem
<b>student nr.:</b>	9912347
<b>cluster:</b>	Databases
<b>graduation advisor:</b>	dr. M.M. Fokkinga
<b>2<sup>nd</sup> graduation advisor:</b>	dr. A. Wombacher
<b>external company:</b>	InDialoog, Enschede
<b>external advisor:</b>	Ing. J. Posthuma
<b>date:</b>	30/10/2008

## Summary

Data transformation has been an important issue since the beginning of the it era. And with an ever growing amount of data and all the different forms it comes in, data transformation and standardization is still a hot issue.

With the use of XML many companies claim to have found a way to interchange there data fairly easy. Of course XML is now widely accepted and is used in many connections as a bridge between multiple parties.

But still XML is only a standardized structure which leaves its filling to the creator. Which leaves us with the old problem: You say "potato," I say "potahto...". One might call an element 'E-mail' while another calls it 'email', and these are still fairly alike and in the same language.

With the fast amounts of data today, also fast amount of naming variations exist. One solution would be to let al people agree on the naming of elements, which of course in practice is impossible. Although it would be good practice if people would consider naming conventions, which will make semantic matching a bit easier.

Semantic matching is the key in data transformation. The meaning of elements must be known in accordance with its context. Getting semantically correct matches is still a tricky undertaking, especially in an open context. People all still the best judges in this matter, but are of course losing it to computers on raw processing speed. Therefore it may be concluded that we need the best of both worlds to solve data transformation issues as efficiently as possible.

This thesis report aims to provide a clear and workable data transformation method. The focus will not be on technical solutions but on creating a user friendly and efficient data transformation prototype. A setting where people can easily input there semantic knowledge and the computer can use this knowledge in its fast processing.

After a look at data transformation insights, the system is designed with as key feature the matcher which uses automated and manual matching in an iterative feedback setting. Following three iterations will be done to produce a basic data transformation prototype.

In iteration one the system base will be designed and developed which is to provide a solid/flexible base for semantic matching. Reusability and flexibility are important in this phase so that system components can easily be altered if ever needed later on. Iteration two aims to provide a better user interaction for visual transformations. This will be done by the visualization of the leading XML transformation language XSLT. VisualXSLT will provide a component based gui to build transformations on the fly. Iteration three addresses partial mapping reuse though transitive matching and ranking based on semantic knowledge.

# Table of contents

<b>1. Introduction.....</b>	<b>5</b>
1.1 Background.....	5
1.2 Objectives.....	5
1.3 Overview.....	6
<b>2. Project overview.....</b>	<b>7</b>
2.1 Assignment formulation.....	7
2.2 Research questions.....	7
2.3 Goals.....	8
2.4 Project approach.....	8
<b>3. Data transformation.....</b>	<b>10</b>
3.1 Transformation parts.....	10
3.2 Matching.....	13
3.2.1 Automated schema matching.....	14
3.2.2 Schema matching survey.....	16
3.2.3 Human interaction and efficiency.....	18
3.3 Input, Output and Intermediate data.....	19
3.3.1 Input and output data.....	19
3.3.2 Intermediate data.....	20
<b>4. Main design.....</b>	<b>21</b>
4.1 Scalability.....	21
4.2 Data overview.....	23
4.3 Data stores.....	24
4.3.1 Raw & output XML schema store.....	24
4.3.2 Personal schema store.....	25
4.3.3 Data store.....	25
4.3.4 Mapping store.....	25
4.4 Element overview.....	26
4.4.1 Connection layer.....	26
4.4.2 Schema valid parser.....	27
4.4.3 Matcher.....	27
<b>5. First iteration, basic system.....</b>	<b>30</b>
5.1 Goals / Requirements.....	30
5.2 Required components .....	30
5.3 Research questions.....	30
5.4 Design.....	31
5.4.1 Parser rules, schemas and meta data.....	31
5.4.2 Mapping format.....	32
5.4.3 Manual matching.....	33
5.4.4 Non-partial mapping store reuse.....	34
5.5 Implementation.....	35
5.6 Evaluation.....	35
<b>6. Second iteration, Visual XSLT.....</b>	<b>37</b>
6.1 Goals / Requirements.....	37

6.2 Visual aiding survey.....	37
6.3 Required components.....	40
6.4 Research questions.....	41
6.5 Design.....	41
6.5.1 Visual elements.....	41
6.6 Implementation.....	48
6.7 Testing.....	49
6.7.1 Test Goals.....	50
6.8 Evaluation.....	50
6.8.1 Comparing the prototype.....	51
6.8.2 Test results.....	52
6.8.3 Conclusion.....	54
<b>7. Third iteration, Partial mapping reuse.....</b>	<b>56</b>
7.1 Goals / Requirements.....	56
7.2 Research questions.....	57
7.3 Research.....	57
7.3.1 Reuse in Coma.....	57
7.3.2 Discovering mappings.....	58
7.4 Design.....	61
7.4.1 Storing partial matches .....	61
7.4.2 Match candidate discovery.....	63
7.5 Evaluation.....	70
<b>8 Conclusions &amp; Recommendations.....</b>	<b>71</b>
8.1 Conclusions.....	71
8.2 Recommendations.....	72
<b>Appendix A. References.....</b>	<b>74</b>
<b>Appendix B. Data conversion example.....</b>	<b>75</b>
<b>Appendix C. Parse rules, schemas, meta data.....</b>	<b>79</b>
<b>Appendix D. Connection and driver classes.....</b>	<b>81</b>
<b>Appendix E. Visual classes.....</b>	<b>82</b>
<b>Appendix F. Test results E1.....</b>	<b>83</b>

# 1. Introduction

This thesis report is about data transformation [1], in particular data transformation that aims to generate many different forms of documents out of a single source. The aim of this document is to provide a clear and workable data transformation method. This means a complete walk through from data input to data output. The focus will not be on technical solutions but on creating a user friendly and efficient data transformation. Of course technical solutions can be a part of this.

This chapter will give background information on the project and set its global goals. A short overview of the thesis chapters will be given at the end.

## 1.1 Background

Data transformation is widely used in IT businesses. There are numerous data types and many ways of converting on type to another. Also the context of data will not always be clear, meta data might be missing or be in an unknown legacy or binary format. Because of this diversity, data transformations can be time consuming. They are likely to involve a lot of manual interaction, are hard to reuse or can only be done by users with programming experience. A new data type like XML [2] is far more friendly for data transformation, because it contains meta data, is human readable and has a widely accepted standardized structure.

There are businesses that use 1-on-1 transformations as a base for their systems. 1-on-1 transformations can be done quickly and are relatively easy to do. A 1-on-1 transformation system has the drawback that it will grow quadratically with every added data format. This will make such a system poorly scalable and costly to maintain and expand. Building a true many-to-many transformation system will take more time at first but will repay its effort in scalability, reusability and maintainability.

InDialog is a communication / IT company implementing a range of web based systems. These systems use different sources as external data which have to be stored, processed and presented in different formats. Transformation are now manually implemented to fit the main system used, the content management system 'Ariadne' [3]. This is time consuming and does not fit future needs for expansion. Therefore a central storage system has to be build which covers the aspects of data communication, transformation and storage.

## 1.2 Objectives

The objective of this project is to investigate and prototype a data transformation method for a many-to-many data transformation. The goal is to create a workable and efficient transformation walk through rather than creating a 100% technical solution. Of course technical solutions will be a part of the project, but their function will be to aid in user friendliness and a more efficient total.

To build a scalable many-to-many transformation system the 1-on-1 transformation idea has to be thrown overboard. If an intermediate data store is used transformation options can be reduced from  $n^2-n$  to  $2n$ . This intermediate data must be easy to process and have enough meta data to feed an automated transformation to multiple output types. When designing the system key features like re usability, interchangeability must be kept in mind to create a flexible system. A flexible system must leave the system open for later change, so no major redesigning has to be done when new features or problems arise.

The proposed method should incorporate a modular buildup of different components in an ETL (Extract, Transform and Load) [4] environment. There will probably always be the need of manual analysis and mapping for the simple reason that a computer does not know the context of standalone data. But to reduce manual operations the system can use automatic matching methods, for instance element matching and structure matching. Also manual operations can be simplified by using support features like visual aided mapping. The result of transformations can be made reusable by storing them for later transformation-by-example processing. [5][6][7][8]

## **1.3 Overview**

### **Chapter 2, *Project overview:***

Describes the project formally and gives insight in the approach taken to tackle this project.

### **Chapter 3, *Data transformation insight:***

Gives an insight of the parts needed for data transformation and usable techniques that are currently on the market.

### **Chapter 4, *Main design:***

Describes the main design of the system, which will be used as basis for the prototype.

### **Chapter 5, *First iteration, basic system:***

Describes the basic system which will be used for the other iterations.

### **Chapter 6, *Second iteration, Visual XSLT:***

Describes a visual representation of XSLT which will be used to visually aid users in the system.

### **Chapter 7, *Third iteration, Partial mapping reuse:***

Describes the partial mapping reuse based on previously stored matches.

### **Chapter 8, *Conclusions & Recommendations:***

Gives conclusions about the project and recommendations for re-enacting and further development this project.

## 2. Project overview

This chapter describes the project formally and gives insight in the approach taken to tackle this project. The project assignment, research questions and goals are formulated and the iterative approach used in this project will be explained.

### 2.1 Assignment formulation

***Design and implement a many-to-many data transformation prototype. This will embody the design and implementation of a user friendly, low effort / high efficiency, transformation walkthrough based on ETL (Extract, Transform and Load) processing with a intermediate data store.***

### 2.2 Research questions

Out of the assignment formulation keywords are selected. These keywords represent the main subjects of this assignment. These keywords are then used to construct the research questions, which are used to divide the assignment in smaller research parts.

*Keywords:*

1. ETL processing
2. many-to-many data transformation
3. intermediate data store
4. low effort / high efficiency
5. user friendly

Out of these keywords the research questions are constructed. The questions are chosen to cover the important aspects of that research part. The answering of these questions must result in the information needed to perform the assignment formulated.

A numbered reference between keywords and research questions is shown below.

nr.	research question	chapter
1	What element are necessary for ETL processing?	3.1
2	What information is useful for data transformation?	3.1
2, 3	What are common / useful data formats?	3.1 / 3.3
3	What format can be used for the intermediate data?	3.3
4	How can low effort / high efficiency be realized?	3.2
4, 5	What is a good optimum between automated and manual matching?	3.2

nr.	research question	chapter
4, 5	How can user interaction be used?	3.2
4	What forms of (automated) transformation can be used?	3.2

## 2.3 Goals

To produce a system conform the assignment formulation the following goals will have to be met:

goal	chapter
Design and implement a basic ETL system	3.1 / 4
Define a intermediate data format	3.3
Design an automated matching element within the base system	3.2 / 4.3.3
Incorporate reusability in data, matching and programming	5.4 / 7
Design and implement a user interface for visual aided mapping	6
Evaluate user friendliness, user efforts and efficiency.	3.2.3 / 4.3.3 / 5.4.4 / 6 / 7

## 2.4 Project approach

To tackle this project an iterative approach will be taken. This is done to keep the project more easy to handle and less vulnerable for major time consuming errors. At every iteration goals and milestones will be set to build up the system step by step. After every iteration the system must be analyzed to see if the goals have been met and the milestones are reached.

The development phase of the project will be divided into three iteration steps. Every iteration step will take approximately four weeks. A single iteration will consist of the following parts:

- setting goals
- making choices and setting boundaries
- designing the parts
- implementation of the designed parts
- evaluation of the goals

The project will be split up in three iterations. The idea is to start with a broad and simple system and to add more difficult parts on top of that piece by piece. This means that with every iteration a functional system will be the result. The benefit of these functional iteration results are that the system can be evaluated as a whole with every iteration. The main goals of the three iterations are:



- Iteration 1:** Implement a broad base system able to import, store and output data. Data transformation will be manual in this iteration.
- Iteration 2:** Expand the base system with visual aided matching. Research, design and implement a visual representation for data matching.
- Iteration 3:** Expand the system with non-partial mapping reuse. Research, design and implement a non-partial mapping reuse prototype.

### 3. Data transformation

Data transformation [1] is a large domain, the objective of this chapter is to give an insight of what is important and what is useful within this domain for this project. The subjects described in this chapter are referenced from the research questions in chapter 2.2 and a part of the goals in chapter 2.3. The different elements and information needed in data transformation and ETL [4] processing will be shown. Common and useful data formats will be reviewed and the format possibilities for the intermediate data will be discussed. Matching will be discussed as an important feature within data transformation. The use of human interaction and user friendliness will be discussed in combination with automated transformation and in comparison with efficiency and effectiveness.

#### 3.1 Transformation parts

To be able to design and implement a basic ETL system it is first necessary to get an insight of important features of such a system. To get an insight of what these important features are the following research questions will be answered:

- What elements are necessary for ETL processing?*
- What information is useful for data transformation?*
- How can user interaction be used?*

#### System overview

A global view of a transformation system can be divided into three main areas of interest:

- input/output data formats
- ETL processing
- intermediate data store

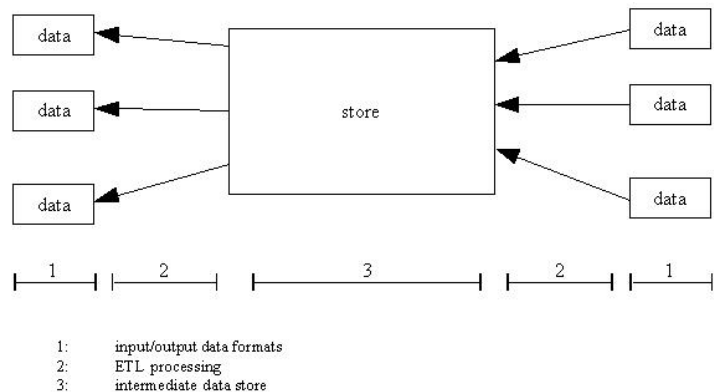


Illustration 1: main areas

These three main areas of interest can of course be refined into smaller areas. If we walk through the system from input to output we will encounter the different key components. The first component is the **external data** that will be presented to system to be transformed. The external data can be delivered to the system by a variety of data containers. External data can be contained in files, streams, databases, etc. To keep the connection to the external data transparent a **connection layer** will be needed to connect and read/decode data in a uniform way from an internal perspective.

The goal is to read data and transform it into a format suitable for intermediate storing. The external data will be delivered in a variety of data formats, these different formats must be transformed into the data format suitable for intermediate storing. After reading the data we have raw data in many different formats. The first objective is to transform this data to a shared format that can easily be processed for further use. For this shared format XML [2] will be used because there are many tools available to processes it and it is widely used in open communication. The transformation of the read data to the shared format will require **parsers** for the different external formats. A parser will transform the read data to a **raw XML** format.

The next step is to process the raw XML to match the data format of the intermediate store. This can be done by the use of **matchers**. Matchers can find similarities between the raw XML and the intermediate format. Automated matchers will not give a 100% result so it must be possible for a user to adjust matching manually. The end result of the matching will be the representation of the external data in the **intermediate data** format and the **mapping** used for this transformation.

After the data has been stored, the data can be requested to be outputted in a specific output format. To make writing to a specific data output format easier and to give a user options to manipulate the data, the stored data will first be transformed to a XML representation of the output format. This transformation will be done by a **matcher**, which again can be adjusted by the user. The end result of the matching will be a **XML representation** of the output data and the **mapping** used for this transformation.

The XML representation can be transformed to the output data format by **parsers**. A parser will transform the XML to the specific **output data** format which than can be written to its requested output. This writing will be done by a **connection layer** responsible for encoding and outputting the data.

The parts named above can be bound together into a main design for the system, see illustration 2. When looking at illustration 2 it is obvious that the left side of the system is a mirror image of its right side. Individual parts, connection layer, parsers and matchers are defined on either side. This suggests re usability, this will be kept in mind when designing the individual system part. [5][6][7][8]

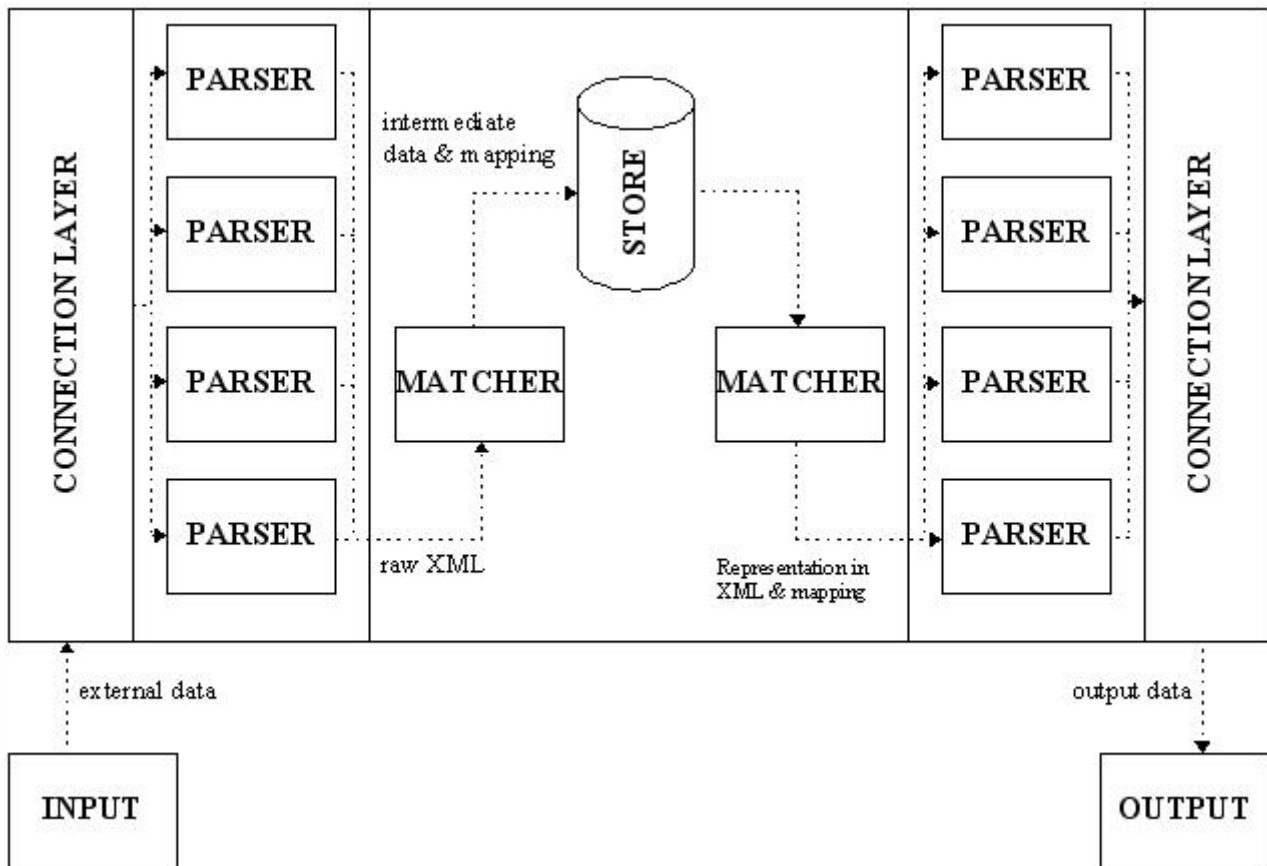


Illustration 2: the big picture

### Useful data information

There are steps in the design that can provide useful information for the data transformation. Of course information can be found in the processed data itself. Depending on the format of the data, data may contain meta data giving extra information about the data, its context, references etc.

Useful information about the data can also be gotten from the data environment. This information can be found in references within the data domain. For example creation data of a file or keys of a database table. Another way to obtain useful information is by user input, a user can be given the opportunity to add additional meta data, hereby describing parts of the data environment and semantics.

When the data is matched by the system the result of this matching can give extra information about the data. The total mapping can be stored for reuse, more specific the elements matched within the mapping can be used to describe references between elements and semantics. Also user interaction of what matches are good and what not are very useful since they produce a realistic semantic reference. These references and semantics can later be reused to aid in new matchings. [5][6][7][8]

## **Conclusion**

*What elements are necessary for ETL processing?*

Main elements necessary for ETL processing are: Connection layers, parsers, matchers (manual & automatic) and data stores.

*What information is useful for data transformation?*

Useful information for data transformation can be found in the input data and its meta data, the mapping result and possible user input. These sources give information about the data's context, references and semantics and thereby aid the understanding of the data. This information can be useful within a mapping process, it can provide more complete data and extra references for better matching. The context, reference and semantic information can also be stored to aid future mapping processes, this re usability can improve future matching results because it provides additional information to the mapping process which then is evolving with every mapping.

*How can user interaction be used?*

User interaction can be used to input additional information to the system and judge matching results.

## **3.2 Matching**

A key feature of the system is the matching. The goal of matching is to get the right semantic match between elements. Matches can be found in a variety of ways, manually, through automated algorithms, learning networks, iterative approaches, etc. With all kinds of different techniques at hand, the goal is to get the best semantic results at the lowest effort. To get an insight how to reach this goal, the following questions will be answered.

*What forms of (automated) transformation can be used?*

*How can user interaction be used?*

*What is a good optimum between automated and manual matching?*

*How can low effort / high efficiency be realized?*

An important part of data transformation is matching. Matching is the process of finding elements that are alike and classifying their relation. A structured form of matching is schema matching. Schema matching compares two schema's with each other and tries to find semantic correspondences between the schema elements. The result of this matching is called a mapping, this mapping contains possible matches found between elements of the different schema's.

Schema matching is very common nowadays, it is used in many applications working with multiple data sources. Major application domains of schema matching are: Data warehousing, Search engines and E-commerce. But schema matching is also done at a large scale by many smaller applications in need of synchronizing and merging data. With the introduction of XML, schema matching has become more widely and openly used, a lot of open source tools can be found aiding XML matching and transformation.

Schema 1	Schema 2
Cust	Customer
C#	→ CustID
CName	→ Company
FirstName	→ Contact
LastName	Phone

Table 1: schema matching example

Whether a matching result is semantically correct can in the end only be judged by an end user. So there is still a lot of manual matching done by domain experts. This will of course be a time consuming undertaking. Automated schema matching can be used to lighten the matching process.

### 3.2.1 Automated schema matching

There are many different schema matching approaches and many different forms of schema matchers. Schema matchers usually use some kind of algorithm to walk through the schema's and match information between them. A new approach here is the use of machine learning techniques a.k.a. artificial intelligence.

The **LSD** (Learning Source Description) [13] system is such a matcher, it uses previous acquired data to learn from and improve its matching results. The LSD system is build up out of three basic elements:

1. a training phase, training a matcher with test data
2. a multi learning strategy, selecting and combining different matching candidates from different matchers
3. a ranking step, resulting in the favouring of matchers that gave a good result

Schema's can be matched by using many different criteria and approaches. Two main approaches are:

- **schema based matching**
- **instance based matching**

Schema based matching looks at information that can be derived from the schema itself. It looks at the structure of the elements, relations, element names, data types, etc. Instance based matching looks at the information available in the contents of instances (data values).

Different individual matchers can also be combined to try to get a better matching result. Two main approaches are:

- **hybrid matching**
- **composite matching**

Hybrid matchers are a combination of different matching techniques working together in a fixed predefined way. This approach has the power of combining the best of multiple worlds, eliminating individual weaknesses. Composite matchers combine different individual matcher. They all compute their own result which will be evaluated and combined into one end result. This approach is very flexible, different matchers (individual and hybrid) can be brought together for a specific task at hand.

These matchers can be implemented in different ways. A generic matching implementation is shown in illustration 3.

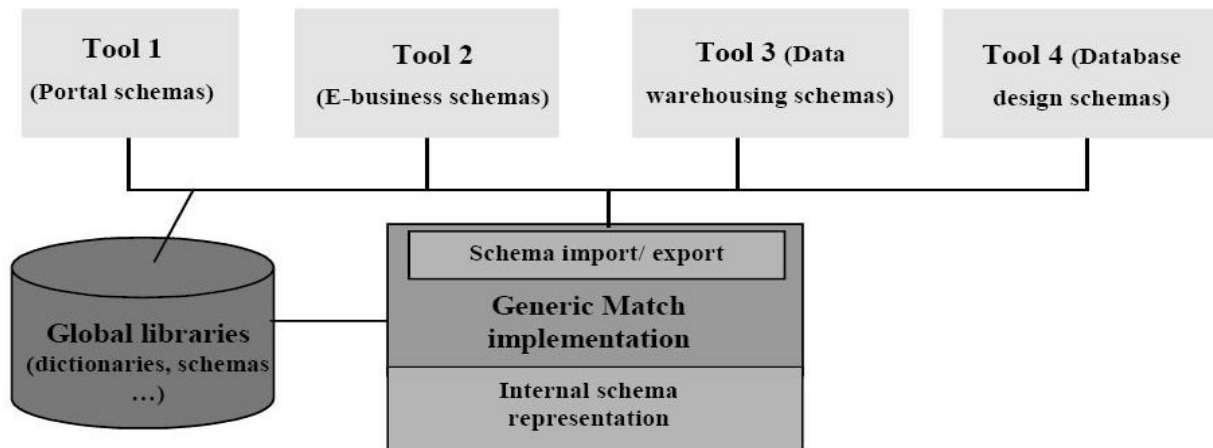


Illustration 3: high level architecture of generic match [15]

Rahm and Bernstein [15] came up with the graphical representation of schema matching approaches show in illustration 4. The characteristics of these individual matchers will not be further discussed here.

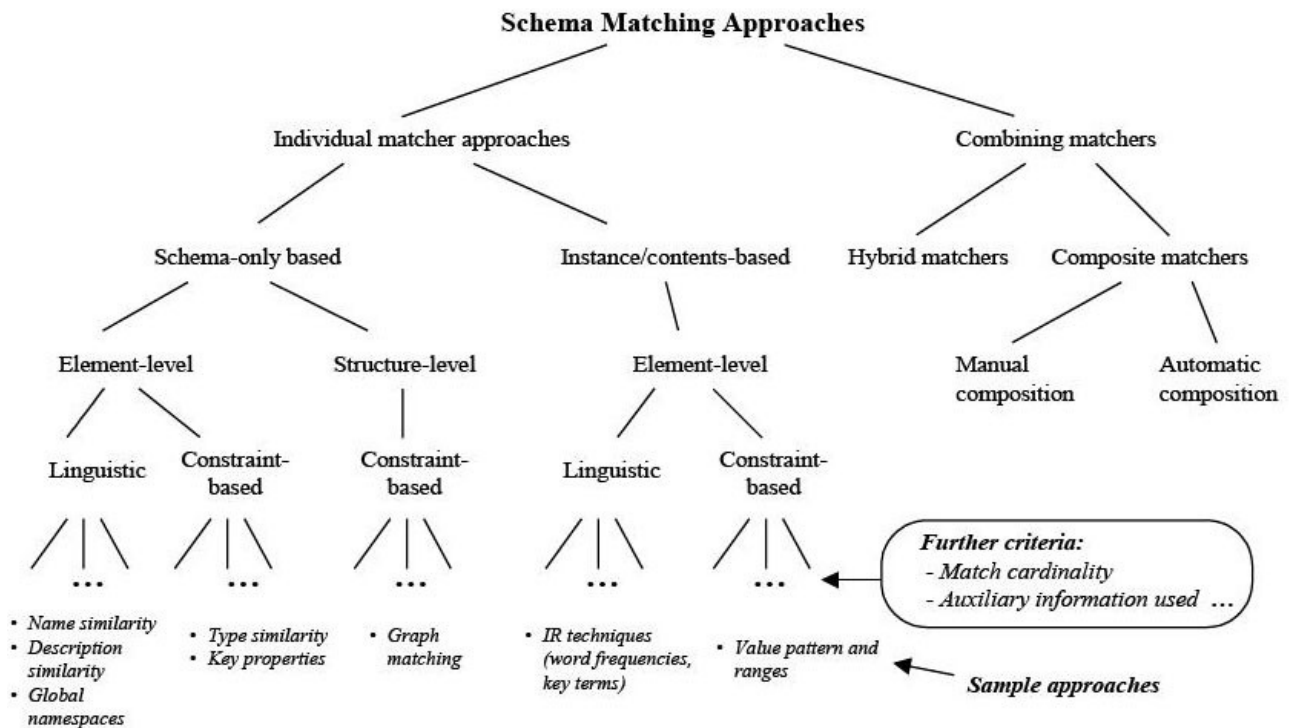


Illustration 4: classification of schema matching approaches [15]

### 3.2.2 Schema matching survey

Nowadays a lot of development is done in data transformation, especially schema matching is popular. Schema matching maps elements from a source schema to elements from a target schema. Still a lot of work within schema matching is done manually because automated matching cannot yet give a satisfactory result since it lacks the semantic view of an end user.

A lot of work is done in improving automated schema matching. The major approaches for schema matching are visible in illustration 4. It is proven that single match algorithms will not always generate good results on a wide domain range. To make matching more flexible and thus more efficient the trend is to use multiple match algorithms or matchers. This allows users to select matchers for specific application domains which will optimize the matching result.

For individual matchers, the following classification criteria are considered:  
[9][13][15]

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

Instance data can give insight in the semantics of schema elements. This can be valuable when useful information cannot be retrieved from a schema. Instance data can be used to aid constructing schemas when a schema is missing or help in analysing the correctness of a schema interpretation. With instance data and possible auxiliary data a better semantic view can be made because it gets its knowledge from the actual contents.

Schema matching only gets its information out of the properties of schema elements (name, description, data type, relationship type, constraints, structure). This means semantic values of the content can only be derived if these properties reflect their content. If this is not the case user interaction or some form of artificial intelligence is needed. Different approaches of schema matching are: element matching, structure matching, linguistic based matching and constraint based matching.

Matcher will in general find multiple match candidates. These candidates can be compared and normalized to identify the best match candidates. The use of different simultaneous matching approaches can give extra insight in the correctness of match candidates.

**Element vs structure matching:** match can be performed for individual schema elements, such as attributes, or for combinations of elements, such as complex schema structures.

Element matching matches individual elements from one schema to individual elements from another schema without considering their underlying elements. Structure matching matches element by fully or partial matching elements and their sub elements. For more complex matchings the effectiveness of structure matching can be enhanced by considering known equivalence patterns. For element matching effectiveness can be enhanced by knowledge of element equivalence, which of course can also be applied within structure matching.



**Language vs constraint:** a matcher can use a linguistic based approach (e.g., based on names and textual descriptions of schema elements) or a constraint based approach (e.g., based on keys and relationships).

Linguistic matchers use names and text of elements to find semantic similarities. A good investment is the use of thesauri or dictionaries. By exploiting synonyms, hypernyms and homonyms better semantic references can be made.

Constraint matchers use data types, value ranges, uniqueness, optionality, relationship types and cardinalities etc. to determine similarities between elements. This will often lead to imperfect matches, as there may be several elements with similar constraints. But constraints can still be used to limit possible match candidates.

**Matching cardinality:** the overall match result may relate one or more elements of one schema to one or more elements of the other, yielding four cases: 1:1, 1:n, n:1, n:m. In addition, each mapping element may interrelate one or more elements of the two schemas.

Matching cardinality can be viewed in two ways, globally and locally. Global cardinality looks at the cardinality of all the elements within a matching. Local cardinality looks at the cardinality of individual elements. When matching multiple elements at once, expressions are used to specify the more complex relation between these elements. Most existing approaches use 1:1 local matches and 1:1 or 1:n mappings. More work is needed to explore more sophisticated criteria for generating local and global n:1 and n:m mappings, which are currently hardly treated at all

**Auxiliary information:** most matchers rely not only on the input schemas but also on auxiliary information, such as dictionaries, global schemas, previous matching decisions and user input.

Auxiliary information like thesauri, dictionaries and user input can provide useful (miss)match information at a low effort. The reuse of common schema components and previous mapping are also promising reuse oriented approaches. Often schema's matched are in some way similar to a previous matching, so reuse can improve efficiency. Names, types, key, constraints and schema fragments can be reused, especially when working in a local domain these elements will have some form of standardisation which will improve re usability. Matches and schema fragments can be stored for reuse, but this will only be functional if there is the possibility to check them for similarities. This of course is a match problem in itself.

The different features mentioned can be found in schema matching programs on the market: SemInt, LSD [13], SKAT, TranSCm, DIKE, ARTEMIS, CUPID [9], COMA [16]. Especially COMA looks interesting for its flexible approach towards schema matching and its match iterations.

### 3.2.3 Human interaction and efficiency

Automated matchers nowadays do not have the capability to give a full satisfactory result, so it must be possible for a user to adjust matching manually. Since automated matching will not work 100% stand alone it must be treated as an aiding tool in the transformation process. Automated matching is then a tool to create a higher efficiency.

The manual matching can also be adapted to create a higher efficiency. By using a graphical user interface the matching process can be made more comprehensive, easier to adapt and thereby more efficient. The design of the graphical user interface and its underlying function is of great importance for the efficiency. The design must be clear to the user and must try to “push” the user in the right direction.

A commonly used design is a two view layout. On the left side of the screen the current schema is shown, on the right side the preferred output schema shown. The mapping is visualized by drawing lines between the schema elements that are linked. This view gives a good overall perspective of the matching between both schema's.

Besides the matching of elements the graphical user interface can be used to manage the automated matching process. The automated matching process can be aided by giving certainties and options about the schema matching. This can result in a reduction of the mapping space and thus in a higher efficiency.

Also user interaction can be used to grade the matching result. This feedback can be used to optimize the matching process in the future. This feedback can include an advisory for semantics so matchers can get a better insight on semantics. [13][15][16]

### Conclusion

*What forms of (automated) transformation can be used?*

Schema matching is a good form of transformation between schemas at a low effort. Instance matching can be used for auxiliary data, on its own it will not yield as much matches as schema matching at the same effort. Element matching is a easy way to construct simple matchings. Structure matching is more difficult and will have fewer results but is not bounded to simple matchings.

Language matching is useful, especially within specific domains. Dictionaries and thesauri are forms of auxiliary data that can be used to aid language matching. Auxiliary data is very useful within transformations, it can aid processing and can be used to learn from earlier transformations.

*How can user interaction be used?*

User interaction can be used for simple matchings, more complex matching can of course also be done but this will require a domain expert / programmer. User interaction can be used to judge the outcome of automated matches. User interaction can provide a set of definite matches and non-matches which can be excluded in automated matching and can give semantic value to language elements. When using a combination of automated matchers a user can make a good choice between different matchers.

*What is a good optimum between automated and manual matching?*

Manual matching is a sure solution but can be time consuming and lacks reusability. Manual matching can be used to match the simple cases and the rest can be matched

using automated matching. Automated matching can be used as an aid which can learn from previous matchings. Manual interaction can then be used to judge the automated processing and improve future automated matchings. Is this setting both forms of matching can support one and another.

*How can low effort / high efficiency be realized?*

Low effort / high efficiency can be realized by choosing components that give good result at a low cost, both in development and user time. Good options to use are:

- A graphical interface for manual matching, this can save time and give an easy access to more complex matchings.
- Automated element / language matching, this can yield fast results and is not very complex to build.
- Reuse of matchings by storing element and language matches. The use of thesauri and dictionaries will prove very useful.
- A possibility for combining matchers will give more flexibility and better results. Useful combinations can also be reused.

### **3.3 Input, Output and Intermediate data**

The goal of the system is data transformation. Therefore the data itself is an important subject. In this paragraph the following questions are answered to provide an understanding of what kind of data is handled.

*What are common / useful data formats?*

*What format can be used for the intermediate data?*

#### **3.3.1 Input and output data**

As input and output data all kinds of formats should be possible. Of course it would be unwise to try to implement them all at the first try. A good idea is to start off with a basic set of data formats which will be supported. This set should incorporate some commonly used data formats. The basic set must be easy to expand later on, so this should be kept in mind when designing the system. A good option for an expandable design is the use of managers. Managers locate and assign resources so that only new resources have to be added to such a manager for a system extension.

To select the data formats for the basic set supported by the system it is a good idea to choose data format within the first application domain of the system. The application domain here will contain data formats applicable for data transformation and data formats used in the InDialog domain.

Data formats useful in the data transformation domain are: XML, XML Schema, XSLT and Xquery. Data formats useful in the InDialog domain are: TXT, CVS, Excel, MySQL database, XML, (X)HTML.

### 3.3.2 Intermediate data

The intermediate data format will have to be a fully open, structured and hierarchical format. This will ensure that the data is easy readable, easy to process and it will help future development and compatibility. Also a very large portion of the input and output data formats will in one way or another be structured or hierarchical.

XML fits these needs perfectly, XML is open, structured and hierarchical. The openness makes the data easy the exchange between all sorts of different systems. The structure and hierarchy makes the data easy to read, understand and process. Besides these features XML is already widely used in data transformation.

A survey has been done to determine what this intermediate XML should look like. First a view on data was divided into categories: actual data, structure, styling and layout. Different data contexts have different needs in these categories, text documents will need structure elements as paragraph, page, etc; a address file will need structure elements as name, address, phone number, etc. So will every specific data format add its needed elements to the intermediate XML. It is clear to see that this will not work out, endless elements will have to be added making the intermediate XML too hard to handle in every way.

Since one all consuming intermediate XML will not work out, the choice has been made to define the intermediate XML as a combination of a valid XML and its describing schema. The schema must contain clear element and attribute names which are unambiguous, so the context of the data will be clear for any user. This combination of a XML with a schema will give a clear intermediate data format, which will be easy to process. Also the schema can be useful for matching re usability, if a personal schema matches an already used intermediate data schema a stored mapping can be reused, letting the user skip parts of the matching process.

### Conclusion

*What are common / useful data formats?*

Data formats useful in the data transformation domain are: XML, XML Schema, XSLT and Xquery. Data formats useful in the InDialog domain are: TXT, CVS, Excel, MySQL database, XML, (X)HTML.

*What format can be used for the intermediate data?*

The combination of XML data and a XML schema defining the XML data. This will give a structured and flexible intermediate data format, which will be easy to process and reuse.

## 4. Main design

Now that the main functions and options of the system have been discussed a main design can be made. This main design will define the system at a global level, which will be the main reference for further design and implementation choices.

### 4.1 Scalability

When designing a system scalability is a desired property. Scalability can be defined as followed: “the ability of a system to accommodate an increasing number of elements or objects, to process growing volumes of work gracefully, and/or to be susceptible to enlargement” [21]

But scalability itself is not one single property. Scalability can be desired in different specific parts of a design. There are different types of scalability that apply to different parts of a system. Bondi [21] considers four general types of scalability:

- Load scalability  
*The ability to function gracefully, I.e., without undue delay and without unproductive resource consumption or resource contention at light, moderate, or high loads while making good use of available resources.*
- Space scalability  
*The ability to not let memory requirements grow to intolerable levels as the number of items that are supported increases.*
- Space-time scalability  
*The ability to continue to function gracefully as the number of objects that are encompasses increases by orders of magnitude.*
- Structural scalability  
The ability to not impede the growth of the number of objects it encompasses, or at least will not do so within a chosen time frame that are implemented or standardised.

In the main design of this prototype structural scalability is an important factor. Designing with structural scalability in mind aims to reduce costs and effort in long terms. Designing a system which only performs the present tasks needed can be made very efficient and effective on short term. But when new tasks arise in the future, adapting the system will proof to cost a lot more effort than it would have cost if more scalable design had been made on forehand. The cost of changes after release can be 60 to 100 times higher than changes during the definition phase. Changes during development can be 1,5 to 6 times higher than changes during the definition phase [22]. So it is fair to say that structural scalability pays off in long term.

To design for structural scalability means designing for present needs and keeping an open mind for future needs. This does not mean that a design should keep options open for every possible future change. That could just produce the inefficient design avoided. The aim is for a balanced design structure which is efficient for current needs but which is open enough to support changes in the future at low effort.

Such a balanced design can be helped by taking a good look at the system goals. When examining system goals it is important to realise what the short term goals are and what the long term goals are. Or to realise that some short term goals could need expansion in the future.

To make the prototype structural scalable the design should have a good dividing of functional elements. Dividing functionalities helps keeping a system manageable by creating different sub-systems. A sub-system performing a specific task is easier to replace and reuse. When looking at it from a black box point of view it can be replaced by any another component having the same in- and output definition.

Besides the replacement and reuse of elements it is useful to see which elements have to be expandable. When new input, output or processing types are needed a plug-in structure will have major advantages. This means that functional elements do not have to be adapted, but can be extended by placing new elements beside them which define new options within the existing system.

The features mentioned are concentrated on the single use of functional elements. There can however be different elements which have changing contents and still have to work together. When using a plug-in structure different output values might not be normalized. To keep this cooperation structural scalable a framework has to be set up which unifies different contents in such a way that they still can be used together. Such a framework should for example be able to combine and normalize different values. On the other hand it is also a good idea to define guidelines for the use of plug-ins. This will help to have more control over the plug-in interacting with the system. To create plug-ins within certain limits API's are commonly used. An Application Programming Interface provides a collection of definitions which can be used within save boundaries of the system consuming the plug-ins.

## 4.2 Data overview

Out of paragraph 3.1 an overall view of the data within the system can be made. The system will be fed a data input ( $d_{input}$ ) which will be read by the connection layer. The parser will then transform this data into a XML perspective ( $d_{raw\ xml}$ ) which is valid according to a specific schema ( $s_{raw\ xml}$ ). This schema can then be matched to a personal schema ( $s_{personal}$ ) which represents the data format wish of the user. The matching of these schema's will result in a specific mapping ( $m$ ) which will be stored for later reuse. This mapping will be used to transform the  $d_{raw\ xml}$  into an intermediate data ( $d_{intermediate}$ ) which will be stored for later output use.

The output of data is analogue to its input. The intermediate data is represented by the personal schema and the output data is represented by an output XML schema ( $s_{output\ xml}$ ). These schema's are matched and will result in a mapping. This mapping will be used to transform the intermediate data into a XML representation of the output data ( $d_{output\ xml}$ ) and will be stored for later reuse. The  $d_{output\ xml}$  can then be parsed into its output format and written to the data output ( $d_{output}$ ).

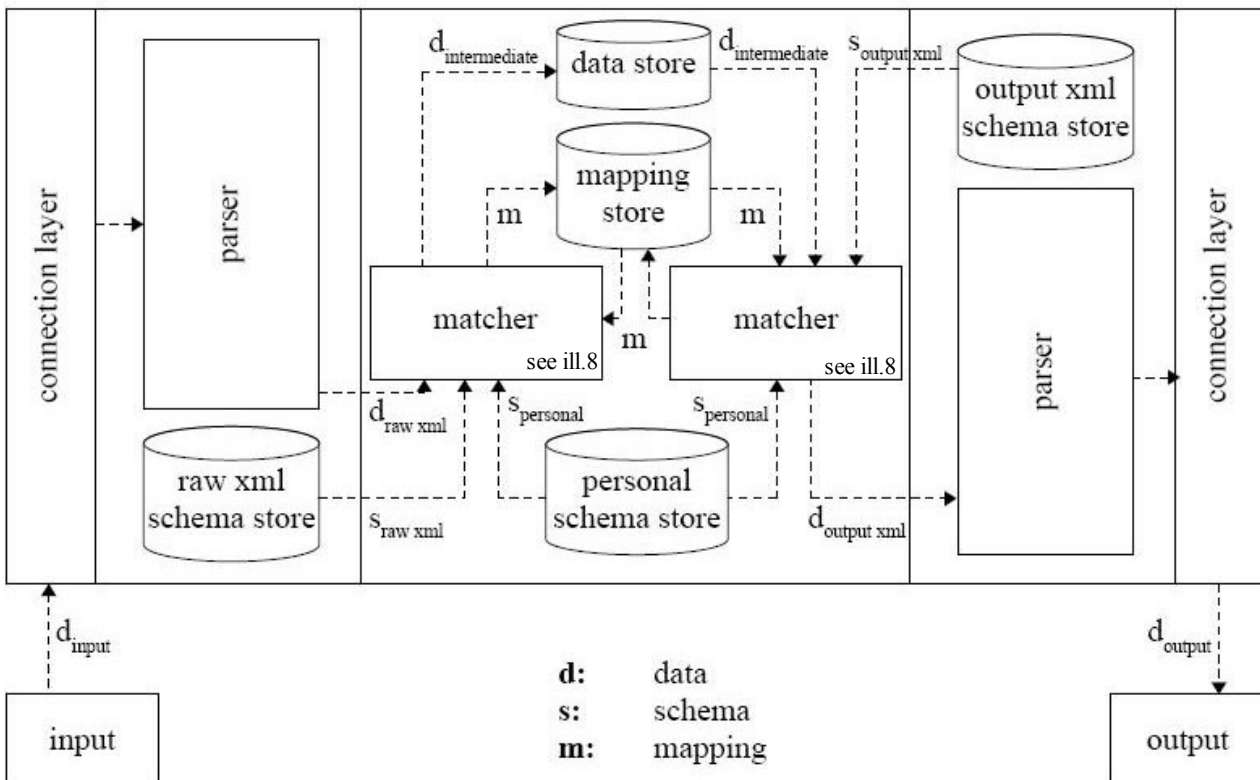


Illustration 5: Data overview

An example of this data overview can be found in Appendix B. Appendix B demonstrates a conversion from input to intermediate data using a TXT input format and a user defined XML personal schema. The actual format of the mapping used will be discussed together with the data store.

### 4.3 Data stores

The data stores are used to store data within the system for later direct or indirect reuse. Direct reuse can for example be seen as the full reuse of a raw XML schema for transforming input data. Indirect reuse can for example be seen as the reuse of element mapping information in a new mapping process.

The data will be stored in a relational MySQL database. Relational databases are fast and scalable. Since the system will not be working with large payloads it will not need a large commercial database. MySQL will suffice in the storing needs.

#### 4.3.1 Raw & output XML schema store

The raw XML schema store and the output XML schema store serve exactly the same purpose, so they will be identical of design. Therefore only the raw XML store is discussed here.

The raw XML schema store is meant for storing and retrieving the XML schemas which define the XML representation of the input data. These raw XML schemas define the output of the parsers who parse the input data to raw XML data.

The raw XML schema store only has to store XML schemas. These schemas refer to specific data formats which can be parsed by the parsers. So to retrieve the data it is wise to also store the reference between the schemas and the data formats.

The data format can be represented by a mime type. A mime type consists of a type and a subtype referring to a specific data format. Mime example: image/jpeg, text/plain, application/msword. When input data is read its mime type has to be extracted so that a valid parser can be selected which supports that mime type and thus outputs data according to the referring XML schema. To keep the schema easy manageable a name will be given to a schema for human reference. This results in the following relational database table:

<b><i>Raw XML schema</i></b>	
<b>id</b>	int(11) primary key
<b>schema</b>	text
<b>mimes</b>	text (comma separated)
<b>name</b>	varchar(32)

Table 2: Raw XML schema store



### 4.3.2 Personal schema store

The personal schema store is meant for storing and retrieving the personal schemas which define the intermediate data format. The personal schemas are used in the mapping process, they define the target schema on the input side of the system and the source schema on the output side of the system.

The personal schemas only represent internal data of the system, so it does not need any outside reference like a mime type. To keep the schema easy manageable a name will be given to a schema for human reference. This results in the following relational database table:

<b><i>Personal schema</i></b>	
<b>id</b>	int(11) primary key
<b>schema</b>	text
<b>name</b>	varchar(32)

Table 3: Personal schema store

### 4.3.3 Data store

The data store is meant for storing and retrieving the intermediate data. The intermediate data consists of two parts, the XML data and a XML schema defining it. Since the XML schema is already stored in the personal schema store only a reference to it has to be stored with the XML data. To keep the data easy manageable a name will be given to the data for human reference. This results in the following relational database table:

<b><i>Data</i></b>	
<b>id</b>	int(11) primary key
<b>ps_id</b>	int(11) foreign key
<b>data</b>	text
<b>name</b>	varchar(32)

Table 4: Data store

### 4.3.4 Mapping store

The mapping store is meant for storing and retrieving mapping results and is thereby the source of re usability in the matching process. The mapping store must be able to provide the system with useful information for matchings. To provide this information the mapping store must contain the mapping results but also auxiliary data like a dictionary or thesauri.

It must be possible to get relevant data out of mapping results. This means that it must be possible to retrieve partial data out of a full mapping result so it can be matched with a particular matching case at hand.

Further insight about the mapping store ((partial) matching reuse and thesauri/dictionaries) will be given in the iteration steps.

## 4.4 Element overview

The system consists of a few key elements, these elements and their functions and options will be discussed. Important features are re usability and user friendliness

### 4.4.1 Connection layer

The function of the connection layer is to provide a connection object on request. By providing a specific connection type and url in that request a connection manager will select a specific connection driver. The connection driver will return a connection object for that specific connection type able to perform functions necessary within that connection domain.

The connection manager receives connection requests with containing a type and a url. Type describes the source type, like 'file' or 'mysql'. The url describes the location of the source. A file type for example can be located locally '/var/www/html/index.html' or remote '<ftp://www.indialoog.nl/index.html>'. Both are file types but they require a different connection driver to connect to.

Drivers can be registered in the connection manager. The connection manager searches within these registered drivers for a suitable driver for the connection request. The chosen driver will return a connection object for that connection type. The connection object contains functions for retrieving data out of the connection through the driver.

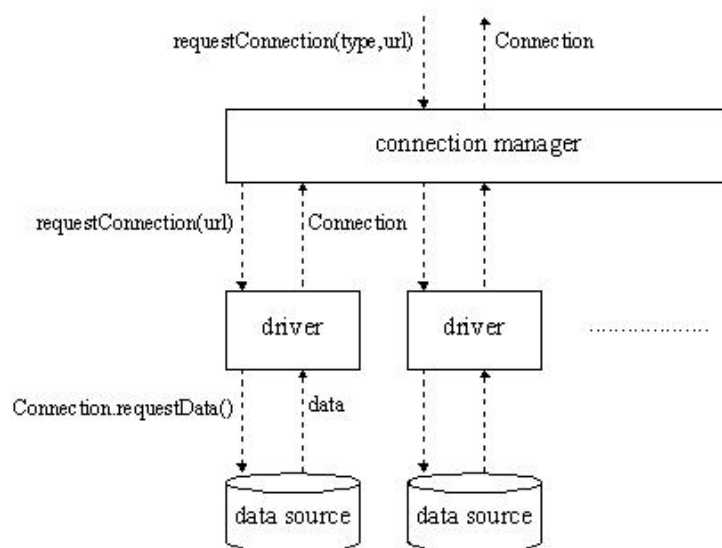


Illustration 6: Connection Layer

The driver is responsible for the retrieving of the data. For the data transformation we are not only interested in the data but also in possible meta data. Meta data can give extra information about the actual data which can be useful for the data transformation or for the end user. The aim is to get as much meta data out of a data source as possible. This means that functions for meta data retrieval must be present within the connection object/driver. Different data sources will have different meta data, so a when a driver is build an overview must be made of which meta data will be available.

#### 4.4.2 Schema valid parser

The function of the parser is to read the input data through the connection manager and parse it into raw XML data. The parser is called with a connection, the parser manager identifies the connection type and selects a parser suitable for this type. The parser is fed the data and meta data ( $md_{input}$ ) from the connection which will be parser into a raw XML format. This raw XML data will be valid with a raw XML schema included in the parser. The parser will return the raw XML data and schema as its result.

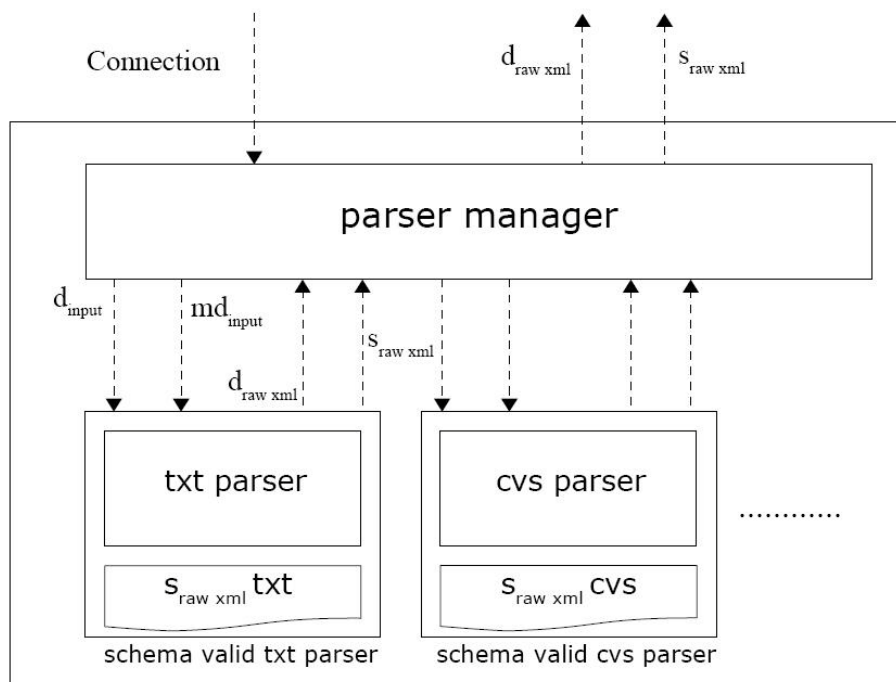


Illustration 7: Schema valid parser

#### 4.4.3 Matcher

The function of the matcher is to match the raw XML schema with the personal schema. This can be done in different ways, manually with the aiding of a graphical interface and automatically by using automated matchers and auxiliary data. Both of these ways can influence one another by using iterations, so they must both have the same definition of what a matching is.

The matching proposed is a flexible combination of schema matchers aided by a graphical interface. The matcher combination approach from the COMA [16] system has been adopted. Automated matching will generate a mapping, if this mapping is correct can only be judged by the end user. So after the mapping process it must be possible for the user to give feedback and adjust the mapping ( $m'$ ). In a simple situation this can be a onetime adjustment, but in more complex cases this can be a process with many re matchings and feedbacks. To implement this no serious changes have to be made in the system when using match iterations. In every iteration the same matching and feedback loop can be walked through similar to the single run matching. As a part of the feedback the user can already define certain matches ( $m''$ ) or exclude parts of the matchings ( $s'_{raw\ xml}$ ,  $s'_{personal}$ ) to reduce matching complexity. To aid the manual adjusting a graphical interface can be used to shorten the human interaction time as well as the comprehension of the mapping result.

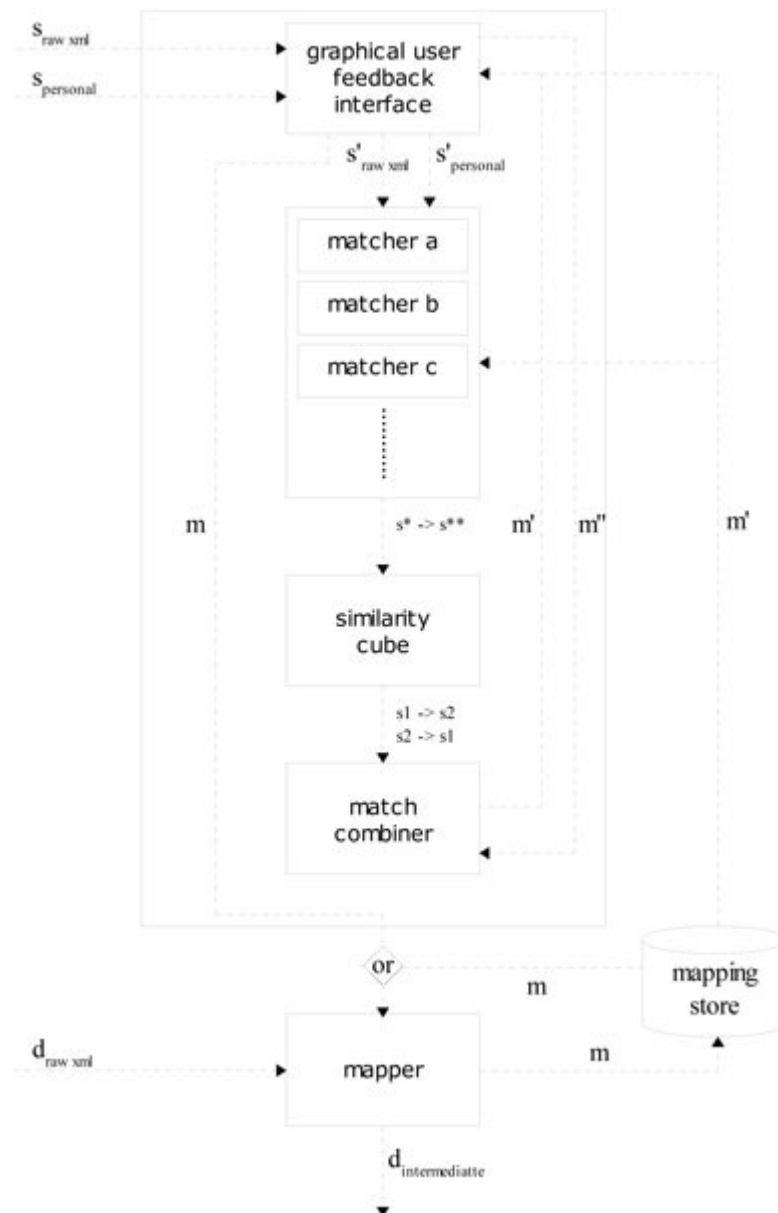


Illustration 8: Matcher (see ill.5)

The combined matchers will give different matching results. Individual results can be combined and corresponding values can be ranked by similarity, this can be done by using a similarity cube. The similarity cube will create two sets of match results (schema1-->schema2 and schema2 --> schema1) defining the match candidates per schema element. Out of these two schemas a combined result mapping can be produced by aggregating the individual similarity values using chosen similarity strategies.

The result mapping can be used to transform the  $d_{\text{raw xml}}$  into  $d_{\text{intermediate}}$ . Also the mapping will be stored in the mapping store so it can be reused later. Partial or full mappings out of the mapping store can be feed to automated matchers or can be used for manual processing. A part of these partial mappings are mapping which are present in the dictionary or thesauri.

## **Conclusion**

In this chapter the main design of the system has been discussed. The design has been made at a global level, which contains all main elements necessary for the functionalities within this project. The main design is set up with large functional blocks which can later on be further designed at a more detailed level.

The main design (ill. 5) is functional symmetrical so functionalities will be easy to reuse on both the input and output side of the system. The connection and parser manager provide a flexible environment where new connections and parsers can be added to the system relatively easy. The matcher is designed to contain the possible future needs for flexible matching so no big changes will have to made later on.

These features provide structural scalability within the design of the system at a global level, which will be a good base to start from for now and for further development.

## **5. First iteration, basic system**

This chapter describes the first development iteration of this project. The goal of the first iteration is to build a base system to use in the further development. Important in this step is to keep an open structure in mind that can be used to interchange different elements later on.

Another important goal of the first iteration is getting a better view of what are good and what are bad options for efficient data transformation.

### **5.1 Goals / Requirements**

The first development iteration has the following goals:

- design and implement a basic transformation system capable of simple manual matching
- supported input data formats must be: txt, cvs and relational database
- getting as much as possible information out of input data
- ability of storing and recovering internal data for non-partial re usability
- supported output data formats must be: txt, cvs and relational database
- putting as much as possible information into the output data

### **5.2 Required components**

The following system elements will have to be designed and implemented to let the system meet its goals:

- a connection layer which also reads possible environmental meta data
- a parser able to parse txt, cvs and relational database data with an output valid according to a specific schema
- a schema store holding the parser schema's
- a schema store holding personal schema's
- a manual matcher, outputting intermediate data and the mapping used
- a data store holding the intermediate data
- a data store holding the mappings

The elements mentioned are able to transform the input data to the intermediate data. Since the system is symmetrical the same elements will be needed to transform the intermediate data to output data.

### **5.3 Research questions**

Most of the main elements in the design are already designed in chapter 4, namely the stores, the connection layer, the parser manager and the matcher. In this first iteration a simplified manual matcher will be used.

This leaves the following parts open for research/design:

1. parser rules and schemas with meta data for txt, cvs and relational database
2. mapping format
3. manual matcher
4. non-partial re usability design

The following research questions are constructed. A numbered reference is made with the list above:

1	What specifications make a parser a valuable addition to this system?
2	What are key features for a mapping format? What mapping format should be used in this system?
3	What functionalities are necessary with manual matching? How can manual matching be tested and valued?
4	What is important data for non-partial re usability? How can non-partial re usability be offered to the user?

## **5.4 Design**

The four system parts named above will be discussed with respect to the research question named.

### **5.4.1 Parser rules, schemas and meta data**

*What specifications make a parser a valuable addition to this system?*

The function of a parser is to parse the input data format to an output data format. To do so a set of rules must be defined which describe how the input data is parsed to the output data. The appliance of these rules must produce a result valid according to a specific schema, so that consistency is guaranteed.

Extra benefit can be gained in including meta data in the parsing process. This meta data can describe environmental knowledge not existing within the data. Thus the parsing of meta data can hereby provide extra insight in processing the output data.

Meta data will differ between different data sources. To be able to process the meta data it must be stored in a uniform way, independent of the actual data stored. This is done by defining a meta element at the top of each schema. The meta element can contain any number of meta data represented by a key and value tuple. The meta data is defined as followed:


	<pre> &lt;meta&gt;   &lt;key title="keyname"&gt;data value&lt;/key&gt;   &lt;key title="keyname"&gt;data value&lt;/key&gt;   ...   ... &lt;/meta&gt; </pre>
---	---

Table 5: Meta data

Parser rules define how source data is parsed. Parse rules are triggered by a specific sequence and generate predefined output data. The parsed output data will be valid to the corresponding XML Schema ( $S_{\text{raw xml}}$ ) of that parser. The parser rules, schemas and meta data can be found in Appendix C.

### Conclusion

*What specifications make a parser a valuable addition to this system?*

A parser in this system gives extra value in being schema valid. The parser provides a schema on the parsed data which can be used in the further transformation matching. Extra meta data produced by the parser gives extra information about the data source and reduces for user interaction.

### 5.4.2 Mapping format

*What are key features for a mapping format?*

*What mapping format should be used in this system?*

The data transformation is defined within a mapping. This mapping defines how a target element is formed out of a combination of source elements, conditions and functions. This mapping has to be stored within a defined mapping format which has to be structured and is fairly easy to process. XSLT is such a mapping format, it is a widely used transformation language. XSLT is based on XML and is thereby well structured, human readable, easy to process and widely supported.

The resulting XSLT can be stored for later reuse if the same transformation has to be redone. Because of the XML format of XSLT it is probably very good possible to retrieve parts out of the full XSLT for reuse.

### Conclusion

*What are key features for a mapping format?*

A mapping format should be capable of mapping a source format to a target format. In practice this means that a mapping format should be a flexible format to be able to serve a large range of source and target data. This flexibility will also mean a more easy interaction with other systems.

*What mapping format should be used in this system?*

XSLT should be used as the mapping format of this system. It is widely used, based on XML and is thereby well structured, human readable and easy to process. This will also provide easy interaction with other systems of InDialog.



### 5.4.3 Manual matching

*What functionalities are necessary with manual matching?*

*How can manual matching be tested and valued?*

The first step in the matching process is a simple manual matching. This can later be expanded to more complex iterated matching with automated functions and user interaction.

The matching used is a schema matching which matches target elements from one schema to source element from another schema. In this simple matching rules are applied that define the transformation of one source element to one target element with possible application of a condition or a function.

rule = (source element, condition, function, target element)

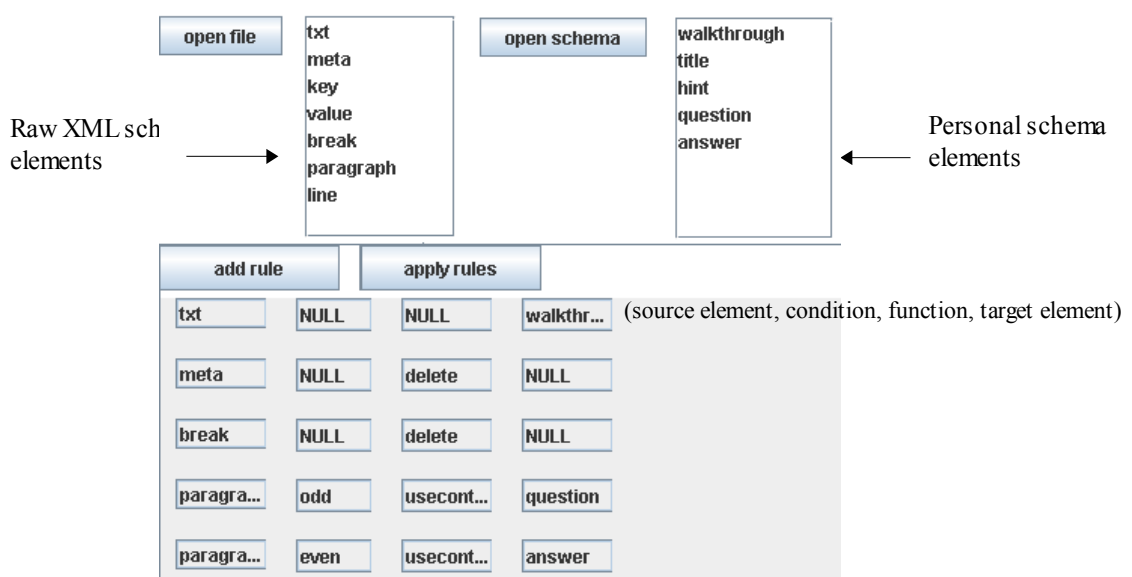


Illustration 9: manual matcher rules

These rules can be applied directly without generating XSLT in between. This is done in the first iteration to save time and to research to benefits and shortcomings of manual matching.

### Conclusion

*What functionalities are necessary with manual matching?*

Manual matching must provide a way of defining a mapping between a source and target schema. This can be done by defining connection between source and target elements. Functions and conditions expand the expressing possibilities of the matching.

*How can manual matching be tested and valued?*

Since this is the first iteration and the matching will be further developed, no extensive testing will be done on this manual matching. An evaluation is given in 5.6 based on personal interaction with the system by the domain specialist.

#### 5.4.4 Non-partial mapping store reuse

*What is important data for non-partial re usability?*

*How can non-partial re usability be offered to the user?*

The mapping produced can be stored so that when the same input schemas are encountered it can be reused. The mapping is a product of the combination of the raw XML schema and the personal schema so the mapping reuse can be stored as a link between these two.

Full mappings can be stored as data as shown in table 6. To keep the data easy manageable a name will be given to the mapping for human reference. The naming is very important for later reference, so user must be encouraged to use naming that expresses the meaning of the mapping.

<b>Mapping</b>	
<b>id</b>	int(11) primary key
<b>xslt</b>	text
<b>name</b>	varchar(32)
<b>rxs_id</b>	int(11) foreign key
<b>ps_id</b>	int(11) foreign key

Table 6: Mapping store

The related id's of the personal and raw XML schema used to generate the mapping are also stored for later references, because multiple mappings are possible between the same schemas. Multiple mappings between these two occur when different interpretation are asked form the same data.

The schema relation within the mapping table can be used to offer the user an insight of possible mapping that can be reused. By querying the mappings with the id's from the personal and raw XML schema used at that time. Or it could even be used to give a suggestion on what schemas to use with certain mappings.

This suggests that this will only work when the user wants to reuse a mapping without any changes. This is only true when the mapping is processed directly after selection. It is also possible to load the mapping and the related schemas so that the mapping can be edited before processing it. This will save the user a lot of time when alike mappings are needed. The user can then load a mapping, slightly alter it and process or save it. Only the time for the alterations is needed, main mapping time is reduced to a minimum.

#### Conclusion

*What is important data for non-partial re usability?*

Important data for non-partial re usability is the XSLT data and the schemas used to create the mapping. The total of these three must be named so an easy reference can be made when mapping are reused.

*How can non-partial re usability be offered to the user?*

Non-partial re usability can be offered to the user in two forms. By loading a mapping and directly processing the mapping. Or by loading a mapping and giving the user the opportunity to edit it before processing or saving it.

## 5.5 Implementation

For the implementation Java is used as programming language. In addition to the common packages extra XML and w3c libraries are used to support the processing of XML data.

For the processing of the XML data the Document Object Model is used instead of SAX. This is done because DOM is easier to use because it can access tree nodes at random whereas SAX goes through data sequentially. This will make implementing more flexible although SAX is potentially faster. Although SAX might be faster, DOM is now chosen above it, because the system will not require an optimum working speed. This is because of the incidental need for transformation within InDialog. When optimization is needed a change to SAX can be suggested.

The Class Diagram of the connection and driver elements of the basic system can be found in appendix D.

## 5.6 Evaluation

*Design and implement a basic transformation system capable of simple manual matching.*

The simple manual matching implemented with the use of rules works good for simple matchings, but leaves little room for more complex matchings. Only 1:1 Element matchings can be done and there is no way to aggregate matching elements.

A good option would be to step away from the rule approach and use an object oriented approach to the matching. The matching elements can then be seen as object which can be connected to one another to present a more flexible way of building matchings. This will make aggregation possible and 1:n and n:1 cardinality. To incorporate this approach with the XSLT mapping format, these matching element can represent XSLT element from the XSLT standard.

*Supported input/output data formats must be: txt, cvs and relational database.*

The supported data formats work well and are easy to expand by adding new parsers and connections. Specific parsers and connections extend from their parent object so they are easy to implement, the including of new parser and connections can be made even easier by the use of dynamic class loading. Dynamic class loading makes it easy to introduce new classes by other parties.

*Getting as much as possible information out of input data.*

Getting meta data from the connection in the parsing process proves to be an easy way to get basic environmental data. It would be useful to make the manual adding of meta data available, so that more semantic values are added.

*Putting as much as possible information into the output data*

Meta data is now put into the parsed data but they are potentially lost in the resulting data. To reduce lost of semantic data a construction has to be found to store the meta for the processed data. This may be done within the data itself which will make it easier to access but may not be possible if it compromises the schema definition of that data. Otherwise a linking will have to be made to keep the reference between data and meta data.

*Ability of storing and recovering internal data for non-partial re usability.*

The storing of mapping results linked to their source schemas used works well when using exactly the same mapping multiple time. When variations have to be made it is time consuming when a totally new mapping must be made. So a good new option would be the possibility to derive new mappings from existing ones.

Non-partial re usability will contribute very little to none to automated schema matching because no individual element can be retrieved. The next logical step is to expand re usability by adding possibilities for partial recovery of mapping results. As a part of partial mapping recovery a dictionary or thesauri can be introduced to aid automated matching.

## 6. Second iteration, Visual XSLT

This chapter describes the second development iteration of this project. The goal of the second iteration is to expand the base system to improve efficiency in matching. To improve this efficiency the development of visual aiding has been chosen.

The development in visual aiding has been chosen above the choice of automated matching and re usability. This is done because of the environment in which the application will be deployed. The application will at first run in an environment where mappings will be relatively small and processing will be at a low frequency.

In this setting visual aiding in the interface will most likely create more direct efficiency than automated matching and re usability. So visual aiding is given preference in this iteration.

### 6.1 Goals / Requirements

The second development iteration has the following goals:

- evaluate good and bad points in visual aided mapping
- design and implement a visual aided matching environment
- testing and evaluating the visual aided matching implementation

### 6.2 Visual aiding survey

Before the specifications can be described a better insight on visual aided matching has to be gotten to determine what elements have to be incorporated. To get this insight on visual aided matching a survey has been done. The object of this survey is to get a view of systems on the market and their workings. The good and bad points will be listed resulting in an advice which elements to use in the second iteration.

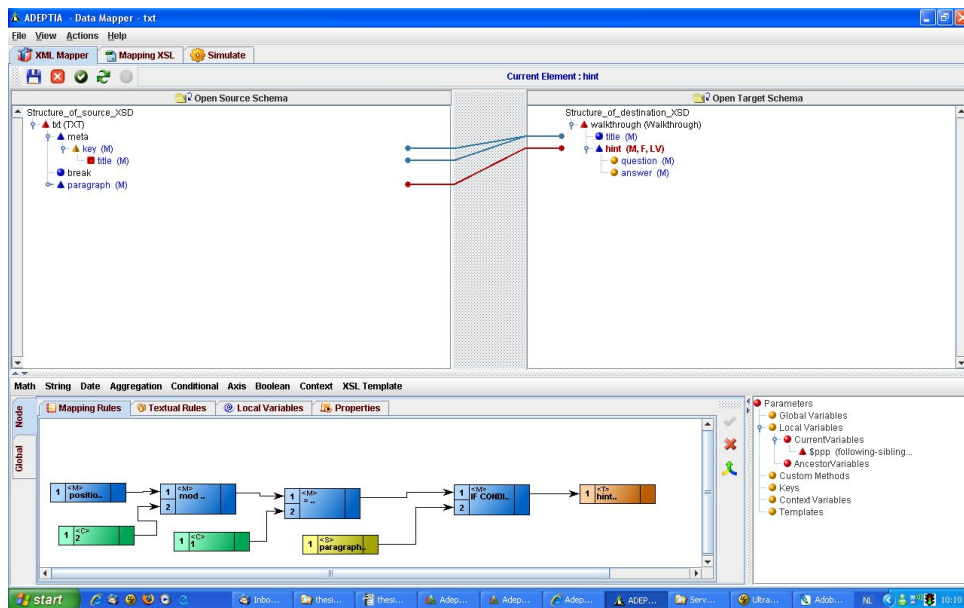
In this survey two widely used commercial matching programs will be evaluated. The 'Data Transformation Server' from Adeptia and 'Stylus Studio 2007 XML' from Stylus Studio. Both offer a wide range of data transformation tool and support them with visual aided mapping.

#### **Adeptia**

The visual matching by Adeptia is done by the hierarchical representation of the source schema on the left and the target schema on the right. The elements within the schemas are connected by lines only. The lines represent rules which are further specified within a different panel.

Adeptia defines four types of rules:

- mapping rules,       made up of visual xslt building blocks (nodes)
- textual rules,       made up of textual xslt rules
- local variables,     made up of xslt variable definitions to be used in other rules
- properties,         provides the for-each functionality



Adeptia provides a large number of nodes for the visual mapping rules, this reduces the need for the manual typing of rules. The nodes provide a preview of their working. If the node is connected the connections are used in the preview. The preview of node workings can be a time saver because a user doesn't have to wait until completion to evaluate its result. It is possible to set the value of a connection instead of really connecting a node, which makes it possible to manually add values or functions which otherwise would need a lot of extra nodes.

It is possible to define templates which contain predefined matchings. The use of templates will create re usability and a less complex visual view. Also global XSLT data can be defined which can be placed at the beginning or the end of the resulting mapping. This can be a good feature for re usability when data contains fixed headers and footers.

The for-each node is not visually displayed but resides within the property rules. This makes very little sense because all other XSLT element can be found within the visual mapping rules. Beside the fact that this is not logical for the user, it emits the use of the for-each node within a mapping, thereby reducing flexible looping of elements.

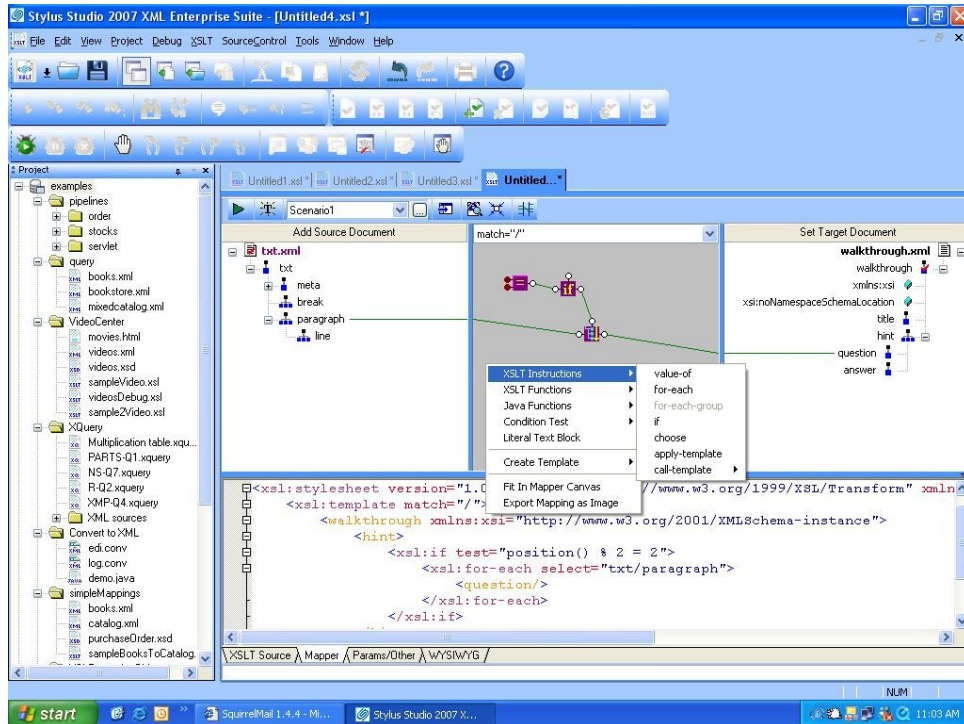
The nodes in the mapping rules receive inputs (attributes, nodes, values) and produce outputs (XSLT elements). This produces an intuitive flow from source to target, but also depicts a linear approach. It focuses on the generating of XSLT elements and their content, this excludes the view on the exterior of XSLT elements.

In some cases it is more intuitive to approach a mapping with an external condition. Such a problem will also be solvable by using only internal conditions, but this will force a user in a specific modelling pattern which can be less efficient, comprehensible and flexible.

The visual matching produces a XSLT code as its result. Although it is possible to alter the generated XSLT code, it is incapable to translate the alterations back to a visual matching. This is not a very important feature because this direction is not taken in the normal work flow. But it might be of value when trying to fine-tune matchings.

## Altova

The visual matching by Altova is done by the hierarchical representation of the source schema on the left and the target schema on the right. The elements within the schemas are connected by lines and nodes. Lines represent the connection between source elements, nodes and target elements. Nodes represent XSLT elements (instructions, functions, conditions).



Altova does not define textual rules, but does support the use of templates. All of the matching is done visual which works very easy, unfortunately not a very large set of nodes are presented. The missing of axis nodes (following, sibling,...) is especially a big loss, because they provide many relational functionalities. The XSLT generation of the visual matching can be previewed directly so results can be judged immediately.

Besides the standard connection for input and output the nodes can be extended with other possible connection ports like sorting ports. Another important feature of the nodes is the presence of a so called 'flow port'. Each node can posses one flow port which defines the exterior condition of a XSLT element. This makes it possible to set a condition on the external side of the current node. The possibility of using both inside and outside conditions gives a more flexible and less linear approach to the visual match building.

Altova can transform the visual matching to XSLT, but is also capable of transforming valid XSLT back to a visual matching. This does not help in the normal work flow, but can give extra insights when making adjustments.

Since all the nodes are displayed between the source and target elements this will soon become complex. A solution given to reduce this complexity is the possibility of hiding links. This does reduces complexity but also removes the global match overview.

## Conclusion

Both of the systems reviewed have their advantages and disadvantages. To give a good overview of both, the pros and cons of the visual matching are displayed in the table below:

### *Adepatia*

<b>pros</b>	<b>cons</b>
hierarchical schema presentation	non-consistent for-each construction
non-complex use of linking lines	linear, content oriented view
easy to use visual mapping rules	four different rule options to manipulate the matching
large amount of nodes to use	
node previews	

### *Stylus*

<b>pros</b>	<b>cons</b>
hierarchical schema presentation	small amount of nodes to use
easy to use visual mapping nodes	
node previews	
flexible, non-linear view by the use of flow ports	
two way transformation possible	

Concluding, good options for visual aided matching are:

- a hierarchical schema presentation
- non-complex linking
- visual mapping nodes for all possible XSLT elements
- code previewing
- the use of flow ports

## 6.3 Required components

The following system elements/functionalities will have to be designed and implemented to let the system meet its goals:

- visual mapping node object, representing XSLT element
  - containing linkable input, output and flow ports
  - capable of code generation
- hierarchical schema presentation with schema elements
- linking between schema elements and visual mapping node objects



## 6.4 Research questions

1. How can the XSLT standard be transformed to a uniform valid visual representation?
2. What properties are necessary for user and system functionality on visual mapping nodes?
3. What possibilities and constraints must linking incorporate?
4. How must (nested) linking be handled?
5. How can the user be aided in its visual interaction?

## 6.5 Design

The elements and functions mentioned in the previous chapter will be designed. The main goal of the visual elements is the aiding of the efficiency for the system user. Because of this goal not only technical aspect will be looked at, but also user interaction related issues.

### 6.5.1 Visual elements

Visual mapping nodes are the main building blocks for visual aided mapping. The nodes describe the XSLT transformations that will be performed. Therefore the nodes must be a logical presentation of XSLT elements. Another important aspect is that the nodes must have a clear way working and connection with each other. This must guarantee an easy working for the system user.

First an analysis is made of XSLT 2.0 to see how it is build up and what characteristics are involved. After that a visual interpretation can be designed which supports both XSLT and a user friendly interface.

### XSLT characteristics

*How can the XSLT standard be transformed to a uniform valid visual representation?*

*What properties are necessary for user and system functionality on visual mapping nodes?*

A XSLT 2.0 element format will look like the following example which is displaying the for-each element:

```
<xsl:for-each
  select = sequence-expression>
  <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each>
```

All XSLT elements are in fact XML elements, thus they have an opening and closing tag which begin with 'xsl:' defining the XML as a XSLT element. After the ':' both tags display the element name.

The opening tag can include zero or more attributes which are defined by  $[name] = [type]$ , where  $[name]$  is a string defining the attribute name and  $[type]$  is the type of the attribute. Predefined types are:

type	meaning
expression	regular expression resulting in an boolean value
sequence-expression	XPath expression value
node-sequence-expression	sequence of nodes
sequence-constructor	sequence of sibling nodes

Other types are: char, string, string options, QName, uri-reference.

Next to the attributes included in the tags content can be placed between the opening and the closing tag. This content can contain fixed XSLT elements for that specific element and a sequence constructor.

Next to the attributes and types cardinalities can be used. This is done by the usual cardinal identifiers \*,+,?. For example `xsl:sort*` means zero or more sort nodes can be connected.

### Visual node

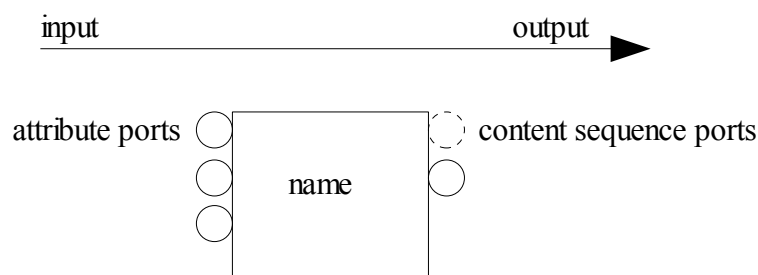
*What properties are necessary for user and system functionality on visual mapping nodes?*

*How can the user be aided in its visual interaction?*

A visual node can now be defined according to the XSLT characteristics. The XSLT tag name can be defined as the node name, the attributes can be defined as input values and the content can be defined as output values of the node.

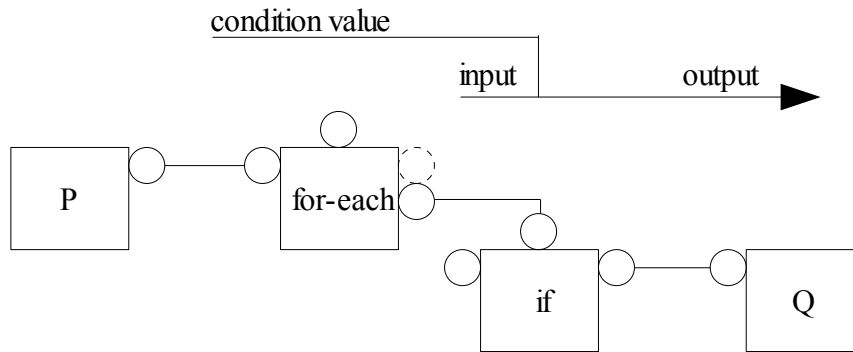
The input and output values from different nodes can be connected with each other to build up a more complex mapping. The connection points of these values shall be called 'ports' from now on. Ports connected nodes and give their output value to the connected input port of the connected node.

To make node connection good understandable for users the input ports will be placed together on the left side of a node and output ports will be placed together on the right side of a node. This is done because the source schema will be placed on the left side and the target schema will be placed on the right side. This creates the global understanding that data flows from left to right, from input to output.



Input ports always define attributes of an XSLT element. An output port can be a sequence of fixed XSLT element specific for that node. An example of this is the use of `xsl:sort` within the `for-each` element. A sequence of these specific XSLT elements must appear before the sequence of the containing element.

External condition values can be added by the use of a so called flow port. By the use of flow ports more complex mappings can be made with the visual nodes. The flow port defines in what case the current node will be applied and is located at the top of the node. For example when a `for-each` node must be executed under a specific condition the flow port of the `for-each` node can be connected to the content sequence port of an `if` node.

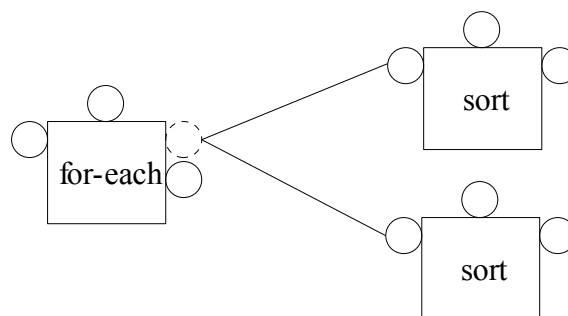


This will result in all P elements being iterated and mapped to Q if the if condition holds. This mapping can easily be switched by the flexible use of the flow ports. If the `for-each` and `if` node are switched a subtle difference results in a completely different mapping result. The P elements are then iterated only when the if condition holds.

The use of the flow port gives room to subtle but easy changes in the mapping, making the combination of nodes more flexible and leaving more room for user interpretation.

The ports of an element can be connected to other ports to add more mapping nodes. In some cases it is more efficient to add the value on a port manually. The manually adding of values will provide more efficiency to users with the understanding of XSLT. Instead of adding multiple nodes a user can add a value in the form of a constant or a XSLT expression.

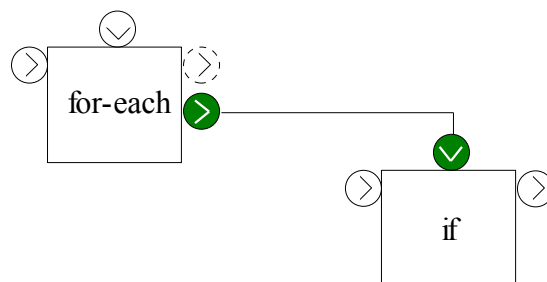
Multiple connections can be made on a port if its cardinality allows it. The port cardinality is obtained directly from the XSLT definition of an element. The default cardinality is 1.



To give a recognizable visualisation to the ports and their connection/value the following visualisation will be used:

<i>port type</i>	<i>visual</i>	<i>reasoning</i>
not connected, fixed	○	the open circle represents it is still open for connection
not connected, optional	⊖	the dashed line represents the choice to use it or not to use it
connected, to port	●	the closed circle represents it is closed for connection, green represents a successful connection
connected, set value	●	black (node background) represents that the value is part of the node

Also the direction of ports can be displayed visually so that the user can instantly see if two ports can be connected or not. This is done by adding arrows in the ports. The user can then make correct connections by adding an outward pointing arrow to an inward pointing arrow. This enforces the global understanding that data flows from input to output.



### **Hierarchical schema presentation and cardinality**

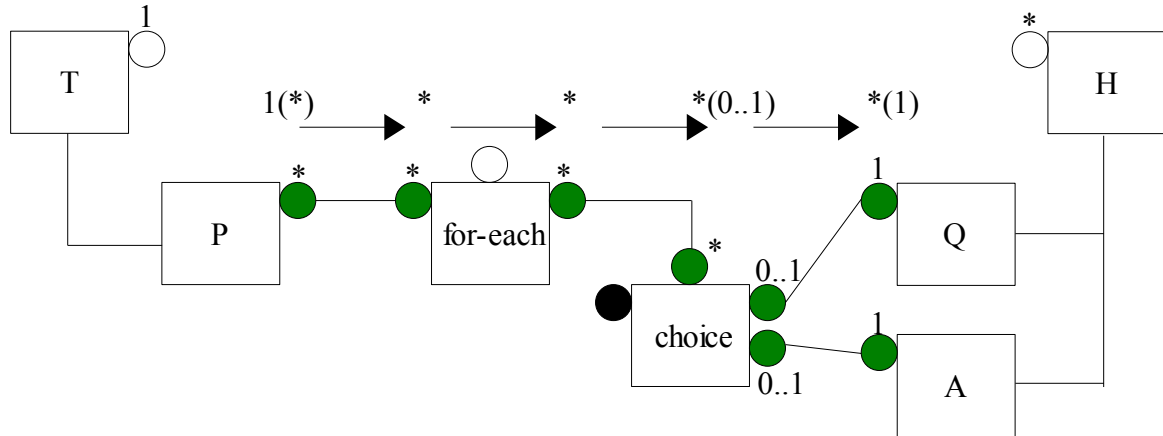
*What possibilities and constraints must linking incorporate?*

For the source and target schema a visual representation has to be made. A good visualisation of these schemas is a tree representation. Since the build-up of a XML schema is in fact a tree structure this complies very well with the visual understanding of a user.

In this tree representation elements and attributes of the XML schema will be represented hierarchical. Next to the visualisation of the tree structure it is possible to represent the cardinality of the elements used. The visual representation of the cardinality will give the user more insight in the working of the schema and prevent the user from making an invalid mapping. Many mapping tools can make mappings that result in an invalid XML data according to the target schema.

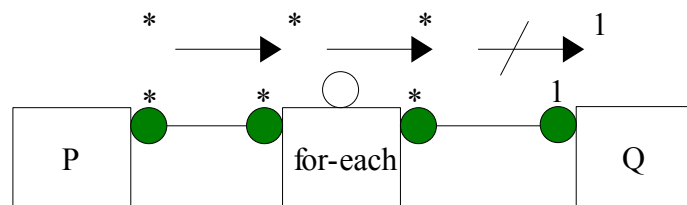
Checking if the cardinality results in a valid XML while building the mapping is not possible without generating the XSLT code, generating the output XML and validating it. This will probably result in a bad performance and may not always be wanted while the user is in the building process. Although this is the best way to check the validity it is not very practical within the building process.

A good alternative is to give an advice on possible mismatches in the building process and give a validation result on completion. A validation result can be made by a schema validator, which are widely available. For the mismatch advice cardinalities must be passed through from the source to the target side within a mapping and checked on the possibility of invalidity. Only an advice can be given because only the cardinalities and nodes workings can be checked in this way and the input data is left out in the equation.



The example above describes a possible match in cardinality. On the left side P can exist \* times in 1 T, this is expressed as  $1(*)$  which can be reduced to \*. The for-each node loops the \* P elements which does not change \*. The choice node gives is an optional switch for input elements, this means that an input element is chosen or not. This result in a 0..1 cardinality for choices, but because this is done for \* elements the result cardinality is expressed as  $*(0..1)$ .

On the right side both Q and A can occur once in \* iterations of H, which is expressed as  $*(1)$ . Now  $*(0..1)$  and  $*(1)$  can be validated, since  $*(0..1) \leq *(1)$  this mapping can be validated as a possible match since it might still be possible that the choice always goes one way which will result in a  $0 \neq 1$  mismatch.



This example demonstrates a mismatch. Because the cardinality of P is \* and it is iterated by the for-each node the cardinality still remains \*. The cardinality of Q is 1, so validating the mapping will result in  $* > 1$  meaning there is a mismatch. But when the actual data only contains one P element the result can in practice still be valid.

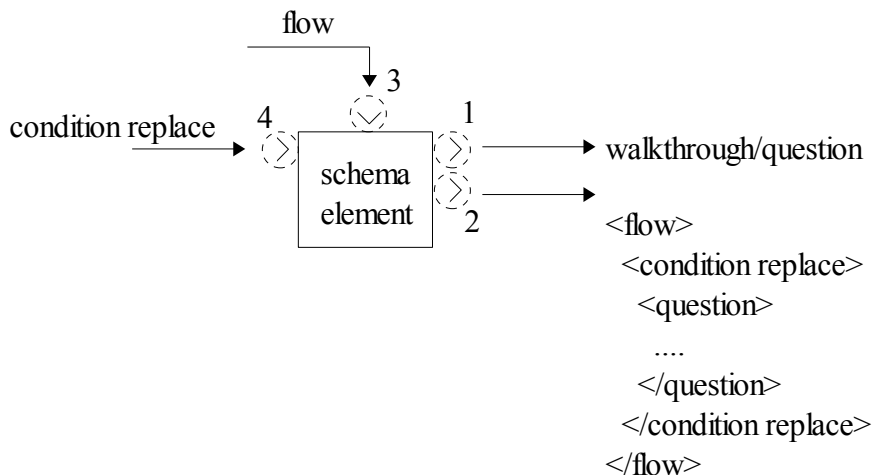
### Schema elements

The visual representation of schema elements displayed above suggest they have output ports on the source side and input ports on the target side. This will in practice not work because schema elements can be used in different representation settings.

A schema element can be used in the following settings:

setting	example
1. returning its path value	text/paragraph
2. returning its tag value (content sequence)	<paragraph>.....</paragraph>
3. defining a external condition by use of a flow port (viewed from question element)	<for-each select="text/paragraph"> <question>....</question> </for-each>
4. getting a inclusion in a parent element (viewed from walkthrough element)	<walkthrough> <question>....</question> </walkthrough>

A schema element can then be made in accordance to the working of a visual XSLT element. Settings 1 and 2 represent content-sequence ports since they produce an output result. Port 3 represents an attribute port which is responsible for the condition inclusion for itself in the result. This can be viewed as a replacement port, which replaces the normal returning tag value with an extension of the connected condition. Port 4 represents the flow port.



Port 2 can for example be connected to the flow port of a XSLT element returning a string value so that the dots in the example above are filled with a certain value. Also other elements can be connected in this way. This makes it possible to connect inside and outside of elements which makes the possibilities for the user very flexible.

### Linking and code generation

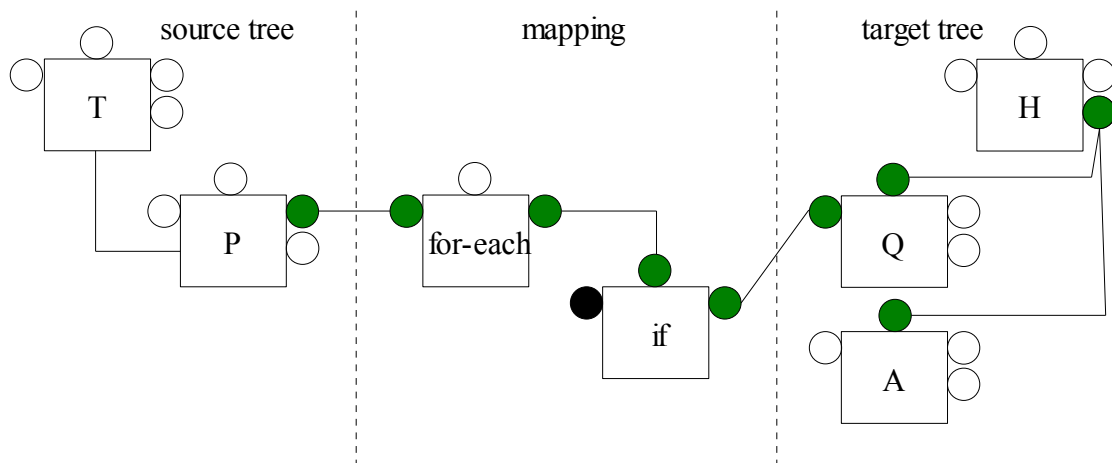
*How must (nested) linking be handled?*

Nodes can be linked to each other by their ports, but not all ports can be connected to each other. Ports can only be linked if the produced output of a port matches the required type of the linked input port. So for every port the accepted or produced type(s) must be defined. This is a fairly easy concept and will not be discussed further.

Less trivial is the generation of the mapping code (XSLT) out of the linking between the source elements, nodes and target elements. A mapping may consist of a lot of linked nodes which do not have a linear path from source to target. How must the code be generated and in what order must the nodes be followed to get the mapping which was intended by the user?

Since the target schema is the goal of the mapping process it will serve as the guiding principle for the code generation. The code generation will start at the root element of the target schema and will follow its path down in hierarchical order. When a linked element is encountered the links have to be walked through from right to left to determine the code generation for that target element.

To generate the code out of the linked nodes a recursion has to be done over the flow ports. Attribute ports only support data for the attributes on the tag. Output and flow port support the order in which a mapping must be generated. So every time a flow port is connected on a node a recursion is made on the node connected to the flow port. The code generated by the node connected to the flow port is then placed outside of the code generated by the node owning that flow port.



<pre>&lt;h&gt; * &lt;/h&gt;</pre>	<pre>&lt;h&gt; *   &lt;if&gt;     &lt;q/&gt;   &lt;/if&gt; &lt;/h&gt;</pre>	<pre>&lt;h&gt;   &lt;for-each&gt;     &lt;if&gt;       &lt;/q&gt;     &lt;/if&gt;   &lt;/for-each&gt;   * &lt;/h&gt;</pre>
-----------------------------------	---	--

In the source tree no connections are needed between schema elements, because there is no need for nesting on the source side.

Above a partial code generating example is displayed. At the first step the H element is encountered which isn't connected but does have linked sub elements, so a h-tag is generated and the next adding point for code is set within the h-tag.

At the second step the Q element is encountered which is connected to a if-node. The if node has a connected flow port, so an if-tag is placed within the h-tag but the next adding point for code is set outside the if-tag.

At the third step the for-each node is encountered which is connected to a source element. A for-each tag is placed outside the if-tag and the next adding point for code is after the for-each closing tag. Encountering a source element stops the connection search. After the Q element the A element is encountered, but since it is not connected and has no linked sub elements no code will be generated. There are no more target elements and thus the code generating is completed as displayed in step 3.

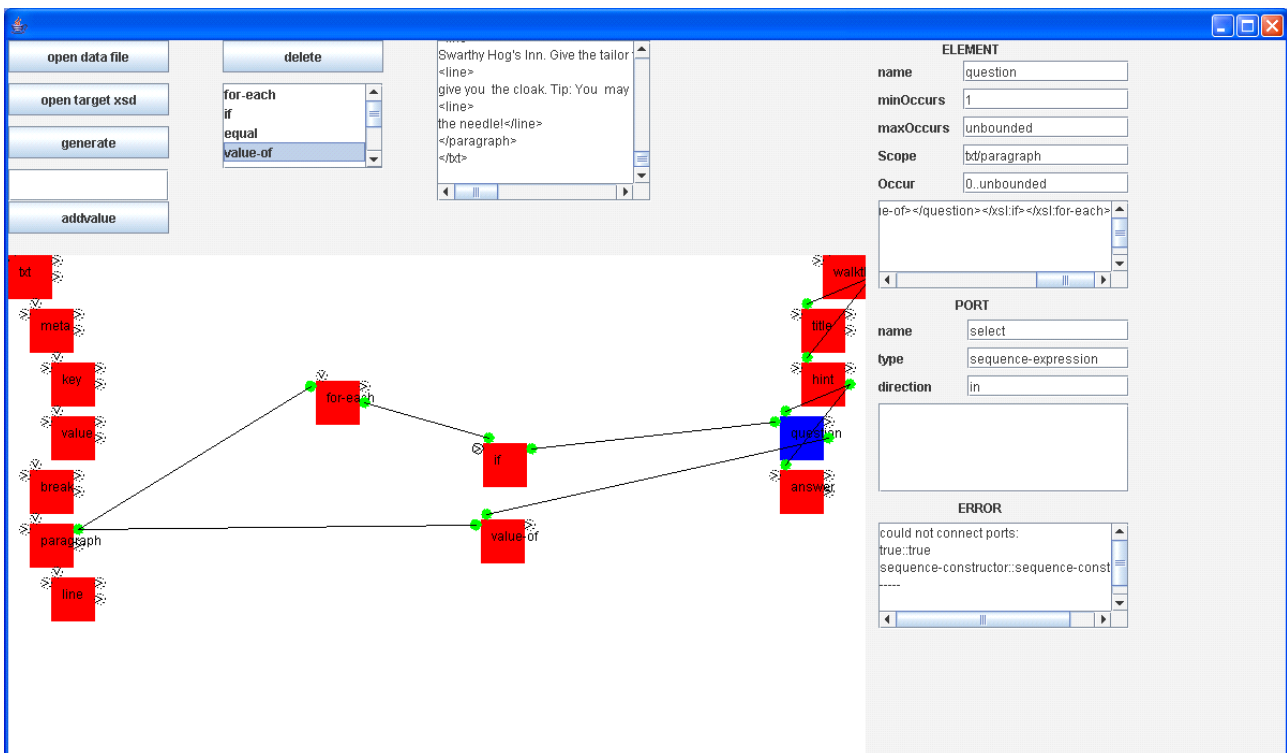
## Scope

Because (nested) links can be used it is possible that certain select values can change the scope of an element. This happens for example when a for-each element iterates over a certain select value. This means that when a path value is selected within such an iteration its scope must be recalculated. If this is not done absolute paths will be miss interpreted in the resulting XSLT code, because there relative paths are expected.

Relative scopes are calculated in the same manner the code generation is realized. By going through the nested structure of elements. A scope is then recalculated by the encountering of a schema element or a select value of a XSLT element.

## 6.6 Implementation

The Class Diagram of the visual XSLT elements can be found in appendix E. Visual elements can be connected to each other through their port connections. Every extension of the VisualElement has its own internal XSLT generating that expresses its XSLT working. The extensions share the same connection structure and are therefore more or less the same.





The VisualElement and the VisualXSLTPort extend from geometry classes which is very useful for displaying the visual elements. Because of their geometry extension the visual elements can directly be painted on a canvas. This lets the visual elements have a handy mixture of functional and visual properties for both the XSLT generation and the visual interaction.

## **6.7 Testing**

The visual aided matching will be tested to get a view of the good and bad points of the implemented system. The main goal of the visual aided mapping is providing the user with a user-friendly and efficient interface. The testing therefore will not be like a regression test but will focus on user interaction.

This user interaction test will be set up in the form of test case scenarios. Different test cases will be performed by test users. The test cases will be set up to use the system in different settings trying to cover its full potential.

These test cases will be performed by two types of users, novice users and expert users. This is done to get a better overall coverage of possible situations. Expert users have more know-how on the theoretical working and will probably follow a more direct path through a system. Novice users will probably follow a more varied path through the system getting in situations not encountered by expert users [20]. They will also have different needs for information feedback, an expert user will search more towards specific information while a novice user will need more information about the basic workings of the system. The novice users will be given a short insight in de working of XSLT transformations to give them the knowledge expected from a normal system user

The tests will be monitored with a record/playback technique [19], this means the states in the testing are recorded and can be evaluated later on. This normally is a time consuming operation, but since InDialog is a small organization only a few test will be performed. The aim is to use two expert users and two novice users. They will be monitored during their test cases and specific situation will be captured for later evaluation. In this monitoring successful and failed attempts will be recorded to get an insight in where problem arise with the working of the interface.

Next to the case tests the user will be asked to give their opinion on several properties of the visual aided matching environment. These properties are retained from a user survey on what properties could be valued as good or bad in general applications. This survey is done on four people within InDialog and result in the following list:

- flexibility
- user friendliness
- intuitive working
- global overview
- undo possibilities
- information retrieval
- categorical approach
- clarity in cause and effect
- concise information
- intuitive visualisation

Users are also asked to give a general description of their finding after working with the visual aided mapping.

### **6.7.1 Test Goals**

The tests are used to get a view of the good and bad points of the implemented system. So the result of the testing must be a list of positive points, negative points and missing additions of the system. Out of all the points the relative ones must be bundled to create a list of improvements for the system.

First the testers will get a short introduction into XSLT and data transformation. After that they will be asked to perform two test cases under supervision in which an observation will be made on how the testers use the system. The observation will focus on discovering pitfalls and positive workings.

To get a goal oriented feedback from the testers the survey is so constructed that it leaves as little room as possible for evasive answers. This is done by giving six options in the multiple choice questions, so that there are enough steps between a absolute good and bad. The use of an even number eliminates the option of choosing the middle option as an easy escape.

Also different open and multiple choice questions are constructed to overlap or ask the opposite. This given more information about the same subject and also stimulates the tester to give more thought about a subject.

Next to the multiple choice questions the testers are asked three open questions about every subject:

- "what do you find positive?"
- "what do you find negative?"
- "what would you like to see improved?"

This will give a more detailed insight in the testers opinion for the specific testing goals. The results can be cross referenced to the answers of the multiple choice answers to get a more complete view of the testers findings.

Both the multiple choice answers and the open answers will be evaluated in reference with the observations during the test cases.

## **6.8 Evaluation**

In this chapter visual aided matching has been discussed and a system prototype for visual XSLT has been made. In this evaluation the prototype will be compared to the matching tools in the evaluation of 6.2. Also to be able to evaluate the different aspects of visual aided matching a survey has been done. This survey was constructed to evaluate the good and bad points of the prototype in reference to visual aided matching for technical and non-technical users.

### 6.8.1 Comparing the prototype

The prototype build is a simple proof of concept of the design in 6.5. The prototype has its focus on the correct and flexible linking of XSLT elements. The matching tools evaluated in 6.2 are large commercial programs which obviously overshadow a small matching prototype. To get a fair comparison, the good visual aided matching options concluded in 6.2 are compared with the matching/linking properties of the prototype. By comparing these options to the prototype, it can be evaluated according to the combined wanted properties of a good visual matching tool. After this comparison the prototype will be compared to the cons noted by the two individual matching tools.

A schematic view of the prototype can be found on page 47, illustration 18 shows a screen shot.

The concluded good options out of 6.2 compared to the prototype:

#### *a hierarchical schema presentation*

The prototype uses a hierarchical schema presentation in the same way as the reviewed tools do. The prototype is not styled in any way, but it is evident that it draws a correct hierarchical presentation of the structure.

#### *non-complex linking*

The prototype uses only one way of linking: connecting nodes through their ports. No alternatives are used which may make the basic linking more complex. The ports are visualized to present their connection, direction and value options. Once the visual presentation is understood, a user should be able to link all XSLT elements in the same way.

#### *visual mapping nodes for all possible XSLT elements*

The prototype only uses a few XSLT elements. The implementation of all XSLT elements would be cost too much time. But the XSLT elements in the prototype have been constructed in such a manner that it should be possible to produce all of the other XSLT elements with the same concept.

#### *code previewing*

The prototype does not contain code previewing directly. It does contain code generation which keeps nesting and scopes in mind. This code generation is the building block for code previewing. If the code generation would be made visible at every change, the code previewing would be a fact.

#### *the use of flow ports*

The prototype uses flow ports. The node element and its ports have been designed in such a way that nodes can be connected in the all the ways possible in XSLT. This provides the freedom of connecting the nodes as wanted by the user, not as dictated by the tool (within XSLT standard).

The individual cons out of 6.2 compared to the prototype:

#### *non-consistent for-each construction*

In the prototype the working of the different graphical XSLT elements are consistent. Due to the port connection design in 6.5 no non-consistent constructions seem to be necessary. Admitted that only a small set of XSLT element have been implemented, all

XSLT elements that satisfy the XSLT characteristics mentioned in 6.5 should keep the graphical XSLT construction consistent.

*linear, content oriented view*

The linear, content oriented view mentioned in 6.2 about Adeptia references to the way XSLT elements are to be connected. Within Adeptia the perspective is that an element should be placed inside another one. This might restrict the freedom of the transformation programmer. In the prototype no perspective restriction are present. Elements can be placed standalone within the prototype and can be connected later on in any perspective that the XSLT standard allows.

*four different rule options to manipulate the matching*

For some complex XSLT options Adeptia uses different rule options. As mentioned before the prototype does not need extra options for XSLT elements satisfying the XSLT characteristics.

*small amount of nodes to use*

The prototype has start out using a small set of XSLT elements. This is done to make a proof of concept. This set should be expanded to offer a complete set of XSLT elements if the prototype is to be further developed. Due to the graphical XSLT element design this should be no problem.

## **6.8.2 Test results**

In the survey three different question forms are used. Multiple choice to get an overall grading. Open questions about positive, negative and improvement points to get a more detailed picture about the subjects. And an option for open remarks.

The following table shows the scores from the multiple choice questions for the chosen subjects out of chapter 6.7. Out of the four test users there are two novice users (N1, N2) and two expert users (E1, E2). The novice users work within the communications department of InDialog. They had no experience with XSLT and have been given a crash course. The expert users work within the programming department of InDialog. E1 has good knowledge of XSLT and E2 has little knowledge of XSLT. The test form and its results of E1 can be found in appendix F.

scale from 0 to 6

	E1		E2			N1		N2				
flexibility	5	5		5	5		6	6		6	6	
user friendliness	4	6		5	6		5	6		3	5	
intuitive working	5	6	6	3	3X		6	5X		4	3X	
global overview	3	6	6	4	6	6	5	6	6	5	5	6
undo possibilities	4	6		4	6		6	6		6	6	
information retrieval	6	6		4	2		3	2		4	5	
categorical approach	5	5		1	2		6	6		4	4	
clarity in cause and	6	6	4	6X	X		2X	X	X	X	X	
concise information	6	6		6	5		6	6		6	4	
intuitive visualisation	6	6		6	4		5	5		3X		

*Illustration 10: Test results*

X: question skipped due to lack of knowledge of XSLT

The following text shows the overall results from the open questions. In these results the case observations and the multiple choice scores are considered.

### **flexibility**

The flexibility is marked as good, the open questions also give a positive result. The flexibility in matching is described as 'complete freedom in matching'. As improvement the addition of frequently used expressions is mentioned.

### **user friendliness**

The user friendliness is marked as good. The open questions reveal that the test users are positive towards the visualisation of the XSLT elements and the connection. But they would like extra information on the use of ports and a possible end result.

### **intuitive working**

The intuitive working is marked as average. The open questions reveal that a lack of XSLT knowledge is a pitfall in intuitive working. The test users describe a need for a clear guidance through a matching.

### **global overview**

The global overview is marked as good. The open questions reveal that the test users find the source, target and matching parts and their interaction easy to use. The test users predict a problem in overview when matchings become more complex.

### **undo possibilities**

The undo possibilities are marked as good. Although it is still a prototype the test users give a positive feedback on full deletion of elements and their connections at once. The click area is considered a bit small result in miss clicking ports. In addition the test users would like to see undo/redo and copy/paste functionalities.

### **information retrieval**

The information retrieval is marked as average. The non-technical testers find the information too technical. Both would like more practical information about the working of the matching and feedback about errors and results.

### **categorical approach**

The information retrieval is marked as average. The test users find the difference between source, target and mapping good to understand. But they would like to see more guidance in the steps to take to make a matching.

### **clarity in cause and effect**

The clarity in cause and effect is marked as fairly good. The non-technical test users say cause and effect are hard to predict because of their lack of XSLT knowledge. The technical users don't have this problem because of their XSLT and programming knowledge. To clarify cause and effect the test users would like more information about the matching they are working on.

### **concise information**

The concise information is marked as good. As already mentioned above the information is fairly technical. Additional options mentioned are tooltips, a help page, examples, result previews.

### **intuitive visualisation**

The intuitive visualisation is marked as good. The test users find the visual XSLT elements easy to use and comprehend. The non-technical testers don't see a direct link with XSLT but still find the visualisation intuitive. They look at it as a way to make simple matchings without XSLT knowledge. The technical testers find the visual XSLT elements a good visualisation of the XSLT standard and see it as an easy way to make fast matchings.

## **6.8.3 Conclusion**

The testing has been done by a very small test group within InDialog. Because of the minimal size of the test group no real objective result can be deducted. To get an objective result the testing will have to be done within a large scale setting. The current results however can be viewed in context of the demanding standard of InDialog. And could be used as a starting point for a larger scale testing.

*The following conclusion is based on the test result within the context of the demanding standard of InDialog:*

Out of the test results it can be concluded that the visualXSLT gives a good visual representation of XSLT. An important element of the system is providing the user with information about its working. Novice users have the need for information about the workings of XSLT, port and connections. One possible way to provide this information is the use of tooltips. Tooltips can give quick information about an object and do not need extra user interaction since they are displayed on mouseover events. Another way is to give more insight in the matching process by using fixed guided steps.

An open point is the behaviour of the system with more complex mappings. Will the overview become too hard to comprehend? What effect will different visual adjustments (use of hiding or transparency) have on this?

The visualXSLT element are build according to the XSLT standard, this means that every possible XSLT construction can be reproduced in visualXSLT. This can be verified by the fact that XSLT is a XML structured language, where element (tags) can be placed either completely inside or completely outside another element. This behaviour is copied in the visualXSLT design by the use of content ports and flow ports which define the outside and

the inside of an element.

Only a small number of XSLT elements have been implemented in this prototype, but since all XSLT elements have the same generic build up, it is considered that the visualXSLT implementation completely covers the XSLT language domain.

If this visualXSLT prototype is compared to the matching systems discussed in chapter 6.2 the most important difference is that it provides an exact copy of the XSLT workings whereas the systems discussed have wandered from a 1-on1 match with XSLT which makes some XSLT mappings hard or even impossible to reproduce without manual coding. Out of the comparison in 6.8.1 it can be concluded that the prototype has stayed true to its design idea. The prototype followed the pros, and evaded the cons which had been concluded in 6.2. Therefore the prototype shows good promises for further development.

## 7. Third iteration, Partial mapping reuse

This chapter describes the third development iteration of this project. The goal of the third iteration is to expand the system of the second iteration with partial mapping reusability.

Non-partial mapping reuse has been discussed in chapter 5.4.4. This is a fairly trivial activity wherein total mappings can be reused if source and target schema are exactly matched from a previously made mapping. Non-partial mapping reuse can be useful when there is a need for bulk transformation, in which a large number of similar transformations will be processed.

In many cases similar connections have already been made by previous matches or only small adjustments are necessary. The previously made matching between elements can be used to aid new matchings. If element A and B are a match in a large number of other cases there is a big change they will match in this case and the partial mapping between A and B can be reused.

### 7.1 Goals / Requirements

A lot of research is done on automated matching to improve efficiency in manual matching. These automated matching systems use different approaches (see chapter 3.2) to find possible matches, and are usually capable of adjusting their dissensions on previous matching results. In this project the choice has been made to not use automated matching at first, but to first focus on the use of data from previous mappings. This is done for three reasons:

- Because of the non-intensive use of the system it will not directly need automated matching to improve efficiency. Mapping reuse data gained out of previous mappings will provide enough mapping support at first. The previous mapping reuse can be build in a way so that automated matching can be added later on.
- A different approach is tried which focuses on the value of manual created matches. Manual created matches will have more semantic value and therefore it will be used as a solid base for partial reuse mapping. The results can later on be used to aid automated matching. Mappings are then made out of the perspective of the end-user instead of by artificial intelligence.
- In the scope of this project the use of automated matching will probably mean a larger time investment, thereby prolonging the project beyond its planned length.

The third iteration has the following goals:

- research partial mapping reuse options
- make a design and implementation for storing and recovering partial mappings
- design and implement an algorithm for generating match candidates, which includes transitive matching, based on previous matches
- evaluate the partial mapping reuse based on previous matches



## 7.2 Research questions

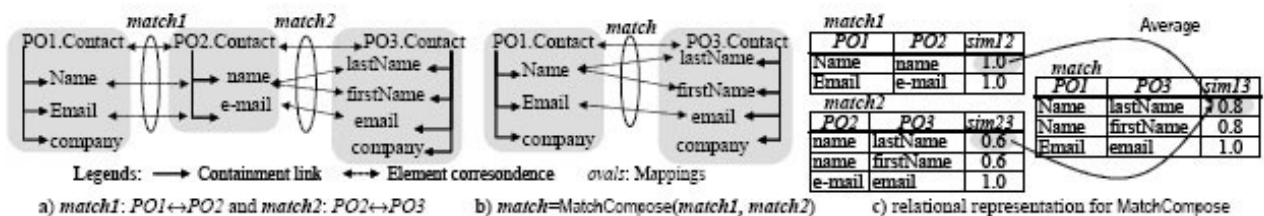
1. What are good partial mapping reuse options, which fit in the current system architecture?
2. What is a good way for storing and recovering partial mappings?
3. How can match candidates be discovered out of stored partial mappings?
4. How can transitivity be added to match candidate discovery?
5. How can the partial mapping reuse be evaluated?
6. What are differences and resemblances with automated mapping?
7. What are good and bad points of partial mapping reuse?

## 7.3 Research

What are good partial mapping reuse options, which fit in the current system architecture?  
 What is a good way for storing and recovering partial mappings?

A small survey has been done on the possibilities of partial mapping reuse. There is a lot of research done on automated mapping which can also be used for reuse in manual mapping. Two papers in particular are used to create a design for the partial mapping reuse. The first one is "COMA - A system for flexible combination of schema matching approaches" [16], in particular chapter 5 which is about the reuse of previous match results. The second one is "An automatic tool for discovering complex mappings" [17] which provides a general framework for mapping generation through relevant decompositions and identifications.

### 7.3.1 Reuse in Coma



Illustratie 11: MatchCompose example [16]

Coma uses a 'MatchCompose' operation that assumes a transitive nature of the similarity relation between elements. This assumes that when  $a$  is similar to  $b$ , and  $b$  to  $c$ , then it is very likely that  $a$  is also similar to  $c$ .

$contactFirstName \leftarrow 0.5 \rightarrow Name \leftarrow 0.7 \rightarrow firstName$

multiply:  $0.5 * 0.7 = 0.35$

average:  $(0.5 + 0.7) / 2 = 0.6$

In the above example  $contactFirstName$  can be found to be similar to  $firstName$  through transitivity. Different computation can be used to calculate the combined similarity.

Coma uses the Average and Dice function instead of multiplying which doesn't reflect good similarities with values between one and zero.

*Average*: dividing the sum of the similarity values of all match candidates by the total number of set elements.

*Dice*: returns the ratio of the number of elements which can be matched over the total number of set elements.

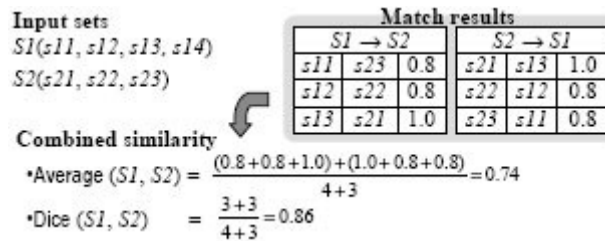


Illustration 12: Computing combined similarity [16]

Coma stores previous matches for later reuse. When a new match problem occurs the store can be searched for previous match results which are usable in that case.

If multiple matchers were used a similarity cube is used which aggregates different values to a combined similarity value. After that, match candidates are determined and ranked according to their similarity values. Out of the match candidates combined similarities are computed.

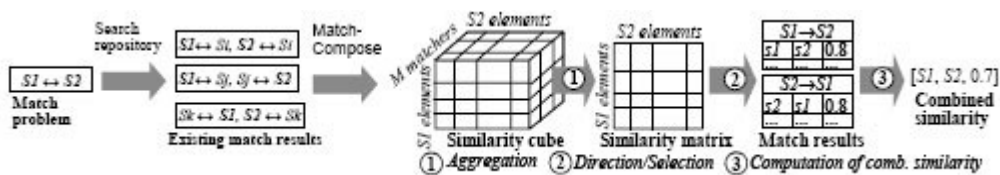


Illustration 13: Schema reuse matcher [16]

### 7.3.2 Discovering mappings

"An automatic tool for discovering complex mappings" [17] describes a general framework for mapping generation. The main idea is that the input schemas are split up in relevant decompositions for which then partial mappings can be determined.

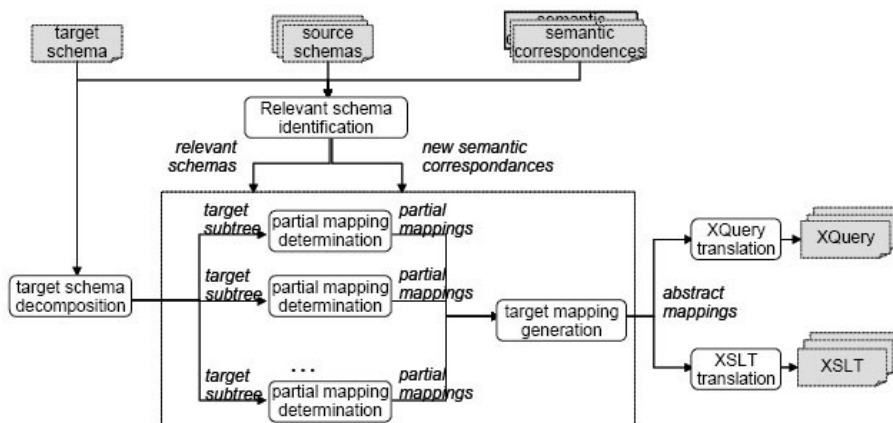


Illustration 14: General framework for mapping generation [17]

Target subtrees are determined out of the target schema. Each subtree consists out of one rootnode and one or more mono-valued child nodes. This results in a decomposition like the one displayed in the illustration below (st1,st2, st3).

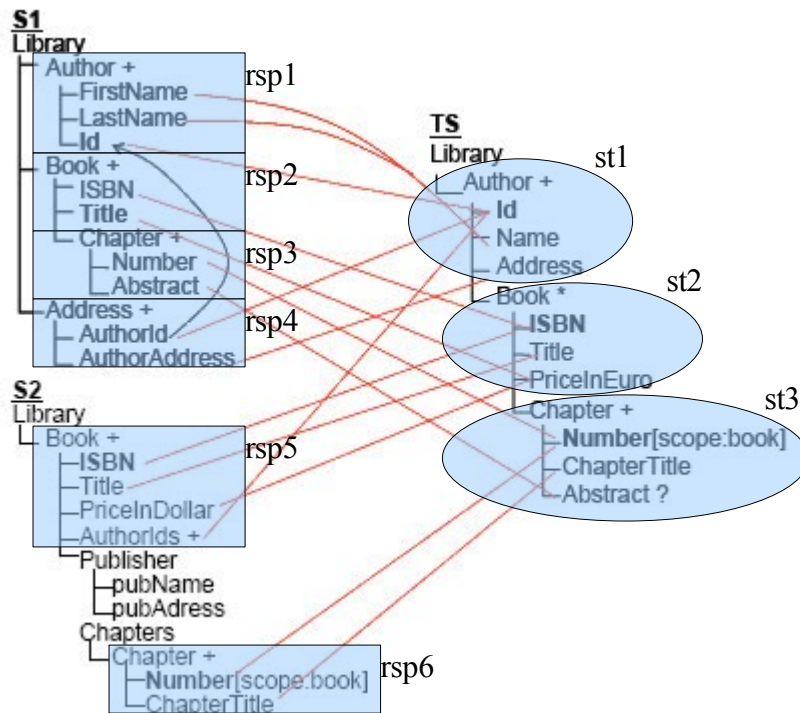


Illustration 15: Schema decomposition example [17]

Based on semantic correspondences the partial mappings can be processed in three steps. The above target decomposition is also done for the source schema, but in this case only the relevant schema parts are considered. Relevant schema parts are elements which have a connection with the target schema (rsp1,2,3,4,5,6).

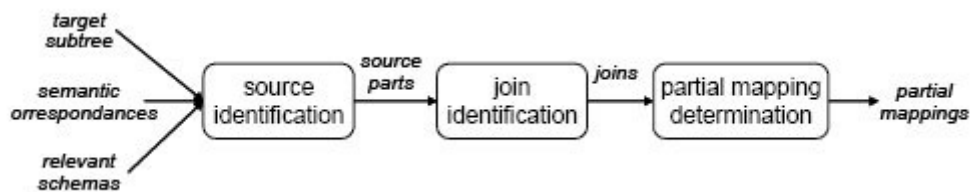


Illustration 16: Determining partial mappings process [17]

When the target subtrees and relevant source parts are determined, joins are searched that can be used to combine these source parts. After that partial mappings can be determined for each target subtree. A more detailed description of these techniques and the algorithms used can be found in [17]. An example of the working of the joint operations is depicted in illustration 16.

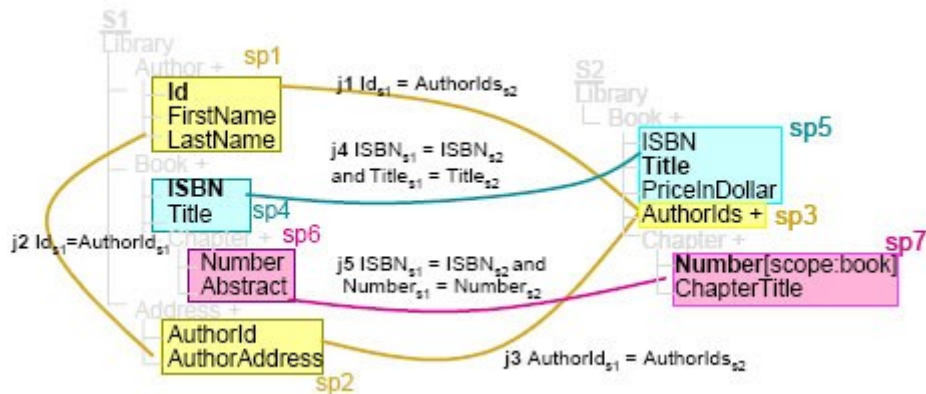


Illustration 17: Join operations [17]

## Conclusion

*What are good partial mapping reuse options, which fit in the current system architecture?*

Both papers mentioned above contain options that can be used in the design of partial mapping reuse in this project. Good options are:

- The decomposition of schemas into subtrees, this groups elements together that semantically belong to the same object. Also non relevant objects within schemas are discarded which simplifies the mapping reuse and avoids 'contamination' with unwanted data.
- The composition of partial mappings not only based on direct matches and rankings but also on transitive similarity. This will give more similarity options for matching and thereby creating a flexibility in possible synonym discovery.

*What is a good way for storing and recovering partial mappings?*

Both papers don't directly mention ways of storing or recovering partial mappings. But out of the data used it is possible to get a good view of what data will be needed for storing and recovering partial mappings.

For a simple match one element is linked to another element. Apart from their names more information can be used to improve match efficiency. When using subtrees the elements used belong to an object sharing a semantic for the element within that object. The rootnode can be viewed as the main description for that semantic. Coma also mentions that schemas from the same application domain usually contain many similar elements, which are typical to that domain. The knowledge of subtree and domain context can therefore lead to good reusable match candidates.

Out of the above it can be concluded that it is good to store and recover partial mappings with the context of their domain and subtree.

## 7.4 Design

*How can match candidates be discovered out of stored partial mappings?*

*How can transitivity be added to match candidate discovery?*

Based on the research in chapter 7.3 a design will be made for partial mapping reuse within this project. The design will be split up in three parts: the storing of partial matches out of manual made mappings, the discovery of match candidates between two schemas based on the stored partial matches and the discovery of transitive matches.

### 7.4.1 Storing partial matches

The first thing that has to be done is storing the partial matches. This must be done in a way that preserves as much data as possible for later reuse. As concluded in chapter 7.3 not only the match nodes self but also the node context is important for a better semantic understanding.

The context of a match node will be derived from two instances. The first is the subtree in which the match node resides, the second is the domain in which the match node resides. To store this information for every match found, next to the match itself the subtrees involved will be stored.

A stored subtree will contain the domain, the headnode and the monovalued childnodes of the headnode. A match will contain the domain, the headnode and the node involved from the source and the target element which are linked together.

Also the XSLT of a match will be stored to provide more insight of the match internals. If only the match itself would be stored the only thing derived will be that element A is linked to element B. This will eliminate any future need for semantics, because the XSLT can give a more complete view of the construction used and possible other elements involved in a match.

To give a direct insight in what type of link is stored a type description is added to the match. The type description will show if the match is a structured or a value link. A structure type match is considered as a match that is only used to build the structure of a mapping and not actually transfers data from the source a match to the target. A value type match is considered as a match that does transfers data from the source to the target. If actual data is transferred in a match it is likely to have more semantic value then when a match is only used to construct a hierarchy. An expected behaviour is that the nodes in a subtree will mostly be of the value type and that headnodes will mostly be used for structure type matches. If this turn out to be true the match type can give extra insight in match reuse.

### Generating and storing subtrees and matches

When a made mapping is stored both the source and the target schema will be analysed to retrieve all subtrees within them (see [17]). Out of these subtrees only the relevant subtrees are stored and out of these relevant subtrees the matches are generated and stored. Below an example displays how matches and subtrees are generated and stored.

The example considers a mapping between persons and personnel.

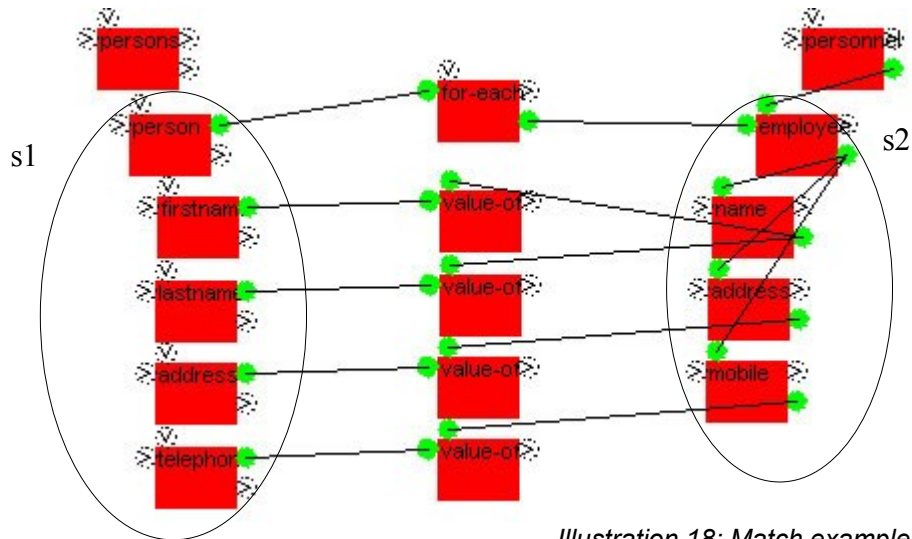


Illustration 18: Match example

1. First the subtrees are determined for the source and target schema which will result in s1 and s2.
2. Out of the subtrees the relevant subtrees are determined. Both s1 and s2 are connected so they are both considered relevant.
3. The relevant subtrees are stored in the database:

<b>subtree</b>	("','1','persons','person','firstname,lastname,address,telephone')
id (int)	
mappingid (int)	("','1','personnel','employee','name,address,mobile')
domain (varchar)	
headnode (varchar)	
nodes (text)	

4. Out of the target subtrees the matches with the source subtrees are determined and stored.

<b>match</b>	("','persons','person','person','personnel','employee','employee','structure','1','<xslt.....')
id (int)	
domain_a (varchar)	("','persons','person','firstname','personnel','employee','name','value','1','<xslt.....')
headnode_a (varchar)	
node_a (varchar)	("','persons','person','lastname','personnel','employee','name','value','1','<xslt.....')
domain_b (varchar)	
headnode_b (varchar)	("','persons','person','address','personnel','employee','address','value','1','<xslt.....')
node_b (varchar)	
type ('structure','value')	
mappingid (int)	
xslt (text)	("','persons','person','telephone','personnel','employee','mobile','value','1','<xslt.....')

## 7.4.2 Match candidate discovery

The goal of the match candidate discovery is to generate match candidates for elements within a chosen source and target schema. These candidates will be generated by reusing previous matches which are recovered from the match store. For the match candidate discovery a design has been made that uses parts of [16] and [17].

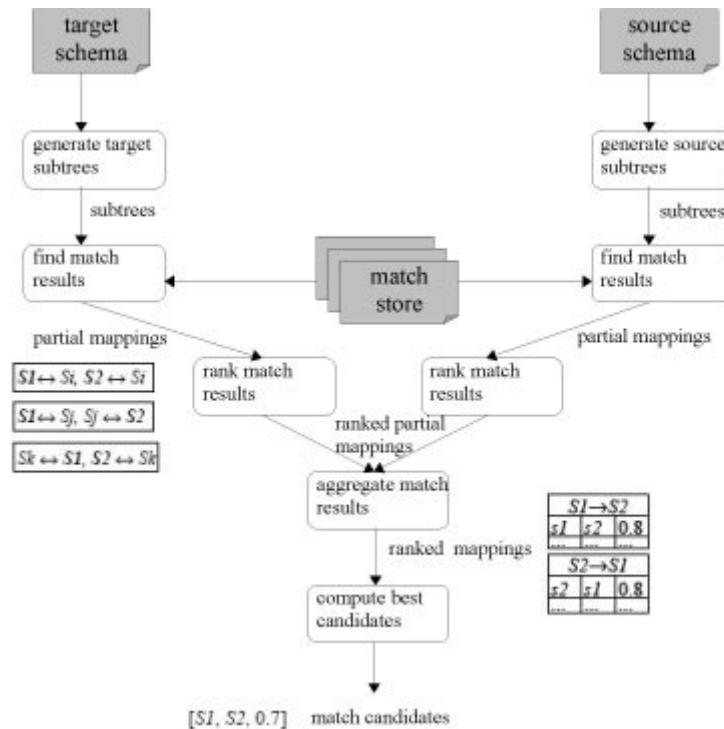


Illustration 19: Schema reuse matcher

### generate subtrees

Out of the source and target schema, subtrees are generated. The generation of subtrees is done in exactly the same way as mentioned in 7.4.1. The subtrees can be used to find match results in combination with previously stored matches.

### find match results

For the found set of subtrees every subtree is checked for possible match combinations within the match store. Match combinations are found if an endnode of a subtree matches a endnode of a stored match. An endnode is the last non-empty node available within a match element triplet (domain\_a, headnode\_a, node\_a), which results in either the value of the childnode or the value of the headnode if the childnode value is empty.

This will result in a set of matches found in the match store where the endnode match a node from the schema. The nodes beside the endnode within a match element triplet don't have to match. A set found out of s2 in illustration 17 might look like this:

#### Source matches

(personnel,employee,) <-> (persons, **person**,.)  
 (personnel,employee,name) <-> (persons,person, **firstname**)  
 (personnel,employee,name) <-> (persons,person, **lastname**)  
 (y,employee,y) <-> (x,x, **lastname**)

(personnel,employee,address) <-> (persons,person,**address**)  
 (personnel,employee,mobile) <-> (persons,person,**telephone**)  
 (x,x,phonenumber) <-> (y,person,**telephone**)  
 (b,employee,b) <-> (a,person,**telephone**)

**rank match results**

Next the found matches will be ranked. This ranking will represent how much the found match corresponds with the context of the element of the schema. This is done by looking if the other nodes within the match element triplet correspond. This will tell if the match originates from the same domain or object context. If this is the case a higher ranking will be given because matches found in the same context suggest a better chance of semantic correctness.

If the domainnode and headnode of a found element triplet match that of the schema element it is classified as a domain specific link and is higher valued within that domain. If there is no match on the domainnode and the headnode the link is lower valued as it is then considered to be a more general link.

The lower value of a general link is compensated by the more frequent appearance of such a link throughout different domains.

<i>domainnode match</i>	<i>headnode match</i>	<i>ranking</i>
true	true	1,4
false	true	1,3
true	false	1,2
false	false	1,1

The above ranking is chosen so that the result of a multiplication of two domain specific links is more than the multiplication of two general links, but also so that the multiplication of more general links is able to result higher. Two domain specific links will result in  $(1,4)^2 = 1,96$  and are comparable with seven general links  $(1,1)^7 = 1,95$ .

The ranking values chosen now are not founded on an existing theoretical principle but they will have to be tested and adjusted if necessary. The ranking can be implemented as a weight factor used in a neural network. The weight can be adjusted manually or automatically by training and testing on predefined values.

A ranked set found out of s2 in illustration 18 might look like this:

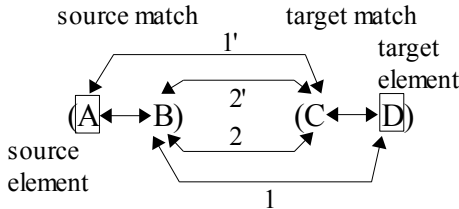
**Source matches**

(personnel,employee,) <-> (persons,**person**,) = 1.4  
 (personnel,employee,name) <-> (persons,person,**firstname**) = 1.4  
 (personnel,employee,name) <-> (persons,person,**lastname**) = 1.4  
 (y,employee,y) <-> (x,x,**lastname**) = 1.1  
 (personnel,employee,address) <-> (persons,person,**address**) = 1.4  
 (personnel,employee,mobile) <-> (persons,person,**telephone**) = 1.4  
 (x,x,phonenumber) <-> (y,person,**telephone**) = 1.3  
 (b,employee,b) <-> (a,person,**telephone**) = 1.3



## aggregate match results

Both the source and the target schema provide a set of ranked partial mappings. These two sets have to be combined to find possible matches between the source and target elements. All matches within the two sets will be compared to each other to see if they have corresponding elements so that they can be combined. The combining of matches is done as illustrated below:



**Match 1:** (D = B)  
 A ↔ (B=D)  
 A ↔ D

**Match 2:** (C = B)  
 A ↔ (B=C) ↔ D  
 A ↔ D

Match 1 and 2 will cover all possible combined matches. It might seem that match 1' is thereby left out of the combined matches. That would imply that the (A ↔ B) && (C ↔ D) where A=C will not result in (A ↔ D). Below is the proof that this match will still be found.

1. ((A=C) ↔ B) && ((C=A) ↔ D)
2. A = C, so both can be substituted by X  
 (X ↔ B) && (X ↔ D)
3. if (X ↔ D) exists in the target set, it means it must also be present in the source match set.  
 (X ↔ B) && (X ↔ D)  
 (X ↔ D) && (X ↔ D)
4. (X ↔ D) && (X ↔ D) will result in (X ↔ D)
5. X can be substituted by A  
 (A ↔ D)

When matches can be combined also their ranking must be combined. The combination of the ranking is calculated based on the individual rankings of the matches and the depth of the match. If a match is combined in the first aggregation the match depth is considered to be zero. The rank combination between two matches is calculated as followed:

$$\frac{\text{match1.rank} + \text{match2.rank}}{2} * \frac{1}{\text{depth} + 1}$$

The second part of the equation (1 / (depth + 1)) ensures that the result ranking decreases with every aggregation step. This is analogue with the thought that a match constructed on a deeper aggregation level is less important. The case here is also that the ranking chosen is not founded on an existing theoretical principle but it will have to be tested and adjusted if necessary. The example below illustrates the result combination of the source matches used before with the target matches.

### Target matches

(personnel, <b>employee</b> ,)	↔ (persons, person,)	= 1.4
(personnel, employee, <b>name</b> )	↔ (persons, person, firstname)	= 1.4
(personnel, employee, <b>name</b> )	↔ (persons, person, lastname)	= 1.4
(y, y, <b>name</b> )	↔ (x, x, x)	= 1.1
(personnel, employee, <b>address</b> )	↔ (persons, person, address)	= 1.4
(personnel, employee, <b>mobile</b> )	↔ (persons, person, telephone)	= 1.4
(b, b, <b>mobile</b> )	↔ (a, a, a)	= 1.1

**Combined matches**

(personnel, <b>employee</b> ,)	<-> (persons, <b>person</b> ,)	= 1.4
(personnel,employee, <b>name</b> )	<-> (persons,person, <b>firstname</b> )	= 1.4
(personnel,employee, <b>name</b> )	<-> (persons,person, <b>lastname</b> )	= 1.4
(personnel,employee, <b>name</b> )	<-> (x,x, <b>lastname</b> )	= 1.25
(personnel,employee, <b>address</b> )	<-> (persons,person, <b>address</b> )	= 1.4
(personnel,employee, <b>mobile</b> )	<-> (persons,person, <b>telephone</b> )	= 1.4
(personnel,employee, <b>mobile</b> )	<-> (y,person, <b>telephone</b> )	= 1.29
(personnel,employee, <b>mobile</b> )	<-> (a,person, <b>telephone</b> )	= 1.29

**compute best candidates**

When the matches have been combined all possible matches between the source and target schema are presented. It is possible that there are multiple options for an element match, in the example above the element *name* can be matched to *firstname* or *lastname*. But if looked at from the other side, the element *firstname* only has *name* as its match candidate.

For both the source and target perspective the match candidates are grouped by schema element. The options per schema element will be ranked according to their number of appearances relative to the total number of options.

$$\text{rank of match option} = \text{average}(\text{match ranks}) * (\# \text{options} / \text{total options})$$

$$\text{rank}(\text{name} \leftrightarrow \text{lastname}) = ((1,4 + 1,25) / 2) * (2 / 3) = 0,88$$

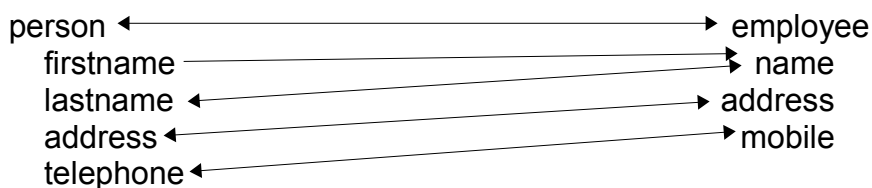
**Source perspective**

lastname:	name	= 1,33
telephone:	mobile	= 1,33
address:	address	= 1.4
firstname:	name	= 1.4
person:	employee	= 1,4

**Target perspective**

mobile:	telephone	= 1,33
employee:	person	= 1,4
address:	address	= 1.4
name:	lastname	= 0,88
	firstname	= 0,47

In this case only the element *name* in the target perspective has two options. If the best ranked options are chosen this will lead to the following combined perspective:

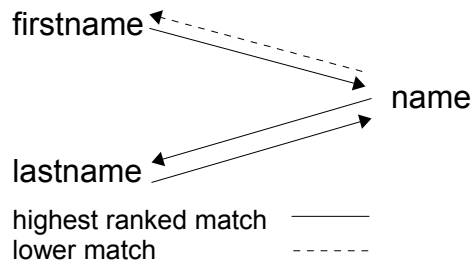


In this combined perspective the double arrows illustrate that there is agreement between the source and target perspective on a match. The match between *firstname* and *name* is not agreed upon which will mean it will be excluded from the match candidates. This way of candidate selection will work fine for single matches, but in illustration 17 it can be seen that *firstname* and *lastname* are parts of a combined match on *name*. Combined matches will get lost in this way of candidate selection, therefore an extra check will be done to discover possible match combinations.

### discovering combined matches

To discover combined matches all candidates from the source and target perspective will be considered. A difference will be made between the highest ranked matches and the other matches. The highest ranked matches will be granted preference before the other matches.

Sets will be formed with elements that are matched together, in the example  $\{firstname, lastname, name\}$  will be such a set. For every set the matches will be generated step for step, beginning with the highest ranked matches.



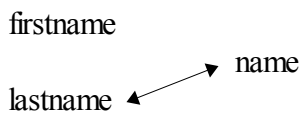
Three types of matches can be distinguished:

1. a match which is linked through two highest ranked matches
2. a match which is linked through one highest ranked match and one lower match
3. a match which is linked through two lower ranked matches

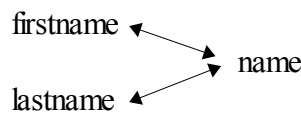
Per type the matches will be generated ordered by match rank, until the maximum number of possible links is reached. The maximum number of possible links is equal to the largest number of elements in a set originating from the same schema,  $\#\{firstname, lastname\} = 2$ .

<i>type 1</i>	<i>type 2</i>	<i>type 3</i>
1: name - lastname	2: name - firstname	

1:



2:



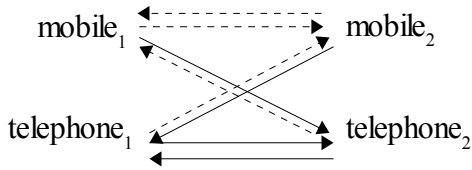
The above example illustrates that a combined match is found. This is still very trivial because there were also two possible options. The following example illustrates a combined match selection with more options where the element *mobile* has previous matches with *telephone* which will prevent it from matching with another *mobile* element.

#### Source perspective

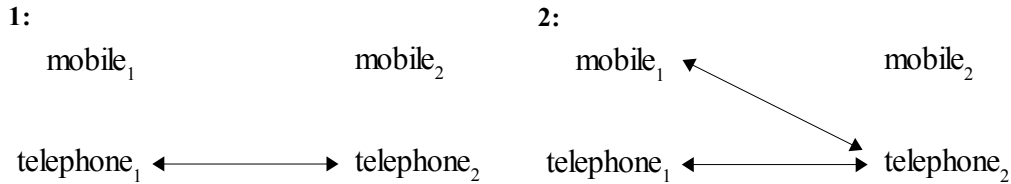
mobile: mobile = 0,38  
 telephone = **0,89**  
 telephone: mobile = 0,47  
 telephone = **0,76**

#### Target perspective

mobile: mobile = 0,38  
 telephone = **0,78**  
 telephone: mobile = 0,53  
 telephone = **0,76**



<i>type 1</i>	<i>type 2</i>	<i>type 3</i>
1: telephone <sub>1</sub> - telephone <sub>2</sub> (0,76)	2: mobile <sub>1</sub> - telephone <sub>2</sub> (0,71) 3: telephone <sub>1</sub> - mobile <sub>2</sub> (0,63)	4: mobile <sub>1</sub> - mobile <sub>2</sub> (0,38)



This example has eliminated two of the four options based on ranking choices. With respect to the non-combined matching an extra option was found where else out of four options only one would have been chosen. In the example above *mobile<sub>1</sub>* is matched to *telephone<sub>2</sub>* where it would be expected to match with *mobile<sub>2</sub>*. This was a result of previous matches between *mobile* and *telephone*.

If the match above is corrected and the match *mobile<sub>1</sub> - mobile<sub>2</sub>* is stored, a following match result will be adjusted. When there is a small number of match references in the database a large shift of ranking can occur. If the *mobile - mobile* match is enhanced it can be possible that through aggregation the *telephone - telephone* match is lower ranked then wanted and is overtaken by the *telephone - mobile* match. This problem only occurs if there is a small number of match references in the database. When there are larger numbers of correct matches in the database this effect is cancelled out. The following example illustrates the same match but with more correct matches (for both options) in the database:

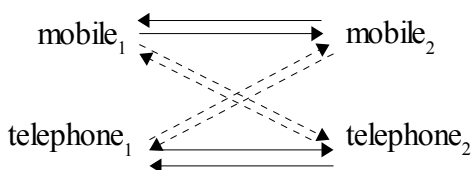
**Source perspective**

mobile: mobile = **0,91**  
 telephone = 0,35  
 telephone: mobile = 0,43  
 telephone = **0,78**

**Target perspective**

mobile: mobile = **0,91**  
 telephone = 0,31  
 telephone: mobile = 0,49  
 telephone = **0,78**

The effect of more correct matches in the database can not only be seen in semantic correct match results, but also in the difference between rankings of a correct and an incorrect match. If more correct data is available the differences will get more distinct.





## 7.5 Evaluation

### *How can the partial mapping reuse be evaluated?*

In this chapter no evaluation has been done on the partial mapping reuse that has been suggested. To get a better view of its working, results of this approach should be evaluated as well as the weighing used in the match computations.

A good approach for this kind of evaluation is running the match process like a neural network training. Large sets can be run while weighing factors can be adjusted. The results can then be compared to control sets to see how effective the computation has run with certain weighing. In this way computations of partial mapping reuse can be compared and evaluated.

### *What are differences and resemblances with automated mapping?*

The discovery of matches out of stored partial matches is in fact an automated mapping process. The main difference is that in this case only existing links can be used to generate new ones. With automated mapping also linguistics or certain structural patterns can be used to find potential new matches. These features use dictionaries or other references to produce match possibilities. This is basically not very different from the reuse of partial mappings. Both use a store of predefined matches which help to identify new potential matches.

Potentially this can be generalized so that different forms of predefined match forms can be used in the same way to find new potential matches. This would make it easier for new match forms to be integrated.

### *What are good and bad points of partial mapping reuse?*

Good points:

- the reuse of semantically correct matches, because the stored matches were made by human choice.
- as a side effect synonyms could be discovered through match aggregation.

Bad points:

- new matches must already have some relation within the store to be found.

The partial mapping reuse idea provides a good semantic base for matching. To make it more flexible a good option would be to extend it with extra external matching aids. The use of external matching aids like dictionaries or thesauries can improve its effectiveness. The main idea behind this is to let stored semantics and automated matches enhance each others working. Both approaches could give feedback and together deliver a more precise result.

## 8 Conclusions & Recommendations

This chapter will discuss the conclusions and recommendations of this project. This will be done by giving feedback on the project objectives / goals.

### 8.1 Conclusions

As a guideline for the conclusions, the goals mentioned in chapter 2.3 will be discussed.

#### **Design and implement a basic ETL system**

In chapter 3 and 4 the basis of the transformation system has been designed. It is designed with the thought of separating functionalities to keep the system easy adaptable. The main setup is a basic ETL system with an intermediate store. The internal buildup of the ETL system is symmetrical in design, so that the input and output side of the system can use the same building blocks. Also the building blocks themselves (parsers, connectors, matchers,..) have been designed to provide good adaptability. The flexible design chosen is a good base for further development.

#### **Define an intermediate data format**

In chapter 3 the possibilities of an intermediate data format have been discussed. It is concluded that no specific formats can be used to define the intermediate data format. As an open format XML is chosen with the restriction that it has to be valid with its XML schema. The XML and the schema together form the intermediate data format. The chosen intermediate data format of XML and XML schema provide a good and flexible connectivity with systems used at the moment, since XML is now a heavily used standard.

#### **Design an automated matching element within the base system**

A matching element has been designed with an eye for flexible matching. A flexible combination of schema matchers aided by a graphical interface has been chosen. The matcher combination has been adopted from the COMA system. The matching element has been set up to provide automated and manual interaction. The enduser can give feedback on the results and use iterations to optimize the results. A graphical interface is a valuable tool in this setting. It provides the user with faster understanding and interaction in the cases where manual matches are needed

#### **Incorporate reusability in data, matching and programming**

The main focus in this project has been on matching reusability. Data and programming reusability has been left undiscussed, but is encouraged as a good practise. Two options of matching reusability have been discussed. Partial and non-partial matching reusability. Non-partial is fairly trivial, but still needs a good naming reverence for good usability. Partial mapping reuse has been discussed throughout chapter 7. A schema reuse matcher (ill. 18) has been designed based on [16] and [17]. Relevant subtrees are generated matched and ranked to find partial match options. In this process semantic values are embedded by taking in account environmental

information. This works by applying different weights in the ranking for different environmental matches. Environmental information taken in account is:

- the nodes matching within a match triplet
- the depth of an aggregated match
- possible combined matches

Although weights in the partial mapping reuse were not evaluated, it presents a promising matching idea based on semantics.

### **Design and implement a user interface for visual aided mapping**

In chapter 6 visual aided mapping has been discussed. A visual implementation of XSLT has been made and evaluated. The visualXSLT proves to be a valuable addition for matching processes that require manual tweaking. The visual aided mapping still needs some matching knowledge, but gives an efficient tool for manual interaction.

### **Evaluate user friendliness, user efforts and efficiency.**

Since the best semantic values are gained from end-users; user friendliness, user effort and efficiency are evaluated throughout this project. User interaction and process automation have been merged into one design to work efficiently together. Within this interaction visual aiding is adapted to optimize user friendliness, efforts and efficiency. Non-partial mapping has been introduced in chapter 6 to provide more efficiency in matching reuse.

In general it can be concluded that a many-to-many data transformation prototype has been presented which focuses on semantics and user efficiency. The system parts have been build from a semantic point of view and with an eye on user friendliness and efficiency. The system is modelled based on ETL processing using an intermediate store. In the store and the processing reusability and interaction have been optimized by the use of leading open standards and modular building blocks.

This conclusion fulfils the assignment formulation, so it may be concluded that the project is completed successfully.

## **8.2 Recommendations**

The project presents a data transformation prototype. This means it still leaves room for extra research and development. In this paragraph recommendations will be done for future research and development on this project.

### **research & development within the scope prototype**

Within the current scope of the project there are open issues that can be addressed before looking at expansion possibilities.

#### *Testing the matching*

In chapter 7 different parts in the matching process have been discussed. As mentioned in these chapters the weights and algorithms used have not been thoroughly tested.



To get a proven method the algorithms used should be tested by using training and control sets. In this testing also the weights used should be examined so that the result will be optimized. Only by testing the matching a good view can be gotten on its effectiveness and thereby possible changes can be made.

### *User interface*

In chapter 6 visualXSLT was introduced and tested. Still the user interface is that of a prototype. As mentioned in the testing and conclusions of chapter 6, the user interface can be improved by adding more guidance to the user. Guidance can be provided in different ways:

- the use of a problem walkthrough (a.k.a. a wizard)
- providing extra information of the elements at hand (use of tooltips etc.)
- a better look and feel (styled elements, better selecting etc.)
- providing more information on the matching (result preview, access to submatches, etc.)

There will of course be a lot more to improve, a good way to find out these points is to test the prototype in practice.

### *Testing in practice*

To get the prototype to a consumer level it should be tested in practise. This should reveal practical flaws and missing features at a fast pace. Testing in practise can also be used to get a closer insight in the matching problems for specific domains.

### **future research & development of prototype**

Also outside the current scope, research and development can be done. A key feature can be the implementation of automated matching which has not been used within this project. With that, extra features like thesauries and dictionaries can be researched and integrated to optimize results.

A good development would be the research and development of different types of matchers within the semantic based system. Using different types of matchers will reveal their pros and cons, which will give more insight in producing better result in specific situations.

Aside from the matching, the prototype can be enhanced by expanding the accepted input and output types. The system can be widened by adding extra features like for example a semantic matching environment dedicated to giving semantic meaning to elements, domains, matches and their connection.

A lot can be thought up to extend functionality and interaction, important questions to keep in mind are therefore: "will this feature simplify interaction?", "will this feature optimize results?"

## Appendix A. References

- [1] Wikipedia. Data transformation: a process to convert from a source data format into destination data involving data mapping and code generation. [http://en.wikipedia.org/wiki/Data\\_transformation](http://en.wikipedia.org/wiki/Data_transformation).
- [2] W3C. XML: Extensible Markup Language, a W3C-recommended general-purpose markup language derived from SGML (ISO 8879). <http://www.w3.org/XML/>.
- [3] Muze. Ariadne: Open Source, Multilingual Web Application Server and Content Management System. <http://www.ariadne-cms.org>.
- [4] Wikipedia. ETL: Extract, transform, and load, a process used in data warehousing for data transformation. [http://en.wikipedia.org/wiki/Extract\\_transform\\_load](http://en.wikipedia.org/wiki/Extract_transform_load).
- [5] S. Bossung, H. Stoeckle, J. Grundy, R. Amor and J. Hosking. Automated Data Mapping Specification via Schema Heuristics and User Interaction. In *Automated Software Engineering archive Proceedings of the 19th IEEE international conference on Automated software engineering*, pages 208-217. IEEE Computer Society, 2004.
- [6] K. Strehlo. Tactics to Data Transformation Nirvana. *White paper*. Data Junction, 2002.
- [7] Itemfield, The Complexity Advantage: Solving the Requirements of Complex Data Transformation in Business Integration. *White paper*. Itemfield, 11/2004.
- [8] Altova. Data Integration: Opportunities, challenges, and Altova MapForce™ 2005. *White paper*. Altova, 10/2005.
- [9] J. Madhavan, P.A. Bernstein and E. Rahm. Generic Schema Matching with Cupid. In *27<sup>th</sup> International Conference on Very Large Data Bases*, pages 49-58. Morgan Kaufmann Publishers Inc., 2001.
- [10] M. Smiljanic, M. v Keulen and W. Jonker. Formalizing the XML Schema Matching Problem as a Constraint Optimization Problem. In *DEXA 2005, 16th International Conference on Database and Expert Systems Applications*, pages 333-342. Springer, 08/2005.
- [11] M. Smiljanic, M. v Keulen and W. Jonker. Using Element Clustering to Increase the Efficiency of XML Schema Matching. In *22nd International Conference on Data Engineering Workshops*, page 45. IEEE Computer Society, 2006.
- [12] M. v Keulen, A. d Keijzer and W. Alink. A Probabilistic XML Approach to Data Integration. In *21<sup>st</sup> International Conference on Data Engineering*, pages 459-470. IEEE Computer Society, 2005.
- [13] A. Doan, P. Domingos and A. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *ACM SIGMOD Record archive*, volume 30, issue 2, pages 509-520. ACM, 06/2001.
- [14] M. Smiljanic, M. v Keulen and W. Jonker. Effectiveness Bounds for Non-Exhaustive Schema Matching Systems. In *22nd International Conference on Data Engineering Workshops*, page 83. IEEE Computer Society, 2006.
- [15] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. In *The VLDB Journal*, Volume 10, issue 4, pages 334-350. Springer-Verlag New York, Inc., 12/2001.
- [16] H.H. Do and E. Rahm. COMA: a system for flexible combination of schema matching approaches. In *28th international conference on Very Large Data Bases*, pages 610-621. VLDB Endowment, 2002.
- [17] Z.Kedad and X. Xue. An automatic tool for discovering complex mappings. *Technical Report*. University of Versailles, 2005.
- [18] E. Rahm, H.H. Do, D. Aumueller and S. Massmann. Matching Large Schemas with COMA++. In *Talk at Microsoft Research, Redmond*. University of Leipzig, 03/2005.
- [19] A.M. Memon. GUI Testing: Pitfalls and Process. In *Computer archive*, volume 35, issue 8, pages 87-88. IEEE Computer Society Press, 08/2002.
- [20] Wikipedia. GUI software testing: the process of testing a product that uses a graphical user interface. [http://en.wikipedia.org/wiki/GUI\\_software\\_testing](http://en.wikipedia.org/wiki/GUI_software_testing).
- [21] A.B. Bondi. Characteristics of scalability and their impact on performance. In *2nd international workshop on Software and performance*, pages: 195-203. ACM, 2000.
- [22] R.S. Pressman and D. Ince. Software Engineering, A Practitioner's Approach. In *5th edition*, chapter 1. McGraw-Hill Higher Education, 2001.

## Appendix B. Data conversion example

paragraph	What do I do in the streets of Serenia? eol
break	eol
	Hint: eol
paragraph	After you visit one of the shops in the town of Serenia, you can find a silver coin in the street. It will be located just outside the tailor's door at the entrance to an alley. There is also a barrel in the alley entrance. Be sure to look in the barrel and take the fish. eol
break	eol
paragraph	What do I do at the Tailor's Shop? eol
break	eol
	Hint: eol
paragraph	You can get a blue cloak from the tailor. But first you will need to get the golden needle from the haystack beside the Swarthy Hog's Inn. Give the tailor the golden needle and he will give you the cloak. Tip: You may need an army of help to find the needle! eol eof

Table 7:  $d_{input}$ , txt format

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="txt">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="meta" minOccurs="0">
          <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
              <xs:element name="key">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="title" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:sequence maxOccurs="unbounded">
          <xs:element name="break" minOccurs="0"/>
          <xs:element name="paragraph" minOccurs="0">
            <xs:complexType>
              <xs:sequence maxOccurs="unbounded">
                <xs:element name="line"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Table 8:  $S_{raw\ xml}$

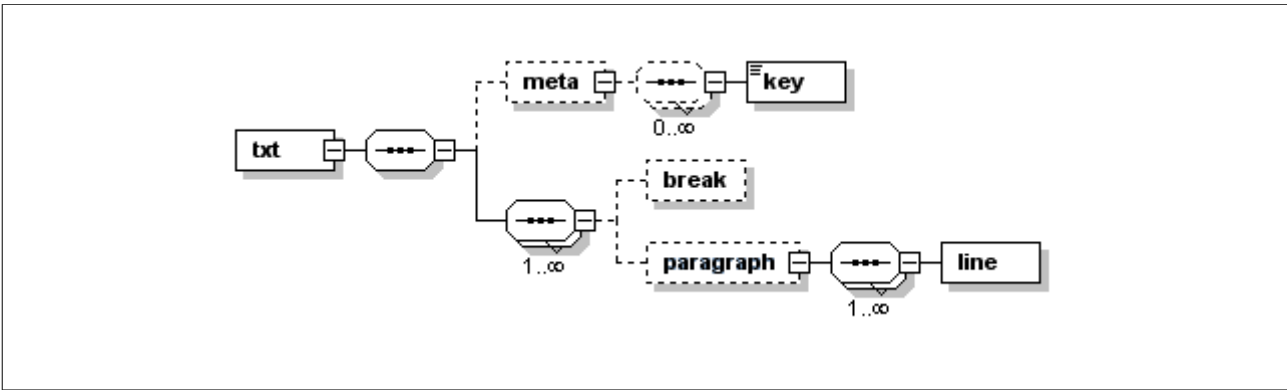


Illustration 20:  $S_{raw\ xml}$

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="walkthrough">
    <xs:annotation>
      <xs:documentation>Comment describing your root element</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title"/>
        <xs:element name="hint" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="question"/>
              <xs:element name="answer"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Table 9:  $S_{personal}$

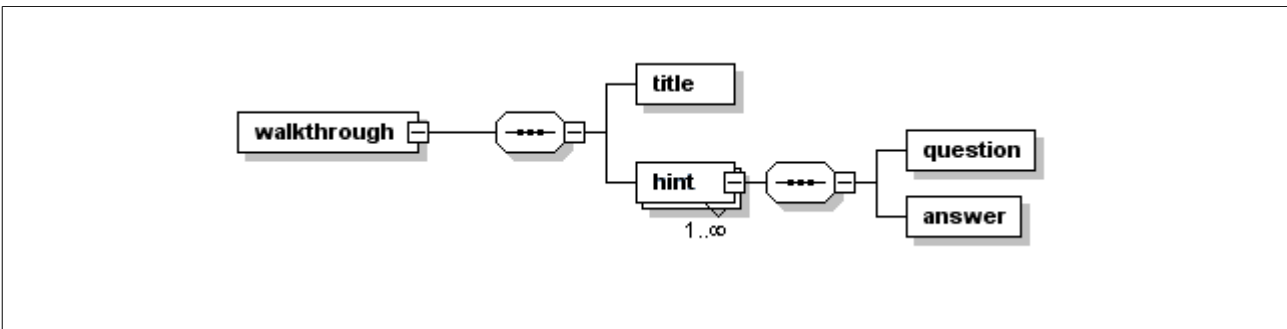


Illustration 21:  $S_{personal}$

txt	--> walkthrough
paragraph (odd)	--> question
paragraph (even)	--> answer
@key=filename	--> title

Table 10: *m*, intuitive mapping

```

<txt filename="kq5a.txt">
  <paragraph>
    <line>What do I do in the streets of Serenia?</line>
  </paragraph>
  <break />
  <paragraph>
    <line>Hint:</line>
    <line>After you visit one of the shops in the town of Serenia, you can</line>
    <line>find a silver coin in the street. It will be located just</line>
    <line>outside the tailor's door at the entrance to an alley. There is</line>
    <line>also a barrel in the alley entrance. Be sure to look in the</line>
    <line>barrel and take the fish.</line>
  <paragraph>
  <break />
  <paragraph>
    <line>What do I do at the Tailor's Shop?</line>
  </paragraph>
  <break />
  <paragraph>
    <line>Hint:</line>
    <line>You can get a blue cloak from the tailor. But first you will</line>
    <line>need to get the golden needle from the haystack beside the</line>
    <line>Swarthy Hog's Inn. Give the tailor the golden needle and he will</line>
    <line>give you the cloak. Tip: You may need an army of help to find</line>
    <line>the needle!</line>
  </paragraph>
</txt>

```

Table 11: *d<sub>raw xml</sub>*

```
<walkthrough>
<title>kq5a.txt</title>
<hint>
  <question>What do I do in the streets of Serenia?</question>
  <answer>
    Hint:
    After you visit one of the shops in the town of Serenia, you can
    find a silver coin in the street. It will be located just
    outside the tailor's door at the entrance to an alley. There is
    also a barrel in the alley entrance. Be sure to look in the
    barrel and take the fish.
  </answer>
</hint>
<hint>
  <question>What do I do at the Tailor's Shop?</question>
  <answer>
    Hint:
    You can get a blue cloak from the tailor. But first you will
    need to get the golden needle from the haystack beside the
    Swarthy Hog's Inn. Give the tailor the golden needle and he will
    give you the cloak. Tip: You may need an army of help to find
    the needle!
  </answer>
</hint>
</walkthrough>
```

Table 12:  $d_{intermediate}$

## Appendix C. Parse rules, schemas, meta data

### Txt parser:

#### Grammar:

$S \leftarrow B^*Q^* \mid \epsilon$ $Q \leftarrow P \mid PB^*S$
--

B = Break

P = Paragraph

This means that a txt consists of breaks and paragraphs, or be empty. Paragraphs are separated by at least one break. Breaks can follow breaks. Paragraphs can not follow paragraphs.

The txt parser reads the source data per line.

#### Possible line reads:

empty line (el):

only a end of line character is read

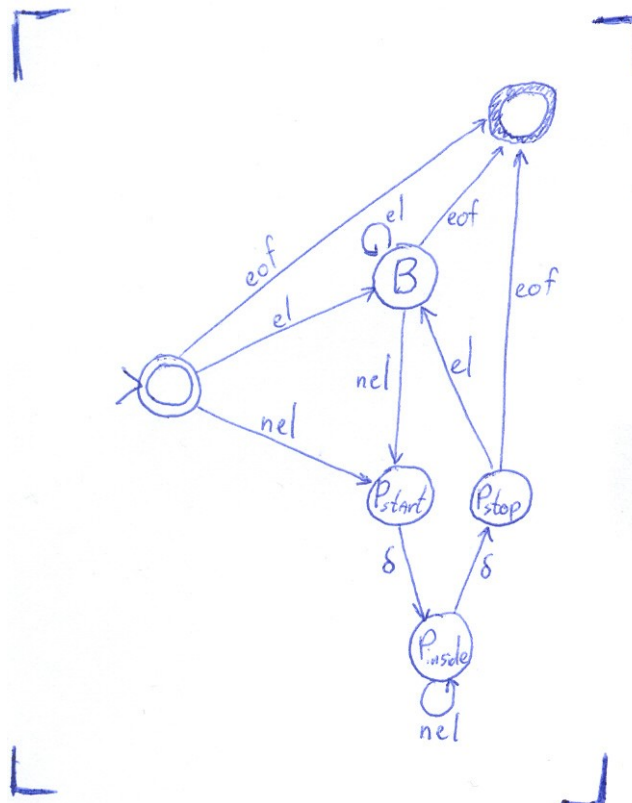
non-empty line (nel):

more then one character is read before the end of line character

end of file (eof):

the end of file is read

#### State machine



### *XML Schema*

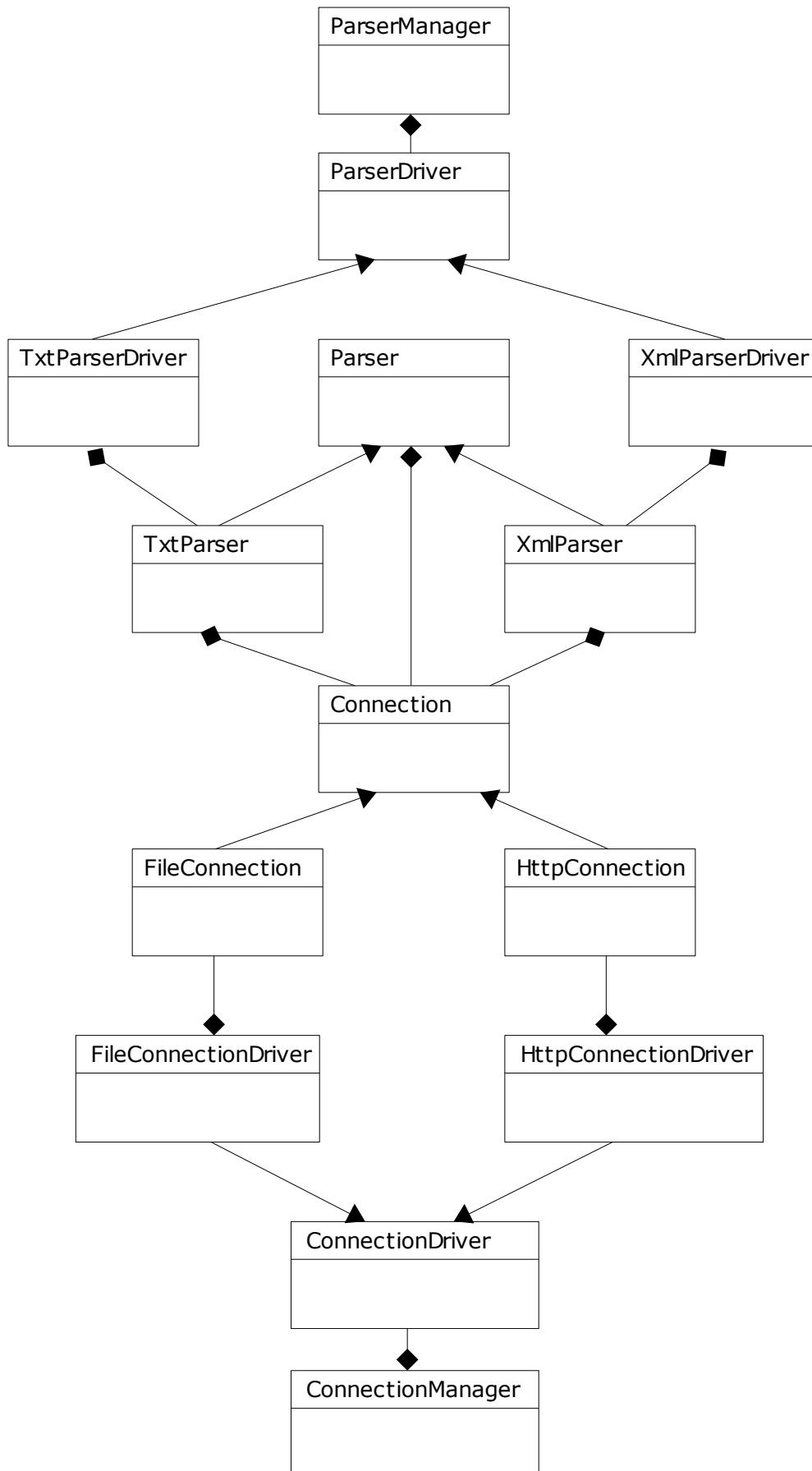
See Appendix B, table 8 and illustration 20.

#### *Meta data examples*

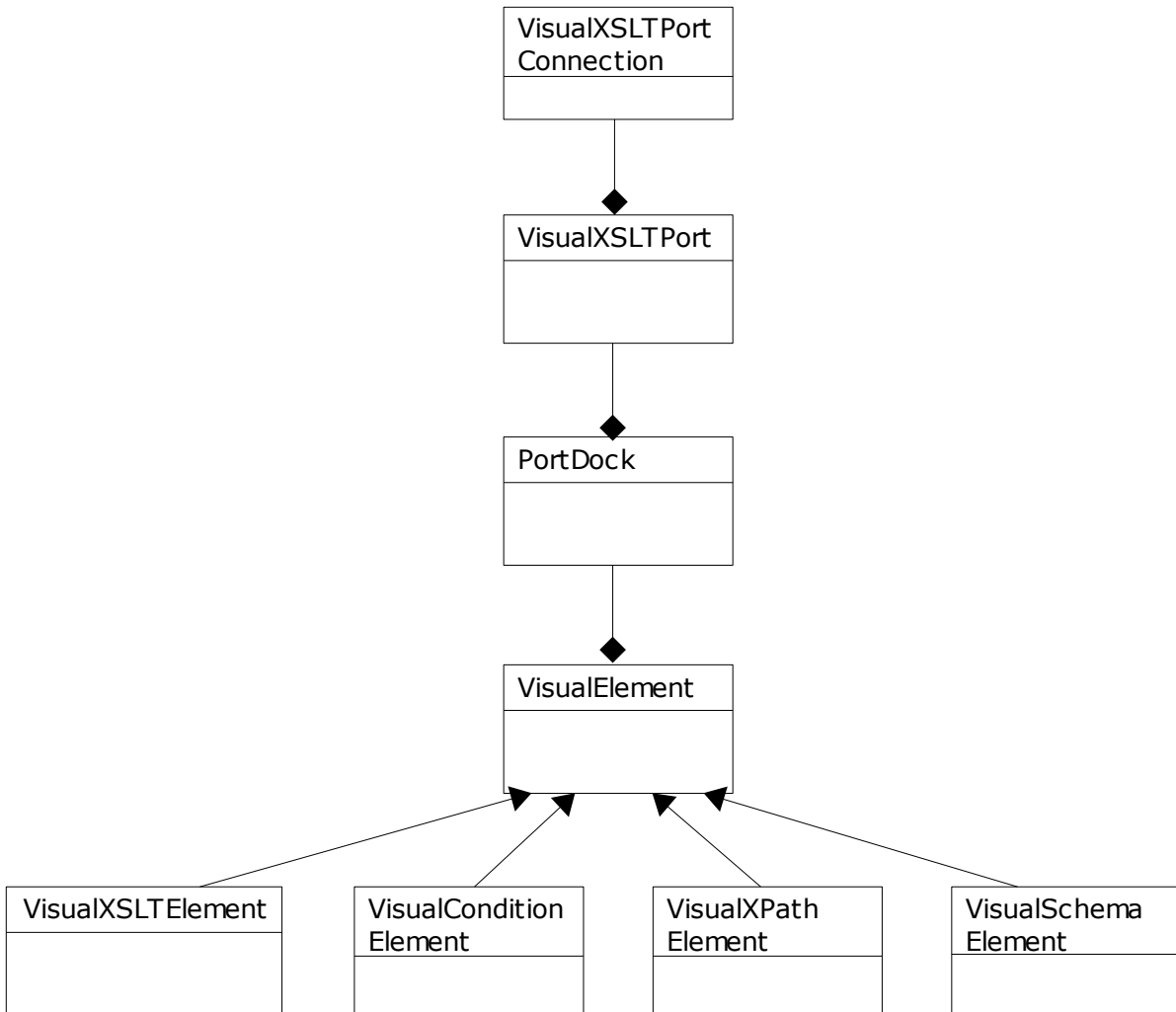
<b>name</b>	<b>value</b>	<b>source</b>
filename	filename.ext	file
creation time	timestamp	system
filesize	integer	file
description	text	user input
domain	text	user input
etc.	etc.	etc.



## Appendix D. Connection and driver classes



## Appendix E. Visual classes



# Appendix F. Test results E1

Stefan

**Survey Prototype Visual Matching**

slecht      goed  
nee          ja  
O O O O O O

**Flexibiliteit**

Wat vind je van de flexibiliteit van het matchen?	O O O O ● O
Kun je makkelijk matchings maken en aanpassen vanuit je mentale beeldvorming van het matching probleem?	O O O O ● O

Wat vind je positief qua flexibiliteit?  
*Verschillende mogelijkheden om van 1 centrale bron meerdere doelen te maken (html, comma, ...)*

Wat vind je negatief qua flexibiliteit?

Welke toevoegingen zou je willen zien qua flexibiliteit?

**Gebruiksvriendelijkheid**

Wat vind je van de gebruiksvriendelijkheid van het matchen?	O O O ● O O
Zijn de matching onderdelen makkelijk inzichtelijk?	O O O O O ●

Wat vind je positief qua gebruiksvriendelijkheid?  
*Duidelijke interface m.b.t. source & target document. element eigenschappen zijn duidelijk*

Wat vind je negatief qua gebruiksvriendelijkheid?  
*Bij complexe matchings kan het onoverzichtelijk worden*

Welke toevoegingen zou je willen zien qua gebruiksvriendelijkheid?  
*Meer overzicht bij complexe matching*

**Intuïtieve werking**

Wat vind je van de intuïtieve werking van het matchen?	O O O O ● O
Is de werking analoog aan je mentale beeldvorming van een matching?	O O O O O ●
Is de werking analoog aan je mentale beeldvorming van XSLT?	O O O O O ●

Wat vind je positief qua intuïtieve werking?

Overzichtelijkheid geeft je gelijk inzicht in het feit dat je mappings moet maken

Wat vind je negatief qua intuïtieve werking?

Moeilijk te beginnen als je de applicatie niet gebruikt

Welke toevoegingen zou je willen zien qua intuïtieve werking?

### Overzichtelijkheid

Wat vind je van de overzichtelijkheid van het matchen?	0 0 ● 0 0 0
Is de opdeling in bron, mapping en doel inzichtelijk?	0 0 0 0 0 ●
Wat vind je van de koppeling van elementen en lijnen?	0 0 0 0 0 ●

Wat vind je positief qua overzichtelijkheid?

De opdeling van bron, mapping & doel

Wat vind je negatief qua overzichtelijkheid?

De koppelingen worden onoverzichtelijk bij complexe mapping

Welke toevoegingen zou je willen zien qua overzichtelijkheid?

Oplossing voor complexe mapping.

### Herstel mogelijkheden

Wat vind je van de herstel mogelijkheden binnen het matchen?	0 0 0 ● 0 0
Moet je veel moeite doen voor kleine aanpassingen?	● 0 0 0 0 0

Wat vind je positief qua herstel mogelijkheden?

Applicatie gaat slim met delete om, verwijst ook de lijnen (koppelingen)

Wat vind je negatief qua herstel mogelijkheden?

Mogelijkheid undo/redo, copy/paste niet aanwezig maar begrijpelijk in prototype

Welke toevoegingen zou je willen zien qua herstel mogelijkheden?

Geen, is & blijft prototype

### Informatie voorziening

Wat vind je van de informatie voorziening van het matchen?	0 0 0 0 0 0 0
Is de informatie toereikend voor inzicht verschaffing in de koppelingen?	0 0 0 0 0 0 0

Wat vind je positief qua informatie voorziening?

Informatie van elementen & mapping duidelijk  
visueel

Wat vind je negatief qua informatie voorziening?

Welke toevoegingen zou je willen zien qua informatie voorziening?

### Categorische aanpak

Zie je een categorische aanpak in het prototype?	0 0 0 0 0 0 0
Wat vind je van de categorische aanpak in het prototype?	0 0 0 0 0 0 0

Wat vind je positief qua categorische aanpak?

Duidelijke weergave bron, mapping & doel

Wat vind je negatief qua categorische aanpak?

Poorten zijn in eerste instantie onduidelijk

Welke toevoegingen zou je willen zien qua categorische aanpak?

Mouse over (tooltip) bij poorten

### Helderheid oorzaak gevolg

Is de oorzaak en gevolg uitwerking helder in de matching?	0 0 0 0 0 0 0
Is de oorzaak en gevolg uitwerking analoog aan XSLT?	0 0 0 0 0 0 0
Kom je onduidelijke gevolgen tegen?	0 0 0 0 0 0 0

Wat vind je positief qua oorzaak en gevolg?

Je kunt makkelijk het resultaat bekijken.  
Hier kan je makkelijk via debuggen.  
→ doel

Wat vind je negatief qua oorzaak en gevolg?

Soms kun je lijnen ~~na~~ connecten die eigenlijk niet mogen, binnen zelfde element

Welke toevoegingen zou je willen zien qua oorzaak en gevolg?

Duidelijke foutmelding bij verkeerde koppelingen.

#### Bondigheid informatie

Is de verstekte informatie bondig?	0 0 0 0 0 0 0
Is de verstekte informatie te bondig?	0 0 0 0 0 0 0

Wat vind je positief qua bondige informatie?

Alle info is zeer duidelijk, meer voor een non-programmer wel te kechnisch

Wat vind je negatief qua bondige informatie?

Welke toevoegingen zou je willen zien qua bondige informatie?

#### Intuïtieve visualisatie

Is de visualisatie intuïtief voor de matching als geheel?	0 0 0 0 0 0 0
Is de visualisatie intuïtief voor de werking van XSLT elementen?	0 0 0 0 0 0 0

Wat vind je positief qua intuïtieve visualisatie?

Je krijgt het gevoel dat je met een xslt editor werkt!

Wat vind je negatief qua intuïtieve visualisatie?

Welke toevoegingen zou je willen zien qua intuïtieve visualisatie?

**Open Opmerkingen:**

Duidelijke applicatie wat na een korte uitleg goed gebruikt kan worden.

De interface is duidelijk, en een perfecte visualisatie m.b.t matching.

Visualisatie zou onoverzichtelijk kunnen worden bij complexe documenten.

Voer niet iet'ers wel makkelijk in gebruik

Top prototype, zie hier veel toekomst in, ben benieuwd of InDialog verder zal gaan met doorontwikkeling, zou het absoluut adviseren.