Scalable and Reuse-Oriented Data Integration

A Distributed Semi-Automatic Approach

Matthijs Gerard Dabroek

Department of Information and Computing Sciences
Utrecht University

A thesis submitted for the degree of
Master of Business Informatics

First supervisor
dr.ir. J.M.E.M. van der Werf
Utrecht University

Second supervisor
dr. F.J. Bex
Utrecht University

External supervisor
W. Wezelman, MSc.
ING Bank N.V.

October, 2016

Abstract

In the current information age, it is crucial for an organization to integrate all of its available source systems to provide deep insights, adhere to regulations, or provide a competitive edge. However, data integration often proves to be a tedious and costly process. In this study we aim to answer the question: can we formulate and construct a semi-automatic distributed system to enable scalable and reuse-oriented data integration?

To aid the process of data integration we propose a system that takes advantage of previously provided associations between source schemas. To provide a common language between disparate sources, we introduce the use of an ontology to our proposed solution. We attempt to semi-automatically match attributed from the schemas of source systems to entities of the ontology, by utilizing a self-learning aspect and a feedback loop. We attempt to achieve this by applying a dual approach, using both semantical and structural aspects of the source and the ontology.

We constructed a proof-of-concept and performed a user acceptance study to evaluate our approach and validate our solution. Our contribution is two-fold: we *distribute* our data integration system, thereby contributing to the scalability of the system, and we *reuse* previously obtained results to enable semi-automatic matching.

We performed both quantitative and qualitative analysis to evaluate the accuracy and feasibility of our system. The outcome suggests that our solution has merit and shows that end-users have positive expectations towards its use and performance.

Contents

List of Figures

Acknowledgements

First of all, I want to thank Jan Martijn van der Werf, my academic supervisor, for providing me with the continuous input and feedback needed to pinpoint on the topic, and steer the research in the right direction. I also want to thank Floris Bex, as my second academic supervisor, for taking the time to read and evaluate my work.

A special thanks to my company supervisor, Wouter Wezelman, for giving me the opportunity to perform this research in the context of a large organization. His supervision, good advice, and backing enabled me to speak to the right people, have proper resources, and bring this research to a successful conclusion.

Next, I want to thank several other colleagues, Fekke Odijk, Sandra Wennemers, Siem Lakeman, Erik Baauw, and Peter Leenes, for taking time and providing me with a complete and in-depth insight into the processes and procedures of the organization. Also, Stefan de Jong, Lucian Baghiuc, and Jacob Goense, for their input and knowledge on the local source systems and related processes.

Likewise, big thanks to data scientists Robert Rodger and John Muller, for giving me insight into machine learning techniques, data science approaches, and other useful feedback on my proof-of-concept. Similarly, my sincere gratitude to everyone else I've talked to in the organization, who has helped me develop my ideas and gave their input on my proof-of-concept.

I want to thank my fellow master students, especially Marja Tiekink and Courtney Schriek, for their patience in our bi-weekly lunches, while listening to me rambling on about my work.

A big thanks to my father, Gerard Dabroek, for reading my work and for his comprehensive review and helpful suggestions. And last but not least, Katia Agnamazian, for being my team member and trusted confidant, providing me with all the patience, love, inspiration, and support needed to endure and finish this project.

Chapter 1

Introduction

1.1 Problem Statement

Major corporations deal with a plethora of (legacy) systems developed in different programming eras, each with their own standards and paradigms. Data is spread across multiple systems, formats, languages, and divisions, frequently resulting in an overly complex network of connected IT. Information is often hard to find, slow to collect, and difficult to analyze. In the current information age, coupling systems to create added business value and deeper business insights is inevitable to keep a competitive edge. Information on demand depends on the ability of the enterprise architecture to find and combine all key data in time, no matter the format or location.

For financial institutions, data integration is vital not only to add business value, but also for accurate statutory and compulsory regulatory reporting. Whether it is externally, for regulator and investor communities, or internally, for managerial decision-making, customer intelligence, or advanced analytics, information from many systems across the enterprise is required. Aggregating and interpreting the correct information scattered over numerous source systems, is a tedious and complex process. An abundance of sources need to be identified and a vast amount of data has to be interpreted. After correctly identifying the required information, it has to be extracted from the source systems, transformed into usable data, and loaded into a data warehouse for further interpretation. Many data owners, business analysts and system experts, are working in parallel to manually identify information and connect source systems. Previously identified knowledge is often ignored and reuse overlooked. Manually creating these so-called mappings between source systems and a common agreed-upon standard tends to be slow, repetitive, costly, and error prone.

Yet aggregating business data on a large scale proves to be extremely difficult, complex, and demanding (Gagnon, 2007). Companies commonly use traditional manual extract, transform, load (ETL) procedures to translate data from one source system to the other, a tedious and highly customized process. Because of this, business value and agility is lost, due to production costs and limited time.

What is needed is to integrate the data from several disparate sources. Several definitions for data integration exist in literature. Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data (Lenzerini, 2002). Data integration is a set of techniques that enable building

systems geared for flexible sharing and integration of data across multiple autonomous data providers (Doan et al., 2012).

The notion of data integration is becoming more and more relevant while the magnitude of available data sources increases steadily and data creation is occurring at a record rate (Villars, Olofson, & Eastwood, 2011). Data growth challenges and opportunities are often referred to with the term big data. Gartner defined three – now widely used – characteristics of big data, i.e. increasing volume (amount of data), velocity (speed of data in and out), and variety (range of data types and sources) (Beyer & Laney, 2012). But piles of data are useless to businesses if there is no good way to retrieve, observe, and interact with that data to generate tangible information and practical knowledge. A strong data integration strategy can help companies to harness information coming in from every direction and use it to support their business goals.

A common issue in large enterprises is that the conceptual model of data differs greatly between systems. Systems can hold similar entities with completely different attributes, or common attributes with a different semantic representation. This issue is not only common between applications, but can also occur within applications, particularly when that application is divided into separate components. Since the advent of the microservice architecture, data integration has gained a renewed wave of attention. As more and more applications are being deployed in the cloud, monolithic applications are being broken down into independently deployable and scalable services (Lewis & Fowler, 2014). This involves rethinking data sharing between different components of the system.

A drawback of traditional data integration methods is the centralization of authority and governance over the integrated systems, as depicted in Figure 1.1. Data integrators need to have a global overview of all the resources spread over the source systems. Experts need to be involved and consulted every time a change is introduced in a source system. This creates bottlenecks in the development of these systems.

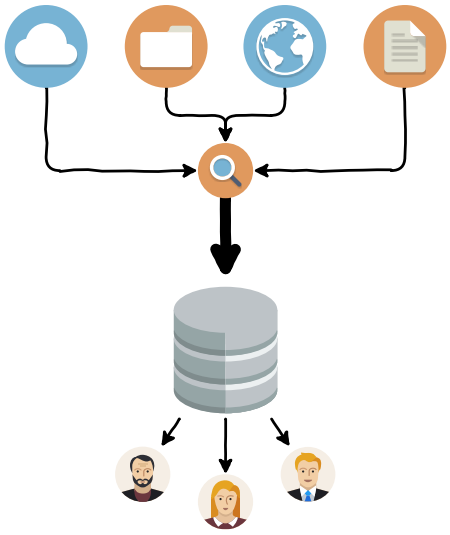


Figure 1.1 Traditional Integration Approach

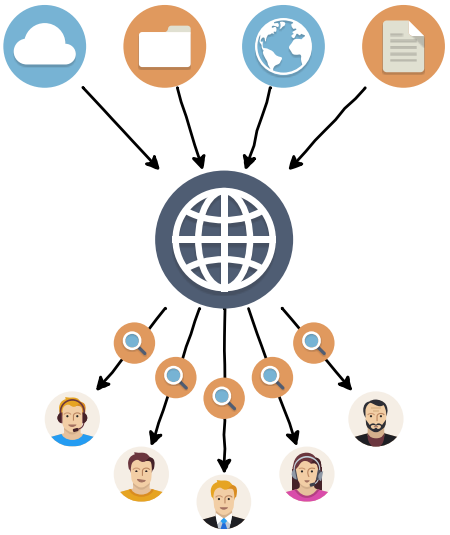


Figure 1.2 New Integration Approach

Therefore, new ways to integrate data sources are required, since traditional ways of integrating simply require too much modelling, maintenance, and coordination among the owners of the data sources. Owners of data sources need to be able to collaborate without any central authority or global standardization, as depicted in Figure 1.2. The ability to scale up to more sources requires us to redesign architectures to exploit the power of large clusters. That means challenges like schema matching will need to be tackled in a much more parallelizable and scalable way.

We hypothesize that an important piece missing in existing distributed solutions is scalable reuse between source systems. None of these systems retain acquired knowledge in a scalable and resilient fashion. What is lacking is the capability of the system to distribute, cache, and reuse previously made associations by expert knowledge or machine learning.

Much research has been conducted and tremendous progress has been made on the topic of data integration (A. Halevy, Rajaraman, & Ordille, 2006). Despite the amount of progress, there are still a number of open research challenges and issues, to two of which this thesis tries to find a solution. First is a scalable, distributed solution to the tedious process of matching and interlinking the data models between the distinct data sources. Second is the lack of industry support for decentralized data integration solutions.

1.2 Objective

The objective of this research is to provide means and a proof of concept to aid the process of distributed data integration, without shifting the responsibility of the system experts to a centralized authority, to improve scalability and resilience. The goal is to provide an unobtrusive solution, without disrupting - or with minimal impact on - the established process, by building a knowledge system where mappings are being created and stored to easily retrieve data that is needed. This way we aim to enable continuous dynamic integration of regularly connecting and disconnecting peers.

The goal of this research is to aid the process of mapping local resources to a shared, common language. We aim to do this by:

- improving *scalability* and *resilience* by distributing the process;
- reducing *complexity* and *costs* by localizing responsibility;
- improving *consistency* by reducing replication of data;
- improving *efficiency*, *accuracy* and *automation* by reusing previous results.

1.3 Outline

In Chapter 2 we illustrate our research approach. We start by presenting our research questions, followed by the combined research methods approach we selected, to show how they will support our research.

In Chapter 3 we present the results of our literature study. We define the main concepts like data integration, and introduce important related concepts. We also explore several architectures and techniques that can support our objective.

In Chapter 4 we present our devised solution. We describe the different aspects of the solution, present a supporting architecture, and explain how it can be applied to support the process of data integration.

In Chapter 5 we explain how we devised our proof-of-concept. We explain how we implemented several of the components of our proposed architecture and give a preview of the systems' interface.

In Chapter 6 we evaluate our work. We describe the details of our case study and present the results of both the quantitative and qualitative analysis we preformed.

In Chapter 7 we discuss our findings and their implications. We discuss the threats to the validity of our research, address the limitations of our research, and provide some recommendations for the case study organization.

In Chapter 8 we describe explicitly how we answered the questions posed in Chapter 2 and provide several suggestions for future work on this research topic.

Chapter 2

Research Approach

2.1 Research Questions

2.1.1 Main Research Question

Based on the before-mentioned problem statement and objective we have formulated the following main research question.

Can we formulate and construct a semi-automatic distributed system to enable scalable and reuse-oriented data integration?

Our main research question contains two key properties of the system we aim to construct. Semi-automatic, to reduce the dependency on system experts and reuse previously acquired knowledge, and distributed, to ensure the scalability of the system.

2.1.2 Sub Research Questions

The main research question can be divided in several sub questions. Together they should provide a satisfying answer to our main research question.

I. *How can distributed data integration be accomplished?*

In order to provide source system experts with full control and governance over their data flowing into the system, it is key to decentralize the process.

II. *Which techniques support distributed data integration?*

Next to the peer-to-peer negotiation and consistency algorithms, there are numerous techniques to support the data integration process. In our research we will look at protocols and algorithms for peer-to-peer connections, distributed hash tables, and consensus and consistency algorithms. We aim to assess and discern the optimal techniques needed for our proposed solution.

III. *How can these techniques be combined in a system architecture?*

We aim to provide an architecture in which we combine the techniques found for *SRQII* into an integrated system. Theory and practice should come together to form a satisfying system architecture.

IV. *Can the applicability or usability be demonstrated with an implementation or proof-of-concept?*

We will create an implementation of our proposed solution to demonstrate that the concept is feasible and performant.

V. *Can it be applied in large-scale enterprises and does it work?*

By applying our concept in a real-world environment, we intent to show that our implementation can be of benefit in a practical and commercial context.

2.2 Research Method

For our research we combine design science research with case study research. This way we combine (Figure 2.1) a structured and theoretical approach with a rigorous hands-on approach, in which we can build, refine, and evaluate our theories. This method will be supported by a thorough literature research to support and structure our stated questions and propositions.

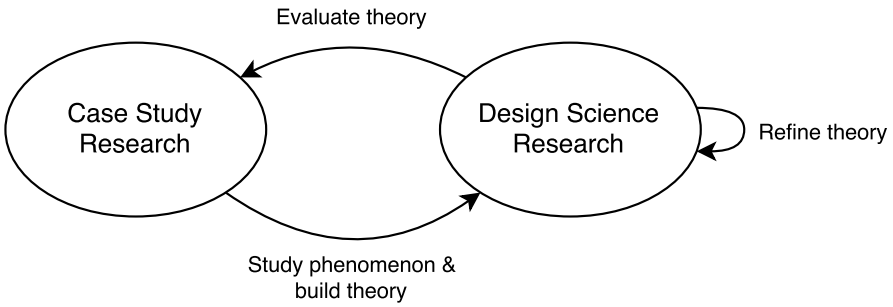


Figure 2.1 Combined Research Approach

2.2.1 Design Science Research

We address the problem based on the four basic problem solving principles of Pólya (1945) taken from his book, 'How to Solve It'; understand the problem, devise a plan, carry out the plan, and look back. Additionally, we follow the design science approach described by Hevner, March, Park, and Ram (2004). Design science is all about problem solving in a structured way to evaluate if your solution actually solves your problem. Design science is contrasting from natural science in the fact that we examine and stage artificial phenomena, instead of observing natural phenomena occurring in the "real world".

We start in the problem domain by defining our problem as it exists in the real world. By analyzing the problem, we elevate to a conceptual level referred to as the abstraction domain. In the abstraction domain we can devise or construct a solution to our generalized problem. By implementing the solution, we return to the physical level by creating a tangible artifact that allows us to evaluate our solution on our initial problem.

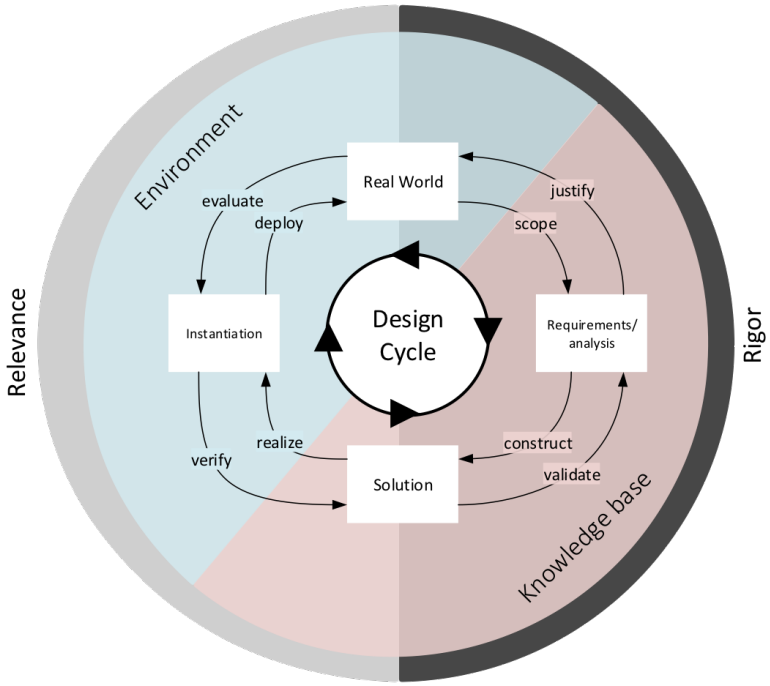


Figure 2.2 Design Cycle, reprinted from Schenkhuizen (2016)

This cyclic approach, as depicted in Figure 2.2, can be done iteratively to fine-tune our solution and make sure we actually solve our problem.

We make use of a running example to illustrate our solution and make the theory more easily to grasp. As domain of the running example we chose the average medium to large-sized enterprise. Eventually we evaluate our solution by applying it in a case study at a major corporation.

2.2.2 Literature Research

First a literature study is conducted to gain insight in the maturity of the research area. Next, we aim to identify gaps in the current research and provide new insight into the field by devising and constructing a generic solution through a mash up of existing techniques. In our research we ended up exploring a wide variety of topics like peer-to-peer algorithms, ontology matching, the semantic web, and machine learning.

The literature research consists of a semi-structured literature review using the snowballing process. Snowballing refers to a continuous, recursive process of searching, scanning and aggregating references of articles, books, and other research documents. Our review of the literature aims to confirm that a gap exists and that our proposed research can lead to interesting scientific insights.

2.2.3 Case Study Research

Our case study research follows the research design as proposed by Yin (2003) in his book, Case Study Research. We will elaborate on the five components of a research design mentioned in the chapters of his book.

As indicated in the previous section, our case study research begins with a thorough literature review and the careful and thoughtful posing of research questions and objectives. We perform literature research to rely our analytical strategy on theoretical propositions. We aim to follow explicit procedures when doing our research, to protect against threats to validity, maintaining a chain of evidence. In this effort we follow Yin (2003) his blueprint of case study research as a "linear but iterative process". We organized this research according to the six elements of case study research: the plan, design, preparation, data collection, analysis, and reporting.

> The essence of a case study, the central tendency among all types of case study, is that it tries to illuminate a decision or set of decisions: why they were taken, how they were implemented, and with what result. (Schramm, 1971)

In general, case studies are the preferred strategy when "how" or "why" questions are being posed, when the focus is on a contemporary phenomenon within some real-life context (Yin, 2003). Therefore, a case study provides detailed contextual views on our phenomenon of interest. In other words, we place our solution in a real-world context, to observe and analyze its efficiency and effectiveness. As the quote by Schramm (1971) states, we try to shed light on the set of steps we have taken to explain why, how, and to what outcome they have led us.

A single-case design case study will be performed at a Dutch multinational banking and financial services corporation headquartered in Amsterdam with a worldwide workforce exceeding 75-thousand. On one hand this case study is intended to identify and validate the problem, and on the other hand to propose, construct, and evaluate a generic solution. Finally, we aim to design an architecture and proof-of-concept that aids the process of data integration through reuse.

To support our approach, interviews with practitioners will be conducted to assess if our problem statement is valid and to justify our approach is addressing the correct problem. Afterwards, to support our evaluation, we will conduct interviews with practitioners to assess if our proof-of-concept is doing what it should do, which is to aid the process of integration. Furthermore, to judge if there is any platform of support by looking at user acceptance.

Chapter 3

Data Integration

Integration comes from the Latin word *integer*, meaning whole or entire. It generally refers to combining components so that they work together or form a whole. Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data (Lenzerini, 2002). It is relevant to numerous applications in both commercial and scientific context, like in enterprise information integration, medical information management, geospatial information systems, and commercial applications.

Data integration has been the focus of research since the early eighties (Draffan & Poole, 1980; Smith et al., 1981). Since then, there has been extensive theoretical research and commercial development in this area. In the past five years, a tremendous effort has been made in producing comprehensive overviews of this field and many of its peripheral subjects (Bellahsene, Bonifati, & Rahm, 2011; Özsu & Valduriez, 2011; Doan et al., 2012; Shvaiko & Euzenat, 2013).

Data integration involves joining, transforming, enriching, and cleansing data, but what it doesn't enforce is *how* the integration takes place. Traditional data integration began with extract, transform and load (ETL) tools designed to automate efforts to pull data from source systems, convert it into a consistent format and load it into a data warehouse or other target database. This process is commonly referred to as the data warehouse approach.

More recently, as data sources continue to increase and get updated more frequently, organizations need a more flexible approach. The data warehouse approach simply does not provide the real-time or near-real-time decision-making that is required by some applications. The rise of the service-oriented architecture also stimulated the interest in more decoupled or loosely coupled solutions to data integration. This is when the field of data virtualization started to develop. Data virtualization enabled a more agile approach to data integration. In the following chapter we will explain several architectures, techniques, and related concepts of data integration.

3.1 Data Integration Architectures

Integration can either be done *physically* or *logically* (Jhingran, Mattos, & Pirahesh, 2002). In the former, the source databases are integrated and the integrated database is materialized. These are better known as *data warehouses*. The integration is performed

by ETL tools, which perform a multi-phase process. This process consists of extracting data from sources, transforming the data to match the global conceptual schema, and loading them in a target database. This process is depicted in Figure 3.1. In the latter, also referred to as *virtual integration*, the global conceptual (or mediated) schema is entirely virtual and thus not materialized at all (see Figure 3.2).

Traditionally, there are two approaches for designing a virtual data integration system. In the *global-as-view* (GAV) approach, one defines the concepts in the global schema as views over the sources, whereas in the *local-as-view* (LAV) approach, one characterizes the sources as views over the global schema. Calì, Calvanese, De Giacomo, and Lenzerini (2001) describe an approach to accessing data integration systems by specifying the global schema in terms of a conceptual data model.

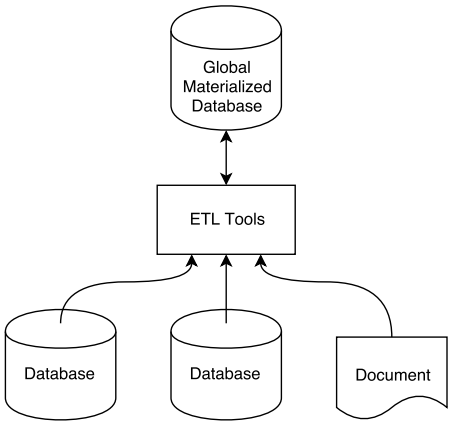


Figure 3.1 Physical Integration

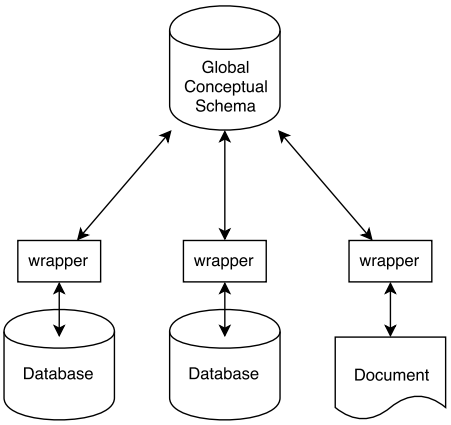


Figure 3.2 Logical Integration

Note that physical distribution does not necessarily imply that the computer systems are geographically apart; they could very well be in the same location. It simply implies that the communication between them is done over a network instead of through shared memory or shared disk, with the network as the only shared resource. In virtual integration the data remains in the sources, and is accessed when needed at query processing time. In our research we focus solely on virtual integration.

3.2 Data Integration Techniques

3.2.1 Schema Mapping and Matching

One of the most vital operation for all data integration systems is identifying how a source database schema relates to the target, integrated schema. *Schema matching* is the problem of generating correspondences between elements of two schemas (Bernstein, Madhavan, & Rahm, 2011). As it turns out, creating mappings is one of the main bottlenecks in developing data integration systems. This task is often difficult because it requires a deep understanding of the semantics of the data sources. Hence, focusing on techniques that reduce the time required from a person to create mappings is essential.

There are two main sources of information to investigate when trying to match data of two source systems. First of all, there is the source schema generally consisting of entities with their attributes, and the relationships between these entities. This metadata is often referred to as the source description and proves to be a vital source of information. Without this metadata, it would be much more difficult to apply matching strategies. The source schema can be analyzed with a different level of granularity. This means that matching can be performed for individual schema elements (i.e. attributes) or for combinations of elements (i.e. complex structures of attributes).

The second fundamental source of information is the data itself. The raw data contained in a system provides all kinds of hints about what kind of data we are dealing with. This can also be used to match entities over different source systems. This is referred to as the *instance-level* approach, opposing the previously described *schema-level* approach.

Matches between source schemas and matches between data can be found using several different techniques. In the following sections we will briefly describe a few of them. We describe two approaches of matching, one that compares the names of schema elements and another that compares the data instances.

Name-based Matching

String matching deals with the problem of finding strings that refer to the same real-world entity (Doan et al., 2012). String matching is an important technique used in many data integration tasks, including both schema matching and data matching. But similar entities in different source systems are often labeled in a completely different way. It is common for element names in source descriptions to comprise of acronyms or abbreviations. Matching may require to split these names on certain delimiters (i.e. capitals) and expanding acronyms and abbreviations using domain-specific dictionaries. Rahm and Bernstein (2001) name several commonly used metrics in their survey of approaches to automatic schema mapping: equality of string (in the same or similar namespace), equality of canonical name representations–after stemming and other preprocessing (also referred to as root form or word stem), equality of synonyms and/or hypernyms, and similarity based on common substrings, string metrics like edit distance, and soundex (i.e. a phonetic algorithm for comparing names by sound, as pronounced in English). Lastly, matching can be done with a dictionary approach. This enables domain-specific vocabulary to be replaced with more generic terms. Examples of these approaches can be found in Table 3.1. This approach is also referred to as the *language-based* or *linguistic* approach.

Instance-based Matching

Merely matching the schema-level is often not enough to find matching elements. This is where we can employ the expressiveness of the data instances. Instance-level data can give important insight into the contents and meaning of schema elements. This is especially true when useful schema information is limited, as is often the case for semi-structured data.

Matching approach	Example
Equality of string (in the same or similar namespace)	Name equals name (but product name does not equal employee name)
Equality of canonical form	EName equals employee name and EmpNO equals employee number
Equality of synonyms	Employee equals worker and salary equals wage
Equality of hypernyms	Manager is-a person and employee is-a person, implying manager ≅ employee
Similarity of names based on common substrings, edit distance, or soundex	managedBy ≅ manager and reportsTo ≅ Reports2
User-provided or dictionary name matches	handledBy ≅ employee

Table 3.1 Semantic-based Matching Approaches (Rahm & Bernstein, 2001) with Examples

Since schema matching tasks are often repetitive, and often performed in a particular domain, similar concepts tend to reoccur. This observation implies schema matching should be able to improve over time. Machine learning has a strong case for repetitive tasks. It enables the system to improve matching accuracy over time.

A variety of classification techniques have been proposed to perform such an instance matching or classification, such as rules, neural networks, and machine learning techniques. Two common learners are the *rule-based* learner and the *naive Bayes* learner.

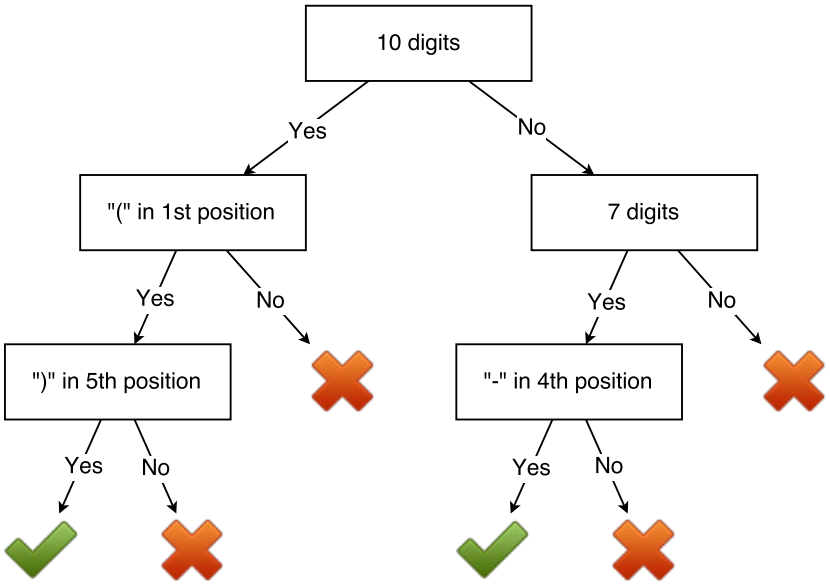


Figure 3.3 Decision Tree to Determine a Phone Number, adapted from Doan et al. (2012)

A rule-based learner examines a set of training examples and computes a set of rules that can be applied to test instances. These rules can be represented as decision trees, as can be seen in Figure 3.3. After that, new data can be positively or negatively classified by simply applying the earlier produced rules.

The naive Bayes learner examines the attributes of a given instance and assigns to the instance the most likely class, given the occurrences of attributes in the training data. Which means it estimates the probability of a correct classification by computing the deviation for each of the existing classifications and predicts the one with the closest similarity, which must be the most likely.

Instance-level matching can also be performed by exploiting available auxiliary information. For example, previously found mappings obtained from matching different schemas can be reused to improve the mapping accuracy. Examples of this technique are further discussed in the section about existing machine learning systems. Consider that there is not always the possibility to use the instance-level to improve the quality of the matching process.

3.2.2 Semantic Integration with Ontologies

The application of *ontologies* for the sharing and reuse of knowledge among software systems is not a new idea. Hull (1997) already mentioned an "ambitious" framework by DARPA, supporting automated data integration using ontologies. Uschold and Gruninger (2004) predicted that ontologies in particular and semantics-based technologies in general will play a key role in achieving seamless connectivity.

Gruber (1991) introduced the concept of "building libraries of shareable, reusable knowledge in which common ontologies play a central role as a knowledge coupling construct". Ontologies are first mentioned in an information science context by Gruber (1991) as "coherent sets or vocabularies of representational terms, together with textual and formal definitions, that embody a set of representational design choices". Later, Gruber (1993) presented the more widely accepted definition of an ontology as an "explicit specification of a conceptualization". In (1997), Borst defined an ontology as a "formal specification of a shared conceptualization". Studer, Benjamins, and Fensel (1998) merged these two definitions stating that: "An ontology is a formal, explicit specification of a shared conceptualization." Guarino, Oberle, and Staab (2009) extensively describe all the elements of this definition in their chapter "What is an Ontology?" in the *Handbook on Ontologies*. In short, they state that without at least a minimal shared ontological commitment from ontology stakeholders, the benefits of having an ontology are limited. This is why it is important for those ontologies to be well-founded, to support large-scale interoperability. More recently, Lenzerini (2011) coined the notion *ontology-based data management* (OBDM) for this approach.

A *domain ontology* represents concepts which belong to a particular domain. They give meaning in the context of this domain, while in the context of another they could mean something entirely different. Examples of this are window (display rectangle in a graphical user interface, or a time period), pipe (smoking pipe, or instrument), and tree (perennial woody plant, or data structure).

Not only *between* domains there can be confusion about the semantics of a term, also *within* a domain multiple meanings for a term can exist. For example, in computer science the words channel (grayscale representation of a primary color in a digital image, or a tool used for inter-process communication), but also pointer (data type used in programming, or the graphical image which follows movements of the pointing device), and many many more.

Why is using an ontology important? It introduces a set of terms used by all individuals involved in the domain, domain model, architecture, and implementation, and with it, it brings unambiguousness. The goal is to avoid translation, because as Evans (2004) points out in his book on Domain-Driven Design,

Translation blunts communication and makes knowledge crunching anemic.

And every time concepts are translated between systems, a direct ability to think clearly about these entities is lost, and space for error is introduced. The game of *Chinese whispers* is often used as a metaphor to illustrate this problem. One person whispers a word or sentence to the next, which is passed down through a group of people until the last person announces it out loud. Errors usually accumulate in the passing of the message, so the announcement of the last person differs significantly from the one of the first.

To summarize, a shared ontology introduces a formal naming of types, properties, and interrelationships of the entities that exist in a particular domain of discourse. Using the same common, ubiquitous language brings an unambiguous way of communicating and establishes structure and relationships which will prove to be of value later in this research.

Several systems have been devised using ontologies as conceptual schemas for information integration systems. Wache et al. (2001) analyze existing information integration systems from an ontology point of view, and discuss several aspects of how ontologies are used to support the integration task. Noy (2004) provides a brief survey of ontology-based approaches to semantic integration developed by researchers in the ontology community. This paper gives a convenient overview to the major themes in this research field. Later, we will show an example of how ontologies can be applied in the context of data integration.

3.3 Distributed Data Integration

The rise of the World Wide Web and availability of a magnitude of structured data sources fueled the exploration of large-scale data integration. Özsu and Valduriez (2011) provide a book-long disquisition on distributed databases. A. Halevy et al. (2006) describe some of the important bodies of work done in the data integration field, and propose further research on some challenges to data integration research today. One of these challenges involves the need of *distributed* data integration. A distributed architecture lets IT manage data in separate systems and create a logical data model that can be used to integrate information for analysis without moving it to a single location.

3.3.1 Peer-to-Peer Infrastructure

The successful adoption of *peer-to-peer* (P2P) file sharing systems, like Kazaa and BitTorrent, emphasizes the problem of interoperability in current enterprise information systems. Peer-to-peer is defined by Zaihrayeu (2006) as a distributed communication model in which peers have equivalent functional capabilities in providing each other with data and services. P2P removes the central authority and introduces a new, social perspective that relies on self-organization. Moving to P2P data integration seems like a natural step. Shifting from a single, centralized mediated schema towards an arbitrary number of peers where each peer runs a local data integration system which integrates both its own data and data from other peers.

Several P2P solutions for data management in distributed data integration systems have been proposed and formalized (Ng, Ooi, Tan, & Zhou, 2003; A. Y. Halevy et al., 2004; Calvanese, De Giacomo, Lembo, Lenzerini, & Rosati, 2006). The fundamental approach of *peer data management systems*, or PDMSs, is to eliminate the reliance on a central, authoritative mediated schema. This model is conceptually related to *peer-to-peer computing*.

Although the topic of distributed data integration has been studied extensively for quite some time, and several P2P data integration systems have been proposed, there are still a lot of challenges to be faced. Because of the fact that almost all of the work on this topic has been fragmented (Özsu & Valduriez, 2011), the market has yet to successfully adopt integrated solutions.

At the basis of every P2P system lies a P2P network, built on top of a physical network, which in most cases is the Internet. It is commonly referred to as the *overlay network*. The overlay network usually has a different topology than the physical network and all the algorithms focus on optimizing communication over the overlay network. In the optimization they try to minimize the number of hops that a message needs to make from source to destination.

Özsu and Valduriez (2011) provide a comparison of the main types of P2P networks, concluding that there is room for improvement in each class of P2P networks. In the next sections we will briefly discuss all three types of P2P networks.

Unstructured Networks

Unstructured networks, depicted in Figure 3.4, do not predefine any structure in the overlay topology. Nodes randomly connect with each other; giving the network it's ad-hoc nature. Therefore, unstructured networks are easy to construct (Özsu & Valduriez, 2011). All nodes fulfill the same role, meaning that the network is less sensitive to high rates of *churn* (i.e. the rate in which nodes join and leave the network), rendering the network more robust.

Structured Networks

Structured networks, as depicted in Figure 3.5, were developed to address the scalability issues of unstructured networks (Özsu & Valduriez, 2011). Structured networks, like the name suggests, are structured by imposing an overlay topology on the nodes. This

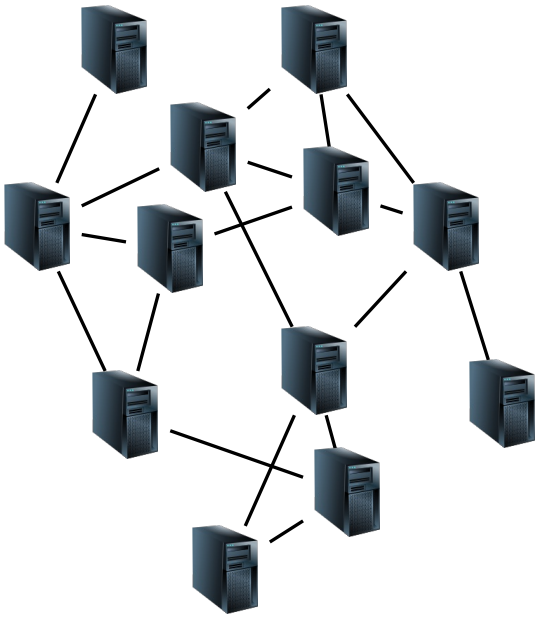


Figure 3.4 Unstructured Peer-to-Peer Network

specific protocol ensures that any node can efficiently search the network for a resource, achieving higher scalability at the expense of a lower autonomy.

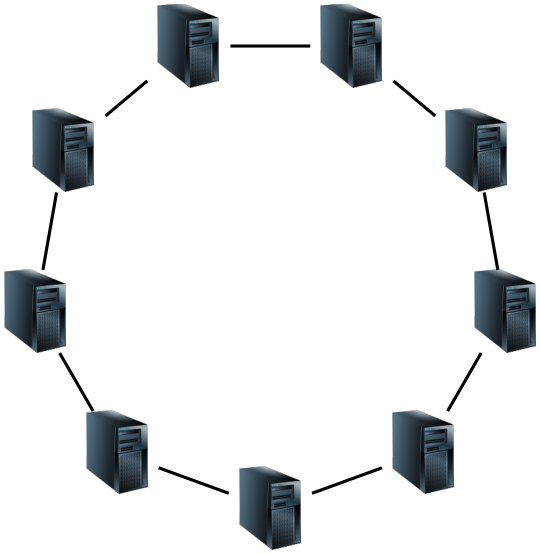


Figure 3.5 Structured Peer-to-Peer Network

Yet, to be able to route traffic in an efficient manner, nodes in a structured network need to store information about specific neighboring nodes. This makes the network less resilient to high rates of churn. The most popular implementation of this infrastructure

is by the use of *Distributed Hash Tables* (DHTs) (Özsu & Valduriez, 2011), in which consistent hashing is used to assign values to specific peers. We will dive deeper into the details of DHTs later.

Hybrid Networks

Hybrid networks use a combination of peer-to-peer and client-server models. In this infrastructure a central server commonly helps peers to find each other. Every hybrid network makes a trade-off between scalability, flexibility and autonomy. The centralized functionality provided by a structured server/client network provides robustness, and the node equality afforded by the pure peer-to-peer unstructured networks provides flexibility and autonomy. Hybrid networks have proven to be more efficient than pure networks.

3.3.2 Replication and Scalability with DHTs

Distributed hash table (DHT) systems are a class of peer-to-peer routing infrastructures that build an overlay network to enable enable scalable wide-area storage and retrieval of information (Zhang, Goel, & Govindan, 2003). It implements the idea of a hash table in a distributed fashion. A hash table is a data structure used to implement an associative array, a structure that maps keys to values. Just like the hash table, key-value pairs are stored in a DHT, but any contributing peer can retrieve a value associated with a given key. The responsibility for the maintenance of mapping keys to values is distributed among the peers, in such a way that churn causes minimal disruption. This allows for scalability of the network and resilience to peer arrival, departure, and failure.

Chord is a protocol and algorithm for a peer-to-peer distributed hash table developed at the Massachusetts Institute of Technology (Stoica, Morris, Karger, Kaashoek, & Balakrishnan, 2001). In Chord, nodes and keys are arranged in a ring (as shown in Figure 3.6). Chord specifies how keys are assigned to nodes, and how a node can retrieve the value for a given key by locating the node responsible for that key. Therefore, keys can be located by a simple sequential search for the node that ought to own the key.

To improve search performance, Chord maintains additional routing information. This routing table is referred to as the *finger table* of each node. Each finger table has $O(logN)$ entries, each entry twice as far away from the previous node in the identifier circle (see Figure 3.6). This allows the search procedure to perform a binary search of the identifier circle, and therefore approach the proximity of the target key much quicker than a sequential search.

There are several established algorithms available from which we can choose to distribute the mappings over the system. Since DHT systems are best for *key-based search*, which we need to lookup mappings, we investigated several existing implementations. Chord is the most popular of the four leading distributed hash table protocols, presented in 2001 along with *CAN*, *Tapestry*, and *Pastry*. There are several academic works comparing the performance of these DHT implementations. Li, Stribling, Gil, Morris, and Kaashoek (2004) conclude that these protocols can achieve similar performance if parameters are sufficiently well-tuned. Yet, a recent study (Medrano-Chávez, Pérez-Cortés, & Lopez-Guerrero, 2015) shows *Kademlia* to be superior to Chord in high churn

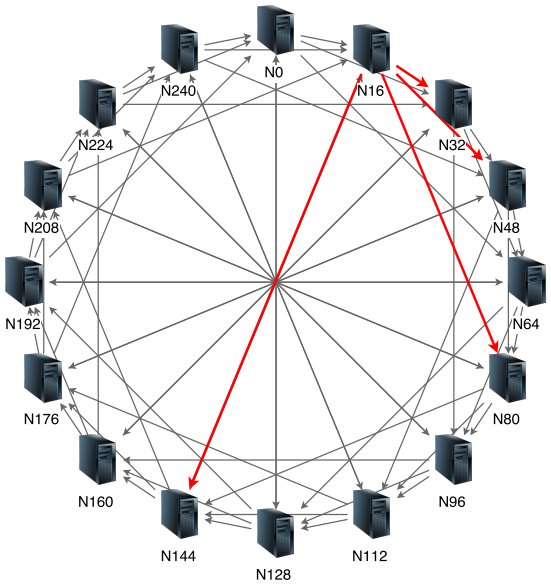


Figure 3.6 Chord Network showing Associated Peer References

scenarios. Kademlia (Maymounkov & Mazieres, 2002) is well known for its use in *file sharing networks* (e.g. BitTorrent, Kad Network, Gnutella).

3.3.3 Scalable Computation

To enable scalable computation, *MapReduce*, a programming paradigm that allows for massive scalability, can be used. MapReduce is a highly scalable programming model that enables *parallel processing* across huge datasets (Dean & Ghemawat, 2008) using a large number of nodes, collectively referred to as a *cluster*. MapReduce processes data close to where it is stored in order to improve efficiency by reducing transfer distances. The model consists of the following sequential steps. First the *map* step, in which a certain process or function is applied to local data. Second, the *shuffle* step, in which similar data gets redistributed so that data with the same key is located on the same node. And lastly the *reduce* step, in which the nodes process each group of data with the same key in parallel.

Several studies have been performed to study the feasibility of distributed learning. For instance, Caragea, Silvescu, and Honavar (2003) and Bhaduri, Wolff, Giannella, and Kargupta (2008) studied the feasibility of distributed *decision-tree induction* in peer-to-peer systems.

With the power of the MapReduce paradigm, several systems for distributed machine-learning with MapReduce were devised. Gillick, Faria, and DeNero (2006) use *Hadoop* and the MapReduce framework to evaluate its usefulness for standard machine learning tasks. Chu et al. (2007) adapt the MapReduce paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms like linear regression, k-means, logistic regression, naive Bayes, and several more. They did this to harness the power of *multicore systems*, however the same technique can be applied in distributed systems.

MLbase (Kraska et al., 2013) aims at both end-users and researchers, trying to make *machine-learning* accessible by providing a simple and declarative solution. It hides underlying complexity of distributed computation by providing a set of high-level operators.

The original goal of *Mahout*, a suite of machine learning libraries designed to be scalable and resilient, was to implement the algorithms mentioned in the paper of Chu et al. (2007). It is meant as a commercial friendly, stable, scalable suite of machine learning tools.

3.4 Existing Distributed Integration Approaches

The areas of schema matching and schema mapping have been the topic of research for decades. A lot of different techniques have been devised and have been fine-tuned in several iterations. Following, is a selection of techniques that have been formulated and, some of them, developed in the past.

3.4.1 Pairwise Schema Mapping

In the *Piazza* peer data management system (PDMS) proposed by A. Y. Halevy et al. (2004) each end-point defines the mapping between the local schema and the schema of any other peer that contains data that are of interest. Piazza provides "an interlinked collection of semantic mappings between peers' individual schemas". Piazza relies on the transitivity of the defined mappings and tries to extract mappings between schemas that have no defined mapping. Nevertheless, they use a centralized "global" catalog and cache the mappings at each peer to provide performance to the system. The system also assumes to exist in a relatively stable environment and sees joining a PDMS as a heavyweight operation.

Hyperion (Arenas et al., 2003) generalizes this approach to deal with autonomous peers that form connections at run-time, using mapping tables to define value correspondences among heterogeneous databases. In contrast to Piazza, their proposal assumes absence of central authority, no global schema, ephemeral participation of peers, and continuously evolving coordination rules among peers.

PGrid (Aberer et al., 2003) also relies on pairwise mappings between peers, however it assumes initial mappings are constructed by skilled experts. Just like Piazza they are relying on the transitivity of these mappings. To extract new mappings, PGrid employs a *gossip algorithm* that relates schemas of the peers between which there is no predefined schema mapping yet.

3.4.2 Mapping with Machine Learning Techniques

Machine-learning approaches are especially useful in data integration scenarios where new elements have to be matched against an existing collection of entities. The *LSD* (Learning Source Descriptions) system (Doan, Domingos, & Halevy, 2001) introduced the idea of employing current machine-learning techniques to semi-automatically find

mappings. LSD firsts asks the user to provide the semantic mappings for a small set of data sources, then uses these mappings together with the sources to train a set of learners.

The *COMA* system (Do & Rahm, 2002) considered a generic form of reusing past matches, and also considered composing multiple matching approaches in a flexible way.

GLUE (Doan, Madhavan, Domingos, & Halevy, 2002) uses the previously described approach of automatically extracting mappings between shared schemas. Given two ontologies, for each concept in one ontology GLUE finds the most similar concept in the other. It uses multiple learning strategies, each using a different type of information either in the data instances or in the taxonomic structure of the ontologies. On top of that, GLUE exploits a variety of *heuristics* and *constraints* to further improve mapping accuracy.

3.5 Related Concepts

Data virtualization is an umbrella term for the technology that offers data consumers a unified, abstracted, and encapsulated view for querying and manipulating data stored in a heterogeneous set of data stores (Van der Lans, 2012). Simply stated, it makes a heterogeneous set of databases or files look like one integrated database, without requiring technical details about the data, such as how it is structured or where it is physically located. When used in business intelligence systems, it often makes architectures simpler, more affordable, and more agile. Data virtualization also reduces the need for data replication, improving data consistency as well as reducing the risk of introducing data errors. Data virtualization allows updates to be written back to source systems.

In many situations where data virtualization is applied, data integration is applied as well. But data virtualization does not require data integration. When only one data store is virtualized, no data integration is needed. Data virtualization could be used to only transform the technical interface of a data store into one that is more suitable for a particular data consumer.

Data virtualization can be thought of as a *Service-Oriented Architecture* (SOA) for data. Microsoft (2007) defined it as "a loosely-coupled architecture designed to meet the business needs of the organization". SOA is basically a set of architectural principles for a collection of services that communicate with each other. This can either involve simple data passing between services, or services coordinating some activity. But where the traditional SOA approach has focused on business processes, data virtualization focuses on data that those processes use. SOA and data virtualization both aim for time-to-market advantages, as well as business agility.

Data federation is an aspect of data virtualization where the data stored in a heterogeneous set of autonomous data stores are made accessible to data consumers as one integrated data store by using on-demand data integration (Van der Lans, 2012). Data virtualization and data federation are often confused, because the difference between them is subtle. Data federation always implies multiple data stores, whereas data virtualization could encapsulate the implementation of just one data store, omitting technical information about the data. Moreover, data federation attempts to impose a single data model on the data, which makes it far more difficult to enable data governance (i.e. write

back to the source systems).

The goal of *Enterprise Information Integration* (EII) is to provide uniform access to multiple data sources without having to first load them into a data warehouse (A. Y. Halevy et al., 2005). EII translates the user's query into queries on the data sources and integrates the result of those queries so that it appears to have come from a single integrated database (Bernstein & Haas, 2008). According to these definitions, the goal of EII is to make a set of heterogeneous data sources appear as a single, homogeneous data source to a user or system. In that sense, EII is quite synonymous with data virtualization. Just as data federation, the use of the acronym fell out of fashion and should be seen as an "older" term for data virtualization.

Chapter 4

Proposed Solution

In this chapter we introduce the SAMARITAN (Semi AutoMAtic distRIbuTed dAta iNtegration) system, to aid the process of data integration.

Our envisioned solution is about the *utilization*, *sharing*, *replication*, and *reuse* of mappings across a distributed system. For our proposed solution, we use the techniques described in Rahm and Bernstein (2001) on *schema matching* and follow up on the suggestion from A. Halevy et al. (2006) on *reusing human attention*.

To circumvent the issues with central authority we propose to use a common choice for distribution, which is to use the *peer-to-peer* (P2P) network architecture. A. Y. Halevy et al. (2004) propose to implement their global mappings using *distributed hash table* (DHT) techniques, which should increase scalability and robustness of their query reformulation algorithm. We take this notion and apply it to store and distribute the extracted mappings over the network – basically functioning as a *cache* – to be used for further automated mapping in conjunction with the machine learning techniques proposed by Doan et al. (2001).

Our contribution is two-fold: first, we *distribute* the mappings and computation, thereby contributing to the *scalability* of the system, and second, we *reuse* existing mappings to enable *semi-automated matching* based on previously recorded results.

With this proposal we aim to improve the *agility* of enterprises by aiding the process of data integration. Our objective is to *improve scalability and resilience* by distributing the process, *reduce complexity and lower costs* by localizing responsibility, *improve consistency* by reducing replication of data in the enterprise, and *improve efficiency and automation* by reusing previously obtained results.

Because pairwise schema mapping, as described in section 3.4.1, in P2P systems using name-matching or string-matching is computationally expensive when applied to a huge domain, the region referred to as *search space*, we have to find a way to limit the set of items to match with. We aim to limit the search space by applying the use of an *ontology* to our solution, as suggested by Uschold and Gruninger (2004).

In this chapter we examine several different aspects of the system. These aspects are at different levels of abstraction. To deal with this we will look at the system from varying architectural views. A view is a representation of a coherent set of architectural elements as read by system stakeholders, and consists of a representation of a set of elements and the relations among them (Bass, Clements, & Kazman, 2012). We use two

views in particular: the process view, and the physical view. The *process view* deals with the dynamic aspect of the system, explains its processes, and shows the runtime behavior of the system. The *physical view* depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer.

First, we explain about the infrastructure of the system as a whole. We describe the two main elements of our systems: the overall network and the nodes in the network. Additionally, we give an overview of the dynamics of the nodes in the system. Subsequently, we zoom in to a lower level and dive into the intricacies of the matching and mapping processes of a single node, upon joining the system. Lastly, we present a *reference architecture*, mapping these steps unto a system architecture, that supports our solution.

4.1 Infrastructure

4.1.1 Network

The infrastructure of our system consists of a network of interconnected nodes (Figure 4.1). Together they hold and supply a global conceptual schema to which new sources are being mapped semi-automatically. The global conceptual schema can be queried by end users to retrieve and aggregate data from disparate source systems.

Our initial idea was to use an unstructured P2P network, because of the absence of imposed hierarchy on the system. Also the ad-hoc nature of unstructured P2P networks fits closely to the system we aim to construct. However, we choose to go for a structured approach, because of the scalability issues imposed by the unstructured network. In our proposal scalability and resilience trump all other features. Besides that, a structured network ensures a lower *latency* for queries and reduces *message flooding*.

Even though *Chord* is more popular among academics (by number of published papers), there are more reference-implementations for *Kademlia* (because of its use in file sharing networks). Hence we favor the use of Kademlia over Chord to specify the structure of the network and the exchange of information. Note that there is actually very little difference between these systems and their routing tables. The overall principles are very similar. It supports our distributed application architecture in the sense that it partitions storage and computing tasks between peers.

4.1.2 Node

A node is a connection point in a communication network. If the network is a distributed system, the nodes are called *peers* and are effectively both client and server at once. The nodes are equally important and equally privileged participants of the network. A node can serve as either a *data sharing node*, or a *computing node*, or both. A node can represent any system, or group of interconnected systems with a shared interface, as long as it can be controlled via a piece of middleware.

Before a node can join the network it needs some *middleware*; a little extra piece of software that acts as a bridge between the network and the node itself. The middleware, depicted as a small, orange box in Figure 4.1, enables the node to communicate with the network and to orchestrate shared resources.

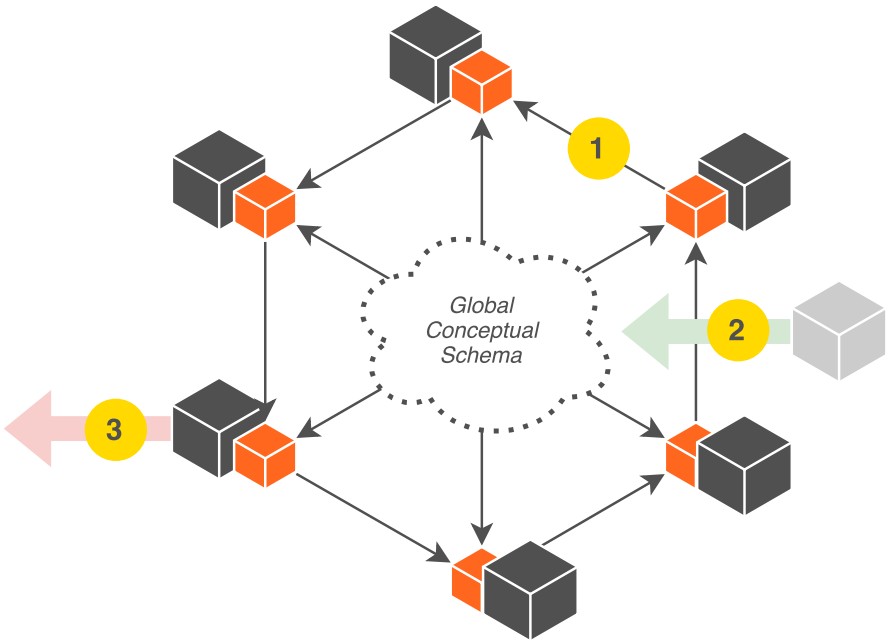


Figure 4.1 Network of Interconnected Nodes

Each node its middleware has several responsibilities. It needs to take care of handling distributed storage, aid in the computation of the distributed matching process, provide a user interface to the end user, and help to maintain the network.

4.2 System Dynamics

When a node joins the network (Figure 4.1; number 2), it undergoes several subsequent processes. First is a *bootstrapping* process, where the newly joining node locates a bootstrapping node that provides initial configuration information to successfully join the overlay network. In this initial phase, the joining node receives a unique identifier and a routing table from another node that is already connected to the network. The routing table provides the addresses of existing peers in the network. The specific implementation is described by Maymounkov and Mazieres (2002) in their paper on Kademlia.

A node is not required to offer source data. When the node does decide to share its data, the node first shares its source schema, which is an abstract collection of metadata. This can be a JSON schema, XML schema, relational database schema, or any other structured or semi-structured schema that formally describes the elements in the data offered by the node. The schema is parsed and processed in the *matching* process, which we will describe in-depth later this chapter.

The network stores the schema elements and the location of the peer, and will try to determine which entities can be mapped to the schema of the new node. In this connecting phase the result of the automated matching is presented to a system expert. This expert has the possibility to accept or reject the proposed mapping. Any expert

decision needs to be stored, since it is as useful to know what *does not match* as it is to know what *does*. To the elements of the schema of which no match is found, instance data is requested to perform an additional mapping process. The successful mappings will also be stored and replicated over the network.

Because the matching process can be a very computationally expensive process, it is not performed by one centralized server, since we can make use of the computing power available to use via other nodes in the system, reflected by number 1 in Figure 4.1. P2P is in used extensively for file sharing purposes, but to a much lesser extend for P2P computing. By using idle computing power of several of the nodes in the network, we can reduce the overall cost of, and load on the system. To improve scalability and performance of the distributed processes, we intend to *parallelize* them. This way we can take advantage of the distributed environment we are working in and exploit the power of large clusters. We can apply the previously described techniques, like MapReduce, to distribute the computation of new matches over the nodes in the P2P network. As depicted in Figure 4.2, the group of elements to match are split and distributed to several different nodes in the system. These, in turn, compute the matching suggestions from the elements to the shared ontology. The partitioned results are returned to be aggregated to a combined end result, ready for inspection by a system expert.

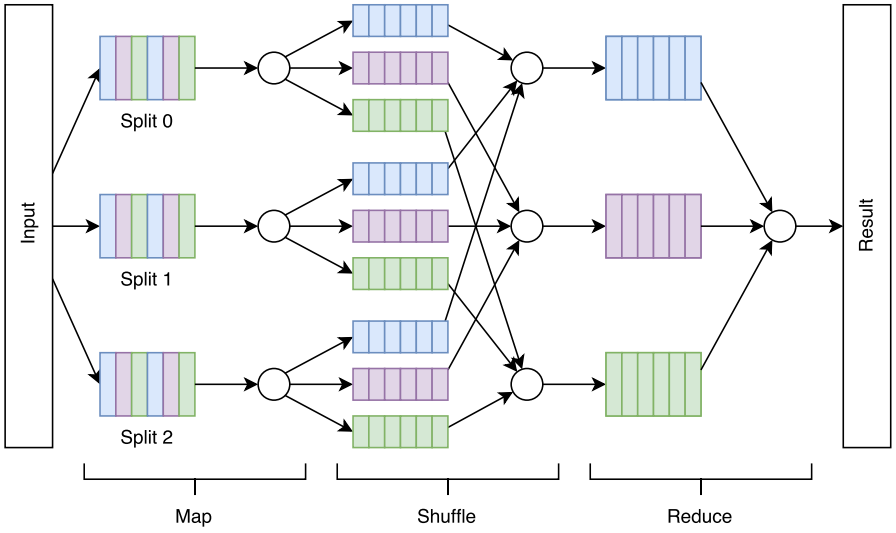


Figure 4.2 Distributed Sorting Process with MapReduce

Not only computing power is shared, also storage is distributed in this network. The matching process produces a set of valid mappings. To make use of these mappings, we distribute them across the system, to be reused to provide suggestions when another node joins the network. These mappings are replicated from node to node, to produce a resilient system. Nodes can disconnect, or leave the system, without it losing "memory" of previously made mappings, since these are retrievable from other nodes in the system.

From the above-mentioned process, several distributed "tables" or "collections" arise. One containing all the available elements in the system, another containing all the successfully obtained mappings of current and previously connected nodes.

When a node disconnects or in case of a node failure, illustrated in Figure 4.1 by the arrow with number 3, the data available in this node is no longer obtainable or accessible to the network. The nodes left in the network need some kind of mechanism to detect that a node has left the network and remove it from their routing table. This is called *routing table management*. In order to keep the routing table up to date, each node periodically searches for the peers it should have. If one of the peers it has in its routing table does not respond, the peer removes it from its routing table, and replaces it with the closest peer it finds. In a Kademlia network, each peer reference in the routing tables goes both ways. When leaving the network, a node only needs to send a leave message to all the nodes in its own local routing table, which can subsequently remove the disconnecting node and look for the next best peer.

Next to removing the node from the routing table, the system needs to know that the data associated with that node is no longer available. Nevertheless, we still want to make use of the mappings we created for these elements. This is why we only have to remove the reference of the node from the first replicated table, which maintains the schemas of the available sources. The second replicated table is unaltered and can still be used for future mappings.

To summarize, once a node is connected to the network, it can share several pieces of information. If the node has been connected before, it can share its successful mappings from its data to the global ontology, if not, it commences with the matching process. Additionally, it can share its own source data upon receiving a query from the network. Next to sharing its own information, the node can accept to store replicated information from the network, such as the successful mappings from any other source tot the global ontology, and the node can aid in the distributed computation of the matching process of other nodes. When a node disconnects from the network, its source data is unavailable to the network, but its successful mappings can still be reused in the matching process.

4.3 Matching Process

Before the data of a node can be shared with the system, the matching process has to be performed and finalized. The matching process addresses the problem of identifying that two objects are semantically related. Our approach to solving this problem is two-fold. First, we plan to give insight into the source systems' *semantics* by expanding abbreviations and acronyms, and assign meaning to the sources' elements, with the use of expert knowledge and dictionaries; the *semantic-based* approach. The system will retain this information and reuse it to learn how to rank and present meaning to the system experts. With these semantics we can search through the definitions of our conceptual schema. We apply the same approach to retain and reuse the connections made by system experts, between the identified semantics of the source schema and the ontology. Secondly, we can use the *internal relationships* (i.e. foreign keys) extracted from the source schema to improve suggestions for relationships of previously identified entities; the *structure-based* approach. In Figure 4.3 you can see the matching process in *Business Process Model and Notation* (BPMN) on the highest granularity level. The plus sign in an activity indicates that it is a composite activity with a complete diagram below.

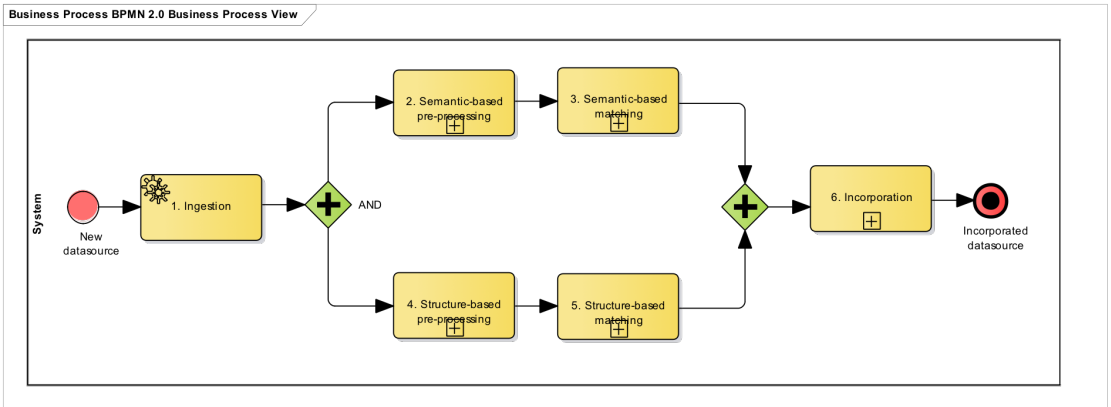


Figure 4.3 Business Process View of the Matching Process

Both the semantic-based and structural-based approaches run in parallel and are part of a multi-phased process (Appendix A): an *ingestion phase*, a *preprocessing phase*, a *matching phase* and a *feedback phase*. In the ingestion phase the source schema is offered and parsed. In the preprocessing phase several types of analysis are performed, and a graph is constructed. The matching phase takes care of the full text search, graph search, and scoring process. In the feedback phase the end user is able to provide feedback to the suggested mappings.

4.3.1 Semantic-based Approach

Ingestion Phase

The first thing we need is a component to digest incoming schemas. These schemas can be ingested by allowing different source formats like JSON, XML, but also database exports (a.k.a. data dumps) denoted in a *data definition language* (DDL), usually SQL. This to identify three concepts from the sources, being the entities, their attributes, and their constraints. The last step of ingestion is storing this freshly obtained (raw) source information in a distributed fashion in our system.

Preprocessing Phase

After ingestion, we take all extracted strings and try to assign meaning to them. We start this process by breaking all the strings up in separate terms, which-in context of lexical analysis-is referred to as *tokenization*. It uses different strategies to try to break up strings in understandable words. This is necessary because data definitions are often obfuscated when they are encoded by source systems. Usually because of Oracle having a long standing limit on table names of 30 characters, and IBM DB2 for z/OS (mainframe) limiting table names to only 8 bytes, which resulted in extremely abbreviated naming conventions.

A *naming convention* is a set of rules for choosing the character sequence to be used for identifiers like database names. That the choice of naming conventions can be an

extremely controversial issue, is clearly visible in the immense variety of conventions being used interchangeably. One obvious example is the use of multiple-word identifiers. Since most programming languages do not allow whitespace in identifiers, a way of delimiting word is necessary, since simply concatenating words results in unreadable identifiers. Common examples of delimiting words are PascalCase, camelCase, and snake_case. Whilst the first two are separated by *letter-case* (i.e. upper- or lowercase), the latter one is separated by a *delimiter* (e.g. the underscore).

Hence, to be able to interpret all the words or strings in these identifiers we have to split these concatenated strings back into a set of separated strings, referred to as *tokens*. After tokenization, we can add synonyms, filter stop words, modify and prepare the tokens for the matching process. This is done to improve the quality of the search results later on.

Matching Phase

To prevent creating a bottleneck in the form of exponential computational complexity, we refrain from matching every single entity to every other. This problem is addressed by introducing an ontology. If there is no ontology readily available, Noy and McGuinness (2001) provide a guide to ontology development, borrowing some ideas from principles of *object-oriented design*.

To compare the semantics of the source schema and the shared, common language we use several techniques from the fields of *natural language processing* (NLP) and *information retrieval* (IR). For instance, *query expansion* (QE) is the process of reformulating a query to improve retrieval performance in information retrieval operations. QE involves techniques such as *stemming* (which enables finding all the various morphological forms of a token by reducing each token to their word stem or root form), but also adding synonyms and removing stop words. We can apply several heuristics to identify abbreviations and acronyms. To be able to expand synonyms and acronyms, we need to include a dictionary.

Before we start matching the semantics of the source schema with those of the ontology, we have to give the end user the possibility to verify and correct the semantics we assigned to the tokens. The end users' feedback will subsequently improve our semantic process over time.

Next we can use the semantics to perform a textual search over the ontology. The ontology should not only consist of the name of the entity, but also a description or definition. This way we are able to perform a search by related concepts mentioned in the ontology terms' descriptions. Our previously mentioned stemming process attempts to remove the differences between different forms, thus increasing the likelihood of a positive hit. *Understemming* and *overstemming* are issues to be reckoned with. Understemming reduces retrieval; failing to return relevant documents. Yet, overstemming reduces precision; returning irrelevant documents when they shouldn't be.

After stemming we can match elements based on string equality or string similarity, by calculating several string metrics, as referred to in our literature study. We can combine several of these techniques to extract and apply semantics to the raw metadata extracted from the source systems.

Finally, we need to score the results of the matching phase. Most search engines use a similarity algorithm called *TF/IDF*, short for term frequency–inverse document frequency, which shows the significance of a term in a collection of documents. The score increases corresponding to the number of times a term appears in a document (TF), and is counterbalanced by the number of times the terms appear in the total collection (IDF). Consequently, terms that appear in many documents get a lower relevance score than more infrequently occurring terms.

We utilize TF/IDF for one of the main features of our system: increasing relevance through the reuse of confirmed interrelations. By adding more and more feedback from specialists, the TF/IDF statistic will score relevant documents higher, rendering our system more and more accurate.

Feedback Phase

After the matching phase we can present the end user with ranked ontology suggestions for positive or negative feedback. If the end user finds a match in the ontology directly, he can select and commit it right away. If no proper suggestion is found, the end-user should be presented with a way to search for the correct entity anyway. This alternative search and subsequent commit is incorporated and related to the previously applied semantics, forming new associations, and consequently improving suggestions.

4.3.2 Structure-based Approach

Our second matching approach, structure-based matching, takes the relational information of the source databases into account, and therefore starts with extracting constraints from the source schema to link its concepts together. To achieve this, we use a *regular expression* to find primary keys and foreign keys between tables in our source schema. We use these foreign keys to construct a representation of relationships between the entities in our source system. This representation is generally referred to as a *graph*. We can use this graph by comparing it to our ontology, which is a graph of related concepts as well. We call this the structure-based approach. A simplified description would be *relationship matching*; we compare relationships in the source with relationships in the ontology. We use this technique to suggest related entities in the ontology.

The problem of finding correspondences between two (sub)graphs is called *graph isomorphism*, or graph matching, and is a notorious problem in computer science. The term isomorphism consists of the words *iso*, meaning "equal", and *morphe*, meaning "form" or "shape" and is defined as a mapping of similarity or correspondence of form between elements of two sets, preserving existing vertices between the elements. Simply put, two graphs which contain the same quantity of graph vertices, linked in the same way, are said to be isomorphic.

Ullmann (1976) described a recursive backtracking algorithm for solving the subgraph isomorphism problem. Though, the running time of his procedure is, in general, exponential, or best case scenario, polynomial. Recently, some progress has been made by Babai (2015), publishing an algorithm that appears to be more efficient (quasi-polynomial time) than achieved before.

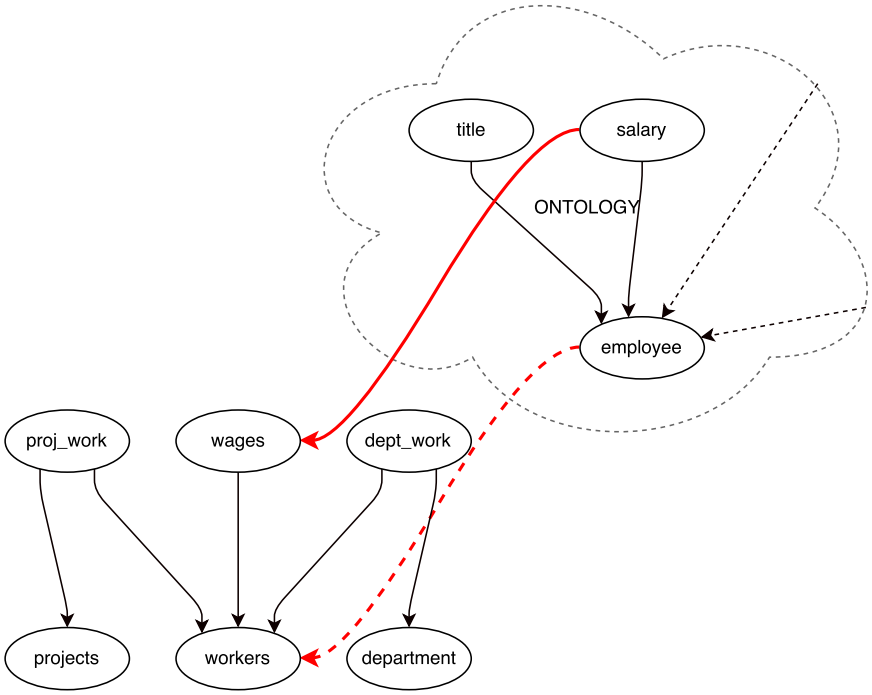


Figure 4.4 Subgraph Isomorphism

When a match between the two graphs is found, we can start to give suggestions for relationships of previously identified matches between the source and ontology. By way of illustration, source database A has a constraint between the tables *wages* and *workers* (see Figure 4.4). In the ontology B these entities are called *salary* and *employee*. If a system expert identifies and assigns the term salary from A to wages from B, we can consecutively suggest that the related schema entity workers could potentially represent *employee* in the ontology. Hence, we increase the score of this option, to make it rank higher on the provided list of suggestions.

4.3.3 Scoring

The last step in our matching process is scoring or ranking of the results. Ranking is a central part of many information retrieval systems. From our different approaches, we receive a different kind of output. The objective is to combine these results in a way that makes sense, by merging the result of both searches and prioritizing results that are common in both result sets. We have to assign *relevance* to each of our supplied suggestions. We can do this by assigning a certain weight or influence to every separate module. This means that a successful match in one module of approach, could have more influence on the scoring of a certain result than another.

To improve our matching accuracy, we need to have a strategy to reduce or circumvent our *error rates*. The first type of error, *false positive* (type I error), represents a document

selected by the system, but ignored or rejected by the expert. These errors should theoretically diminish automatically the more feedback the system receives from end users. More relevant suggestions will get pushed to the top, consequently, less relevant suggestions will get pushed down and off the list of suggestions.

The second error, *false negative* (type II error), represents a document not selected by the system, but searched for by expert. In our user interface, we have incorporated a search field with *autocomplete* functionality for every item to be identified. This way if our list of suggestions is inadequate, the end user can still find the proper term. By associating the term with our entity, it will automatically become more relevant and be selected by the system in a subsequent search.

4.3.4 User-feedback

A key aspect to our system is to reuse previously made relationships to improve the ranking of suggestions. This is where user-feedback can be utilized. Imagine, one of our proposed suggestions is ten times affirmed, and three times rejected to be correct. Another of our suggestions is two times affirmed and never rejected. Which suggestion should rank higher, and which should rank lower? We use both positive and negative feedback to rank our results.

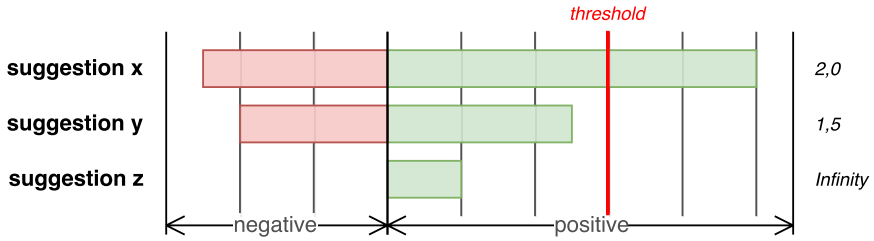


Figure 4.5 User-feedback

As the amount of user-feedback grows, several different suggestions can be selected as correct mappings to the global conceptual schema. These suggestions receive, next to their weighted score from the semantic mapping, also a score based on previous user-feedback. Every time a suggestion is selected and chosen this is recorded as *positive* feedback, and every time a suggestion is selected but ignored this is recorded as *negative* feedback, as depicted in Figure 4.5. The ratio between positive and negative feedback determines the user feedback score, which weighs in the total score of the suggestion. It can happen, that the score of a certain suggestion is infinity, because there is no negative feedback recorded. In this case we introduce a threshold, to prevent a suggestion with a single positive feedback and no negative feedback from tipping the scales.

4.4 Mapping Process

When a node decides to share its data with the system, it needs to follow the above-mentioned matching process, upon joining the network. As described, this process requires

a human to interact with the system to provide feedback. There are several ways for the user to interact with the semantic process, which is performed by the system. These strategies are part of the mapping process.

The mapping process, performed by end users, is strongly aided by the reuse of knowledge in the matching process. In contrast to the semantic approach of the matching process, the mapping process addresses the more high-level problem; ignoring the specifics concerning the operation of the system.

The mapping problem refers to the identification of correspondences between a source and a target schema. A specific strategy or approach is required to resolve this problem. We discovered two different approaches that we can apply to address the mapping problem: a *bottom-up approach* and a *top-down approach*. The former generates a bigger amount of reusable knowledge; the latter significantly reduces the time needed to finish the process.

4.4.1 Bottom-up Approach

The first way we can tackle this problem is by analysing every element and every derived token of the source system. We label this as the *bottom-up approach*. Defining all the elements in the source systems is still a tedious procedure. It doesn't differ greatly from the original problem of schema mapping, except for the semantics we can already provide and prefill to make the life of a modeller at least a little easier. The more often this process is repeated for separate source systems, the smarter and thus more accurate the suggested mappings will be. We aim to basically provide a "click-to-confirm" interface. Where the end-user simply has to provide simple true or false feedback.

4.4.2 Top-down Approach

More often, only a small subset of a source database is needed for reporting. Taking the bottom-up approach would demand too much time and resources. This is why in the *top-down approach* we only define what information is needed from the source systems and define only the appropriate elements. The down-side of this approach is that less feedback is provided to the system, which reduces the variety of information the system holds and can cope with. On the other hand, the system will not be flooded with irrelevant information that is rarely needed by external systems anyway.

4.5 Reference Architecture

The *reference architecture* shows how the previously described process steps are mapped onto the modules of the system. Our proposed architecture provides the system to be used in two different ways. One way is producing information and enables re-use. This is done via a *user interface*, providing the end-users with the ability to contribute to the accuracy of the system by giving their feedback. The other way is purely about consuming information previously stored in the system, and is solely used to give back suggestions for a single entity. This second way should only be utilized after the system has undergone a sufficient amount of training. Its interface is defined as a web *application*

programming interface providing a programmable interface to a set of services which can be implemented by existing systems.

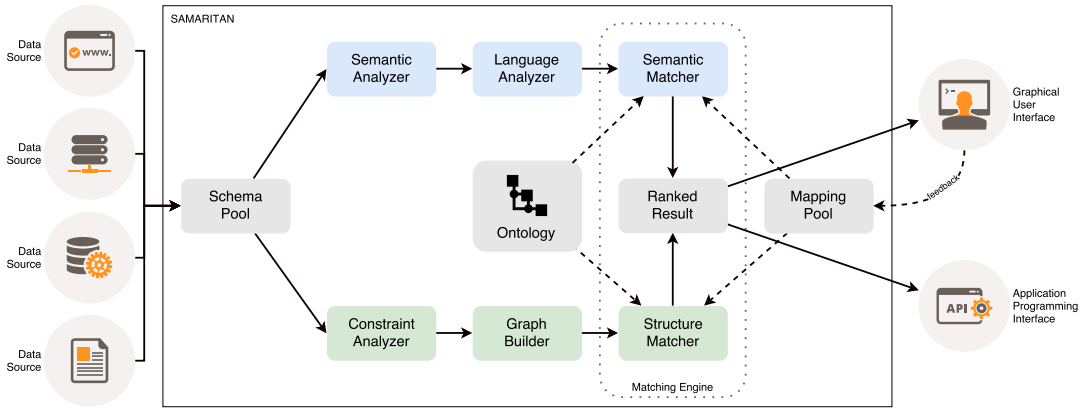


Figure 4.6 System Architecture

Our system is decomposed in several modules or components. Decomposing the system into modules gives it the attribute of maintainability. This way we aim to easily add more matching strategies to our system in future work.

The system architecture, as depicted in Figure 4.6, shows the ingestion of the source schema into the schema pool. Next both matching strategies are executed in parallel, allowing for both semantic and structural analysis of the data. The provided ontology is used to discover correspondences, rank them and present them to the graphical user interface. After the user has given feedback about the suitability of the suggested matches, this feedback flows back into the system. Accepted mappings are stored in a mapping pool, out of which requested mappings can be served.

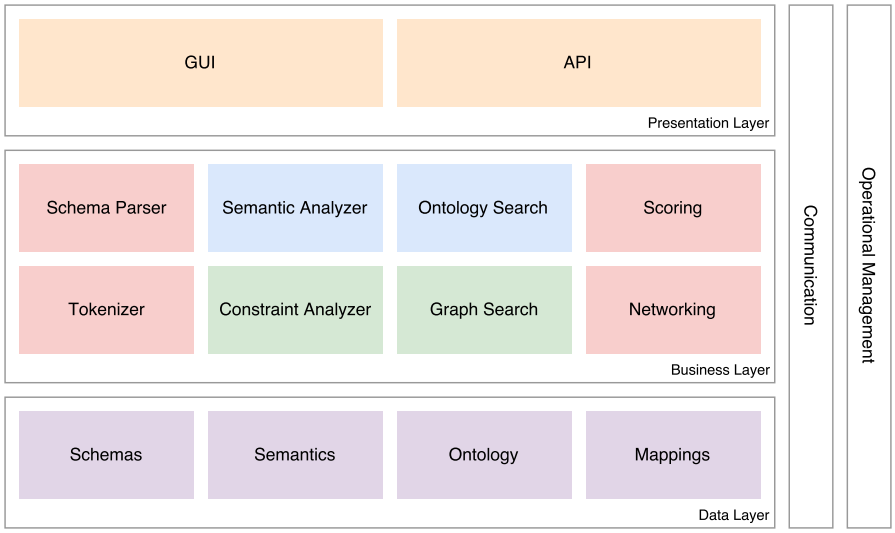


Figure 4.7 Three-layer Application Architecture

The system is subdivided in three distinct layers (see Figure 4.7): *business layer*, *presentation layer*, and *data layer*. All of these layers and their components are located in *every* node in the network (whether they are data sharing nodes, computing nodes, or both), and are therefore completely distributed. In the following sections we will go deeper into the function and interaction between these layers, and take a look at how they are organized.

4.5.1 Business Layer

The business layer holds the *logic* for the whole matching process. This process consists of a couple of processes, splitting the application in several different components or modules with different responsibilities. Each module represents encapsulations of a group of related responsibilities. In Figure 4.7 the modules depicted in red are more generic modules, they take care of shared processes like parsing, tokenization, and scoring. The blue and green modules respectively represent the semantic-based approach and structure-based approach.

4.5.2 Presentation Layer

The presentation layer, or interface layer, is built to provide an interface for end users to provide continuous feedback to the matches the system is producing. The input provided by the end users is fed back into the system and stored for later use. The presentation layer provides two interfaces to interact with: the *user interface* (UI) and *application programming interface* (API).

The UI provides a means for the end user to interact with the system in an intuitive manner. The user is able to provide a source schema, either by providing a SQL-file or connecting directly to the source database. The system extracts all necessary information from the source schema and starts the initial matching phase. The initial matching phase consists of assigning meaning to all the tokens extracted from the source schema. The first results are presented to the end user to be assessed and either approved or dismissed. By having this intermediate step to provide meaning to the source, a reusable dictionary is being constructed, since words and meaning are being interrelated.

There is a second interface module providing a versatile interface which can be used in conjunction with many other architectures. The API outputs exactly the same suggestions as the UI, yet in a machine consumable fashion. Like this the matching power of the system can be reused by other architectures.

Providing an API next to the UI, therefore has significant impact on the usability of the system. By following the principle of *information hiding*, we hide implementation details so that the end-user is not encumbered with complexity inside the system. We also hope to push the adoption of our solution by providing an easy-to-use programmable and integrable interface. The goal is to be plugged into existing architectures, without disrupting the established process. So our solution should be easy to integrate directly into existing ETL approaches, to enable semi-automatic translation of loaded data objects. In short, this *pluggable architecture*, that allows to plug in functionality using versatile modules, is intended to benefit acceptance through ease of integration.

4.5.3 Data Layer

The data layer is designed to provide the system with its distributed characteristics. It is where the data lives, once it is gathered from the source systems. Its data is organized and categorized in a way that enables reuse. The source schema elements, semantics, ontology, and mappings are stored here and have to be readily available for fast retrieval.

Chapter 5

Proof-of-Concept

Because of the limited time we had for this research we could not possibly implement all the aspects as presented in the previous chapter. We had to choose a limited but sufficient set of features, to be able to complete a working product, yet still show the feasibility of our proposed solution. Since the proof-of-concept was developed in association with the case study organization, the proof-of-concept is partially linked to the context this organization, although we have tried to keep most parts of our system as generic as possible. In this chapter we formulate our approach to achieve this goal.

5.1 Business Layer

To start the ingestion phase, we present users with a way to manually provide a source schema. Since SQL as source description is most common, for now we decided to implement the support for this source format only. We used several regular expressions to pattern match strings from the schema definition, and extract the entities, attributes, constraints, and other metadata needed.

For the preprocessing of the elements we extracted from the source schema we used *Elasticsearch*, a search engine based on the *Apache Lucene* project. It provides a distributed, full text search engine with an HTTP web interface. It also provides a set of very powerful algorithms called *analyzers*. Analyzers are composed of a single tokenizer and zero or more token filters. Tokenizers are used to break a string down into a stream of terms or tokens. A simple tokenizer might split the string up into terms wherever it encounters whitespace or punctuation. But as discussed in the previous chapter, technical identifiers from source systems rarely adhere to sensible natural language constructs, and are usually obfuscated with naming conventions.

To split these identifiers into tokens we use a more complex analyzer; the *pattern analyzer*[1]. The pattern analyzer can separate text into terms via regex, short for *regular expressions*, which is a sequence of symbols and characters that expresses a pattern to be searched for (see Figure 5.1). To illustrate, we can give the analyzer the string *EmpNO*, and it will break it up and return the tokens *Emp* and *NO*. If we give it *department_id*, it will break it up in *department* and *id*.

[1] https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-pattern-analyzer.html

```
  ([^\p{L}\d]+)                 # accept non letters and numbers,
| (?<=\D)(?=\d)                 # or non-number followed by number,
| (?<=\d)(?=\D)                 # or number followed by non-number,
| (?<=[ \p{L} && [^\p{Lu}]])    # or lower case
  (?=\p{Lu})                    #       followed by upper case,
| (?<=\p{Lu})                   # or upper case
  (?=\p{Lu}                     #       followed by upper case
    [\p{L}&&[^\p{Lu}]]          #       then lower case
  )
```

Figure 5.1 Regex Pattern to Detect camelCase Identifiers with Explanation

Next, the token filters accept a stream of tokens from the tokenizer and can *modify* tokens (e.g. lowercasing or stemming), *delete* tokens (e.g. remove stop words) or *add* tokens (e.g. synonyms). We use several of the token filters. Elasticsearch has built-in preconfigured stemming algorithms for about thirty languages. The stemming filter is available as one of numerous built-in token filters. These filters are not exclusive and can be combined as needed. Another example of a token filter we used, is the synonym token filter. This filter allows to handle and match synonyms, possibly imported from external sources, during the analysis process to increase chances of a match. We use this filter to expand common abbreviations and acronyms in the context of our case study organization. The key to optimal matching results is the proper configuration of the tokenizer and token filters.

5.1.1 Ontology Search

For the *ontology search*, we employed the proprietary common language or ontology of the case study organization, partially made available to us (it is still under development at the time of writing). We use the ontology to perform full text searches, again using Elasticsearch, on the tokens extracted from our source systems. For this we loaded the ontology, including its term descriptions, into Elasticsearch.

Elasticsearch uses a structure called an *inverted index*, which is designed to allow very fast full text searches. An inverted index consists of a list of all the unique words that appear in any document, and for each term, a list of the documents in which it appears. In our case this index consists of all the terms from our ontology, including their associated descriptions. By default, results are returned sorted by relevance—starting with the most relevant documents, represented by a positive floating-point number. The standard similarity algorithm used in Elasticsearch is TF/IDF. On top of that, Elasticsearch uses a field-length norm, which takes into account the length of the field the term was found in. The longer the field is, the less likely it is that term in the field will be relevant (i.e. a title field will be more relevant than a content field). We use this numerical statistic later on, as a weighting factor for the ranking of our suggestions. For a more in-depth theory of the scoring mechanism of Elasticsearch, we refer to the article[2] on their website on

[2]https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html

scoring theory.

Another very powerful feature of Elasticsearch is that it allows us to configure different analyzers both at *index time* and at *search time*[3]. This means we can break up our obfuscated identifiers at index time, and store the tokens in such a way that we can search for them in natural language at search time. This is exactly what we need, since we do not want to search for the identifiers in the obscure format in which they are encoded in the source schema.

5.1.2 Structure Search

As mentioned in the process section, we need to compare two graphs for our structural matching approach. The first graph we construct from the ontology; the second graph we construct from the constraints that we extracted from the source schema.

For the structural search we tried to use a small, yet elegant, open-source library[4] made by Herold, to compute isomorphism between subgraphs of the ontology graph and the graph we constructed. Unfortunately, this approach did not render the desired result, which is why we decided to only use relational information from identified graph elements. Practically, this means that when an element in the source is identified with a term from our ontology, we suggest all the related ontology terms to all the related elements of the source schema. For example, a schema entity *Worker* has three relationships: *Wage*, *Division*, and *Title*, as depicted in Figure 5.2. Worker is identified in the ontology as *Employee*. Now for all its relationships, all related ontology terms are proposed. Eventually, this tactic will affect the scoring if a similar match is found in another matching approach.

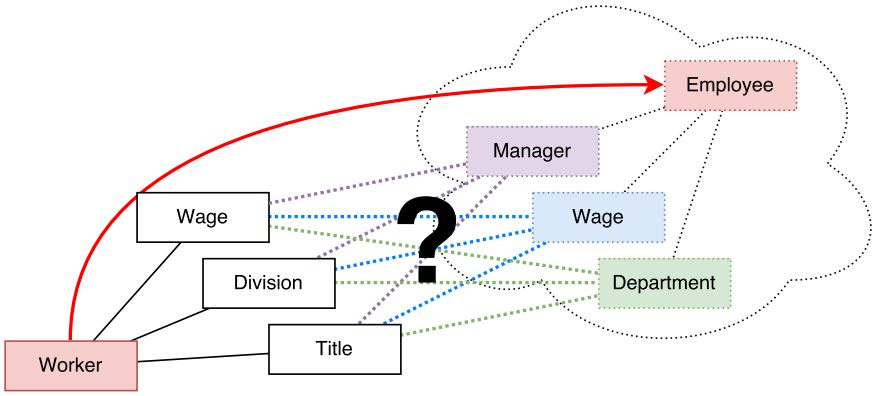


Figure 5.2 Relational Matching

[3]https://www.elastic.co/guide/en/elasticsearch/reference/current/search-analyzer.html
[4]https://github.com/wires/graph-isomorphisms

5.1.3 Scoring

Elasticsearch provides us with a scoring function based on which we can present our results for the full text search. However, as mentioned before we need to incorporate the structural matching results in our list of suggestions. To do this we need to combine the results of both searches in a logical manner. For this we merge the result of both searches and prioritize results that are common in *both* result sets. Hence, if a semantically matching term is suggested for one of these schema entities, and this term also appears in the related ontology terms, we increase the score of the suggestion.

5.2 Presentation Layer

5.2.1 Technology

To create an interface that is easily deployable and accessible from anywhere in an enterprise, we chose to use web technology to construct our presentation layer. The most significant framework being *React*, an "open-source JavaScript library for building user interfaces". React uses declarative views that make the codebase more predictable and easier to debug. It is maintained by Facebook and a community of individual developers and is currently being used on the sites of Netflix, Imgur, Airbnb and other notable companies.

An important feature of React is the use of a *virtual DOM* (Document Object Model); an abstract way of representing a structured document (e.g. the user interface) via objects. React stores this data structure *in-memory*, computes the resulting differences from a user action, and then updates the displayed application efficiently. Efficiently meaning that the React library only re-renders the components on the page that were actually changed, which is usually only a small subset of the total amount of components.

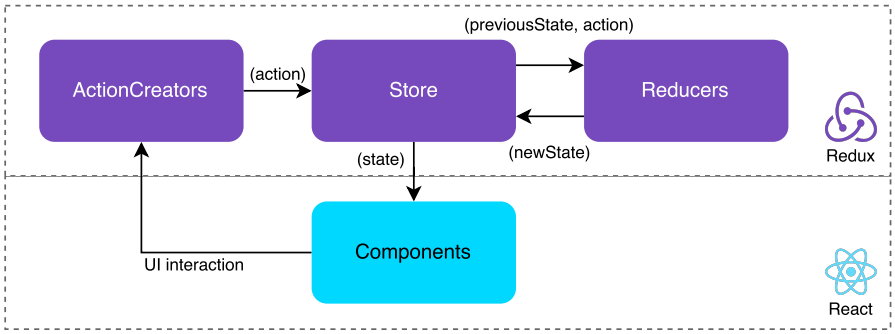


Figure 5.3 React and Redux Application State

Our interface consists of encapsulated components each managing their own state. This state is controlled by another open-source framework, *Redux*. Redux is a "predictable state container for JavaScript applications". It stores the whole state of the application in an object tree inside a single store. The only way to change this state tree is to emit an action, an object describing what happened. To specify how every action transforms the state tree, you write special functions called reducers. So instead of mutating the state

directly, you specify the mutations you want to happen with plain objects called *actions*. Then you write a special functions called *reducers* to decide how every action transforms the application's state (see Figure 5.3).

All of this is to make the presentation layer *predictable*, because there is always only one way to change the state of the application. This means we know where to look if something unintended might happen.

5.2.2 User Interaction

The user interface provides a way for the end user to upload an SQL-file. After the user uploads the file, the system will parse the document and return all the tokens it encountered, ready for semantic analysis. We applied certain heuristics to automatically accept the proposed meaning or leave it up to the user to fill out the meaning manually. We do this to save time by preventing the user to having to fill out every single meaning, for every single token. These heuristics consist of trying to find abbreviations and acronyms based on word length and amount of vowels in the term. When a term has a certain "established" significance, meaning it has been confirmed by an end user often enough to receive a score above a certain threshold, we preselect the semantics without any user interference.

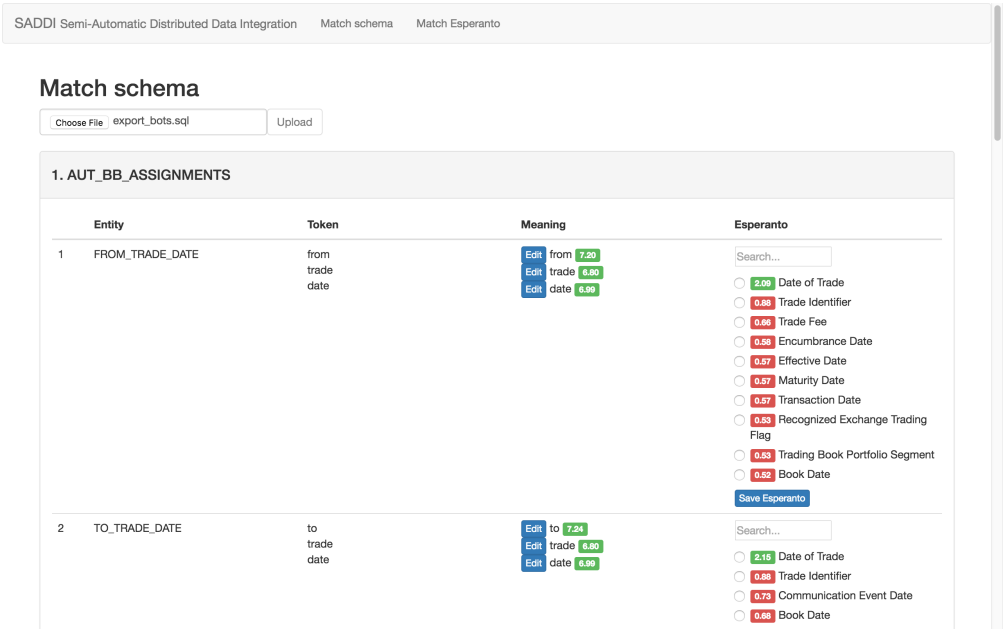


Figure 5.4 User Interface

After that, the user can either accept or change the assigned meaning (see Figure 5.4), on which the ontology term gets refreshed. Every time the user alters the meaning of the tokens, the ontology terms are analyzed again and refreshed based on the new information.

If the ontology term could not be found using the semantic approach, the user has the possibility to manually search for the correct term. By the use of *autocomplete* (see

Figure 5.5), we try to *predict* the rest of a word while the user is typing. Autocomplete aims to speed up human-computer interaction, and successfully does so when it correctly predicts the term that was intended with only a few characters typed.

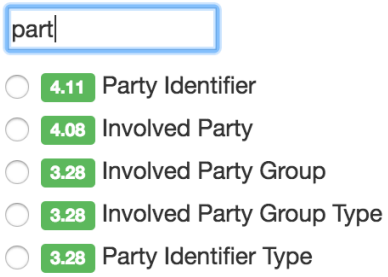


Figure 5.5 Autocomplete Functionality

Like most schema matching scenarios, there is a human in the loop, therefore it is important to have excellent graphical support for viewing results and interacting with them (Falconer & Noy, 2011).

Whenever a user interacts with the user interface and adds semantics or creates relationships between elements, it has an effect on the suggestions of other elements. Under the hood, every new piece of information causes a tremendous amount of recalculation. Relationships have to be sought and new suggestions have to be scored and ranked. To do this for all elements in the source schema at once is unnecessary, because the end user cannot process this amount of information anyway. That is why we solely take the tokens currently being under investigation by the end user and recalculate only the ones that were affected by the change.

Imagine a term was assigned to one of the attributes on the page. We compute if the attribute its tokens are associated with any other item currently displayed, or if the attribute has a relationship with an item that is currently displayed. If so we only recalculate the suggestions of this item.

5.3 Data Layer

After all the literature research done on distributed storage, we have discovered that this topic is worth a thesis on its own. By using an existing solution in our proof-of-concept we have tried to avoid the complexity of reinventing the wheel on this topic of research. Yet, we still wanted to use the scalability of a distributed system.

Elasticsearch comes with an integrated storage solution, which uses a technique called *sharding*. Under the hood, sharding makes use of the techniques we have discussed in our theoretical research on distributed storage (i.e. consistent hashing). This technique does not only involve storage though, it also involves concepts like *adaptive load balancing*, *routing*, and *partitioning* in distributed computing.

Scaling up can be done either *vertically* (bigger servers) or *horizontally* (more servers). Vertical scaling has its limits. Better and more fault-tolerant scalability comes from horizontal scale. Elasticsearch gives us the ability to add more nodes to the cluster and

to spread load and reliability between them. It knows how to manage multiple nodes to provide scale and high availability. This also means that our application doesn't need to care about managing it. So to summarize, out-of-the-box, Elasticsearch is set for scale, and can be distributed over a cluster. This means it is not only the storage solution we were looking for, but can also handle the distributed computation.

Chapter 6

Evaluation

To evaluate our proposed solution we have performed a case study in real-life context. We have chosen to do this at an organization that has an immense amount of data sources. Going from some sources being introduced less than a week ago, to dating back to several decades ago.

Through our case study we tried to answer the following questions. What is the current situation at our case study organization? How can we place our solution in the context of the organization? And does our solution improve or aid the current process? The following sections will try to answer these questions one by one.

6.1 Case Study Description

A classical but critical mistake is often made in *point-to-point interfaces* between different source systems. Source data from one system is transferred and transformed to meet the internal requirements of the target system without adhering to an overall domain model. This results in an unmanageable network (Figure 6.1) of interfaces with an abundance of non-standardized data transformations.

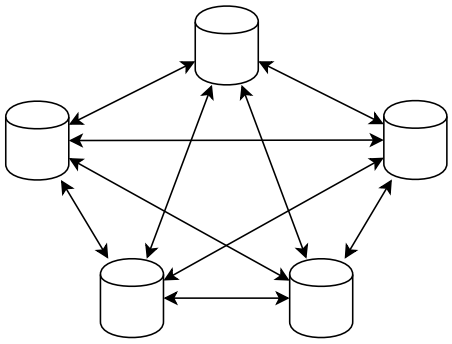


Figure 6.1 Point-to-point

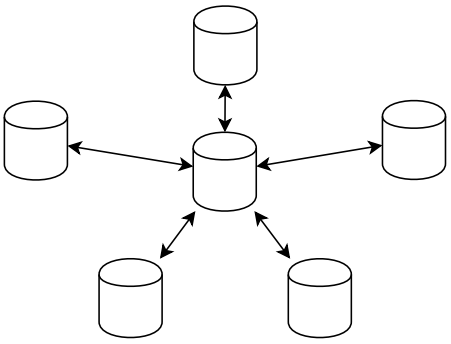


Figure 6.2 Hub-and-spoke

This is exactly what happens when data get transformed from one source system to another and aggregated by a third party. As illustrated earlier by the game of *Chinese whispers*, data gets misinterpreted, mislabeled and misused. The consequence of this for

our case study organization ING Bank N.V., is that incorrect labeling or missing data lineage affects consistency and hinders submission of raw or summarizing data needed by regulators to evaluate the bank's operations and overall financial health.

To address this problem ING wishes to *harmonize* the definitions of data that they exchange across systems by introducing a shared common language they call *ING Esperanto*, based on IBM's Industry Models for Banking (BDW). A global data definition process is set up to achieve harmonized data definitions and to create ING Esperanto. Harmonizing data definitions is necessary to enable a single source of truth and to ensure an accurate data lineage.

ING Esperanto is a business data glossary, describing the terms that are used on a daily basis, to avoid any ambiguity or miscommunication. In essence, the Esperanto translates to three distinct components. A set of *Business Terms* that all of ING needs to adhere to. These are defined, and managed by the respective Business Owners. Focused on the information we need to exchange globally and thus need to agree upon globally. A *Logical Data Model*, based on IBM BDW, linked to the Business Terms. A *Physical Data Model*, reflecting technical tables (rows and columns).

In our case study we used ING Esperanto as an ontology to limit the search space of our schema matching solution.

The newly proposed architecture of our case study organization exists of several data lakes, in which data from various sources flows together (see Figure 6.2). This data is ingested by the data lakes via APIs. We aim eventually to be able to plug our solution in between of the existing systems, to enable semi-automatic translation of freshly inserted data objects.

6.2 Data Collection

6.2.1 Interviews

To get a grasp on the current situation of the ING, and how we could plug our solution into ING's current process, we performed interviews with several people across the organization, that touch upon the topic of global data management. From *Data Modellers* to *Enterprise Architects*, from *Ops Engineers* to a *Chief Data Officer*. These interviews not only enabled us to develop a great insight into the current processes at the organization, but also facilitated our acquaintance with new and supporting technologies like Elasticsearch.

6.2.2 Ontology Extraction

To build our ontology from ING's business terms, we used the available, yet poorly documented, API of IBM's *InfoSphere Information Governance Catalog* (IGC). (Though, fortunately, it provides an API, otherwise we would have to write a scraper to extract these terms and related context.) We extracted the business terms and corresponding context (structure) for each term to construct the ontology. Based on this context, we were able to write a script to generate the edges between the terms for our graph.

6.3 Results

To assess if our solution actually improves or aids the current process and is a success, we pose that two criteria need to be met. We need to answer the questions; how *efficient* is our solution, and how is the solution *received*? So first, the system needs to be efficient enough to employ it without actually achieving a reverse effect of our objective, i.e. slowing the process down. Second, the system needs to be accepted and adopted by the target group.

6.3.1 Quantitative Analysis: Accuracy

How do we avoid irrelevant suggestions and find what we are looking for? To measure *relevance* or *accuracy* we can use two measures called *precision* and *recall*. Precision is defined as how many retrieved documents are really relevant and recall as how much of all the relevant information is found. For example, when our matching system returns 30 documents, only 20 being relevant, while failing to return 40 additional relevant documents, its precision is 20/30 and its recall is 20/60. So, precision is how useful the results are, and recall is how complete the results are.

With that, precision can be seen as a measure of exactness or quality, whereas recall is a measure of completeness or quantity. To perform statistical analysis, one should take as the null hypothesis that all, and only all the relevant documents are selected. This outcome corresponds to maximum precision (no type I errors) and maximum recall (no type II errors).

$$\text{precision} = \frac{|\{\text{relevant documents}\}| \cap |\{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|} \tag{6.1}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\}| \cap |\{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|} \tag{6.2}$$

Looking at Figure 6.3, we can also define precision as the amount of true positives (left half of the circle), divided by the amount of all selected documents (the full circle). Recall can also be defined as the amount of true positives (again the left part of the circle), divided by all relevant documents (left half of the figure).

For the first criterion we have performed several automated tests, as well as some manual analysis. For these tests, we constructed five database schemas in the organizational domain. These database schemas consist of tables describing the organizational structure, like departments, employees, addresses, phone numbers, etc. We also constructed a small ontology to fit the domain of our database schemas.

In our first test, we took the order of our database schemas and permuted it to every single permutation possible. In our case this equaled five factorial, meaning 120 possible permutations. We excluded the effects of outliers or extremes, by taking the average difference in efficiently of assigning meaning to tokens of 120 differently ordered sets of five databases. In this test we excluded user input, and relied solely on our algorithms to automatically select semantics by applying heuristics and the use of a dictionary. As a dictionary we used WordNet 3.1 (Miller, 1995), a lexical database for the English language, containing 155.287 unique words grouped in 117.659 sets of synonyms.

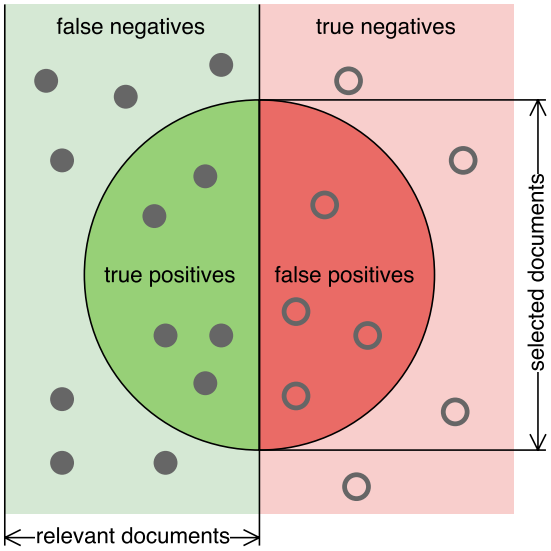


Figure 6.3 Precision and Recall

Our second test consisted of selecting five different permutations and performing the above-mentioned method manually. In this case we could include user input and assigned meaning to every token of the databases. This test covered less permutations, but showed to be much more reliable.

Table 6.1 Confusion Matrix

	Selected	**Not selected**
Relevant	True positive	False negative (Type II error)
Not relevant	False positive (Type I error)	True negative

Our third test consisted of a single run of defining semantics to a single permutation of the databases, again performed by an end user, but this time we tried to measure how accurate the system identifies the corresponding ontology. We did this by measuring the four quarters of the *confusion matrix* (see Table 6.1). With these measures we defined the precision and recall; giving an indication of the accuracy of the system.

Our first test, averaging the result of 120 permutations, shows an insignificant one percent increase in accuracy of automatically selecting the proper semantics. Nevertheless, the system on average identifies 72,4% of the tokens automatically.

The second test, applying user input, shows an increase in accuracy of 4,38% in automatically selecting semantics for the tokens. At the same time our system identified 73,89% of the tokens automatically. This suggests that our system benefits from user input.

The last test, running a real-world scenario, we used our user interface to relate the identified tokens to the properties of the ontology. For this we identified all the elements

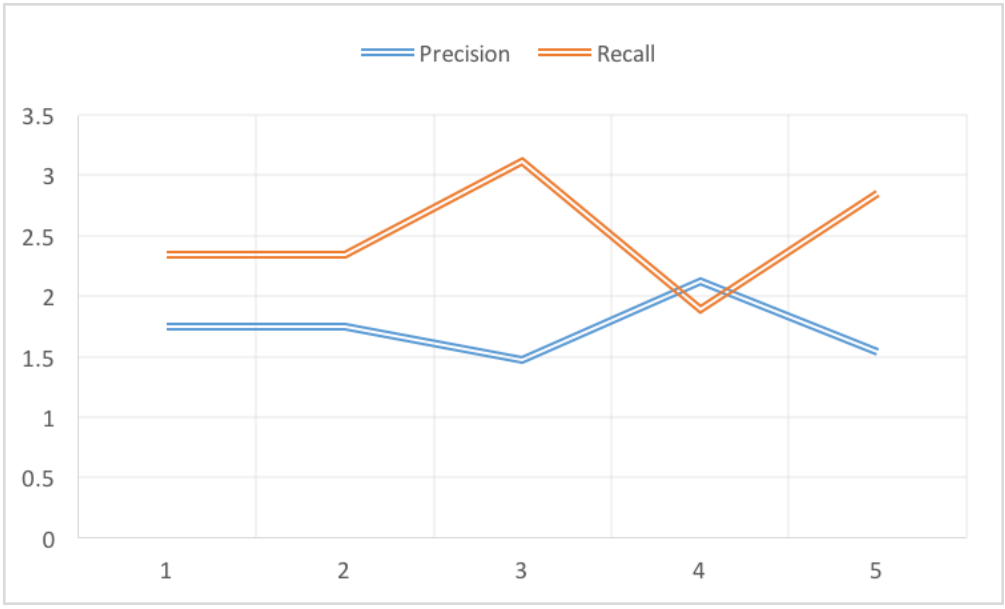


Figure 6.4 Measures of Relevance

of a single set of schemas. Running consecutively through this set, we obtained the following values for precision and recall, as depicted in Figure 6.4.

6.3.2 Qualitative Analysis: User Acceptance

To validate and provide structure to our user acceptance analyses we have chosen to follow the *unified theory of acceptance and use of technology* (UTAUT). UTAUT is a technology acceptance model formulated by Venkatesh et al. (2003) which tries to explain the degree of acceptance of the use of information technology. The model was formulated based on conceptual and empirical similarities across eight technology acceptance models. They state that for new software systems to improve productivity, they must be accepted and used by employees in organizations.

The theory holds that there are four core determinants of intention and usage; *performance expectancy*, *effort expectancy*, *social influence*, and *facilitating conditions*. The first three are direct influencing factors of *behavioral intention*; the fourth is a direct determinant of *use behavior*. Four *moderators* (i.e. gender, age, experience, and voluntariness of use) are proposed, modifying the influence of the four core determinants on behavioral intention and use behavior (Figure 6.5).

Venkatesh et al. (2003) define performance expectancy as "the degree to which an individual believes that using the system will help him or her to attain gains in job performance". Effort expectancy is defined as "the degree of ease associated with the use of the system". Social influence is defined as "the degree to which an individual perceives that important others believe he or she should use the new system". And facilitating conditions are defined as "the degree to which an individual believes that an organizational and technical infrastructure exists to support use of the system".

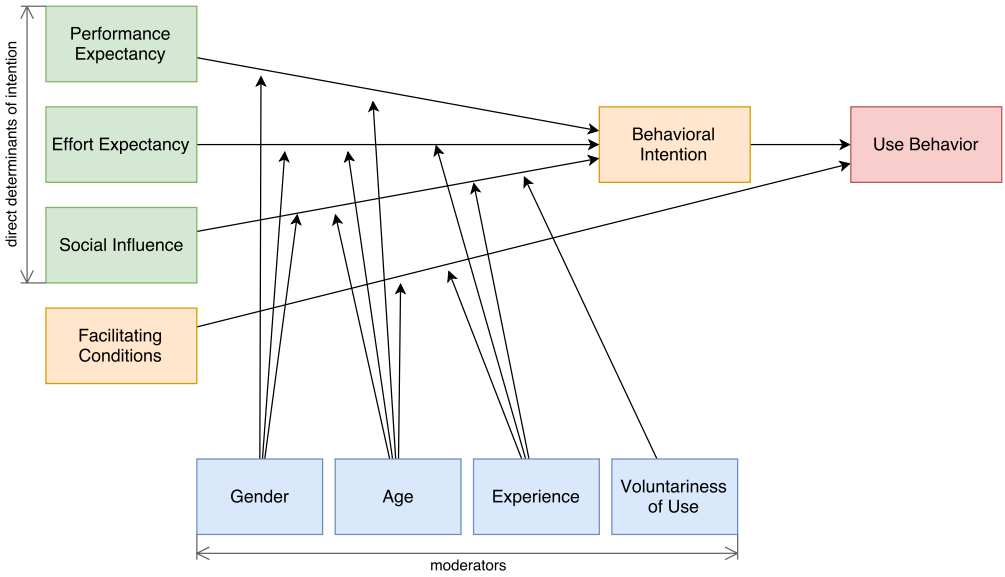


Figure 6.5 UTAUT Research Model, adapted from Venkatesh et al. (2003, p. 447)

Using a short survey, we employed the UTAUT to explain adoption and organizational use of our proposed solution. With questions addressing the before-mentioned factors, the survey measured adoption and its relation to the determinants of intention. Questions were posed in statements on which participants could indicate their level of agreement on a 5-point *Likert* scale (see Appendix B).

The survey was send to a group of fifteen employees across the organization, with whom we have had one or multiple meetings. These people were involved and properly informed about the extent of the research and proof-of-concept. We have received ten valid responses from our target group, composed of some very similarly and some very diversely answered questions. Figure 6.6 shows the study results in a Likert plot. The percentage on the left-hand side of the plot refers to the *strongly negative* (1) and *negative* (2) attitudes. The percentage in the center refers to the total of *neutral* (3) response, and the percentage on the right-hand side reflects the *positive* (4) and *strongly positive* (5) attitudes.

The answered surveys were exclusively filled out by males, age equally distributed, ranging from mid-twenties to end-fifties. They see themselves as experienced in the topic of data integration (Q4), with a predominant part (60%) on the right-hand side of the 5-point scale. With the highest scores, the expectation (Q5) towards the performance of the proposed solution is high. The expectation towards the learning-curve (Q6) is more diverse, ranging from one to four, and is the only answer with the center of gravity on the left side of the neutral. Most of the respondents perceive social influence (Q7) as an influencing factor, yet the sentiment seems subtle since we measure no extreme outliers. Responses on facilitating conditions (Q8) are bell-shaped over the neutral, suggesting that there is no consensus between the respondents on the existence of supporting organizational and technical infrastructure. The most diversely answered question is on the voluntary use of

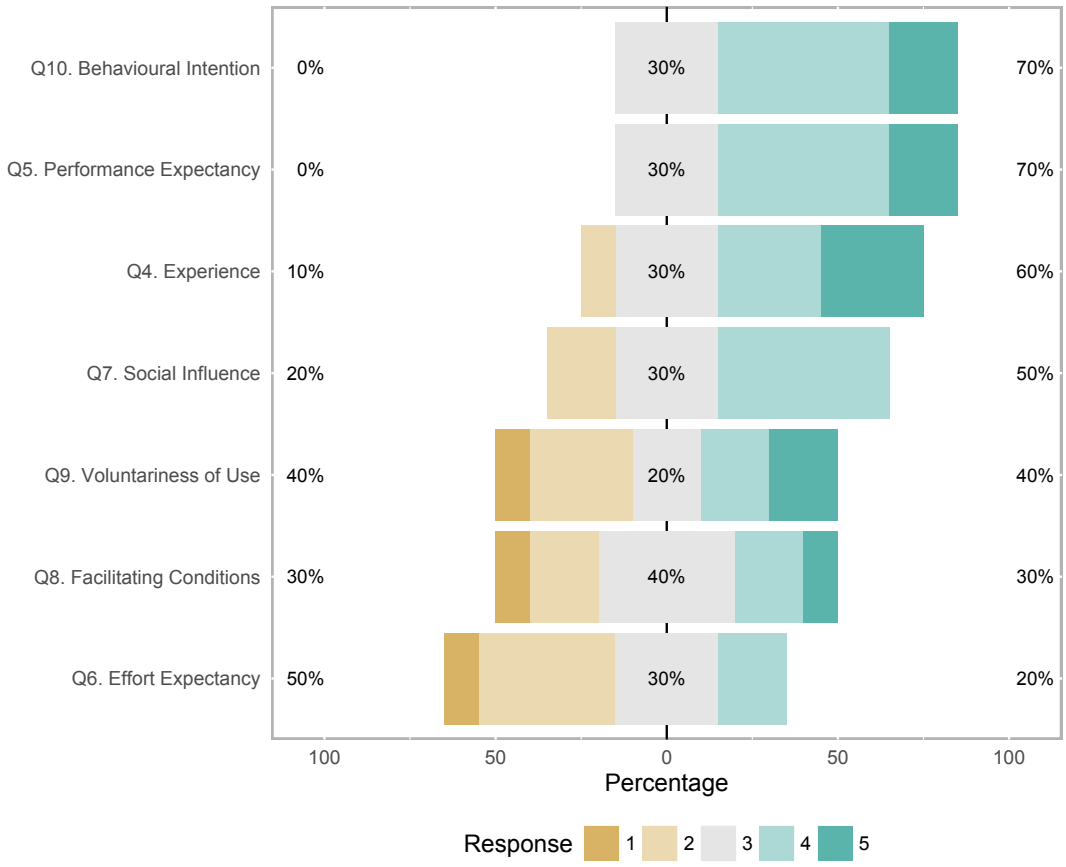


Figure 6.6 User Acceptance Study Results

the system (Q9), giving us an almost evenly distributed answer over the options, making it is hard to say anything about the perceived voluntariness of use of the proposed system. The last question on behavioral intention (Q10), shows how likely is it the respondent would use the proposed solution. The response only shows answers on the positive end of the scale, implicating it would be very likely the respondents would actually use the proposed system. All in all, the respondents are predominantly positive in their response, showing an favourable attitude towards the proposed system.

Chapter 7

Discussion

7.1 Findings and Implications

In our research, we have outlined our research and proposed a strategy on how data integration can be aided by distributing and automating the process. In our literature research we have touched upon several different techniques that support distributed semi-automatic data integration.

We have formulated an approach to semi-automatic data integration and designed a system architecture, combining several techniques and results from our literature research. We have implemented a proof-of-concept to demonstrate the feasibility and applicability of our proposed solution. From analysis of user interaction with our proof-of-concept we can conclude that our implementation of our proposed solution had a positive influence on the speed of the mapping process. We attribute this improvement to the provision of a fair amount of accurate suggestions and by providing means for quick searching through the ontology.

Lastly, we have employed a case study at ING, a Dutch multinational banking and financial services corporation headquartered in Amsterdam, to show our solution is fit to be applied in large-scale enterprises. According to outcome of our statistical analysis and our survey following the UTAUT technology acceptance model, there is a predominantly positive attitude towards our solution. The questioned employees appear to have medium to high expectations and believe the system to have a positive influence on the existing process.

To summarize, we have formulated and constructed a semi-automatic distributed system to aid scalable and reuse-oriented data integration demonstrating its feasibility and applicability in real-world context. We believe the combination of our matching strategies and a distributed solution is a unique and proper approach to scalable data integration, but hinges on the use of a decent and suitable ontology.

7.2 Validity

Since research design is based on following a list of *logical* steps, we can judge the quality of our research according to several logical tests. Yin (2003) lists four commonly used tests to establish the quality of any empirical social research, thus also relevant to case

studies. In his book he distinguishes the several *tactics* for dealing with these four tests when doing case studies. In the following sections we will discuss the *construct validity*, *internal validity*, *external validity*, and *reliability* of our research.

7.2.1 Construct Validity

In verifying the construct validity of our research it is critical to confirm that we identified the correct operational measures for the concepts we studied. Case study tactics (Yin, 2003) for construct validity are to use multiple sources of evidence, establish a chain of evidence, and to have key informants draft the case study report. To ensure the validity of the constructs of our study, we conducted extensive literature research and contextual research in the case study company. This way we have been able to support our hypotheses and give structure to the extent of our research.

For the evaluation of the performance of our solution, it can be questioned if precision and recall are the proper measures. One could argue that recall is not important, as long as there are at least some good hits returned. And precision could be could be irrelevant as long as at least some of the top ranking hits returned are adequate or satisfactory.

7.2.2 Internal Validity

Evaluation of internal validity is for explanatory or causal studies only, and not for descriptive or exploratory studies. Since we do not seek to establish causal relationships, whereby certain conditions are believed to lead to other conditions, we also do not apply any of the tactics proposed by Yin.

7.2.3 External Validity

The domain to which this study can be generalized is huge. As we explained in our problem statement, every organization in the world has the problem of reconciling data within in the organization to be able to extract business value. By applying different ontologies, this research can be applied to many more domains and contexts. Yin (2003) his tactic to evaluate external validity, is the use of theory in single-case studies. Therefore, we have built the aspects of this study upon previously conducted research and proposed future work.

7.2.4 Reliability

To verify the reliability if this research, we have to demonstrate that the operations of this study, like the data collection and the matching quality, can be repeated with similar results. Yin (2003) proposes to use a case study protocol and develop a case study database to provide reliability.

Even though our approach can be repeated, it could show dissimilar results. It is difficult to guarantee similar or stable results, since the amount of data we had to our disposal in our case study was limited. The system is supposed to benefit from large amounts of data and feedback. It is difficult to simulate large-scale user feedback with limited time and resources.

7.3 Limitations

7.3.1 Case Study Design Method

Research Design

We understand and openly acknowledge the strengths and limitations of case study research. We see case study research, like any other form of research, as a complement to the strengths and limitations of other types of research. As a research method, the case study is used in many situations, to retain the holistic and meaningful characteristics of real-life events. Not surprisingly, the case study has been a common research method in a great variety of research fields.

Generalization

A common concern about case studies is that they provide little basis for scientific generalization. The short answer is that, like experiments, case studies are generalizable to theoretical propositions and not to populations (Yin, 2003). Therefore, the objective of the case study is not to do statistical analysis, but to generalize our theoretical conclusions.

Single-case Design Case Study

This research, though limited to a single case, represents a problem that is not exclusive nor restricted to our case study organization. When extended to multiple organizations we expect to see similar results as we encountered in our study.

7.3.2 Instance-based Matching

In our literature research, we discussed instance-based matching, next to name-based matching. Because of the sensitivity of the data in our case study company, we have decided to keep this approach out of scope.

7.4 Recommendations

Due to time constraints and practical difficulties we haven't been able to deeply study every aspect of our research. Nevertheless, we would like to make some suggestions to the case study organization regarding the topic of data integration.

7.4.1 Reuse of Infrastructure

Most of the current data transfer in the organization is done via existing *XFB* (AXway File Broker) interfaces. TCP and FTP are the basic protocols used for XFB, and data is transferred in file formats. By making use of existing organizational infrastructure we could reduce the amount of configuration needed for a new system, therefore reducing the time of implementation. The use of our API, functioning as a kind of *middleware*, could be a workable solution, demanding the smallest footprint as possible on the existing infrastructure.

7.4.2 Reuse of Existing Mappings

The data lakes of ING ensure that only relevant information is extracted and stored. This simplifies the process of data integration significantly, since irrelevant data does not have to be identified. Manually constructed mappings, even though a fairly limited amount, are currently already stored in the data lakes themselves. That means part of the information and infrastructure is already in place. What is needed is a way to reuse these existing mappings to ease the job of data modellers and data owners.

7.4.3 Simplified Storage Solution

Another interesting topic we encountered, worthy of further research is the use of *bigtable*. Bigtable is a compressed, high-performance data storage system built on Google technologies designed to scale into petabytes, across a great range of machines. It can be described as a sparse, distributed multi-dimensional sorted map. In contrast, a data warehouse database is a normal relational database kept on third normal form, to eliminate data redundancy. However, for business intelligence reports where multi-dimensional modelling is prevalent this is not efficient. Several employees have suggested bigtable as storage solution for ING's data integration process. Mostly because the complexity of a structural or normalized solution is computationally expensive. A flat structure reduces complexity and improves scalability. Although, this approach would have its effects on our structure-based approach.

Chapter 8

Conclusions

We started this research in Chapter 1, by posing the problem of the complexity and tediousness of data integration and formulated our objective: to provide means and a proof of concept to aid the process of distributed data integration. In Chapter 2 we formulated several research questions and illustrated our research approach, following the case study research method described by Yin (2003) and design science research method described by Hevner et al. (2004).

In Chapter 3 we discussed our literature research, where we introduced the concepts of data integration and several related concepts. To answer question

I. *How can distributed data integration be accomplished?*

we showed different architectures and techniques for distributed and scalable schema mapping and schema matching, and presented the benefits of the use of ontologies. By doing this we built upon previously performed research on distributed integration approaches.

We have explored several ways of how distributed data integration can be accomplished by comparing several different distributed infrastructures to answer question

II. *Which techniques support distributed data integration?*

and concluded that a structured peer-to-peer infrastructure is optimal for what we aim to achieve. We have discerned and assessed several techniques like distributed hash tables, string matching, and machine learning to support our objective. From these we have selected several existing approaches needed to formulate a proposed solution.

In Chapter 4 we have formulated a solution to a distributed semi-automatic matching and mapping system to answer question

III. *How can these techniques be combined in a system architecture?*

We devised a corresponding strategy and reference architecture, to respectively explain how they interact and show how different layers and modules are composed.

As described in Chapter 5, we have implemented our solution through a proof-of-concept to assess its feasibility and performance, to answer question

IV. *Can the applicability or usability be demonstrated with an implementation or proof-of-concept?*

For the proof-of-concept we have used web technologies to ensure ease of deployment and accessibility. We have used open-source libraries like React and Redux to construct a user interface and API. To speed up the user-computer interaction, we have employed a combination of several newly devised heuristics and existing techniques.

Even though our proof-of-concept differs in many ways from our initially envisioned solution, our contribution remains the same; we have constructed a distributed system, thereby contributing to the scalability of the system, and we have reused information to enable semi-automated matching based on previously recorded results.

In Chapter 6 we illustrated our case study research at a large multinational organization, answering question

V. *Can it be applied in large-scale enterprises and does it work?*

We employed our proof-of-concept to evaluate the proposed solution, and validate the accuracy and feasibility of our system. By applying our proof-of-concept in a real-world context, we showed that the data integration process benefits from our solution.

We have found that by applying heuristics we can partially and semi-automatically assign semantics to our sources. We also concluded that with the limited amount of data we had to our disposal it is difficult to give a conclusive answer about the accuracy of the system.

Next to that, we conducted a user acceptance study following the UTAUT technology acceptance model. We have concluded from our study, that the system is received predominantly positive, with high expectations on the performance and efficiency of the proposed system.

Lastly, in Chapter 7 we have discussed our findings and their implications. We have discussed the several aspects and threats to validity relevant to our research, have noted the limitations of our study, and made several proposals to extend our current work and add future work.

8.1 Future Work

We have only been able to implement the proposed bottom-up strategy in our proof-of-concept. Implementing the top-down strategy could prove to add significant business value, since it mimics the information retrieval processes of an organization more closely.

Next to that, we encourage researchers to build upon our current system with the matching strategies provided in our literature research. There are many more approaches to matching, which could be incorporated in our system by adding a module to our business layer that hooks into the scoring function. Likewise, as initially proposed, we would like to see the content-based matching strategy to be added to our system.

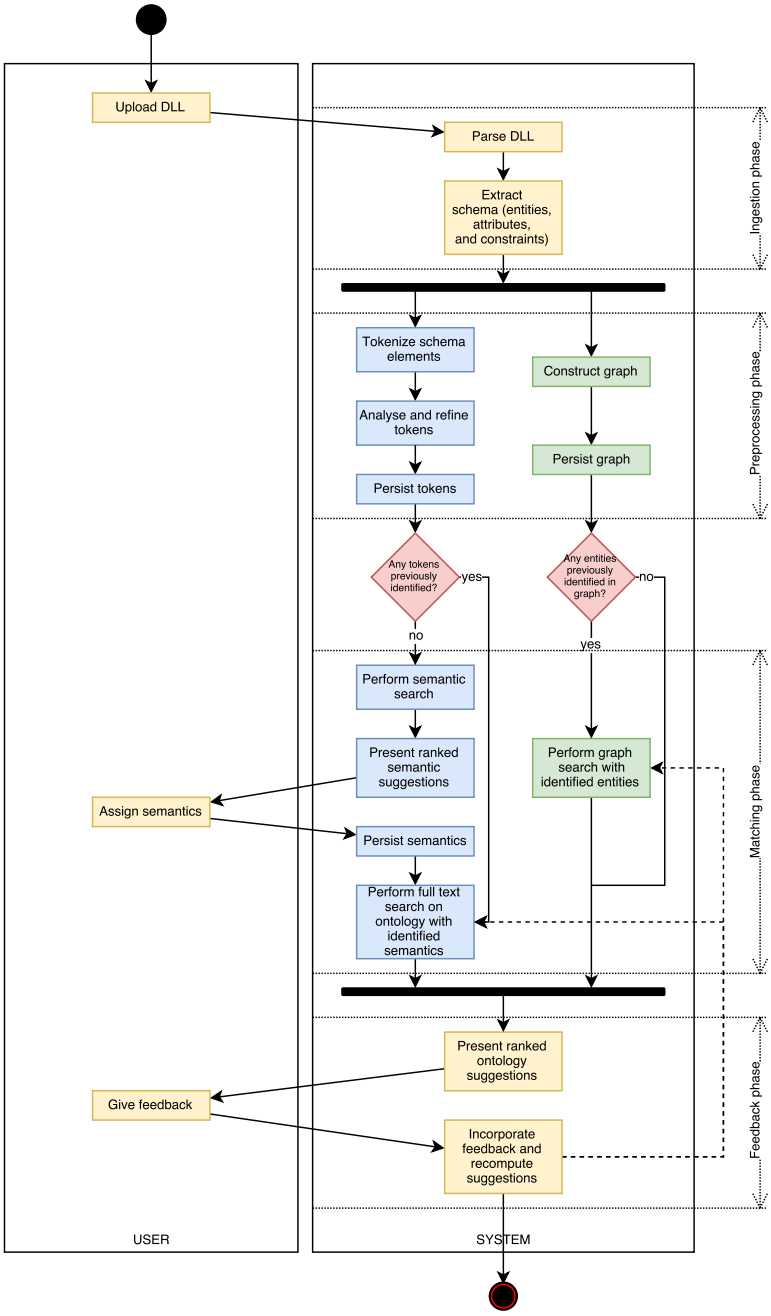
Unfortunately, we were unable to let the proper end users play with our system to measure a difference in efficiency in the integration task. For future work it would be interesting to research what effect our system has on the work of a data integrator.

To increase the scalability of our structure matching approach, we propose to implement the use of a *graph database* like Neo4j. This approach could prove faster, and more scalable than our current approach.

Lastly, we hypothesize, based on the work of Falconer and Noy (2011), that the user interface is of tremendous importance for the accuracy, and with that effectiveness of our system. If the performed task is perceived as difficult or annoying, the process fails with it. For example, large schemas cannot be comprehensively observed on a single screen. It could be beneficial to partition them and present them in fragments to be matched independently.

Appendix A

Activity Diagram of Matching Process



Appendix B

User Acceptance Survey

As you might know, for the last months we have been working on a system that aids the process of translating source descriptions, or source schemas, to ING Esperanto. (ING Esperanto is the ING flavor of the IBM industry model for banks and other financial institutions.) We have implemented a proof-of-concept where we transform and apply semantics to raw source data to semi-automatically find matches in the business terms of ING Esperanto.

We have conducted interviews with several people across the organization, that touch upon the topic of global data management. We have tried to explain the extent of our proposed system, and now we would like to receive your feedback on it.

1. What is your job title or function in the organization?
    a) Open answer
2. What is your gender?
    a) Male
    b) Female
    c) I'd rather not say
3. What is your age?
    a) 20-29 years
    b) 30-39 years
    c) 40-49 years
    d) 50-59 years
    e) 60-69 years
4. How much experience or knowledge do you have on the topic of data integration?
    a) 1. no experience
    b) 5. very experienced

5. What is your expectation towards the performance of the proposed solution? (Do you expect it to aid the process of data integration?)
    a) 1. low expectation
    b) 5. high expectation
6. How much effort do you expect it will take to get familiar with the proposed solution?
    a) 1. little effort
    b) 5. lot of effort
7. Do you perceive that 'important others' believe you should use the new system?
    a) 1. no, not at all
    b) 5. yes, a lot
8. Do you believe that an organizational and technical infrastructure exist to support use of the proposed system?
    a) 1. doesn't exist
    b) 5. already exists
9. Do you perceive the use of the system as being voluntary?
    a) 1. involuntary
    b) 5. voluntary
10. How likely is it you would use the proposed solution? (only answer if applicable)
    a) 1. not likely
    b) 5. very likely

Please enter your email address if you wish to receive the final outcome of this survey.

References

Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., & Schmidt, R. (2003). P-grid: a self-organizing structured p2p system. *ACM SIGMOD Record*, *32*(3), 29–33.

Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R. J., & Mylopoulos, J. (2003). The hyperion project: from data integration to data coordination. *ACM SIGMOD Record*, *32*(3), 53–58.

Babai, L. (2015). Graph isomorphism in quasipolynomial time. *arXiv preprint arXiv:1512.03547*.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Addison-Wesley Professional.

Bellahsene, Z., Bonifati, A., & Rahm, E. (2011). *Schema matching and mapping* (Vol. 57). Springer.

Bernstein, P. A., & Haas, L. M. (2008). Information integration in the enterprise. *Communications of the ACM*, *51*(9), 72–79.

Bernstein, P. A., Madhavan, J., & Rahm, E. (2011). Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, *4*(11), 695–701.

Beyer, M. A., & Laney, D. (2012). The importance of 'big data': a definition. *Stamford, CT: Gartner*, 2014–2018.

Bhaduri, K., Wolff, R., Giannella, C., & Kargupta, H. (2008). Distributed decision-tree induction in peer-to-peer systems. *Statistical Analysis and Data Mining*, *1*(2), 85–103.

Borst, W. N. (1997). *Construction of engineering ontologies for knowledge sharing and reuse*. Universiteit Twente.

Calì, A., Calvanese, D., De Giacomo, G., & Lenzerini, M. (2001). Accessing data integration systems through conceptual schemas. In *International conference on conceptual modeling* (pp. 270–284).

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2006). Data management in peer-to-peer data integration systems. *Global Data Management*, 177–201.

Caragea, D., Silvescu, A., & Honavar, V. (2003). Decision tree induction from distributed heterogeneous autonomous data sources. In *Intelligent systems design and applications* (pp. 341–350). Springer.

Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., & Olukotun, K. (2007). Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, *19*, 281.

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Do, H.-H., & Rahm, E. (2002). Coma: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on very large data bases* (pp. 610–621).

Doan, A., Domingos, P., & Halevy, A. Y. (2001). Reconciling schemas of disparate data sources: A machine-learning approach. In *Acm sigmod record* (Vol. 30, pp. 509–520).

Doan, A., Halevy, A., & Ives, Z. (2012). *Principles of data integration*. Elsevier.

Doan, A., Madhavan, J., Domingos, P., & Halevy, A. (2002). Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on world wide web* (pp. 662–673).

Draffan, I., & Poole, F. (1980). The classification of distributed data base management systems. *Distributed Data Bases: An Advanced Course*, 57.

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Falconer, S. M., & Noy, N. F. (2011). Interactive techniques to support ontology matching. In *Schema matching and mapping* (pp. 29–51). Springer.

Gagnon, M. (2007). Ontology-based integration of data sources. *2007 10th International Conference on Information Fusion*, 1–8.

Gillick, D., Faria, A., & DeNero, J. (2006). Mapreduce: Distributed computing for machine learning. *Berkley, Dec*, *18*.

Gruber, T. R. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. In *Proceedings of the second international conference on principles of knowledge representation and reasoning* (pp. 601–602).

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, *5*(2), 199–220.

Guarino, N., Oberle, D., & Staab, S. (2009). What is an ontology? In *Handbook on ontologies* (pp. 1–17). Springer.

Halevy, A., Rajaraman, A., & Ordille, J. (2006). Data integration: the teenage years. In *Proceedings of the 32nd international conference on very large data bases* (pp. 9–16).

Halevy, A. Y., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., ... Sikka, V. (2005). Enterprise information integration: successes, challenges and controversies. In *Proceedings of the 2005 acm sigmod international conference on management of data* (pp. 778–787).

Halevy, A. Y., Ives, Z. G., Madhavan, J., Mork, P., Suciu, D., & Tatarinov, I. (2004). The piazza peer data management system. *IEEE Transactions on Knowledge and Data Engineering*, *16*(7), 787–798.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, *28*(1), 75–105.

Hull, R. (1997). Managing semantic heterogeneity in databases: a theoretical prospective. In *Proceedings of the sixteenth acm sigact-sigmod-sigart symposium on principles of database systems* (pp. 51–61).

Jhingran, A., Mattos, N., & Pirahesh, H. (2002). Information integration: A research agenda. *IBM systems Journal*, *41*(4), 555–562.

Kraska, T., Talwalkar, A., Duchi, J. C., Griffith, R., Franklin, M. J., & Jordan, M. I. (2013). Mlbase: A distributed machine-learning system. In *Cidr* (Vol. 1, pp. 2–1).

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings of the twenty-first acm sigmod-sigact-sigart symposium on principles of database systems - pods '02* (p. 233). New York, New York, USA: ACM Press.

Lenzerini, M. (2011). Ontology-based data management. In *Proceedings of the 20th acm international conference on information and knowledge management* (pp. 5–6).

Lewis, J., & Fowler, M. (2014). *Microservices: a definition of this new architectural term.* Retrieved from `http://martinfowler.com/articles/microservices.html`

Li, J., Stribling, J., Gil, T. M., Morris, R., & Kaashoek, M. F. (2004). Comparing the performance of distributed hash tables under churn. In *International workshop on peer-to-peer systems* (pp. 87–99).

Maymounkov, P., & Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *International workshop on peer-to-peer systems* (pp. 53–65).

Medrano-Chávez, A. G., Pérez-Cortés, E., & Lopez-Guerrero, M. (2015). A performance comparison of chord and kademlia dhts in high churn scenarios. *Peer-to-Peer Networking and Applications*, *8*(5), 807–821.

Microsoft. (2007). *Service-Oriented Architecture (SOA) in the real world.* Retrieved from `https://msdn.microsoft.com/en-us/library/bb833022.aspx`

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, *38*(11), 39–41.

Ng, W. S., Ooi, B. C., Tan, K.-L., & Zhou, A. (2003). Peerdb: A p2p-based system for distributed data sharing. In *Data engineering, 2003. proceedings. 19th international conference on* (pp. 633–644).

Noy, N. F. (2004). Semantic integration: a survey of ontology-based approaches. *ACM Sigmod Record*, *33*(4), 65–70.

Noy, N. F., & McGuinness, D. L. (2001). *Ontology development 101: A guide to creating your first ontology.* Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, Stanford, CA.

Özsu, M. T., & Valduriez, P. (2011). *Principles of distributed database systems.* Springer Science & Business Media.

Pólya, G. (1945). *How to solve it: a new aspect of mathematical model.* Princeton University Press Princeton.

Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *the VLDB Journal*, *10*(4), 334–350.

Schenkhuizen, J. (2016). *Consistent Inconsistency Management: a Concern-Driven Approach* (Unpublished master's thesis). Utrecht University, The Netherlands.

Schramm, W. (1971). Notes on case studies of instructional media projects.

Shvaiko, P., & Euzenat, J. (2013). Ontology matching: state of the art and future challenges. *IEEE Transactions on knowledge and data engineering*, *25*(1), 158–176.

Smith, J. M., Bernstein, P. A., Dayal, U., Goodman, N., Landers, T., Lin, K. W., & Wong, E. (1981). Multibase: integrating heterogeneous distributed database systems. In *Proceedings of the may 4-7, 1981, national computer conference* (pp. 487–499).

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, *31*(4), 149–160.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: principles and methods. *Data & knowledge engineering*, *25*(1), 161–197.

Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, *23*(1), 31–42.

Uschold, M., & Gruninger, M. (2004). Ontologies and semantics for seamless connectivity. *ACM SIGMod Record*, *33*(4), 58–64.

Van der Lans, R. (2012). *Data virtualization for business intelligence systems: revolutionizing data integration for data warehouses*. Elsevier.

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS quarterly*, 425–478.

Villars, R. L., Olofson, C. W., & Eastwood, M. (2011). Big data: What it is and why you should care. *White Paper, IDC*.

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., & Hübner, S. (2001). Ontology-based integration of information-a survey of existing approaches. In *Ijcai-01 workshop: ontologies and information sharing* (Vol. 2001, pp. 108–117).

Yin, R. (2003). *Case study research: Design and methods*. SAGE Publications.

Zaihrayeu, I. (2006). Towards peer-to-peer information management systems. *International Doctorate School in Information and Communication Technology, University of Trento, Trento, Italy*.

Zhang, H., Goel, A., & Govindan, R. (2003). Incrementally improving lookup latency in distributed hash table systems. In *Acm sigmetrics performance evaluation review* (Vol. 31, pp. 114–125).

SAMARITAN: Distributed Semantic Integration with Elasticsearch

A Guide to Scalable and Reuse-Oriented Data Integration

Matthijs G. Dabroek
Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
m.g.dabroek@students.uu.nl

ABSTRACT

In the current information age, it is crucial for an organization to integrate all of its available source systems to provide deep insights, adhere to regulations, and provide a competitive edge. However, data integration often proves to be a tedious and costly process. In this paper we present SAMARITAN, a semi-automatic distributed system that takes advantage of previously specified associations between source schemas to enable scalable and reuse-oriented data integration. To provide a common language we include the use of ontologies. We constructed a proof-of-concept with Elasticsearch to evaluate our approach and validate our solution. Our contribution is two-fold: we distribute the data integration system, thereby contributing to the scalability of the system, and we reuse previously obtained results to enable semi-automatic matching.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.2.1 [**Software Engineering**]: Software Architectures; H.2.4 [**Database Management**]: Systems

Keywords

big data challenge, distributed data governance, semantic integration, data lake, elasticsearch

1. INTRODUCTION

Big data challenges. In the current information age, it is crucial for an organization to integrate all of its available source systems to provide deep customer insights, comply to law and regulations, or provide a competitive edge by bringing actionable analytics at a timely rate. However, this often requires tedious, slow, and costly processes like data transformation and data integration. The business intelligence landscape tries to keep up with this pace, but traditional data warehousing (DWH) solutions appear to reach their limitations. There are not enough hours in one day to process all the data pouring into the enterprise. It is estimated that a staggering 70% of the time spent on analytics projects is concerned with identifying, cleansing, and integrating data [3]. Yet, many organizations have built enterprise data warehouses (EDWs) and rely heavily on this technology to meet their business's operational and reporting needs. This problem is literary growing, since the rate of data creation is at an all-time high and increasing with a steady pace [10]. The problems emerging from this observation are referred to as *big data challenges* [5].

New architecture. Traditional ways of integrating require too much modelling, maintenance, and coordination among the owners of the data sources [4]. As data sources continue to increase, organizations are reconsidering enterprise data warehouses for a more flexible approach to data management. Big data requires a big, new big architecture [11]. The *data lake* architecture has drawn a lot of attention, because the problems it aims to address pertain to big data challenges. James Dixon, CTO of Pentaho, coined the, now trending, term on his blog[1], with the following description:

> "If you think of a datamart as a store of bottled water - cleansed and packaged and structured for easy consumption - the data lake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples."

Data lake. The definition we prefer is the one from TechTarget[2], which defines the data lake as "a storage repository that holds a vast amount of raw data in its native format until it is needed." In contrast to the EDW, that includes the storage of structured, semi-structured, and unstructured data. The data lake allows for *untreated* data to be digested. Moreover, the costly data transformation process is deferred until the data is actually needed. According to Andrew White, vice president and analyst at Gartner, the primary driver

[1] `https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/`
[2] `http://searchaws.techtarget.com/definition/data-lake`

for data lakes is "the need for increased agility and accessibility for data analysis." The data lake addresses the need for big data analytics within the enterprise without the cost of scaling the EDW to process big data volumes [2]. The data lake differs from the traditional EDW in several ways. The five most noteworthy being that data lakes retain all data, data lakes support all data types, data lakes support all users, data lakes adapt easily to change, and data lakes provide faster insights [1]. Despite their growing popularity, data lakes have received quite some criticism, as analyst firms like Gartner question their long-term viability. If organizations don't coordinate the untreated data that they throw into the lakes, they risk creating convoluted swamps, where data goes in and nothing ever comes out. A data lake certainly benefits IT in the short term in that IT no longer has to spend time understanding how information is used—data is simply dumped into the data lake. But this seems like they are only pushing the problem forward, because when this data needs to be queried, structure need to be applied to the data anyway. Gartner warns about this, stating: "without at least some semblance of information governance, the lake will end up being a collection of disconnected data pools or information silos all in one place" [6].

Metadata and semantics. By its definition, a data lake accepts any data, without oversight or governance. Without descriptive metadata and a mechanism to maintain it, the data lake risks turning into a *data swamp*. And without metadata, every subsequent use of the data means analysts will need to start from scratch. Without any form of data integration at the schema level, it can very quickly become a large collection of unrelated data silos, also referred to as *data puddles*. To keep the data lake useful, quite a lot of metadata—defining the data structure, contents, and lineage—needs to be collected and managed, or governed. Several problems concerning the data lake are addressed by literature. O'Leary states that although data governance generally is considered a non-technical issue, it is a critical component of making the lake work [8]. To address the shortcomings of the data lake, Franz Inc. developed the *Semantic Data Lake*[3] and CapGemini the *Business Data Lake*.[4] IBM introduces *Data Reservoirs*. All of these seem to agree that *data governance* is critical.

Data governance. Data governance (DG) refers to the overall management of the *availability*, *usability*, *integrity*, and *security* of the data employed in an enterprise [9]. The question is, does data governance take away the added benefit of agility? Data governance does not per se require data to be modelled and adhere to a global conceptual schema. It simply requires that there is metadata available and the data that is offered has enough metadata to make it useful for the subsequent *data discovery* process.

[3] `http://allegrograph.com/semantic-data-lake/`
[4] `https://www.capgemini.com/insights-data/data/business-data-lake`

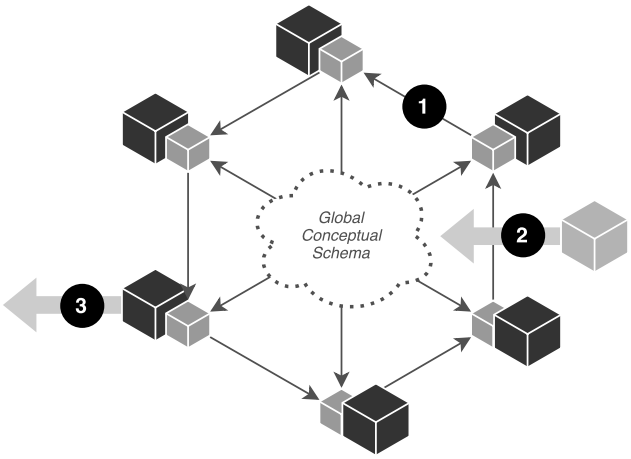


Figure 1: Network of Interconnected Nodes

2. SAMARITAN

Existing solutions still require a centralized governing body or council, just like the DWH approach imposed a central authority. We argue that to enable scalability, a distributed data governance approach is needed, where data owners are responsible for their own data and metadata. They have to make sure their data is useful for the consumers of these data. With this approach, we somewhat mimic data silos from the traditional DWH approach, but in a scalable fashion.

Data governance and semantic consistency are key aspects, and technology should be applied or added to the lake to provide this. We add semantics to prevent data lakes to perish like many congested SharePoint environments in the past. We propose a solution to provide semantic context to ingested data. Our proposed approach is two-fold: to increase agility we distribute the data governance process by making data owners responsible for their own data and metadata, and to aid the process of integration we employ a semi-automated semantic integration system by using Elasticsearch. The objective is to enable the creation of a semi-automated data repository, functioning as a self-service data catalog for end users for faster data discovery.

To address the aforementioned shortcomings of existing solutions, we devised the SAMARITAN system for Semi AutoMAtic distRIbuTed dAta iNtegration. In SAMARITAN we have combined the missing key aspects in a single system. Our proposed reference architecture shows how the different process steps are mapped onto the modules of the system.

2.1 Distributed Data Governance

According to Loshin [7], the first responsibility of data ownership is the *definition of data*. The data owner is responsible for understanding what information is to be brought into a system. Moreover, he is responsible for assigning the meanings to collections of data, and constructing a data model to hold the collected information. Loshin also discusses the centralized versus decentralized distribution of ownership across the enterprise. He poses that the benefits of the decentralized model include decreased management overhead, bureaucracy, and system integration. The costs include lack

of enterprise standardization for data and systems, and the inability to make use of merged data for additional knowledge discovery. We claim that these costs can be reduced by implementing a global conceptual schema, to improve enterprise standardization. Moreover, it allows data owners to decide which data they want to interconnect with this global conceptual network, federating the data sources and enabling knowledge discovery.

The infrastructure of our system consists of a network of interconnected nodes (Figure 1). Together they hold and supply a global conceptual schema to which new sources are being mapped semi-automatically. The global conceptual schema can be queried by end users to retrieve and aggregate data from disparate source systems. Once a node is connected to the network (Figure 1, number 2), it can share several pieces of information. If the node has been connected before, it can share its successful mappings from its data to the global ontology, if not, it commences with the matching process. Additionally, it can share its own source data upon receiving a query from the network. Next to sharing its own information, the node can accept to store replicated information from the network, such as the successful mappings from any other source tot the global ontology, and the node can aid in the distributed computation of the matching process of other nodes (Figure 1, number 1). When a node disconnects from the network (Figure 1, number 3), its source data is unavailable to the network, but its successful mappings can still be reused in the matching process, since these are replicated over the network.

2.2 Dual Matching Approach

Before the source data of a node can be shared with the system, the matching process has to be performed and finalized. The matching process addresses the problem of identifying that two objects are semantically related. Our approach to solving this problem is two-fold. First, we plan to give insight into the source systems' *semantics* by expanding abbreviations and acronyms, and assign meaning to the sources' elements, with the use of expert knowledge and dictionaries: the semantic-based approach. The system will retain this information and reuse it to learn how to rank and present meaning to the system experts. With these semantics we can search through the definitions of our conceptual schema. We apply the same approach to retain and reuse the connections made by system experts, between the identified semantics of the source schema and the ontology. Secondly, we can use the internal relationships (i.e. foreign keys) extracted from the source schema to improve suggestions for relationships of previously identified entities: the structure-based approach.

Both the semantic-based and structural-based approaches run in parallel and are part of a multi-phased process: an ingestion phase, a preprocessing phase, a matching phase and a feedback phase. In the ingestion phase the source schema is offered and parsed. In the preprocessing phase several types of analysis are performed, and a graph is constructed. The matching phase takes care of the full text search, graph search, and scoring process. In the feedback phase the end user is able to provide feedback to the suggested mappings.

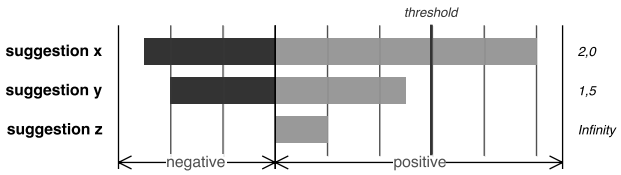


Figure 2: User-feedback

2.3 Semantic Integration with Elasticsearch

Elasticsearch. To support the distributed data governance approach, we propose to perform semantic integration of data using Elasticsearch. Elasticsearch uses a structure called an *inverted index*, which is designed to allow very fast full text searches. An inverted index consists of a list of all the unique words that appear in any document, and for each term, a list of the documents in which it appears. In our case this index consists of all the terms from our ontology, including their associated descriptions. By default, search results are returned sorted by relevance—starting with the most relevant documents, represented by a positive floating-point number. The standard similarity algorithm used in Elasticsearch is TF/IDF, short for term frequency–inverse document frequency, which shows the significance of a term in a collection of documents. The score increases corresponding to the number of times a term appears in a document (TF), and is counterbalanced by the number of times the terms appear in the total collection (IDF). Consequently, terms that appear in many documents get a lower relevance score than more infrequently occurring terms. We use this numerical statistic, as a weighting factor for the ranking of our suggestions. For a more in-depth theory of the scoring mechanism of Elasticsearch, we refer to the article[5] on scoring theory.

Domain ontology. To provide a common or global conceptual language we include the use of an domain ontology. The ontology is a formal description of the domain of interest, and is the central part of the whole system. We start by loading the ontology, consisting of a set of terms and corresponding descriptors, into Elasticsearch. Elasticsearch indexes the ontology and stores an optimal representation for us to match against at a later moment.

Schema digestion. Next, we can digest incoming source schemas to identify the entities and attributes living in the source system. We allow different source formats, just like the data lakes, like semi-structured (e.g. JSON, XML), but also highly structured data, like database exports, denoted in a *data definition language* (DDL), often SQL.

Semantic search. After ingestion, we take all extracted strings and try to assign meaning to them. We start this process by breaking all the strings up in separate terms, which-in context of lexical analysis-is referred to as *tokenization*. This is necessary because data definitions are often

[5] `https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html`

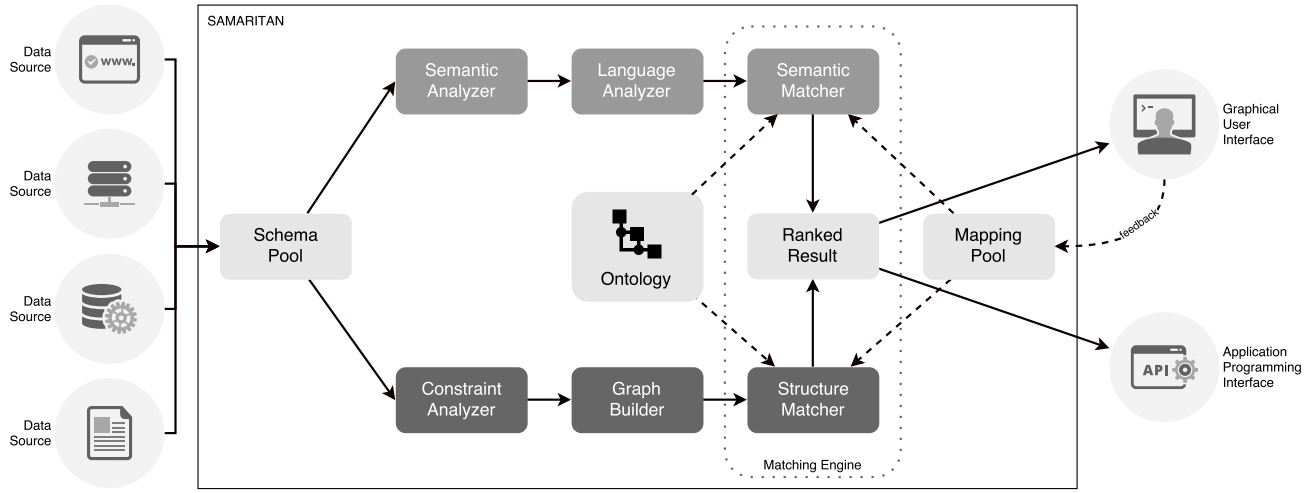


Figure 3: System Architecture

obfuscated when they are encoded in source systems. Usually because of limits on the length of table names, which resulted in extremely abbreviated naming conventions. After tokenization, we add synonyms, filter stop words, modify and prepare the tokens for the matching process. This is done to improve the quality of the search results later on. Before we start matching the semantics of the source schema with those of the ontology, we give the end user the possibility to verify and correct the semantics we assigned to the tokens. The end users' feedback will subsequently improve our semantic process over time. Next we can use the semantics to perform a textual search over the ontology its terms and descriptors. We can match elements based on string equality or string similarity, by calculating several string metrics. After the matching phase we present the end user with ranked ontology suggestions for positive or negative feedback. If the end user finds a match in the ontology directly, he can select and commit it right away. If no proper suggestion is found, the end-user is presented with a way to search for the correct entity anyway. This alternative search and subsequent commit is incorporated and related to the previously applied semantics, forming new associations, and consequently improving suggestions.

Statistical error. To improve our matching accuracy, we need to have a strategy to reduce or circumvent our *error rates*. The first type of error, *false positive* (type I error), represents a document selected by the system, but ignored or rejected by the expert. These errors should theoretically diminish automatically the more feedback the system receives from end users. More relevant suggestions will get pushed to the top, and consequently, less relevant suggestions will get pushed down and eventually disappear from the list of suggestions. The second error, *false negative* (type II error), represents a document not selected by the system, but searched for by expert. The system should provide a way for the end user to manually search for ontology terms. This way if our list of suggestions is inadequate, the end user can still find the proper related term. By associating the term with our entity, it will automatically become more relevant and be selected by the system in a subsequent search.

Ranking through reuse. A key aspect to our system is to reuse previously made relationships to improve the ranking of suggestions. This is where user-feedback can be utilized. Imagine, one of our proposed suggestions is ten times affirmed, and three times denied to be correct. Another of our suggestions is two times affirmed and never denied. Which suggestion should rank higher, and which should rank lower? We use both *positive* and *negative* feedback to rank our results. As the amount of user-feedback grows, several different suggestions can be selected as correct mappings to the global conceptual schema. These suggestions receive, next to their weighted score from the semantic mapping, also a score based on previous user-feedback. Every time a suggestion is selected and chosen this is recorded as positive feedback, and every time a suggestion is selected but ignored this is recorded as negative feedback, as depicted in Figure 2. The ratio between positive and negative feedback determines the user feedback score, which weighs in the total score of the suggestion. It can happen, that the score of a certain suggestion amounts to infinity, because there is no negative feedback recorded. In this case we introduce a threshold, to prevent a suggestion with a single positive feedback and no negative feedback from tipping the scales.

2.4 Reference Architecture

The *reference architecture* shows how the previously described process steps are mapped onto the modules of the system. Our proposed architecture provides the system to be used in two different ways. One way is producing information and enables re-use. This is done via a *user interface*, providing the end-users with the ability to contribute to the accuracy of the system by giving their feedback. The other way is purely about consuming information previously stored in the system, and is solely used to give back suggestions for a single entity. This second way should only be utilized after the system has undergone a sufficient amount of training. Its interface is defined as a web *application programming interface* providing a programmable interface to a set of services which can be implemented by existing systems.

We decomposed the system in several modules or compo-

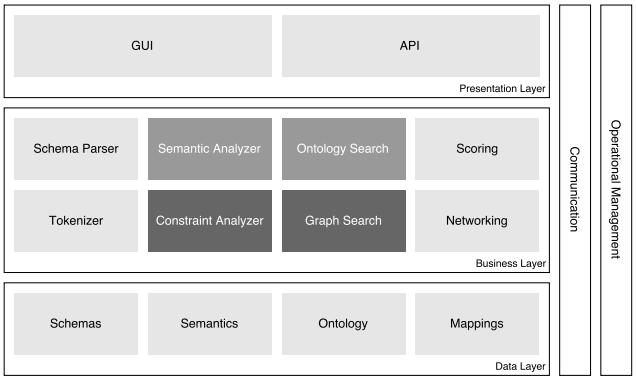


Figure 4: Three-layered Application Architecture

nents. Decomposing the system into modules gives it the attribute of maintainability. This way we aim to easily add more matching strategies to our system in future work. The system architecture, as depicted in Figure 3, shows the ingestion of the source schema into the schema pool. Next both matching strategies are executed in parallel, allowing for both semantic and structural analysis of the data. The provided ontology is used to discover correspondences, rank them and present them to the graphical user interface. After the user has given feedback about the suitability of the suggested matches, this feedback flows back into the system. Accepted mappings are stored in a mapping pool, out of which requested mappings can be served.

Every node in the system is subdivided in three distinct layers (see Figure 4): a *business layer*, *presentation layer*, and a *data layer*. All of these layers and their components are located in *every* node in the network (whether they are data sharing nodes, computing nodes, or both), and are therefore completely distributed.

The business layer holds the *logic* for the whole matching process. This process consists of a couple of processes, splitting the application in several different components or modules with different responsibilities. Each module represents encapsulations of a group of related responsibilities. In Figure 4 the modules depicted in light grey are more generic modules, they take care of shared processes like parsing, tokenization, and scoring. The medium grey and dark grey modules in the center respectively represent the semantic-based approach and structure-based approach, equivalent to the shading used in Figure 3.

The presentation layer, or interface layer, is built to provide an interface for end users to provide continuous feedback to the matches the system is producing. The input provided by the end users is fed back into the system and stored for later use. The presentation layer provides two interfaces to interact with: the *user interface* (UI) and *application programming interface* (API). Providing an API, has significant impact on the usability of the system. The goal is to be plugged into existing architectures, without disrupting the established process, to push the adoption of the solution. It should be easy to integrate directly into existing ETL approaches, to enable semi-automatic translation of loaded data objects. In short, this *pluggable architecture*, that allows to plug in functionality using versatile modules, is intended to benefit acceptance through ease of integration.

The data layer is designed to provide the system with its distributed characteristics. It is where the data lives, once it is gathered from the source systems. Its data is organized and categorized in a way that enables reuse. The source schema elements, semantics, ontology, and mappings are stored here and have to be readily available for fast retrieval.

In addition to these three layers, there is a set of overlapping services that are shared between the layers. The communication services provide communication between the components, by transmitting asynchronous messages via a selection of protocols. Operational management services monitor operational requirements such as scalability and fault tolerance, and can therefore manage the individual components and their resources.

3. DISCUSSION

In this paper, we proposed a strategy on how data integration can be aided by distributing and automating the process. We have deviated from the traditional approach by keeping the data in the source systems. We have formulated an approach to semi-automatic data integration and designed a new system architecture, combining several distinct techniques. We believe the combination of our matching strategies and a distributed solution is a unique and proper approach to scalable data integration, yet hinges on the use of a decent and suitable ontology.

4. REFERENCES

[1] C. Campbell. Top five differences between data lakes and data warehouses, 2015.

[2] Capgemini and Pivotal. The technology of the business data lake, 2013.

[3] M. Chessell, F. Scheepers, N. Nguyen, R. van Kessel, and R. van der Starre. Governing and managing big data for analytics and decision makers, 2014.

[4] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Elsevier, 2012.

[5] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.

[6] N. Heudecker and A. White. The data lake fallacy: All water and little substance. *Gartner Report G*, 264950, 2014.

[7] D. Loshin. *Enterprise knowledge management: The data quality approach*. Morgan Kaufmann, 2001.

[8] D. E. O'Leary. Embedding ai and crowdsourcing in the big data lake. *IEEE Intelligent Systems*, 29(5):70–73, 2014.

[9] M. Rouse. Data governance (dg), 2007.

[10] R. L. Villars, C. W. Olofson, and M. Eastwood. Big data: What it is and why you should care. *White Paper, IDC*, 2011.

[11] D. Woods. Big data requires a big, new architecture, 2011.