TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
15/2018

Semantic Data Lineage and Impact Analysis of Data Warehouse Workflows

KALLE TOMINGAS

TTÜ PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

This dissertation was accepted for the defence of the degree of Philosophy in Computer Science on April 20, 2018

Supervisor: Professor Tanel Tammet
Department of Software Science
Tallinn University of Technology
Tallinn, Estonia

Opponents: Professor Alexandra Poulovassilis
Department of Computer Science and Information Systems
Birkbeck University of London
U.K.

Ph.D Peeter Laud
Research Director
Cybernetica AS
Estonia

Defence of the thesis: May 21, 2018, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

ISSN 2585-6898 (publication)
ISBN 978-9949-83-238-5 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9949-83-239-2 (PDF)

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
15/2018

Semantiline andmevoogude- ja mõjuanalüüs andmelao keskkonnas

KALLE TOMINGAS

TTÜ KIRJASTUS

Table of Contents

ABSTRACT

The subject of the thesis is data flow in data warehouses. Data warehousing is a complex process of collecting data, cleansing and transforming it into information and knowledge to support strategic and tactical business decisions in organizations Our goal is to develop a new way to automatically solve a significant class of existing management and analysis problems in a corporate data warehouse environment.

We will present and validate a method and an underlying set of languages, data structures and algorithms to calculate, categorize and visualize component dependencies, data lineage and business semantics from the database structure and a large set of associated procedures and queries, independently of actual data in the data warehouse.

Our approach taken is based on scanning, mapping, modelling and analysing metadata of existing systems without accessing the contents of the database or impacting the behaviour of the data processing system. This requires collecting metadata from structures, queries, programs and reports from the existing environments.

We have designed a domain-specific language XDTL for specifying data transformations between different data formats, locations and storage mechanisms. XDTL scripts guide the work of database schema and query scanners.

We will present a flexible and dynamic database structure to store various metadata sources and implement a web-based analytical application stack for the delivery and visualization of analysis tools for various user groups with different needs.

The core of the designed method relies on semantic techniques, probabilistic weight calculation and estimation of the impact of data in queries. We develop a method to estimate the impact factor of input variables in SQL statements. We will present a rule system supporting the efficient calculation of the query dependencies using these estimates.

We will show how to use the results of the conducted analysis to categorize, aggregate and visualize the dependencies to address various planning and decision support problems.

The methods and algorithms presented in the thesis have been implemented and tested in different data warehouse analysis and visualization tasks for tens of large international organizations. Some of these systems contain over a hundred thousand database objects and over a million ETL objects, producing data lineage graphs with more than a hundred thousand nodes. The analysis of the system performance over real-life datasets of various sizes and structures presented in the last chapter demonstrates linear performance scaling and the practical capacity to handle very large datasets.

ACKNOWLEDGEMENTS

First, I would like to warmly thank my supervisor, Prof. Tanel Tammet, for the motivation, encouragement and guidance through all these years, as well as the patience and support during all stages of the scientific process and practical works.

I would like to thank all those people who have contributed to the process leading to the completion of the given work. I thank Margus Kliimask and other colleagues from Mindworks Industries for a creative and productive environment, for wonderful ideas and hard work. I would like to thank my colleagues from the Eliko Competence Centre and my fellow doctoral students from Tallinn University of Technology and Graz University of Technology.

Finally, I thank my family members and my friends who have been supportive and have been with me during the long journey of my doctoral studies.

LIST OF PUBLICATIONS

The work of this thesis is based on the following publications:

A Tomingas, K.; Kliimask, M.; Tammet, T. Data Integration Patterns for Data Warehouse Automation. In: New Trends in Database and Information Systems II: 18th East European Conference on Advances in Databases and Information Systems (ADBIS 2014). Springer, 2014.

B Tomingas, K.; Tammet, T.; Kliimask, M. Rule-Based Impact Analysis for Enterprise Business Intelligence. In: Artificial Intelligence Applications and Innovations (AIAI 2014), IFIP Advances in Information and Communication Technology. Springer, 2014.

C Tomingas, K.; Tammet, T.; Kliimask, M.; Järv, P. Automating Component Dependency Analysis for Enterprise Business Intelligence. In: 2014 International Conference on Information Systems (ICIS 2014).

D Tomingas, K.; Järv, P; Tammet, T. Discovering Data Lineage from Data Warehouse Procedures. In: 8th International Joint Conference on Knowledge Discovery and Information Retrieval (KDIR 2016).

OTHER RELATED PUBLICATIONS

E *Tomingas, Kalle; Järv, Priit; Tammet, Tanel (2017). Computing Data Lineage and Business Semantics for Data Warehouse. Accepted for publication in: Lecture Notes in Communications in Computer and Information Science" (CCIS), Springer.*

F *Tomingas, Kalle; Kliimask, Margus; Tammet, Tanel (2014). Mappings, Rules and Patterns in Template Based ETL Construction. In: TUT Research Report Series: The 11th International Baltic Conference on DB and IS, DB&IS2014, Tallinn, Estonia. Tallinn, Estonia.*

G *Tammet, T.; Tomingas, K.; Luts, M. (2010). Semantic Interoperability Framework for Estonian Public Sector's eServices Integration. In: Proceedings of the 11th European Conference on Knowledge Management: Universidade Lusíada de Vila Nova de Famalicão Portugal: 2-3 September 2010, 2: 11th European Conference on Knowledge Management - ECKM 2010, Portugal, 2-3 September 2010. Ed. Eduardo Tomé. Academic Publishing Limited, 988−995.*

H *Tomingas, Kalle; Luts, Martin (2010). Semantic Interoperability Framework for Estonian Public Sector's E-Services Integration. In: Ontology Repositories and Editors for the Semantic Web: Proceedings of the 1st Workshop on Ontology Repositories and Editors for the Semantic Web, Hersonissos, Crete, Greece, May 31st, 2010. (CEUR Workshop Proceedings; 596).*

AUTHOR'S CONTRIBUTION TO THE PUBLICATIONS

A Author contribution to the paper A started with a research problem and methodology setup. It covers model and software development, testing and experimenting, conducting analysis and writing most of the text.

B The author was one of the main contributors and writers of the paper B. The most important part of the work was the development of the methodology to solve data lineage and impact problems based on technologies described in the previous paper A. Additional technical work, like building models, development of software, testing, and analysing the results, were part of feasibility studies and adjustment of the methodology.

C The author was one of the main writers, continuing the development of the methodology and the rule system started in the paper B. The main content of paper B is re-published in the paper C with more additional details, examples and visualizations. The new details, practical data processing and visualizations were the main tasks and the focus of the paper C that was published in the field of information systems rather than computer science.

D The author was one of the main contributors of the paper D. The main tasks and the results of the work were: new formalizations for the data processing rule system, development of the new prototype, performance measurements and new visualization techniques.

E The author was one of the main contributors of the paper E. Again, the main tasks and the results of the work were new formalizations for the data processing rule system, development of the new prototype, performance measurements and new visualization techniques, plus a new business semantics model development.

F The paper F was written as a short and initial version of the paper B that was published at the DB&IS2014 conference in Tallinn and presented in the poster session by the author.

G The author was a member of the semantic assets management development project of the Estonian Information Systems Authority. The paper G concludes the project and presents the ideas of the interoperability framework development. The author was one of the main writers of the paper.

H The paper H is a short initial version of the paper G that was presented at the Workshop on Ontology Repositories in the Extended Semantic Web Conference by the author.

Abbreviations

API	Application Programming Interface
BI	Business Intelligence
DBMS	Database Management System
DDL	Data Definition Language
DI	Data Integration
DL	Data Lineage
DML	Data Manipulation Language
DSS	Decision Support Systems
DW	Data Warehouse
EBNF	Extended Backus-Naur Form
EDW	Enterprise Data Warehouse
EAV	Entity Attribute Value
ETL	Extract, Transform, Load
ELT	Extract, Load, Transform
ER	Entity–Relationship
IA	Impact Analysis
IT	Information Technology
ODS	Operational Data Store
OLTP	On-Line Transaction Processing
OLAP	On-Line Analytical Processing
RDF	Resource Description Framework
SQL	Structured Query Language
XDTL	eXtensible Data Transformation Language
XML	eXtensible Markup Language

Terms[1]

Data Warehouse
A data warehouse (DW) is a collection of corporate information and data derived from operational systems and external data sources. DW is designed to support business decisions by allowing data consolidation, analysis and reporting at different aggregate levels. Data is populated into the DW through the processes of data integration or extraction, transformation and loading.

Data Lineage
Data lineage is generally defined as a kind of data life cycle that includes the data's origins and where it moves over time. This term can also describe what happens to data as it goes through diverse processes. Data lineage can help with efforts to analyze how information is used and to track key bits of information that serve a particular purpose (see also: Data Provenance).

Data Integration
Data integration (DI) is a process in which heterogeneous data is retrieved and combined as an incorporated form and structure. Data integration allows different data types (such as data sets, documents and tables) to be merged by users, organizations and applications, for use as personal or business processes and/or functions (see also: Extract-Transform-Load).

Data Provenance[2]
Data Provenance provides a historical record of the data and its origins. The provenance of data which is generated by complex transformations such as workflows is of considerable value to scientists. Provenance is also essential to the business domain where it can be used to drill down to the source of data in a data warehouse, track the creation of intellectual property, and provide an audit trail for regulatory purposes (see also: Data Lineage).

Enterprise Data Warehouse
An enterprise data warehouse (EDW) is a unified database that holds all the business information an organization and makes it accessible all across the company.

Extract-Transform-Load
Extract transform load (ETL) is the process of extraction, transformation and loading during database use, but particularly during data storage use.

Impact Analysis[3]

[1] https://www.techopedia.com/

[2] https://en.wikipedia.org/wiki/Data_lineage

[3] https://en.wikipedia.org/wiki/Change_impact_analysis

Change impact analysis (IA) is defined as "identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change", and they focus on IA in terms of scoping changes within the details of a design.

Dependency Graph[4]

Dependency graph is a directed graph representing dependencies of several objects towards each other. It is possible to derive an evaluation order or the absence of an evaluation order that respects the given dependencies from the dependency graph.

[4] https://en.wikipedia.org/wiki/Dependency_graph

List of Figures

INTRODUCTION

The amount of available data is growing rapidly in many domains and areas of human activity. Traditional and Internet businesses, social media, healthcare and science are a few examples of the fields where accumulated data and processed information can change the scale and the state of those businesses. The development of the internet, connected information systems, social media, new scientific equipment and the rising Internet of Things (IoT) has brought us to the big data era and scale where traditional data processing technologies and methods do not function, do not perform or simply stop working [1].

There are many reasons why we may want to understand the internal structure and functions of a complex data processing systems like data warehouses. Some of the reasons are related to the need to improve system functions, performance or quality and the ability to evaluate them. Others are related to controlling and managing the system effectively and avoiding unwanted or unpredictable behavior of the system. Data warehouse systems collect data from various distributed and heterogeneous data sources, integrating details or summarized information in local database for further processing and analysis for various applications and purposes. Data warehouses are living, continuously developed, enriched and updated systems with variable load, performance and growing data volumes. Data transformation chains can be very long and the complexity of structural changes can be high. Tracing long and complex data flows or dependencies of data transformation components are serious research tasks without special supporting metadata and tools. Tracing data items back from the final reports or applications to the source items and structures is a data lineage problem. Traceability of internal components dependencies is critical when developing and changing system software or configuration and can be defined as problem of impact analysis. Data lineage allows for tracing internal functional relations of data processing systems and gives insight of data flows for better understanding of what the system does. Impact analysis allows for tracing internal component structures and formal relations of the system and gives an understanding of how a system is built from interconnected components.

In this thesis, we address the data lineage and the impact analysis problems in a generalized and multidisciplinary way to use the same methods and approaches in data warehouse or other decision support, data processing, enterprise integration or service-oriented systems. Our goal is to implement methodology, algorithms, representations, architecture and applications that have a relatively small set of functions for specialized tasks, designed to perform and automate complex analytical tasks. The final system design has to be modular, flexible and robust, but also scalable and efficient to easily adapt heterogeneous environments of real life data processing systems.

The chosen approach combines techniques from multiple fields of information technology and computer science, like metadata capture and loading, unified and open-schema data storing, grammar-based program parsing and resolving, probabilistic semantic interpretation of data transformations and rule-based reasoning, graph-based dependency calculations, data and component flow graph visualization, etc.

Motivation and the Problem Statement

Data warehousing (DW) is a complex process of collecting data, cleansing and transforming it into information and knowledge to support strategic and tactical business decisions in organizations. DW is designed as a rapidly growing, subject-oriented, integrated, time-variant and non-volatile collection of data from heterogeneous data sources, with various connected applications, query engines, fixed or open reporting and analytical tools (see Figure 0.1). Data sources can be volatile and data can be structured (e.g., databases, xml files), semi structured (e.g., log files, emails) or non-structured (e.g., text documents). Data consumers from different domains with various interests (e.g., management information, accounting, customer relationship, sales and marketing, resource planning, forecasting, regulatory reporting, etc.) may have a broad spectrum of requirements and service level quality. The process of source data integration is called Extract, Transform, Load (ETL), and has a specific set of specialized tools for data capturing and processing tasks. The processed and stored data consuming process is called Business Intelligence (BI) and has its own set of tools for reporting, ad-hoc querying, data mining, dashboards and other types of analytics. ETL and BI are not independent components: ETL and data requirements are driven by business needs and BI capabilities are limited by the collected and integrated data.

Figure 0.1 A general scheme of a Data Warehouse process and data flows.

To make reasonable and informed business decisions, we need appropriate data and metadata about context, structure, requirements, processing and timing. Answering questions about used data sources, formulas, structures and freshness of data in analytical systems or reports is challenging and not trivial. Components of data warehouses are distributed over multiple physical locations and a diverse set of software tools, and therefore tracing complex data processing metadata is more complicated compared to using processed data. When the produced data and information is the desired and emergent result of a DW system, then the processing metadata is often hidden and captured into internal structures, relations and programming code of separate components of the data processing system. Emerging results, behavior and functions of such a complex system depend on the subsystems and interconnections (formal and functional) of the system's components. To control, manage or predict the behavior of the system, we must review the elements and the relationships between the components on a detailed level. Large data warehouse systems can have hundreds of thousands of tables/views and millions of columns with tens of millions of estimated dependencies between those components.

We call networks of all dependencies over data warehouse system components Enterprise Dependency Graphs (EDG) and we handle functional and structural dependencies as directed graph edges between component nodes. The problem of data lineage (DL) is seen as a data flow sub-graph construction, calculation and navigation between static data structure components (e.g., tables, views, columns, files, reports, fields, etc.). The problem of component impact analysis (IA) is seen as a sub-graph calculation and navigation between active data transformation components (e.g., ETL tasks and mappings, SQL scripts and queries, DB procedures, reporting queries and components, etc.) and passive data structures.

Data warehouse owners and users are facing various data lineage and impact analysis problems because the chains of data transformations are often very long with complex changes of data structures. More than a dozens of staging steps in a sequence is not a rare case when the transformation steps are generated by the supporting ETL tools. The data models that are designed for OLTP systems are not usually suitable for OLAP systems. Denormalization, aggregation, and new fact inference are some of the practical techniques that require new or changed data structures and new processes to perform the tasks. The management of such a complex integration process is unpredictable, and the cost is uncontrollable due to the lack of information about data flows and internal relations of system components. The consequences can include unmanageable complexity, fragmental knowledge, a large amount of technical work, unpredictable results, wrong estimations, rigid administrative and development processes, high cost, lack of flexibility, quality and trust. These risks are related to the ability to answer the following questions about data lineage and impact analysis problems:

- How can the origins of a data elements, structures and transformation formulas be traced?
- How are the data elements of a specific column, table, view or report used?
- When was the data loaded, updated or calculated in a specific column, table, view or report?
- Which loadings, structures, components and reports are impacted when other components are changed?
- Which data, structure or report is used by whom and when?
- What is the time and cost of making changes in programs or data structures?
- What will break when we change a program or data structure?
- Who is responsible for a data structure, program or formula?

The ability to support and automate answering such day-to-day questions determines the benefits, cost, flexibility and manageability of the system. The dynamics in business, the environment and the requirements ensure that regular changes in data management are required for every living organization. Due to its reflective nature, business intelligence is often the most fluid and unsteady part of enterprise information systems. The most promising way to tackle the challenges in such a rapidly growing, changing and complex field is automation. Efficient automation in this particular field requires techniques from multiple areas of computer science: computer language and semantic technologies, a combination of rule systems and reasoning. Our goal is to aid users with

intelligent tools that can reduce the time required for several difficult tasks from weeks to minutes, with higher quality results and smaller costs.

As an example, showcasing the complexity, a real-life data flow graph (Figure 0.2) is captured and visualized with the methods and tools we introduce in this thesis. The underlying graph structure, rules and algorithms form the basis for understanding and automation of complex analysis tasks.

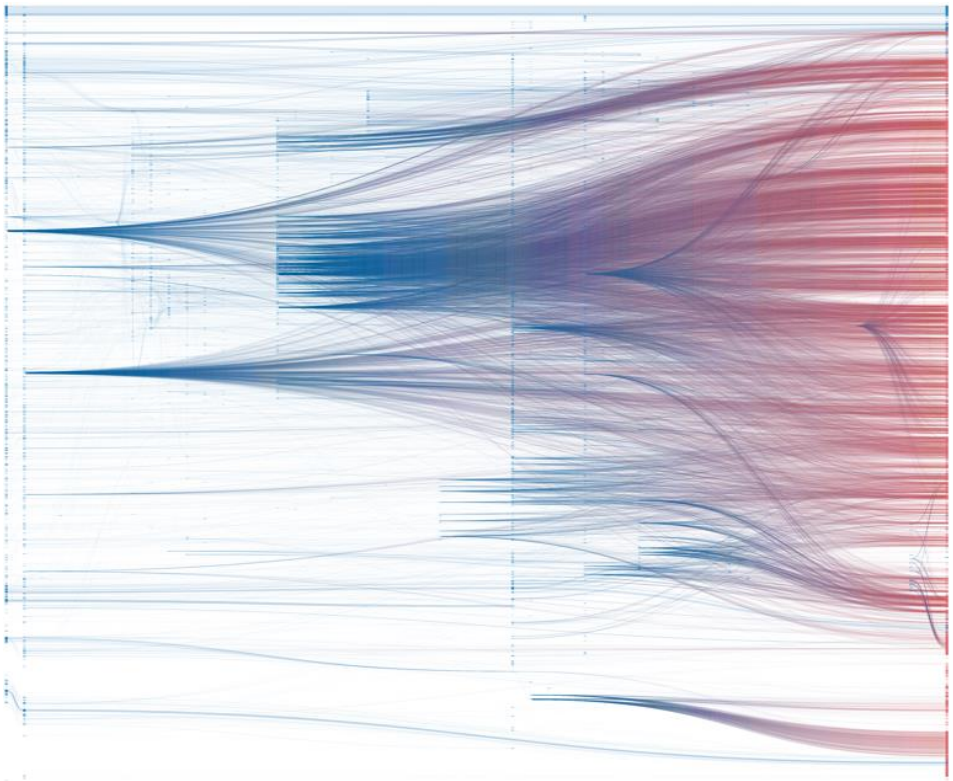


Figure 0.2 Real life Data Warehouse data flows from tables and views (left and middle with blue) to reports (right side with red).

Contribution of the Thesis

The thesis presents a full stack of methods, technologies and algorithms which give analysts a novel way to efficiently solve several existing management and analysis problems in a corporate data warehouse environment.

The work presented lies in the domain of software and knowledge engineering and is based on experimentation with different real-life datasets. The feasibility and usefulness of the results to analysts are validated by practical application on data warehouses of actual large international companies in the financial, utilities, governance, telecom and healthcare sectors. In particular, table 4.1 presents the performance analysis on six large datasets.

The main components of the contribution are:

- A new formalized mapping representation for specifying data transformations between different data formats, locations and storage mechanisms.
- An EAV-style open data model for storing meta-information, ontologies and dependencies of the investigated information system, along with a corresponding graph-based internal representation.
- Algorithms estimating the impact factor of input variables in SQL statements.
- A method for computing the transitive closure of probabilistic dependency chains.
- Data lineage and component dependence visualization methodology.
- Experiments demonstrating the feasibility of the method on large information systems of real companies.
- Analysis and proposals for new ways to apply the lineage analysis to practical problems of finding critical software components, estimating development time, generating documentation and compliance reports.

We describe the underlying technology and abstract mapping concept in our paper A, which forms the foundation for dependency graph representation of data flows and structures (sections 3.2 to 3.4). We draw the methodology framework, system architecture (section 3.1) and define the formal rule system for weighted graph calculation in paper B (sections 3.6 and 3.7). We then extend our rule system with in-memory data structures, illustrate the algorithms with examples and present real-life applications in paper C (section 3.8 and chapter 4). Finally, we present formal definitions and algorithms for graph models and calculations to support semantic data lineage and impact analysis applications (section 3.8), and we present the performance analysis over different real-life datasets in paper D (section 4.3).

The core technologies that are named and used in this thesis and the underlying papers are referenced to their origins in the footnotes. Some of them are closely related to the contribution of the thesis and therefore require additional explanation. The XDTL[5] language and runtime engine are technologies of Mindworks Industries OÜ[6], designed and built by several people inside and outside of the company (including the author of this thesis). The dLineage[7] technology is initially built by the author of the thesis together with my colleague Margus Kliimask, and the XDLT is used as one of the core components of the toolkit. The latter development of modern UI and new features are built with my colleagues form Mindworks Industries.

Organization of the Thesis

The thesis starts with the general introduction and the summary of the contribution.

[5] http://www.xdtl.org/

[6] http://www.mindworks.ee/

[7] http://www.dlineage.com/

The first chapter of the thesis presents an overview of the data lineage and impact analysis fields in data warehousing systems. We will give a simplified example of the problems to be solved. The methodology chapter illustrates our approach to the problems. The first chapter gives a background to the problems that are common for all published papers A to D.

The second chapter gives an overview of the related work in the fields of data lineage, provenance and impact analysis. A focus of the related work chapter is in field of data lineage and data provenance, also other applications in these fields, and the chapter draws wider context to papers B, C and D.

The third chapter of the thesis focuses on the algorithms developed along with the design and the details of our system architecture. We will give detailed presentations and will describe the considerations, options and reasons behind our choices. We will draw a picture of the data model and the basic building blocks with key figures and components that are introduced and used in published papers A to D.

The fourth chapter of the thesis focuses on the details and requirements of our system implementation and the practical case studies in different industries. We will also present new potential application areas. The chapter extends the case studies and the visualizations topics that were introduced in the paper D.

The conclusions chapter summarizes the advantages of our data lineage architecture and system, our contributions and gives suggestions for future work on the topic.

The rest of thesis consists of the four selected publications from the full list of eighth.

1. DATA LINEAGE

This chapter presents a detailed introduction to data lineage and provenance problems, starting with an overview in section 1.1. We continue with an example in section 1.2, with a query example and mapping representation that forms the interconnected data flow graph. We use the same examples in subsequent chapters to illustrate different data linage or impact problems, keeping a connection with different parts in the current thesis.

1.1. Overview of Data Lineage and Provenance

The data lineage, data provenance or pedigree are the overlapping terms used to describe tracing origin sources and derivation of data. The provenance term, in the scientific community, is used synonymously with the lineage term in the database community. Sometimes provenance is also referred to as source attribution or source tagging. Data lineage is a common key component for many different application domains and is also the subject of studies in the field of Computer Science or Data Science. Many business and scientific domains, like scientific data management, big data, machine learning, data warehousing or business intelligence, need provenance or lineage metadata on the origin, rules, transformation, derivation, history, timing, context and background of the used and processed data. Authenticity, integrity, correctness and trustworthiness of information are common requirements for different domains that can be established with effective tracing of data lineage. From scientific and business perspectives, data sets are not very useful without knowing the exact sources, processing methods and rules of derived data sets [2].

Data warehouses [3] and curated databases [4] are typical examples where lineage information is essential. In both databases, comprehensive and often manual effort is usually expended in the construction of the resulting database — in the former, in specifying the ETL process, and in the latter, in incrementally adding and updating the database. Data lineage adds value to the data by explaining how it was obtained. It is important to understand the lineage of data in the resulting database to check the correctness of an ETL specification or assess the quality and trustworthiness of the collected data [5].

There are two levels of granularity in lineage described in previous works: workflow or coarse-grained provenance and data or fine-grained provenance [6]. The coarse-grained workflow lineage describes the data processing components, tasks and programs as a sequence of steps to capture and present general transformations between data sources and targets without specific details. The number of steps and the level of detail can vary between hardware and software platforms and components to transformation programs and sub-components. Fine-grained data lineage describes detailed information and derivation of data items, like data structures, columns, tuples or rows, and represents it as a sequence of transformation steps to trace from sources to targets or vice versa.

Both detail and granularity levels can be seen in combination with up to three types of lineages to answer different questions [7]:

- *Why lineage* refers to context of data transformations and provides justification for input data elements appearing in the output. Why lineage answers questions like how some parts of input data influenced the output data.
- *How lineage* refers to the transformations of the source data elements and answers questions like how inputs were manipulated to produce given output.
- *Where lineage* refers to the locations of the data sources and structures from which the data was extracted and answers questions like where the data comes from or which inputs were used for a given output.

These three notions of why, how and where provenance are used as independent or combined approaches to the data lineage solutions in databases. The previous works that follow and cover these categories are analyzed by Cheney et al. [5] and Tan [6], but there are also works that do not fit neatly into the why, where and how provenance framework. Such works include Wang et al.'s Polygen model [8], Cui et al.'s lineage tracing [9], Widom's Trio system [10] or Woodruff and Stonebraker's work on lineage [11] [5].

To illustrate different lineage types, consider the following simple data loading SQL query from the source table Account (Nbr, Type, State) to the target table Agreement (Agreement_Nbr, Agreement_Type, Agreement_State):

```
INSERT INTO AGREEMENT (Agreement_Nbr, Agreement_Type, Agreement_State)
SELECT Nbr, Type, Coalesce(State,0)
FROM ACCOUNT
WHERE Type = 'A'
AND End_Date is not null
```

The Where lineage for every target table column (Agreement_Nbr, Agreement_Type, Agreement_State) describes where data comes from and corresponds to select list columns (Account.Nbr, Account.Type, Account.State) in the SQL query. The How lineage for each target column column (Agreement_Nbr, Agreement_Type, Agreement_State) describes the column data transformation logic and expressions of each source column (copyOf(Nbr), copyOf(Type), Coalesce(State,0)) in the SQL query. The Why lineage for each target column comes from the conditions part that is present in the where (or join) of the SQL query and describes the context of data transformations like the two predicates here: Account.Type = 'A' and Account.End_Date is not null.

Generic data transformation can be defined as a set of functions $Tr(tr_1..tr_n)$ over source datasets $S_1(s_{1.1}..s_{1.m})$ to $S_{n(}s_{n.1}..s_{n.m)}$ that produce target or output dataset $T(t_1..t_n)$ in context of $C(S_1..S_n)$: $T = Tr(S_1..S_n, C(S_1..S_n))$. General data lineage of target dataset T is defined as a lineage function L: $L(T) = (S_1..S_n)$ and specific where, how and why properties by functions: $L_{where}(T) = (S_1..S_n)$, $L_{how}(T) = Tr(S_1..S_n)$ and $L_{why}(T) = C(S_1..S_n)$. The previous example SQL query column Agreement_State lineage properties can be described as follows:

- L_{where}(Agreement.Agreement_State) = Account.State
- L_{how}(Agreement.Agreement_State) = Coalesce(Account.State,0))

- L_{why}(Agreement.Agreement_State) = Account.Type='A' and Account .End_Date is not null

The previous research on data lineage and provenance has been based on one of two computational approaches in general:

- *The non-annotation approach*, which assumes the execution of a set of transformation functions against the source or input dataset to generate the output dataset in order to compute the data or row level lineage of transformation and target dataset; and
- *The annotation approach*, which carries additional information in transformation to target dataset; this requires modifications of the initial transformation functions and requires extra space for maintaining additional data. The analysis of additional data allows for computation of the data or row level lineage without access to the input dataset.

In this thesis, we focus mainly on the data lineage problem and practical solutions in database environments and use the *data lineage* term instead of *provenance*. We have chosen the *non-annotation approach* to the data lineage problem to support fast start and no impact on the working systems. We also take advantage of data structures and transformations metadata, capture query semantics and make probabilistic score calculation and logic-based inferences about the input or output data, without a need for and access to the real data (i.e. only metadata is used).

1.2. A Motivating Example

As an example of a financial industry data warehouse data lineage and data impact problems, we have constructed our data loading and transformation scenario with four SQL queries and four source tables. The data form the ACCOUNT and LOAN tables are consolidated to one unified AGREEMENT table, then we join the BALANCE table and two new tables, DEPOSIT_SUMMARY and LOAN_SUMMARY, populated with denormalized data for further querying and reporting. The next table (Table 1.1) below presents four SQL DML queries from two different but dependent data loading jobs. The Job1 is responsible for data loading to the DW and the Job2 is responsible for loaded data manipulations and denormalization.

Table 1.1 Data transformation SQL query examples used in DW loading jobs.

SQL Query 1 from Job 1
INSERT INTO AGREEMENT (Agreement_Nbr, Agreement_Type, Agreement_State) SELECT T1.Account_Nbr, T1.Type, T1.State_Code FROM ACCOUNT T1 JOIN ACCOUNT_STATE T2 ON T2.Code = T1.State_Code WHERE T2.State = 'Active' AND T1.Type = 'A'
SQL Query 2 from Job 2

```
INSERT INTO DEPOSIT_SUMMARY (Period_Date, Agreement_Nbr, Agreement_State,
Balance_Amt)
SELECT T3.Balance_Date, T4.Agreement_Nbr, T4.Agreement_State, T3.Balance_Amt
FROM AGREEMENT T4
JOIN BALANCE T3 ON T4.Agreement_Nbr = T3.Agreement_Nbr
WHERE T4.Agreement_Type = 'A'
AND T4.Agreement_State = 2
AND T3.Balance_Date = DATE-1
```

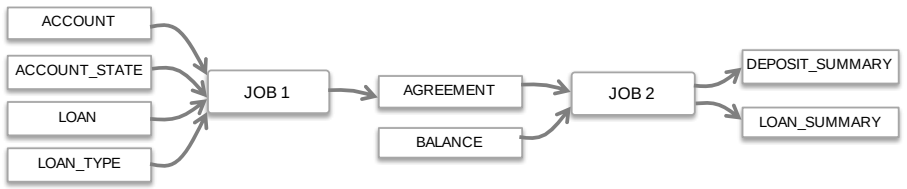
SQL Query 3 from Job 1

```
INSERT INTO AGREEMENT (Agreement_Nbr, Agreement_Type, Agreement_State)
SELECT T6.Loan_Id, 'L', case when T6.State = 'New' then 1 when T6.State = 'Active'
then 2 else 0 end
FROM LOAN T6
JOIN LOAN_TYPE T7 ON T6.Loan_Type = T7.Code
WHERE T7.Type in ('Private', 'Business')
AND T6.State in ('New', 'Active')
```

SQL Query 4 from Job 2

```
INSERT INTO LOAN_SUMMARY (Period_Date, Agreement_Nbr, Agreement_State,
Principal_Amt)
SELECT T3.Balance_Date, T4.Agreement_Nbr, T4.Agreement_State, T3.Balance_Amt
FROM AGREEMENT T4
JOIN BALANCE T3 ON T4.Agreement_Nbr = T3.Agreement_Nbr
WHERE T4.Agreement_Type = 'L'
AND T4.Agreement_State = 2
AND T3.Balance_Date = DATE-1
```

The dependencies between the source and target tables, jobs and the queries can be extracted from the queries and presented as a directed graph. The structures and components are nodes of the graph and dependencies between source and target tables are the directed edges of the graph. The direction of the edge points the data flows from the source to the target structures. The Figure 1.1 has two coarse-grain data flow graphs with the detail level of tables and jobs or tables and queries. We can use those graphs as illustrations for data lineage and impact analysis problems, where data lineage questions can be answered as querying sub-graphs in the target-to-source direction and data or component impact questions can be answered by sub-graph queries in the source-to-target direction. We can also notice that it is not possible to see which table data is moving to the target tables and which is used only for filtering or lookups without going to the fine-grain, column and query components level. For example, we can see that ACCOUNT_STATE and LOAN_TYPE tables are used as sources for the job and query levels, but we do not recognize that the data is not loaded to the AGREEMENT table and is used only for filtering rows with certain types or statuses.



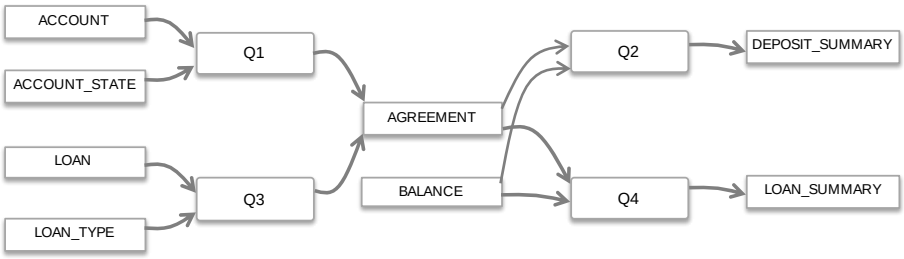


Figure 1.1 DW data transformation flows in table, job and query levels.

The next Figure 1.2 illustrates the fine-grain level of detail, where the query components allow us to construct more complex and detailed dependency graphs to answer data lineage and impact questions at the column level. The transformation queries (Q1…Q4) are parsed to abstract mappings (M1…M4) with all the available source and target tables. Each mapping has data transformation elements (t1.1…t4.3), joins (j1.1…j4.1) and filter conditions (f1.1…f4.1) according to the query structure and expressions. All source and target tables have connected columns according the usage in the query expressions. Additional transformation expressions, key-value constraints and conditions are extracted from the query text and are connected to mappings for further semantic calculations and instance-level data lineage tracing.

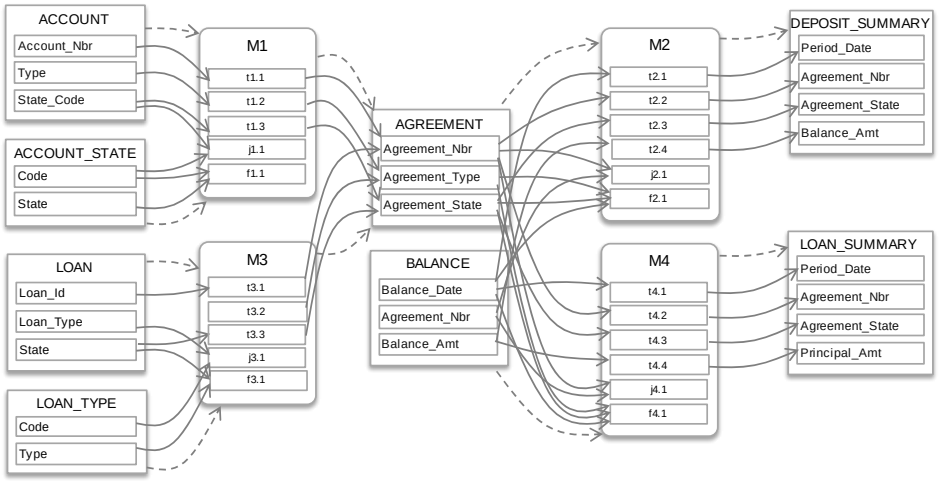


Figure 1.2 DW data transformation flows in table, column and query component levels.

The result of the parse and query processing is a detail-level dependency graph that allows for more precise data lineage and impact analysis in the table and column levels. The graph is a representation of the discrete source and target dependencies between the input and output components without additional knowledge to describe how the data is transformed or filtered in the transformation query. Analysis of the queries Q1…Q4 and predicates from the where clauses shows that different and independent sets of rows produced by

queries Q1 and Q3 from the ACCOUNT and LOAN tables are loaded to the same AGREEMENT table. We also notice that queries Q2 and Q4 are using the same independent sub-sets of rows from the same AGREEMENT table using filtering predicates Agreement_Type = 'A' and Agreement_Type = 'L'.

We can conclude the example by saying that, based on the data structures information and understanding the query semantics in terms of transformation functions and filter predicates, we can make logical inferences about data rows or tuples that are involved or excluded in data lineage workflows.

1.3. Summary

This chapter presented an introduction to data lineage, provenance and impact analysis problems, starting with the overview in section 1.1, followed by the example section 1.2, with queries and mapping representation forms for the interconnected data flow graph that will be used in subsequent chapters to illustrate different data linage or impact problems. These connect with different parts of the current thesis.

2. RELATED WORK

Impact analysis, traceability and data lineage issues are not new. An overview of the data lineage and data provenance tracing studies were collected by Cheney et al. [5], historical and future perspectives were discussed by Tan [6] and the last decade of research activities were presented by Pribe et al. [12]. Lineage and provenance has been studied in scientific data processing areas [7], [8], [9] and in the context of database management systems [2], [6], [16]. Multiple notions of lineage and provenance in database systems have been used to describe relationships between data in the source and in the target: *where* output records came from [7], *why* an output records were produced by inputs [7], [17] and a *how* output record was produced [18]. The query behavior lineage tracking has been used in classical database problems like view update [19] or the expressiveness of update languages [20], and the study of annotation propagation [20], [21] or updates across peer-to-peer systems [22]. The data-driven and data dependent processes and provenance theoretical and practical models described by Deutch et al. [23].

The distinction is made between coarse-grained, or schema-level, provenance tracking [24] and fine-grained-, or data instance-, level tracking [25]. The methods of extracting the lineage are divided into physical (annotation of data by Missier et al.) and logical, where the lineage is derived from the graph of data transformations [26].

We can also find various research approaches and published papers from the early 1990's and later with methodologies for software traceability [27]. The problem of data lineage tracing in data warehousing environments has been formally founded by Cui and Widom [9], [17]. Data lineage or provenance details levels (e.g., coarse-grained vs fine-grained), question types (e.g., why-provenance, how-provenance and where-provenance) and two different calculation approaches (e.g., eager approach vs. lazy approach) have been discussed in multiple papers [6], [28], and formal definitions of the why-provenance have been given by Buneman et al. [7]. Other theoretical works for data lineage tracing can be found in [29] and [30]. Fan and Poulovassilis developed algorithms for deriving affected data items along the transformation pathway [31]. These approaches formalized a way to trace tuples (resp. attribute values) through rather complex transformations, given that the transformations are known on a schema level. This assumption does not often hold in practice. Transformations may be documented in source-to-target matrices (specification lineage) and implemented in ETL tools (implementation lineage). Woodruff and Stonebraker created a solid base for the data-level and operator processing based the fine-grained lineage, in contrast to the metadata-based lineage calculation in their research paper [11].

Priebe et al. concentrated on proper handling of specification lineage, a significant problem in large-scale DW projects, especially when different sources have to be consistently mapped to the same target [12]. They proposed a business information model (or conceptual business model) as the solution and a central mapping point to overcome those issues. The requirement and design level

lineage and traceability solutions for next generation DW and BI architecture described by Dayal et al. [32].

Other ETL-related practical works that are based on conceptual models can be found in [33] and [34]. Ontologies and graphs-based practical works related to data quality and data lineage tracking can be found in [35], [36] and [10]. De Santana proposed the integrated metadata and the CWM metamodel-based data lineage documentation approach [37]. The conceptual modeling approach of ETL workflows described by Bala et al. [38] in the Big Data landscape and Basal [39] presented a semantic approach to combine the traditional ETL approach with the Big Data challenges. Another related work from the field of data lineage and scientific data provenance by Wang et al. [40] brings together challenges and opportunities of Big Data, including volume, variety, velocity and veracity, with the problems of scientific workflow tracking and reproducibility. The cloud-based or distributed systems have their own limitations for data lineage tracing and the data-centric event logging introduced and discussed by Suen et al. [41].

In addition to data lineage and provenance in databases, closely related workflow provenance tracking is an active research topic in the scientific community. The overview of scientific workflow provenance was captured in surveys by Bose and Frew [15] and Glavic and Dittrich [42], and tutorials with research issues, challenges and opportunities were described by Davidson and Freire in [43]. General design and principles of scientific workflow lineage and provenance systems were introduced and discussed by Bose [44], Simmhan et al. [45], Altintas et al.[46], Chervenak et al. [47] and Wu et al. [48], and there are many different flavors and accents, like the collaborative approach from Missier et al. [49] and Altintas [50]; the cloud-based or distributed systems by Cruz et al. [51], Marinho et al. [52] and Wang et al. [53]; the Big Data-oriented approach by Wang et al. [40]; the graph-oriented approach by Anand et al. [54], [55], Acar et al. [56] and Biton et al. [57]; the ontology-driven approach by Bowers et al. [58]; the semantic web and semantic technologies based approaches by Kim et al. [59], Ding et al. [60] and Sahoo et al. [61]; the user- or scientist-oriented systems from Bowers et al. [62]; and the user- or subjective scientist eliminative-based approach by Finlay [63]. The scientific workflow lineage and provenance research does not end here, but continues in different scientific domains, like bioinformatics by de Paula et al. [64] and Buneman et al. [65] or genomics by de Paula et al. [66].

The lineage and provenance problems are not limited with databases, -flows and scientific workflows, but having common challenges in field of curated databases, semantic web, open linked data, e-Sciences and the growing social networking landscape. Some interesting works can be found on the borders of the different domains and disciplines by Chirigati and Freire [67], Hartig and Zhao [68], Moreau [69], [70] and Moreau et al. [71].

In the context of our work, efficiently querying the lineage information after the provenance graph has been captured is of specific interest. Heinis and Alonso presented an encoding method that allows space-efficient storage of transitive closure graphs and enables fast lineage queries over that data [24]. Anand et al. proposed a high-level language QLP, together with the evaluation techniques that

allow storing provenance graphs in a relational database [72]. These techniques are supported by a pointer-based encoding of the dependency closure that supports reducing storage requirements by eliminating redundancy.

Several commercial ETL products are addressing the impact analysis and data lineage problems to some extent (e.g., Oracle Data Integrator, Informatica PowerCenter, IBM DataStage, Teradata Metadata Services or Microsoft SQL Server Integration Services), but those tools and the dependency analysis performed is often limited to the basic functions of a particular system. Another group of commercial tools is formed by the specialized metadata integration products not related to a particular ETL tool, offering a more sophisticated suite of dependency analysis functionality. The examples are ASG Rochade[8], InfoSphere Information Governance Catalog from IBM[9], Data Governance and Catalog from Collibra[10], Informatica Metadata Manager[11], SAP Information Steward[12], Metacenter from Data Advantege Group[13], Adaptive Metadata Manager[14], Troux Enterprise Architecture Solution[15], Metadata Management from Cambridge Semantics[16], Metdata System from AB Initio[17] or MetaIntegration Metadata Management[18], most of which have their own limitations in terms of available functionality and adapters to other products [12].

In addition to full scale metadata management or data governance products, there are several new generation technology companies, who fit into the picture one or another way: Automated SQL query parsing and lineage extraction from SqlDep[19] and Manta[20]; Metadex data lineage solution from CompactBI[21]; Accurity business glossary and data governance solutions from Simplity[22]; Machine Learning based metadata and data lineage discovery solutions from RokittAstra[23]; Data lineage and governance solutions from Synergy[24]; SQL parsing, analyzing, documenting and data lineage discovery tools from General SQL Parser[25]; Data mapping and documenting oriented Mapping Manager

[8] https://www.asg.com/

[9] http://www-03.ibm.com/software/products/en/infosphere-information-governance-catalog

[10] https://www.collibra.com/

[11] https://www.informatica.com/products/informatica-platform/metadata-management.html

[12] http://www.sap.com/community/topic/information-steward.html

[13] http://www.dag.com/

[14] http://www.adaptive.com/metadata-manager

[15] http://www.troux.com/

[16] https://www.cambridgesemantics.com/solutions/metadata-management

[17] https://www.abinitio.com/en/system/enterprise-meta-environment

[18] http://www.metaintegration.com/Solutions/#MetadataManagement

[19] https://www.sqldep.com/

[20] https://getmanta.com

[21] http://www.compactbi.com/

[22] http://www.accurity.eu/

[23] https://www.rokittastra.com/

[24] http://www.meta-analysis.fr/en/la-solution/

[25] http://sqlparser.com/

solution from AnalytixDS[26]; Automated metadata capture, analysis and collaboration tools by AlexSolutions[27]; Data lineage and graph data analysis and visualization tools from Linkurious[28]; Synapse data mapping, analysis, tagging and visualization tools from Sapient[29]; Axon governance, lineage and collaboration tool from Diaku[30]; and finally fully automated, semantic metadata capture, data lineage, impact analysis, business governance and visualization in toolset dLineage[31], that is based on the methodology, algorithms and ideas, that are described in this thesis.

2.1. Summary

This chapter gave an overview of previous works and scientific studies in the field, along with the industry landscape.

[26] http://analytixds.com/products/mapping-manager/

[27] http://alexsolutions.com.au/

[28] https://linkurio.us

[29] https://synapse.sapientconsulting.com/

[30] https://www.diaku.com

[31] http://www.dlineage.com

3. ALGORITHMS AND METHODS

This chapter presents the algorithms and methods we have designed and implemented. The overall architecture follows the methodological pathway presented on a conceptual level in section 3.1. We describe the metadata database design in section 3.2 and different metadata models (metamodels) for databases and data integration in section 3.3. More details about the underlying foundation and mapping design can be found in article A. The logical path with query parsing and resolving techniques continues in section 3.5, with data transformation evaluations and weight calculations. The rule system implemented for graph construction and calculations is discussed in section 3.7, the semantic layers on calculated graphs are discussed in section 3.8. The rule system and graph calculations are discussed at a detailed level in papers B, C and D.

3.1. Overall Architecture and Methodology

The overall architecture is based on an independent metadata collection and storage framework with dynamic schema and unified metamodels, grammar-based query parsing and resolving, probabilistic data transformation weight calculation, rule-based graph calculation and web-based user interface components. The architecture follows the methodology steps (from 1 to 8) presented in Figure 3.1:

1. Scanners collect metadata from different systems that are part of the DW's data flow (DI/ETL processes, data structures, queries, reports, etc.) to the open-schema metadata database (PostgreSQL or Oracle).
2. The SQL parser is based on a customized grammar, the GoldParser parsing engine and the Java-based XDTL engine.
3. The rule-based parse tree mapper extracts and collects meaningful expressions from the parsed text, using declared combinations of grammar rules and parsed text tokens.
4. The query resolver applies additional rules to expand and resolve all the variables, aliases, sub-query expressions and other SQL syntax structures that encode crucial information for data flow construction.
5. The expression weight calculator applies rules to calculate the meaning of data transformation, join and filter expressions for impact analysis and data flow construction.
6. The rule-based reasoning engine propagates and aggregates weighted dependencies.
7. The dependency graph is stored along with the collected metadata in a relational database as binary and directed relations between node objects.
8. The directed and weighted sub-graph calculations, visualization and web-based UI is used for data lineage and impact analysis applications.

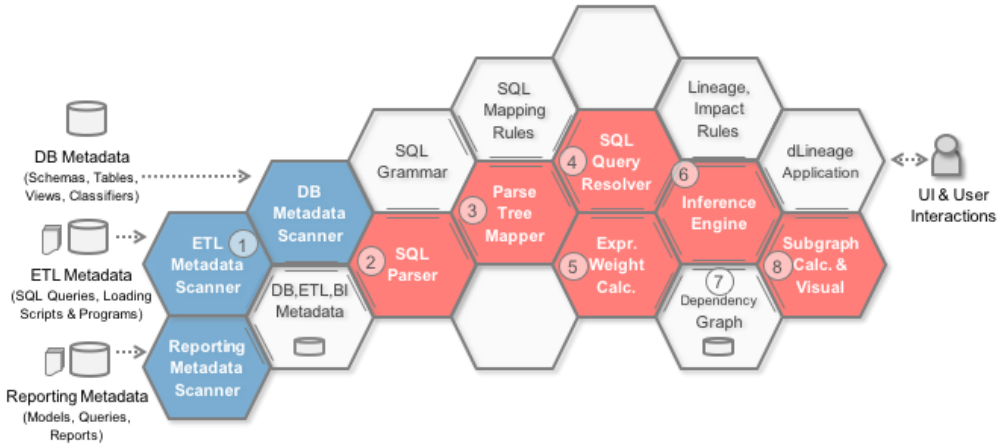


Figure 3.1 Methodology and system architecture components.

The color codes differentiate the data capture components (blue), active data processing components (red) and passive supporting components (white). The double lines in the comb-cell figure express the data flow bonds between the active or passive components.

The base components of the system architecture were introduced in paper A. Our general methodological and architecture scheme is presented in papers B and C and developed further in paper D.

3.2. Metadata Database

Our metadata database is built on a relational database technology for different knowledge management and rule-based analytical applications. The repository is designed according to the OMG Metadata Object Facility (MOF)[32] idea with separate abstraction and modeling layers (M0-M3). The physical data model (schema) is based on principles and guidelines of the EAV (Entity-Attribute-Value)[33] modeling technique suitable for modeling highly heterogeneous data with a very dynamic nature. Metadata models and schema definitions in EAV are separated from physical storage, and therefore modifications to schema on the "data" level can easily be done without changing DB structures, just modifying corresponding metadata. The chosen approach is suitable for open-schema implementations (similar to key-value stores) where the model is dynamic and semantics are applied in query time, but also model-driven implementations with formal and well-defined schema, structure and semantics. The used URI reference mechanism and resource storage scheme makes our metadata repository a semantic data store that is comparable to the Resource Description Framework (RDF) and can be serialized in different semantic formats or notations (e.g., RDF/XML, N3, N-Triples, XMI, etc.) using XML or RDF APIs.

[32] https://en.wikipedia.org/wiki/Meta-Object_Facility

[33] https://en.wikipedia.org/wiki/Entity-attribute-value_model

The physical schema (Figure 3.2) can be seen as a general-purpose storage mechanism for different metadata and knowledge models, and also as a communication medium or information integration and exchange platform for different software agents or applications (e.g., metadata scanners, metadata consumers, etc.). Built-in limited reasoning capability is based on the recursive SQL capability and is captured through data and metadata APIs to implement inheritance and model validation functions. Semantic representation of data allows for extended functionality with predicate calculus reasoners or applying other external rule-based reasoners (e.g., Jena) for more complicated reasoning tasks, like deduction of new knowledge.

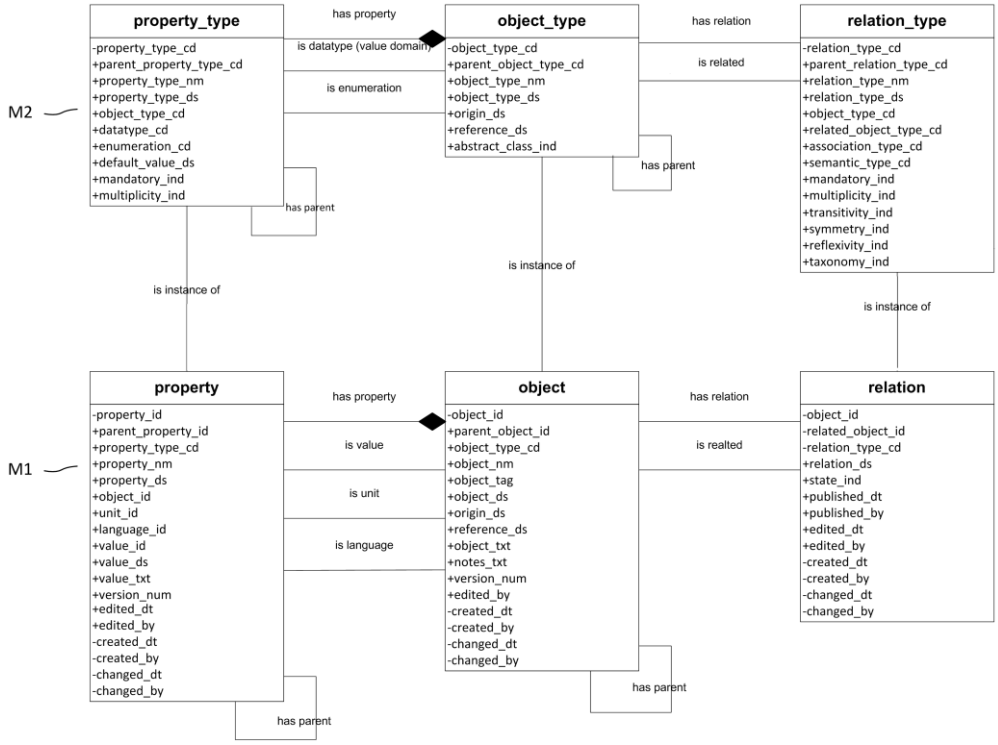


Figure 3.2 Metadata database physical schema tables.

The repository contains integrated object-level security mechanisms and different data access APIs (e.g., data, metadata, XML/XMI, RDF API, etc.) that are implemented as relational database procedures or functions.

An unlimited number of different data models can exist inside our metadata model simultaneously, with relationships between them. Each of these data models constitutes a hierarchy of classes where the hierarchy might denote an instance relationship, a whole-part relationship or some other form of generic relationship between hierarchy members. We designed several predefined metadata models for data lineage and impact analysis data:

- terminology and classification (business meaning and governance);
- relational database (DB and SQL);
- data integration model (ETL);
- reporting model (OLAP, BI, Reporting); and
- mappings model (formalized abstract mappings).

3.3. Design of Metadata Models and Mappings

The relational database metamodel is used to store detailed information about the sources and targets of data transformations. The RDB metamodel focuses on the main database objects, e.g. Schema, Table, View, Column, Datatype, Procedure, etc. The ETL metamodel is based on the OMG CWM[34] reference architecture with base concepts like Folder, Package, Step and Task. The ETL model is focused on the organization and structure of data processing packages, sequences and dependencies of events, relations between elements controlling data processing workflow, etc. The reporting metamodel focuses on Report, Model, Dimension, Hierarchy and Measure elements, taking advantage of the mapping metamodel to store query mappings and related classes, and is used to store information describing the presentation layer. The mappings metamodel used to manage decomposed relationships and expressions in a unified manner.

Various metadata and data integration and ETL models are discussed and used in previous works [73],[74]. We decided to implement our own "soft" models that do not require a database physical schema change when changing the metamodel. The details about the abstract mappings model design, storage and usage is presented in article A.

3.4. Data Capture, Store and Processing with Scanners

The Extensible Data Transformation Language (XDTL) is an XML-based descriptive language designed for specifying data transformations between different data formats, locations and storage mechanisms. XDTL was created by Mindworks Industries as a Domain Specific Language (DSL) for the ETL domain and was designed to keep in mind principles like modularity, extensibility, reusability, decoupled declarative (unique) and procedural (repeated) patterns. XDTL syntax is defined in an XML Schema document. Wildcard elements of XML Schema enables extending the syntax of the core language with new functionality implemented in other programming languages or in XDTL itself. XDTL scripts are built as reusable components that have clearly defined interfaces via parameter sets. Components can be serialized and de-serialized between XML and database representations, thus making XDTL scripts suitable for storing and managing in a data repository. XDTL provides functionality to use externally stored data mappings for the scripts and decoupled from the scripts. Therefore, mappings stored in a repository can exist as objects independent from

[34] https://en.wikipedia.org/wiki/Common_Warehouse_Metamodel

the transformation process and can be reused by several different processes. XDTL acts as a container for a process that often must use facilities not present in XDTL itself (e.g., SQL, SAS language, etc.).

The purpose of a scanner is to extract and capture all relevant metadata about a certain class of data elements and store it in a predefined, structured manner. Scanners components (No1 in Figure 3.1) are collecting external systems metadata, like database data dictionary structures, ETL system scripts and queries or reporting system query models and reports, and all structural information is extracted and stored to the metadata database. The scanned objects and their properties are extracted and stored according to defined meta-models, like relational databases, data integration, reporting and business terminology models. Metamodels contain ontological knowledge about collected metadata and relations across different domains and models. The scanners technology and open-schema metadata database design are described in more detail in our article A.

The database scanner is a program implemented as an XDTL package or script that transforms metadata from a database dictionary into an RDB metamodel. Database scanners are based on ANSI SQL Information Schema[35] specification and are currently being implemented for MsSQL, PostgreSQL, Greenplum, Oracle, Teradata, IBM DB2, Netezza, Vertica and other database platforms. All database scanners are implemented as two-phase processes that materialize (scan) scanned data in a format conforming to Information Schema definition. A separate process (store) stores this temporary information in a permanent storage media (database). Decoupling those processes allows for reusing components created for different database products in multiple combinations.

Application scanning is a procedure implemented as an XDTL package that transforms metadata from application repository or internal representation into an application metamodel. Several application scanners have been implemented for various ETL, OLAP and Reporting tools.

Oracle Data Integrator (ODI) is an ETL tool quite common in DW environments, especially in relation to Oracle databases. The ODI scanner extracts information relevant for impact analysis, i.e., all data sources and targets, column mappings, transformations, JOIN and WHERE conditions, variables, references to external processes, etc.

Business Objects (BO) is a widely used reporting tool used in DW. The BO scanner extracts metadata from a BO application repository and File Store, transforming it into a reporting metamodel. The granularity of the extracted information is relevant to impact analysis requirements.

3.5. Query Parsing and Metadata Extraction

To construct data flows from the very beginning data sources (e.g., the accounting system) to the end points (e.g., reporting system) we should be able to connect the same and related objects in different systems. To connect objects,

[35] https://en.wikipedia.org/wiki/Information_schema

we have to understand and extract relations from SQL queries (e.g., ETL tasks, DB views and procedures) and scripts (e.g., loader utility scripts) and expressions (e.g., report structure) that are collected and stored by scanners. To understand the data transformation semantics that are captured within the query language statements (e.g., insert, update, select and delete queries) and expressions, we have to involve external knowledge about query language syntax and grammatical structure. We used a general-purpose Java-based parser engine[36] and developed a custom SQL grammar that was written in Extended Backus-Naur Form (EBNF)[8]. Our grammar is based on ANSI/SQL syntax, but it contains a large set of dialect specific notations, syntax elements and functions that were developed and trained using large real-life SQL query sets from the DW field. The current grammar edition is based on the Teradata, Oracle, Greenplum, Vertica, Postgres, IBM DB2, Netezza and MsSql dialects.

Example 1. SQL select statement grammar sample in EBNF format:

```
<Select Stm> ::= <Select> UNION <Select Stm>
| <Select> UNION ALL <Select Stm>
| <Select>
<Select>::= SELECT <Columns><Into Clause><From><Where><Group Clause><Qualify Clause>
 <Having Clause> <Order Clause>
<SubqueryStm> ::='('<SelectStm>')`  <Columns> ::=<Restriction>'*'
| <Restriction> <Column List> <ColumnList> ::=<ColumnItem>','<ColumnList>
| <Column Item>
<Column Item> ::= <Column Source>
| <Column Source> <Alias>
| <Column Source> ' AS ' <Alias> <Column Source> ::= <Column Source Item>
<Column Source Item> ::= '('<Column Source Item>')' | <Add Exp>
<From> ::= FROM <Id List> <Join Chain> |
<Join Chain> := <Join> <Join Chain> |
```

Grammar-based parsing functionality is built into the scanners technology and a configurable "parse" command brings semi-structured text parsing and information extraction into the XDTL data integration environment. As the result of the SQL parsing step (No2 in Figure 3.1), we have a large parse tree where every SQL query token has a special disambiguated meaning based on the grammar syntax.

Example 2. Parse tree fragment with grammar rules and parsed text tokens:

```
| +--<SelectStm>::=<Select>
|
+--<Select>::=SELECT<Columns><IntoClause><From><Where><GroupClause><QualifyClause><Ha
ving Clause> <Order Clause>
| | | +--SELECT
| | | +--<Columns>::=<Restriction><ColumnList>
| | | | +--<Restriction>::=
| | | | +--<ColumnList>::=<ColumnItem>','<ColumnList>
| | | | | +--<ColumnItem>::=<ColumnSource><Alias>
| | | | | | +--<ColumnSource>::=<ColumnSourceItem>
```

[36] http://www.goldparser.org/

```
| | | | | | +--<ColumnSourceItem>::=<AddExp>
| | | | | | | +--<AddExp>::=<Exp><Operator><AddExp>
| | | | | | | | +--<Exp>::=<Value>
| | | | | | | | | +--<Value>::=Id
| | | | | | | | | | +--MK.Kood
| | | | | | | | | +--<Operator>::='||'
| | | | | | | | | +--||
| | | | | | | | | +--<AddExp>::=<Exp><Operator><AddExp>
| | | | | | | | | +--<Exp> ::= <Value>
| | | | | | | | | | +--<Value>::=StringLiteral
| | | | | | | | | | | +--'/'
| | | | | | | | | +--<Operator> ::= '||'
| | | | | | | | | | +--||
```

To parse different texts into the tree structure and to be able reduce tokens and parse the tree back to meaningful expressions (depending on search goals), we use a declarative rule set (in JSON format) based on token and grammar rule combinations. Configurable grammar and a synchronized reduction rule set makes the XDTL parse command more suitable for general-purpose information extraction and it captures the resource-hungry computation steps into one single parse-and-map step with a flat table outcome. Parse Tree Mapper (No3 in Figure 3.1) uses 3 different rule sets with more than 100 rules to map the parse tree into data transformation expressions. The defined rules are declared in the following sets and are illustrated in Example 3:

- Stopword list and grammar rules are used to indicate the mapper to flush the buffer and start token collection to construct a new expression;
- Mapword list and grammar rules are used to map collected expressions to meaningful items (e.g., sources, targets, data transformations, joins and filters); and
- Tagword list and grammar rules are used to tag special meaningful tokens in expressions to identify all db objects references (e.g., tables, views, and columns, functions, constants etc.).

Example 3. Mapper rule set sample with sql query tokens and grammar rules:

```
{"parse-map":
 {"stopwords": [
     {"token":"SELECT", "rule": "<Select>"},
     {"token":"FROM", "rule": "<From>"},
     {"token":"WHERE", "rule": "<Where>"},
     {"token":"JOIN", "rule": "<Join>"},
     ... ],
 "mapwords":[
     {"map":"FilterCondition","token":"WHERE", "rule": "<Where>", "group": "0"},
     {"map":"JoinCondition","token":"ON", "rule": "<Join>", "group": "0"},
     {"map":"Source","token":"FROM", "rule": "<From>", "group": "0"},
     {"map":"Target","token":"INTO", "rule": "<Ins Prefix>", "group": "0"},
     {"map":"Transformation","token":",", "rule": "<Column List>", "group": "0"},
```

```
    ... ],
"tagwords":[
    {"token":"Id"},
    {"token":"IntegerLiteral"},
    {"token":"StringLiteral"},
    {"token":"Alias"},
    ... ]
}}
```

After extraction and mapping of each SQL query statement into a series of expressions, we execute the SQL Query Resolver (No4 in Figure 3.1) that contains a series of functions to resolve SQL query structure-specific tasks:

- Resolve source and target object aliases to full qualified (schema name + object name) object names;
- Resolve sub-query aliases to context-specific source and target object names;
- Resolve sub-query expressions and identify them to expand all query-level expressions and identifies to fully qualified and functional ones;
- Resolve syntactic dissymmetry in different data transformation expressions (e.g., insert statement column lists, select '*' statements, select statement column lists, and update statement assign lists, etc.); and
- Extract quantitative metrics from data transformation, filter and join expressions to calculate expression weights (e.g., number of columns in expression, functions, predicates, string constants, number constants etc.).

3.6. Data Transformation Weight Calculation

The problem of origin of data is often related with context, confidence and trustworthiness. We can find papers from literature that focused on mathematical models or algorithms to measure importance, certainty and trust in data processing systems [75] or beliefs, opinions and trust transitivity, propagation and reasoning in agents communication [76]. We notice some similarities in data source confidence, trust calculation and propagation, but our data lineage and impact weight calculation have different purpose. Our data transformation weight calculation is based on probabilistic estimation of data sources usage in data transformations and filtering, and the purpose is to make metadata-based inferences about the data flows and the data usage.

Data structure transformations are parsed and extracted from queries, and are stored as formalized, declarative mappings in the system (articles B and C). To add additional quantitative measures to each column transformation or column usage in join and filter conditions, we evaluate each expression and calculate transformation and filter weights for them.

The Expression Weight Calculation (No5 in Figure 3.1) was based on the idea that we can evaluate column data "transformation rate" and column data "filtering rate" using data structure and structure transformation information captured from the SQL queries. Such a heuristic evaluation allows for distinguishing columns and structures used in transformation expressions or in filtering conditions or

both, and gives probabilistic weights to expressions without the need to understand the full semantics of each expression. We defined two measures that we further used in our rule system for new facts calculation:

- The column transformation weight *Sw* is based on expression complexity estimation in column transformation and calculated weight expresses the source column transfer rate or strength. Weights are calculated in scale [0,1] where 0 means that data is not transformed from the source (e.g., constant assignment in query) and 1 means that the source is directly copied to the target (no additional column transformations).
- The column filter weight *Fp* is based on expression complexity estimation for each filter column in the filter expression and the calculated weight expresses the column filtering rate or strength. Weight is calculated in scale [0,1], where 0 means that the column is not used in the filter and 1 means that the column is directly used in the filter predicate (no additional expressions).

The general column weight *W* algorithm in each expression for *Sw* and *Fp* components are calculated as the column count ratio over all expression components counts (e.g., column count, constant count, function count, predicate count):

$$W = \frac{ColumnCount}{ColumnCount + FunctionCount + StringCount + NumberCount + PredicateCount}$$

All counts are normalized using the expression function list evaluation over the positive function list (e.g., CAST, ROUND, COALESCE, TRIM etc.). If the function in the expression is in the positive function list, then the normalization function reduces according to the component count by 1 to "pay a smaller price" when the used function does not a have significant impact on the column data.

When the column data is mapped from the source column to the target column in the SQL DML statement column expression, then the data transformation weight depends on the complexity of the expression and is between 0 and 1. The following expression samples and the calculated weights for each source-target column pair illustrate the variation of the data transformations:

```
q1: CAST(T1.LogDate AS DATE) as Request_Date                                  => 0.91
q2: T1.First_Name||' '||T1.Last_Name as Full_Name                             => 0.67
q3: MIN(T1.Balance_Amt) as Min_Balance_Amt                                    => 0.5
q4: SUM(ZEROIFNULL(T1.Payment_Amt)) as Sales_Amt                              => 0.33
q5: SUM(CASE T1.Acc_Type IN (2,42) THEN T1.Acc_Amt ELSE 0 END) as Credit_Amt  => 0.2
q6: CASE WHEN T1.Feature_Id is not null THEN 'Y' ELSE 'N' END as Dynamic_Ind  => 0.17
```

The last expression q6 contains parts and measures like *ColumnCount:* 1 (T1.Feature_Id), *FunctionCount:* 2 (Case,WhenThen) and *StringCount:* 3 (null,Y,N). Using those values and the weight definition we calculate the column pair operation `O(T1.Feature_Id,T2.Dynamic_Ind, q6, 0.17)` weight in the expression q6 like this:

$$W = \frac{1}{1 + 2 + 3 + 0 + 0} = \frac{1}{6} = 0.16667 \cong 0.17$$

3.7. Rule System and Dependency Calculation

The defined figures, operations and weights are used with the combinations of declarative inference rules with formal reasoning to calculate possible relations and dependencies between data structures and software components. Applying the rule system to the extracted query graph, we calculate and produce the lineage and impact graphs that are used for the data lineage or impact analysis.

First, we define the rule R_1 to map the column level primitive data transformations to the data lineage graph edges with the aggregation of multiple paths over pairs of nodes. Let $E_{x,y} = \{e \in E_O \mid e.X = x, e.Y = y\}$ be the set of edges connecting nodes *x*, *y* in the graph G_O. The data lineage graph G_L edges are calculated by rule R_1: $\forall x, y \in N\ E_{x,y} \neq \emptyset \implies \exists e' \in E_L$ with a set of properties:

- $e'.X = x \wedge e'.Y = y$
- $e'.M = \cup_{e \in E_{x,y}} e.M$
- $e'.W = max\{e.W \mid e \in E_{x,y}\}$

An inference of this rule should be understood as creating edges *e'* into the set E_L until R1 is satisfied.

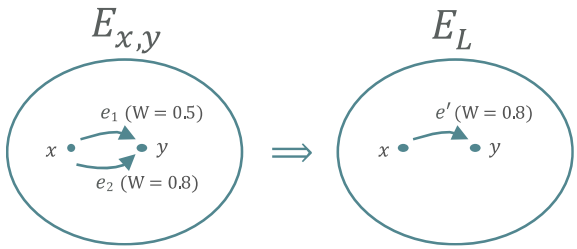


Figure 3.3 Visual representation of data lineage graph inference rule R_1 .

The filter conditions are mapped to edges in the impact graph G_I. Let $F_{M,p} = \{x \mid Parent(x,p) \wedge x\ is\ a\ filter\ in\ M\}$ be the set of nodes that are the filter conditions for the mapping *M* with parent *p in database schema*. Let $T_{M,p'} = \{x \mid Parent(x,p') \wedge x\ is\ target\ in\ M\}$ be the set of nodes that represent the target columns of mapping *M*. To assign filter weights to columns, we use the function $W_f: N \rightarrow [0,1]$. The data impact graph G_I edges are calculated by rule R_2: $\forall p, p' \in N\ F_{M,p} \neq \emptyset \wedge T_{M,p'} \neq \emptyset \implies \exists e' \in E_I$ with a set of properties:

- $e'.X = p \wedge e'.Y = p'$
- $e'.M = M$
- $e'.W = avg\{W_f(x) \mid x \in F_{M,p}\}$

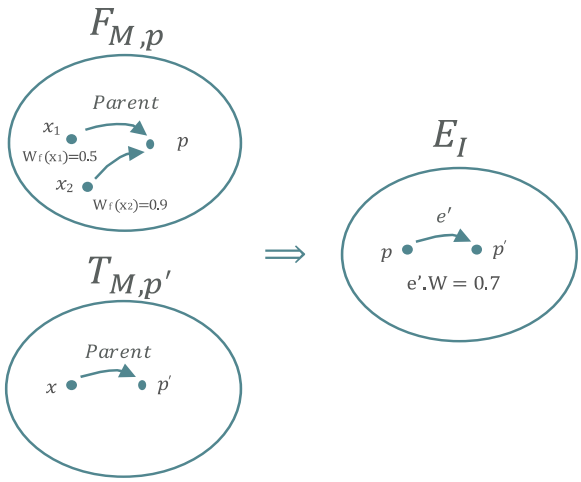


Figure 3.4 Visual representation of data impact graph inference rule R_2 .

To propagate information through the database structure upwards, to view the data flows on a more abstract level (such as table or schema level) or to calculate the dependency closure to answer lineage queries, we treat the graphs G_L and G_I similarly. Let $E_{p,p'} = \{e \in E \mid Parent(e.X, p) \wedge Parent(e.Y, p')\}$ be the set of edges where the source nodes share a common parent p and the target nodes share a common parent p'. The aggregation of the edges to the pair of common parents in the lineage G_L or impact graph G_I are calculated by rule R_3: $\forall p, p' \in N\ E_{p,p'} \neq \emptyset \implies \exists e' \in E$ with a set of properties:

- $e'.X = p \wedge e'.Y = p'$
- $e'.M = \cup_{e \in E_{p,p'}} e.M$
- $e'.W = \frac{\sum_{e \in E_{p,p'}} e.W}{|E_{p,p'}|}$

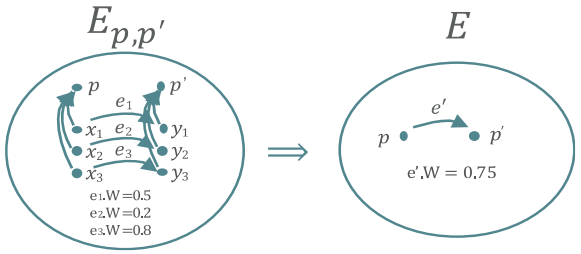


Figure 3.5 Visual representation of data lineage and impact graph inference rule R_3.

Based on the derived dependency graph, we can solve different business tasks by calculating selected component(s) lineage or impact over available layers and chosen details. Business questions like: “What reports are using my data...?”, “Which components should be changed or tested...?” or “What is the time and cost of change...?” will be turned to the directed sub-graph navigation and calculation tasks. We calculate new quantitative measures to each component or

node by number of sources and targets in the graph and we use those results in the UI to sort and select the correct components for specific tasks:

- Local lineage and impact dependency scores are calculated as ratio over sum of local source and target lineage or impact weights. Zero percent means that there are no data sources detected for the object and 100% means that there are no data consumers (targets) detected for the object. About 50% means that there are equal numbers of weighted sources and consumers (targets) detected for the object.
- Global lineage and impact dependency scores are calculated as sums of local dependency scores over connected sources and target chains for each node.

The local dependency calculation algorithm for each connected node is as follows:

$$LD = \frac{\Sigma\big(source(W)\big)}{\Sigma(source(W)) + \Sigma(target(W))}$$

More details about data transformation weight, node score calculations and rule systems are presented in articles B and C. Rule system improvements and current formulations are presented in article D.

3.8. Semantic Layer Calculation

The semantic layer is an additional visualization and specific filter set used to localize connected sub-graphs of the expected data flows for the selected node. All connected nodes and edges in the semantic layer share the overlapping filter predicate conditions or data production conditions that are extracted during the edge construction to indicate not only possible data flows (based on connections in the initial query graph), but only expected and probabilistic data flows.

The main idea of the semantic layer is to narrow down all possible and expected data flows over all connected graph nodes by cutting down unlikely or not-allowed connections in the graph, which is based on additional query filters and semantic interpretation of filters and calculated transformation expression weights. The semantic layer of the data lineage graph will hide irrelevant or highlight relevant graph nodes and edges (depending on user choice and interaction) that makes a distinction when underlying data structures are abstract enough and independent data flows store and use independent “horizontal” slices of data. The essence of semantic layers is to use available query and schema information to estimate the row-level data flows without additional row-level lineage information that is unavailable at the schema level, but is also expensive or impossible to collect at the row level.

The visualization of the semantically connected subgraph corresponding to the selected node is created by fetching the path nodes and the edges along those paths from the appropriate dependency graph (impact or lineage). Any nodes not included in the semantic layer are removed or visually muted (by changing their

color or opacity) and semantically connected subgraphs are returned or visualized in the UI.

The semantic layer calculation is based on the selected node filter set and calculated separately for back (predecessor) and forward (successors) direction using a similar recursive algorithm with a search of overlapping filter conditions.

The illustration of different semantics of connected data flows (see Figure 3.6) is based on previously presented example queries and lineage graphs (see Section 1.2). Tables ACCOUNT and LOAN data are integrated to one AGREEMENT table by queries 1 and 3 (see Table 1.1), which is feeding two separate tables, DEPOSIT_SUMMARY and LOAN_SUMMARY, with queries 2 and 4. This is a typical scenario in DW or OLAP environments and data models where dimension and fact tables are integrating data from different sources and various queries, reports, applications or data marts using that data for different purposes. Based only on database structures and query mappings, we can see how such hub tables are integrating all dimension or fact sources to the one's targets. In other words, we can see and visualize all possible data flows based on query mappings. To distinguish all possible data flows from actual flows based on query conditions and restrictions, we have to go deeper into query conditions analysis to track semantics of data flows.

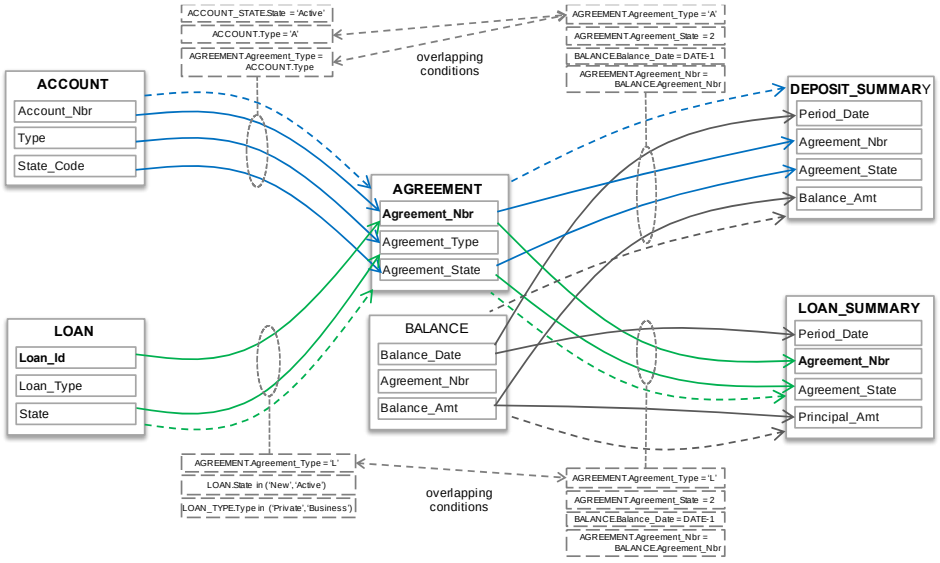


Figure 3.6 Semantic layer illustration for two independent data flows based on overlapping query conditions.

When comparing queries 1-4's mapping and filter predicate conditions, we can see the two separate data flows going to the AGREEMENT table and two separate flows moving out to the DEPOSIT_SUMMARY and LOAN_SUMMARY tables. The data in the AGREEMENT table has the same structure, but different sources and possibly different semantics. The intersection or overlap in query conditions allows us to notice separate slices of filtered

subsets in integrated structures, and such semantic analysis and matching of normalized query conditions allows us to make rule-based inferences about actual data flows. Queries 1 and 2 are dealing with the same data slice and are transforming it from the ACCOUNT to the DEPOSIT_SUMMARY table, and queries 3 and 4 are dealing with the same data slice and are transforming it from the LOAN to the LOAN_SUMMARY table. Those two different data flows in Figure 2.3 are marked with different colors (blue and green).

We can conclude the example by stating that to answer the data lineage questions more precisely we need to look into query semantics in addition to structural mappings. The semantic analysis of query conditions and recursive conditions overlapping search allows us to detect more likely data sources and flows than all possible sources and flows. We can make probabilistic decisions about row level (or set of rows) data flows using database and query metadata without interfering with the work of the actual system.

The details and recursive graph traversal algorithm descriptions of the semantic layer are published in paper D.

3.9. Summary

This system design chapter draws the high level methodological and technical overview of designed and implemented system components, their functions and the form. The system architecture follows the methodological pathway that is defined on a conceptual level in section 3.1. The metadata database design described in section 3.2 and different semantic models (metamodels) for databases, data integration, business intelligence and generalized mappings metadata were described in section 3.3. The metadata capture and scanners were described in section 3.4. More details about the underlying foundation and mapping design can be found in article A. A discussion of logical paths with query parsing and resolving techniques continued in section 3.5, with data transformations evaluation and weight calculation in section 3.6. The implemented rule system for graph construction and calculations was discussed in section 3.7 and the semantic layer on top of calculated graphs was discussed in section 3.8.

4. IMPLEMENTATION AND APPLICATIONS

This chapter presents an overview of the actual implementation along with real-life experiments and relevant statistics. The developed software components and applications are introduced in the section 4.1. A system performance evaluation based on six different real-life datasets and the performance overview details is presented in the section 4.2. Special attention has been given to the dataset visualization techniques presented in the section 4.3. Details of the visualization methods are published in papers C and D. Possible additional application areas are discussed in the section 4.4.

4.1. dLineage.com

The previously described architecture and algorithms have been used to implement the dLineage[37] toolset for data lineage and impact analysis in real organizations. dLineage is packaged as web-based software as a service (SaaS) or a local appliance, prepackaged and configured as a virtual machine (VM) with all the vital components included, such as scanners, parsers and calculation engine, metadata database and web-based user interface with multiple applications. The web-based tools are divided into different applications for different user groups:

- *The technical application* for metadata management, browsing and navigation to keep track of the source systems content and interconnection with all the available technical details.
- *The analytical application* for data lineage and impact analysis, data sources, targets and data flow visualizations.
- *The business applications* for technical metadata management with the help of a connected classification system, business glossary or ontology, and data or business governance with the help of domains, role system and responsibilities.

The scanners and web-based tools of dLineage have been extended and tested in real-life projects and environments to support several popular DW database platforms (e.g., Oracle, Greenplum, Teradata, Vertica, PostgreSQL, MsSQL, Sybase), ETL tools (e.g., Informatica, Pentaho, Oracle Data Integrator, SSIS, SQL scripts and different data loading utilities) and BI tools (e.g., SAP Business Objects, Microstrategy, Microsoft SSRS etc.). The dLineage database is built on PostgreSQL, using an open schema data modeling approach and predefined metamodels, described in sections 3.2 and 3.3. The rule system and dependency graph calculation is implemented in SQL queries and stored as a specialized relation between the scanned node objects. The current implementation uses recursive SQL for subgraph query tasks, which works reasonably well because of a local single object context and a sparse nature of the dependency graph. The number of objects in our test datasets (see section 4.2) were about 1.3 million and

[37] http://dlineage.com

we have tested the recursive SQL approach with three times bigger datasets without any remarkable drawbacks. We have also tested special storage and indexing methods and in-memory database approaches as alternatives for recursive SQL. The most promising approach would be the in-memory structures and algorithms for graph querying, which can be easily adapted and added as application components when needed. The algorithms for interactive transitive calculations and semantic layer calculation (see sections 3.7 and 3.8) are implemented in JavaScript and work in browsers for small and local subgraph optimization and visualization. Visualization of data lineage and impact flows is built using d3.js graphics libraries in combination with Sankey[38] diagram techniques. Additional information can be found on our dLineage[39] online demo site and more technical details are in article D.

The general idea of capturing and visualizing data flows in an organization DW ecosystem are drawn in Figure 4.1. The idea of visualizations using a Sankey diagram is to align all the data sources (e.g., files, interfaces or tables in source database) on the left side, all the final data consumers (ending targets, like reports, export files, API interfaces, etc.) on the right side and all other structures and components between them (depending on sources and targets). Figure 4.1. illustrates a traditional DW environment with several data transformation layers (e.g., source, staging, storage, access, and applications) using a small subset of Human Resource Management System (HRMS) data structures. The data structures and data are copied one-to-one from the source to DW and data transformations are built on the access view layer in this simplified example. Real-life DW environments are usually much more complex with different modeling paradigms (e.g., ODS, dimensional, 3NF or hybrid), which means there will be data restructuring and transformations almost in any layer or stage of data flow.

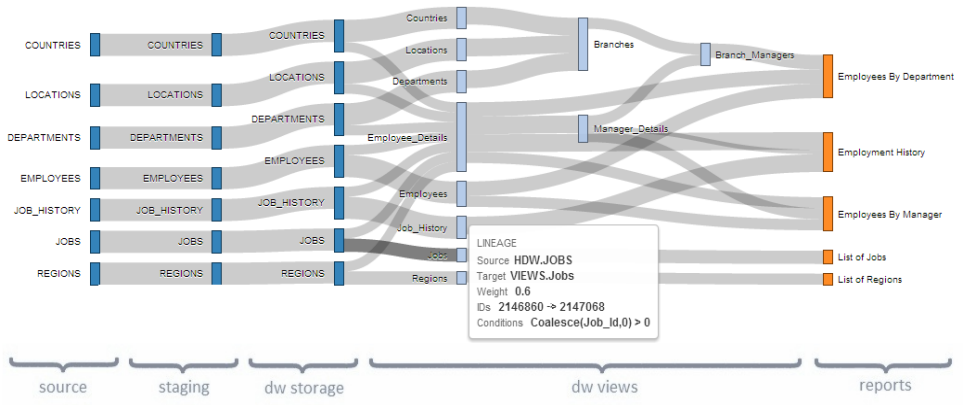


Figure 4.1 Data lineage visualization example in DW environment using Sankey diagram.

[38] https://en.wikipedia.org/wiki/Sankey_diagram

[39] http://www.dlineage.com/

The Analytics application in the dLineage toolset was designed for data lineage and impact graph navigation and visualizations. In the Analytics application, there are two built-in alternative data representation formats: table and graph view; and two complementary content representations: data lineage and impact view. The table view consists of two parts for each selected object: dependent sources and dependent targets, which represent the list of objects that are detected as a source or a target in-context of current focus. Figure 4.2 is an illustration of one report object in a financial reporting hierarchy with more than a hundred different sources (and no targets) that are connected to one report. The table view shows the data lineage or impact graph with calculated metrics (e.g., distance, number of queries, number of sources and targets) and is sorted by the most influential objects first.

The graph view in Figure 4.4 represents the same information about connected sources and targets using a clickable and zoomable Sankey diagram, but in contrast to the flattened table view, the graph view is stretched out from sources to targets and rendered from left to right with all levels and distances clearly visible.

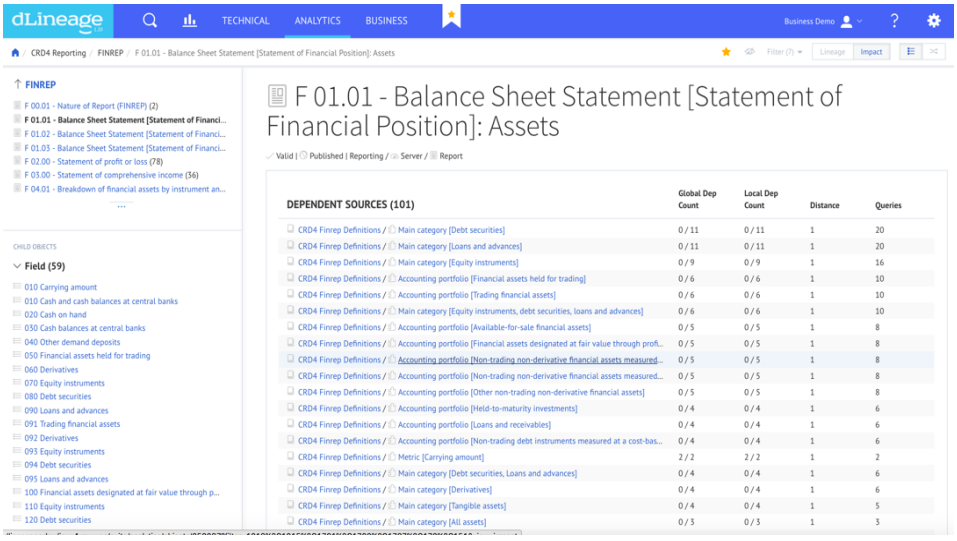


Figure 4.2 dLineage sub-graph table view, source and target objects with calculated metrics.

At the same time, the content filters for lineage and impact graphs based on graph calculation rules (see section 3.7) produces two different dependency relations: lineage (based on data transformation rules R_1, R_3 or R_3) and impact (based on data impact rules R_2 or R_3). Based on the lineage or impact content filters, the user can see and switch between a direct data lineage graph or a dependent component graph. The latter contains also impact graph data that used for data filtering, joining or coding and that do not contribute directly to target structures. In Figure 4.3 and Figure 4.4, one can see the impact view with two or three colored dependency lines, where direct data transformations are in gray and indirect impact dependencies are in red. Both

representations and content filters have their own aspect to emphasize and help in combinations, and together they perform the lineage or impact analysis tasks.

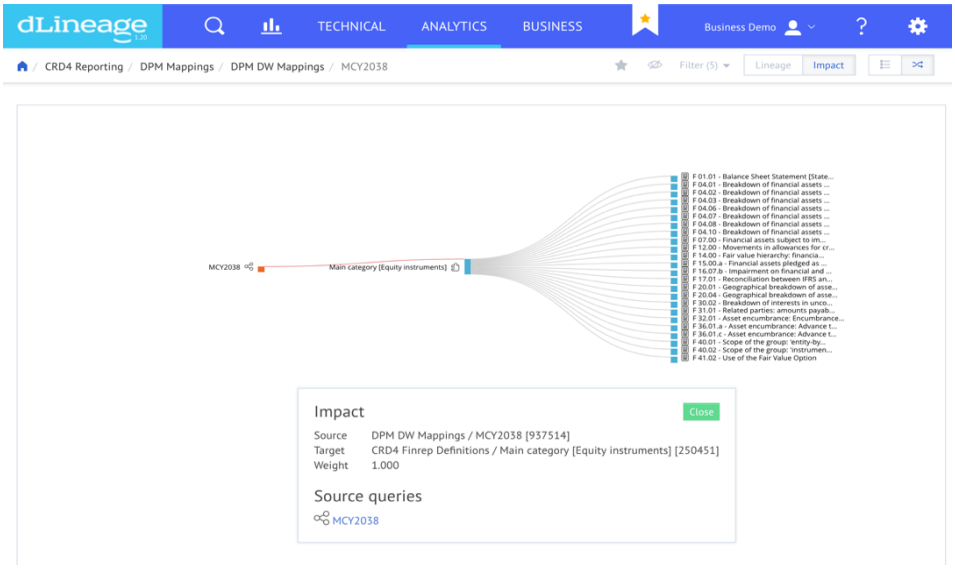


Figure 4.3 dLineage sub-graph graphical view, selected object with all connected targets.

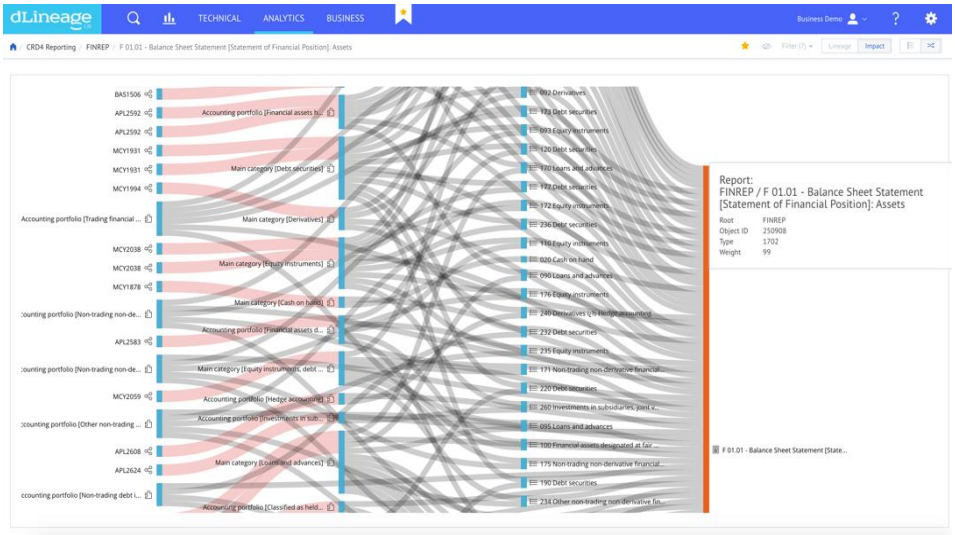


Figure 4.4 dLineage sub-graph graphical view, selected object with all connected sources.

The other applications in the dLineage toolset are built to support related activities to manage metadata scanners, browse and search collected data, manage systems state and heath, analyze discovered dependencies, manage and govern corporate information assets or collect and build business glossaries and definitions to give a

meaning to IT assets. In addition to technical, analytical and business applications, we collect and calculate various measures to estimate system health, integrity, graph connectivity, parse rate and errors, business coverage, errors, etc. Figure 4.5 illustrates the dashboard functionality of the dLineage toolset that visualizes collected measures and data.

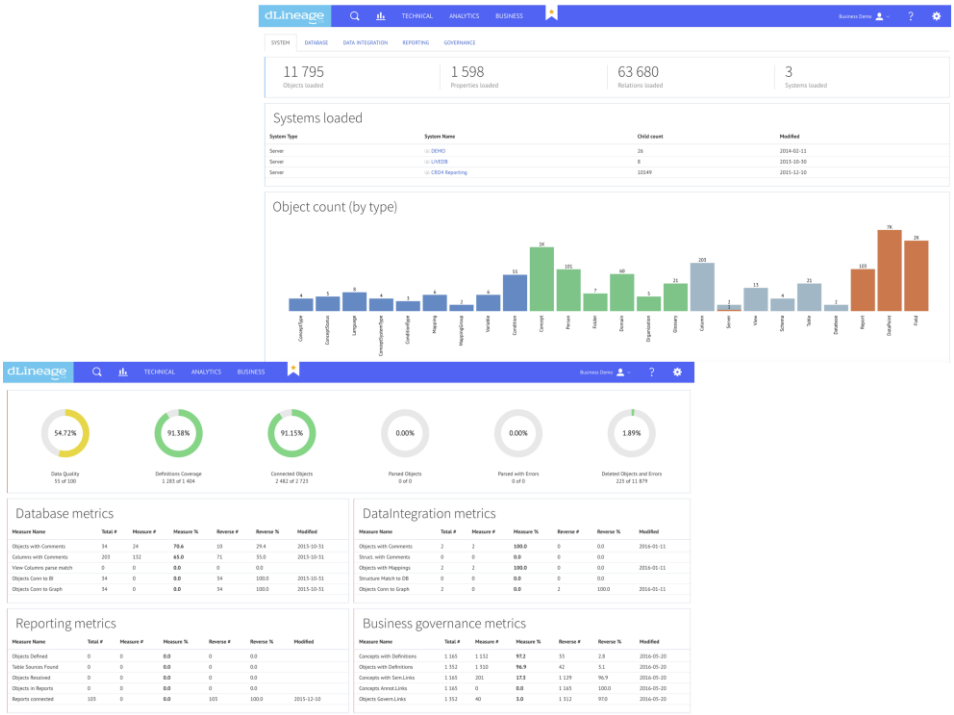


Figure 4.5 dLineage dashboard has aggregated overview about collected metadata and calculated results and metrics.

4.2. Performance Evaluation

We have tested our solution in several real-life case studies involving a thorough analysis of large international companies in the financial, utilities, governance, telecom and healthcare sectors. The case studies analyzed thousands of database tables and views, tens of thousands of data loading scripts and BI reports. Those figures are far over the capacity limits of human analysts not assisted by special tools and technologies.

The following six different datasets with varying sizes have been used for our system performance evaluation. The datasets DS1 to DS6 represent data warehouse and business intelligence data from different industry sectors and is aligned according to dataset size (Table 4.1). The structure of the datasets are diverse and complex, hence we have analyzed the results at a more abstract level (e.g., the number of objects and processing time) to evaluate the system performance under different conditions.

Table 4.1 Evaluation of processed datasets with different size and structure.

	DS1	DS2	DS3	DS4	DS5	DS6
Number of scanned objects	**1 341 863**	**673 071**	**132 588**	**120 239**	**26 026**	**2 369**
DB objects	43 773	179 365	132 054	120 239	26 026	2 324
ETL objects	1 298 090	361 438	534	0	0	45
BI objects	0	132 268	0	0	0	0
Scan time (min)	**114**	**41**	**17**	**33**	**6**	**0**
Number of scripts to parse	6 541	8 439	7 996	8 977	1184	495
Number of parsed query mappings	48 971	13 946	11 215	14 070	1544	635
Query parse success rate (%)	96	98	96	92	88	100
Query parse/resolve perf. (qry/sec)	3.6	2.5	26.0	12.1	4.1	6.3
Query parse/resolve time (min)	**30**	**57**	**5**	**12**	**5**	**1**
Number of graph nodes	73 350	192 404	24 878	17 930	360	1 930
Number of graph links	95 418	357 798	24 823	15 933	330	2 629
Graph processing time (min)	36	62	14	15	6	2
Total processing time (min)	**150**	**103**	**31**	**48**	**12**	**2**

The biggest dataset, DS1, contained a big set of Informatica ETL package files, a small set of connected DW database objects and no business intelligence data. The next dataset, DS2, contained a data warehouse, SQL scripts for ETL loadings and an SAP Business Object for reporting for business intelligence. The DS3 dataset contained a smaller subset of the DW database (MsSql), SSIS ETL loading packages and SSRS reporting for business intelligence. The DS4 dataset had a subset of the DW (Oracle) and data transformations in the stored procedures (Oracle). The DS5 dataset is similar but much smaller compared to DS4 and is based on the Oracle database and stored procedures. The DS6 dataset had a small subset of a data warehouse in Teradata and data loading scripts in the Teradata TPT format.

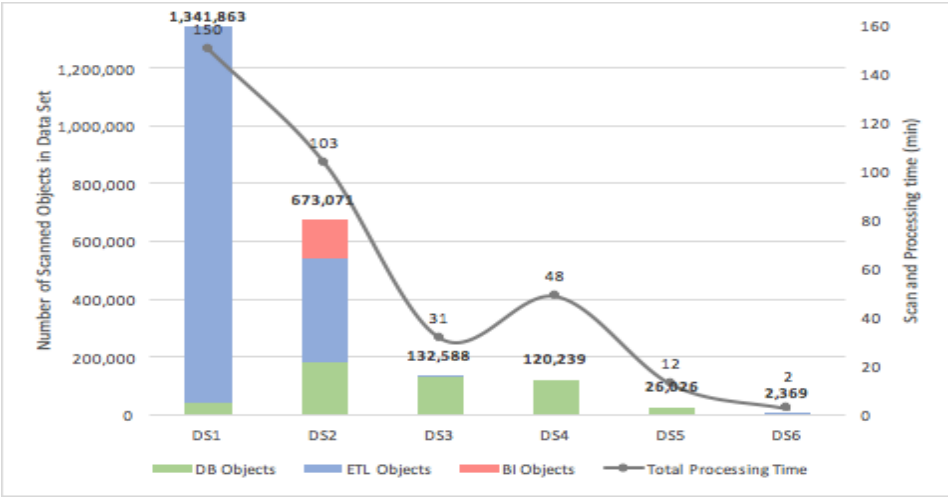


Figure 4.6 Datasets size and structure compared to overall processing time.

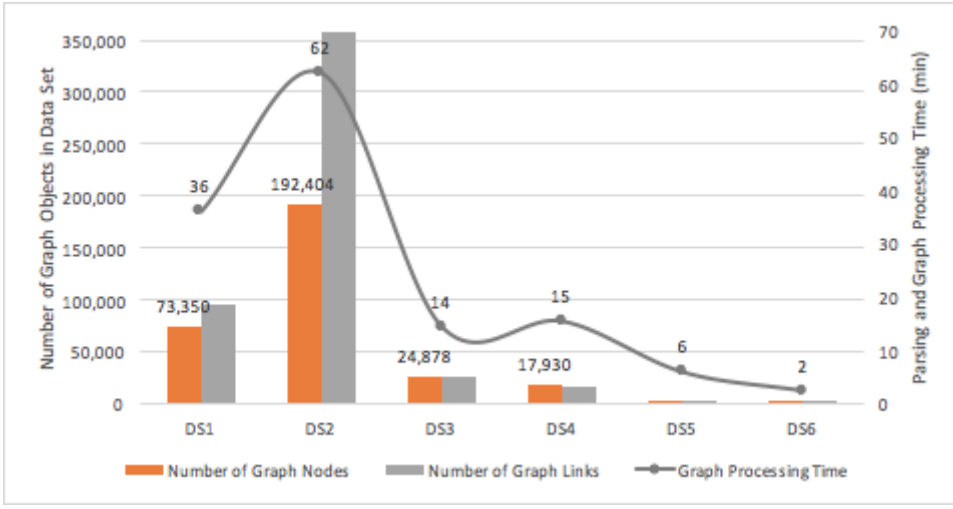


Figure 4.7 Calculated graph size and structure compared to graph data processing time.

The dataset sizes, internal structure and processing time are visible in Figure 4.6, where a longer processing time of DS4 is related to very large Oracle stored procedure texts and loading of those to the database. The initial dataset and the processed data dependency graphs have different graph structures (see Figure 4.7) that do not correspond necessarily to the initial dataset size. DS2 has a more integrated graph structure and a higher number of connected objects (Figure 4.7) than the DS1. At the same time, the DS1 initial row data size is about two times bigger than DS2.

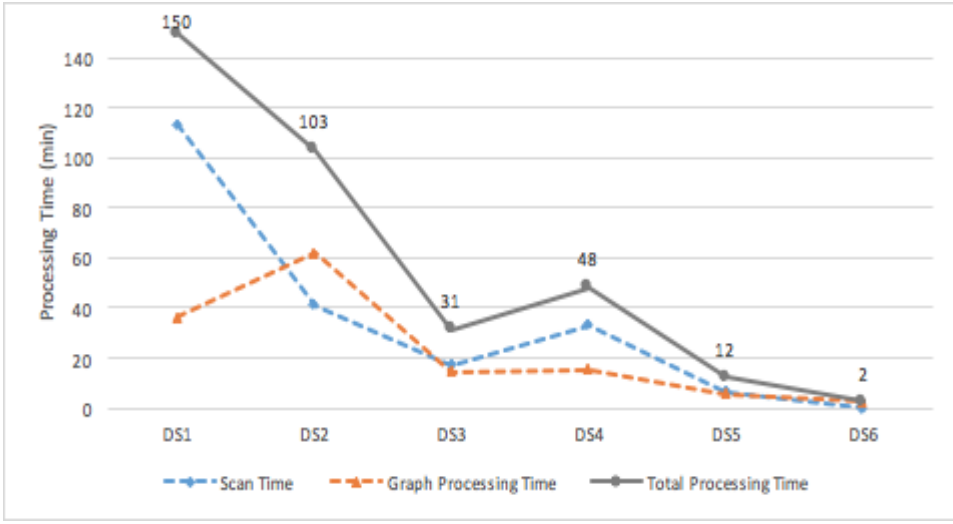


Figure 4.8 Dataset processing time with two main subcomponents.

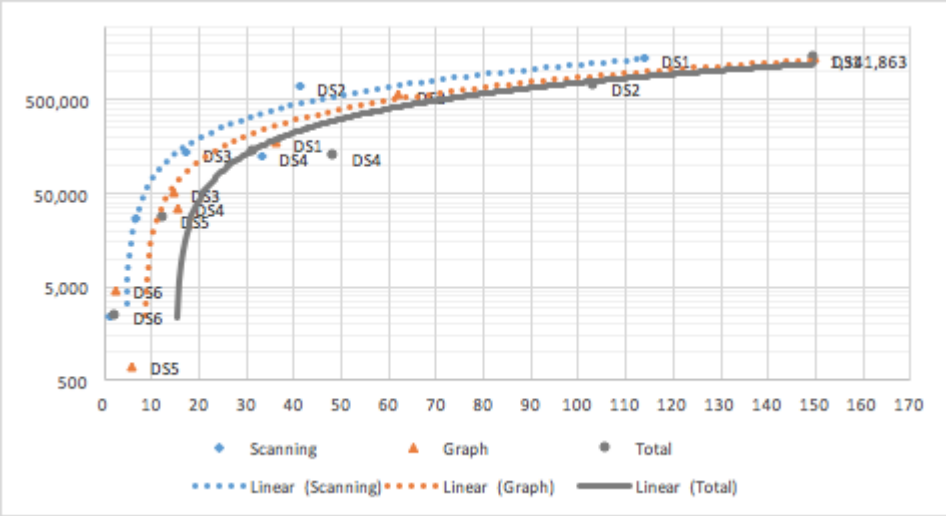


Figure 4.9 Dataset size and processing time correlation with linear regression (semi-log scale).

We have analyzed the correlation of the processing time and the dataset size (see Figure 4.8 and Figure 4.9) showing that the growth of the execution time follows the same linear trend as the size and complexity growth. The data scan time is related mostly to the initial dataset size. The query parsing, resolving and graph processing time also depend mainly on the initial data size, and less so on the calculated graph size (Figure 4.8). The linear correlation between the overall system processing time (seconds) and the dataset size (object count) can be seen in Figure 4.9.

4.3. Visualization

The Enterprise Dependency Graph examples (Figure 4.10 - Figure 4.12) illustrate the complex structure of dependencies between the DW storage scheme, access views and user reports. The examples were generated using data warehouse and business intelligence lineage layers. The details are at the database and reporting object level, not at the column level. At the column and the report field levels, a full data lineage graph would be about ten times bigger and too complex to visualize in a single picture. The following graph from the data warehouse structures and user reports presents about 50,000 nodes (tables, views, scripts, queries, reports) and about 200,000 links (data transformations in views and queries) on a single image (Figure 4.10).

The real-life dependency graph examples illustrate the automated data collection, parsing, resolving, graph calculation and visualization tasks implemented in our system. The system requires only the setup and configuration tasks to be performed manually. The rest will be done by the scanners, parsers and the calculation engine.

The final result consists of data flows and system component dependencies visualized in the navigable and drillable graph or table form. The results can be

viewed as a local sub-graph with a fixed focus and a suitable filter set to visualize the data lineage path from any source to a single report with click and zoom navigation features. The big picture of the dependency network gives a full-scale overview of the organization's data flows. It explicates potential architectural, performance and security problems.

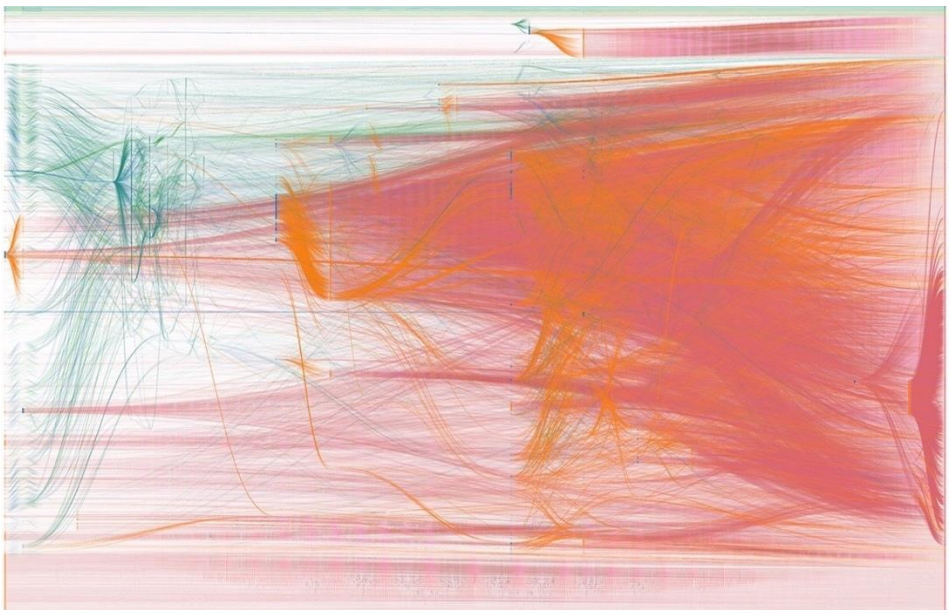


Figure 4.10 Data flows (blue,red) and control flows (green,yellow) between DW tables, views and reports.

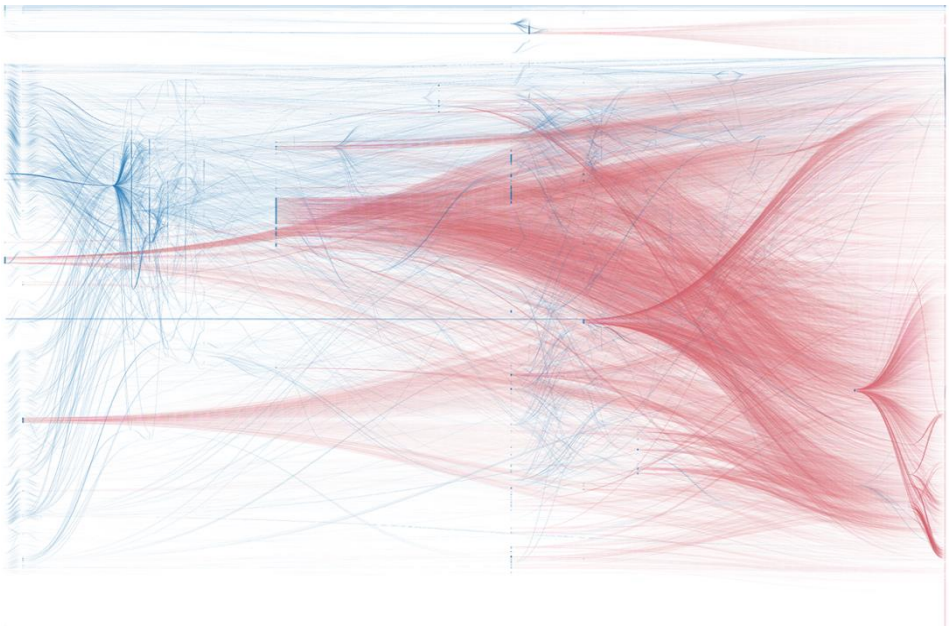


Figure 4.11 Data flows between DW tables, views (blue) and reports (red).

Figure 4.12 Control flows in scripts, queries (green) and reporting queries (yellow) are connecting DW tables, views and reports.

In addition to the visualization of data flows, we have developed the aggregated plot view of graph nodes that will help to analyze database tables, data loading programs or reports in terms of connectedness, complexity and cost. The main idea of the visualization is to draw a two-dimensional plot or bubble chart with a number of connected sources and targets on an X and Y axis that allow us to clearly distinguish more and less connected nodes and the balance between the number of sources and targets or data producers and consumers. The size of the bubble in the chart is a recursively calculated number of child objects that express the complexity of the object and its structure. The color of the bubble is calculated as a sum of all three components – the number of sources, targets and children – expressing the cost of the object in terms of change, development or maintenance.

The more costly objects are located in the upper right corner (see Figure 4.13 and Figure 4.14), with a bigger diameter and colored in red. The less costly objects are located in the lower left corner and colored in blue. The color layer is the fourth dimension of the chart, giving a quick aggregated overview of the selected object set. The bigger and more red an object is, the costlier and more complex it is to change. The smaller and more blue an object is, the less costly and less complex it is to change.

The data axis with its number of sources and targets and bubble size are calculated and drawn in a logarithmic scale. The number of sources, targets and child elements of each object in the same chart can vary with several orders of magnitude, and therefore the logarithmic scale is more suitable for visualization and reading of charts.

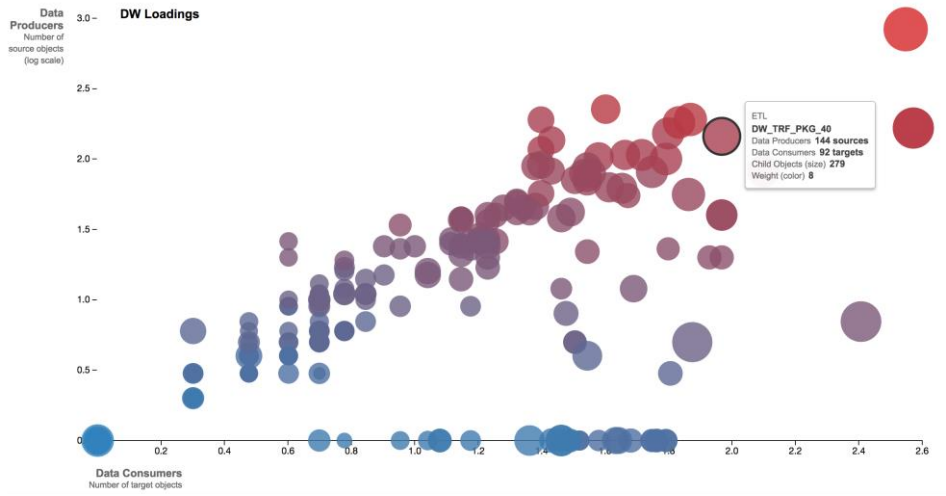


Figure 4.13 Data Warehouse loading packages plot with number of data sources and targets (axis), loading complexity (size) and relative cost (color).

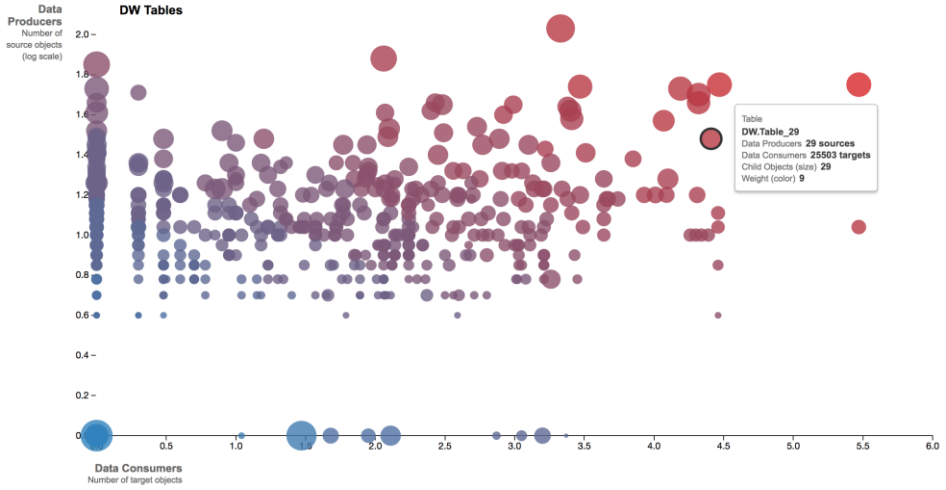


Figure 4.14 Data Warehouse tables plot with number of data sources and targets (axis), loading complexity (size) and relative cost (color).

4.4. Proposed Novel Applications

The previously described architecture and dLineage toolset allows us to address and solve different IT management tasks, based on evidence stored in the dependency graph. In the following section, we describe some practical use cases in addition to data lineage and impact analysis that can be seen as additional applications or plugins for the dLineage toolset.

Planning and Budgeting

The ETL programming is often the most time-consuming, complex and hard-to-predict task in enterprise DW projects, and depends on many variables: analysis and quality of source data, complexity of data mappings and transformations, design of target model, etc. Estimations and budgeting of such tasks are usually based on available input figures and expert opinions, and cannot be easily answered without previous analysis. Automation of these analysis tasks via replacement of expert opinions with traceable calculation and decision algorithms would save money and provide decision support for ETL planning and budgeting projects. We have successfully implemented and used the Excel-based calculation algorithm for ETL programming resources estimation (time and money) in several financial, retail and telecom sector DW projects that were based on available input figures (i.e., number of tables/columns to load, number of tables/columns to design/create/change/drop, number of views/column to design/create/change/drop, number of tasks and packages, etc.), customized weights and constants and calculation models that allowed us to validate and replace the human expert opinion and speed up planning tasks. Such a model, with a manually adjusted weight system for each individual organization, has the ability to imitate the average human expert decisions with accuracy over 90%.

When implementing a similar model on a real DW dependency graph and bringing the existing components with their sources and target object counts, weights and complexity measures, we can build a new evidence-based estimation calculator. Such an approach allows us to automate and speed up the project estimations and make it available via a web-based UI or wizard to end users such as project managers or business experts. The planning and budgeting app allows faster decisions assisted by connected content and might even outperform the average expert estimation because of additional knowledge captured into the dependency graph.

Automatic System Documentation

Relevant systems documentation is an important topic in IT systems development and is especially important in the context of DW development. A crucial part of DW documentation describes actual data mappings, transformations and loads with all the sources and targets. DW development and management can quickly become expensive and error-prone when detailed mappings and dependencies are not available. Design time mapping documents are usually not detailed enough and are outdated by the end of ETL design and programming. The lack of time, project setup and used tools often do not support the online documentation availability all the way to the end of the development phase. Automated documentation generation from actual data transformation programming code or ETL metadata would be the solution.

The toolset with DW systems and programs metadata scanning, parsing, resolving and storing in a unified metadata database is a good starting point for automated documentation. Unified data mappings and constructed dependency graphs consist of all the information required to generate detailed (column level)

ETL mapping documents. A web-based user interface allows for linked and living documentation that is accurate and more usable than traditional design time system documents.

Enterprise Search and IT Asset Management

The overview and management of corporate IT assets is a challenging topic for many organizations. IT systems are physically separated by design or security concerns. Integration of technical artefacts requires extra effort and tools. Different counterparties require the same data, but with different details and viewpoints highlighted, and there are not many tools to support them all from one source. IT architecture, maintenance, support, development and data delivery requirements are different and interested parties are rarely ready to find a common solution. Enterprise asset management with connected dependencies, business terminology, full text search, responsibilities and role systems would be the common solution for different needs.

The core functionality described provides metadata for IT systems which is organized in a suitable format to provide full-scale IT asset management functions. Built in google-like full-text makes every scanned object fast and easy to find. Business applications have functions to build up a full-scale business glossary system in top-down or bottom-up manner and additional role, domain and responsibility systems allows one to implement IT asset governance applications suitable for different needs throughout an organization.

Auditing and Compliance Reporting

Compliance with different internal and external requirements can be critical for many organizations and alignment of the requirements is time-consuming and costly. Specific industry sectors have their own requirement standards or mandatory governance regulation, and compliance with regulations will reduce the risks and business costs or allow the company to operate in the market. Compliance with regulations requires auditing or certification processes, and automation of data capturing, consolidation, measurement and alignment tasks allows for cost savings and quality improvements. The examples of such global regulations would be the Sarbanes-Oxley Act[40] for public and private companies in the US, which was designed to protect investors, competitors and companies themselves; Basel III[41] and Solvency II[42] in financial and insurance industries in the EU for capital requirements and risk regulations; the General Data Protection Regulation (GDPD)[43] directive from EU/EC for personal data usage and protection in online and internet businesses worldwide.

In order to fulfill regulations, we need to catalog the requirements in the form of business ontology and connect IT assets manually or automatically with the

[40] https://en.wikipedia.org/wiki/Sarbanes-Oxley_Act

[41] https://en.wikipedia.org/wiki/Basel_III

[42] https://en.wikipedia.org/wiki/Solvency_II_Directive_2009

[43] https://en.wikipedia.org/wiki/General_Data_Protection_Regulation

requirements. Depending on the specific regulations, we can build a logic-based rule system and connect it with an underlying dependency graph to derive data for requirements, to check internal logic and consistency of requirements and to provide solid, fact-based audit trail and proof of compliance.

4.5. Summary

This implementation chapter concludes the presentation of the designed and implemented software system, performance evaluation and datasets visualization. The developed software components and applications were introduced in section 4.1. System performance evaluation based on real-life datasets and the performance overview details were introduced in section 4.2, and the dataset visualization was presented in section 4.3. Finally, novel further application areas were discussed in section 4.4.

CONCLUSIONS

This thesis presents novel methods, algorithms and experimental results for practical data lineage and impact analysis. We are able to map, aid and automate the solution of management and analysis problems in a corporate data warehouse environment.

Automation of human intensive analysis tasks reduces time and costs, improves quality and leads to better decisions with reduced risks. It may take a week or two for a human analyst to solve moderately complex impact analysis tasks. We show that this time can be reduced to hours or minutes, with the interpretation of the results being feasible for users without the help of domain experts.

The traditional data lineage and impact analysis problems can be compared to the internet search problem before the invention of Google. The analyst of a new system component, functionality or business requirement had to find and read all the relevant documents and/or code bases to trace and model the data sources and dependencies. Our chosen approach to DW impact analysis and data lineage in a closed corporate environment can be compared to Google's approach to web scanning and indexing to build a sophisticated search engine. We scan, collect and map an organization's IT systems and data warehouse environment, data structures, queries, reports and programs, without using the DW data or affecting the normal work and behavior of those systems.

Processing and mapping the collected data to an RDF-style database schema creates a unified physical base for data storage. The unified data representation allows us to define and implement a set formalized rules to build weighted and directed dependency graphs. Probabilistic weight calculation in query parsing and weight propagation by the rule system brings the data transformation semantics to the graph for further usage. The weights are used for node dependency and transitivity calculations, for layer visualization, filtering and object sorting. The weight system is also used in the semantic layer calculation to visualize only the applicable data flow subgraphs for each selected node.

We have implemented all the algorithms described in the thesis and built a web-based dLineage software toolkit for browsing, analyzing and visualizing collected and calculated data. This toolset, algorithms and techniques have been successfully employed in tens of case studies and projects.

The presented case studies and performance analysis with six different real-life datasets demonstrates that our algorithms and implementations are linearly scalable.

We will continue our research and system development in the field of business semantics and governance automation to employ the underlying dependency graph in combination with semantic techniques and ontology learning. Combining different techniques to automate business definitions management and IT asset governance will hopefully allow us to fill another gap in the corporate knowledge and asset management landscape.

REFERENCES

[1] K. C. Viktor Mayer-Schönberger, "Big Data: A Revolution That Will Transform How We Live, Work, and Think.," John Murray, 2013, p. 242.

[2] S. B. Zdonik, "Provenance , Lineage , and Workflows," *Computer (Long. Beach. Calif).*, pp. 1–24, 2010.

[3] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM SIGMOD Rec.*, vol. 26, no. 1, pp. 65–74, 1997.

[4] P. Buneman, J. Cheney, W.-C. W. Tan, and S. Vansummeren, "Curated Databases," *Pod. June 9–12, 2008, Vancouver, BC, Canada.*, pp. 1–12, 2008.

[5] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in Databases: Why, How, and Where," *Found. Trends Databases*, vol. 1, no. 4, pp. 379–474, 2007.

[6] W. Tan, "Provenance in Databases : Past , Current , and Future," *Sigmod 2007*, pp. 1–10, 2007.

[7] P. Buneman, S. Khanna, and W.-C. Tan, "Why and where: A characterization of data provenance," *Int. Conf. Database Theory*, vol. 1973, no. January, pp. 316–330, 2001.

[8] Y. R. Wang and S. E. Madnick, "A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective," *Proc. 16th VLDB Conf.*, no. January, pp. 519–538, 1990.

[9] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," *VLDB J.*, vol. 12, no. 1, pp. 41–58, 2003.

[10] J. Widom, "Trio: A System for Integrated Management of Data, Accuracy, and Lineage," *Proc. 2005 CIDR Conf.*, pp. 262–276, 2005.

[11] A. Woodruff and M. Stonebraker, "Supporting fine-grained data lineage in a database visualization environment," *Data Eng. 1997. Proceedings. 13th Int. Conf.*, no. January, pp. 91–102, 1997.

[12] T. Priebe, A. Reisser, and D. T. Anh Hoang, "Reinventing the Wheel?! Why Harmonization and Reuse Fail in Complex Data Warehouse Environments and a Proposed Solution to the Problem," *Proc. 10th Int. Conf. Wirtschaftsinformatik*, pp. 766–775, 2011.

[13] Y. L. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in e-Science," *SIGMOD Rec.*, vol. 34, no. 3. pp. 31–36, 2005.

[14] S. B. Davidson and J. Freire, "Provenance and scientific workflows," *Proc. 2008 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '08*, p. 1345, 2008.

[15] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: a survey," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 1–28, 2005.

[16] P. Buneman and W. Tan, "Provenance in Databases," *Proc. 2007 ACM SIGMOD Int. Conf. Manag. data*, pp. 1171–1173, 2007.

[17] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the Lineage of View Data in a Warehousing Environment," *ACM Trans. Database Syst.*, vol. 25, no. 2, pp. 179–227, 2000.

[18] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings,"

Proc. twenty-sixth ACM SIGMOD-SIGACT-SIGART Symp. Princ. database Syst. - Pod. '07, vol. pages, no. June, p. 31, 2007.

[19] P. Buneman, S. Khanna, and W.-C. Tan, "On propagation of deletions and annotations through views," *Proc. twenty-first ACM SIGMOD-SIGACT-SIGART Symp. Princ. database Syst. - Pod. '02*, vol. 2002, no. June, p. 150, 2002.

[20] P. Buneman, J. Cheney, and S. Vansummeren, "On the expressiveness of implicit provenance in query and update languages," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006, vol. 4353 LNCS, pp. 209–223.

[21] D. Bhagwat, L. Chiticariu, W. C. Tan, and G. Vijayvargiya, "An annotation management system for relational databases," in *VLDB Journal*, 2005, vol. 14, no. 4, pp. 373–396.

[22] T. Green and G. Karvounarakis, "Update exchange with mappings and provenance," in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 675–686.

[23] D. Deutch, Y. Moskovitch, and V. Tannen, "A Provenance Framework for Data-Dependent Process Analysis," *Proc. VLDB Endow.*, vol. 7, no. 6, pp. 457–468, 2014.

[24] T. Heinis and G. Alonso, "Efficient lineage tracking for scientific workflows," *Proc. 2008 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '08*, no. Section 2, p. 1007, 2008.

[25] P. Missier, K. Belhajjame, J. Zhao, M. Roos, and C. Goble, "Data Lineage Model for Taverna Workflows with Lightweight Annotation Requirements," *Proven. Annot. Data Process.*, pp. 17–30.

[26] R. Ikeda, A. Das Sarma, and J. Widom, "Logical provenance in data-oriented workflows?," in *Proceedings - International Conference on Data Engineering*, 2013, pp. 877–888.

[27] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, 2001.

[28] O. Benjelloun, A. Das Sarma, C. Hayworth, and J. Widom, "An introduction to ULDBs and the Trio system," *IEEE Data Eng. Bull.*, vol. 29, no. 1, pp. 5–16, 2006.

[29] H. Fan and A. Poulovassilis, "Using AutoMed metadata in data warehousing environments," *Proc. 6th ACM Int. Work. Data Warehous. Ol. - Dol. '03*, p. 86, 2003.

[30] P. Giorgini, S. Rizzi, and M. Garzetti, "A goal-oriented approach to requirement analysis in data warehouses," *Decis. Support Syst.*, vol. 45, no. 1, pp. 4–21, 2008.

[31] H. Fan and A. Poulovassilis, "Using schema transformation pathways for data lineage tracing," *Knowl. Transform. Semant. Web*, vol. 3567, pp. 64–79, 2010.

[32] U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson, "Data integration flows for business intelligence," *Proc. 12th Int. Conf. Extending Database Technol. Adv. Database Technol. - EDBT '09*, p. 1,

2009.
[33] A. Simitsis and P. Vassiliadis, “A Methodology for the Conceptual Modeling of ETL Processes,” *CAiSE Work.*, pp. 305–316, 2003.
[34] A. Kabiri and D. Chiadmi, “A method for modelling and organazing ETL processes,” in *2nd International Conference on Innovative Computing Technology, INTECH 2012*, 2012, pp. 138–143.
[35] D. Skoutas and A. Simitsis, “Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data,” *Int. J. Semant. Web Inf. Syst.*, vol. 3, pp. 1–24, 2007.
[36] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita, “Improving Data Cleaning Quality using a Data Lineage Facility,” in *DMDW*, 2001.
[37] A. S. DeSantana and A. M. de C. Moura, “Metadata to Support Transformations and Data & Metadata Lineage in a Warehousing Environment,” in *Data Warehousing and Knowledge Discovery*, 2004, vol. 3181, no. 6th International Conference, DaWaK 2004, Zaragoza, Spain, September 1-3, 2004. Proceedings, pp. 249–258.
[38] M. Bala, O. Boussaid, and Z. Alimazighi, “Extracting-Transforming-Loading Modeling Approach for Big Data Analytics,” *Int. J. Decis. Support Syst. Technol.*, vol. 8, no. 4, pp. 50–69, 2016.
[39] S. K. Bansal, “Towards a Semantic Extract-Transform-Load (ETL) framework for big data integration,” in *Proceedings - 2014 IEEE International Congress on Big Data, BigData Congress 2014*, 2014, pp. 522–529.
[40] J. Wang, D. Crawl, S. Purawat, M. Nguyen, and I. Altintas, “Big data provenance: Challenges, state of the art and opportunities,” in *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, 2015, pp. 2509–2516.
[41] C. H. Suen, R. K. L. Ko, Y. S. Tan, P. Jagadpramana, and B. S. Lee, “S2Logger: End-to-end data tracking mechanism for cloud data provenance,” in *Proceedings - 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013*, 2013.
[42] B. Glavic and K. Dittrich, “Data provenance: A categorization of existing approaches,” *Btw*, pp. 227–241, 2007.
[43] S. Davidson and J. Freire, “Provenance and scientific workflows: challenges and opportunities,” *Proc. 2008 ACM SIGMOD*, pp. 1–6, 2008.
[44] R. Bose, “A conceptual framework for composing and managing scientific data lineage,” in *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, 2002, vol. 2002–Janua, pp. 15–19.
[45] Y. L. Simmhan, B. Plale, D. Gannon, and S. Marru, “Performance Evaluation of the Karma Provenance Framework for Scientific Workflows,” in *Proceedings of the 2006 International Conference on Provenance and Annotation of Data*, 2006, pp. 222–236.
[46] I. Altintas, O. Barney, and E. Jaeger-frank, “Provenance Collection

Support in the Kepler Scientific Workflow System," *Work*, vol. 4145, pp. 118–132, 2006.

[47] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *J. Netw. Comput. Appl.*, vol. 23, no. 3, pp. 187–200, 2001.

[48] E. Wu, S. Madden, and M. Stonebraker, "SubZero: A fine-grained lineage system for scientific databases," in *Proceedings - International Conference on Data Engineering*, 2013, pp. 865–876.

[49] P. Missier, B. Ludäscher, S. Bowers, S. Dey, A. Sarkar, B. Shrestha, I. Altintas, M. K. Anand, and C. Goble, "Linking multiple workflow provenance traces for interoperable collaborative science," in *2010 5th Workshop on Workflows in Support of Large-Scale Science, WORKS 2010*, 2010.

[50] I. Altintas, *Collaborative Provenance for Workflow-Driven Science and Engineering*, vol. 129. 2011.

[51] S. da Cruz, C. Paulino, and D. de Oliveira, "Capturing distributed provenance metadata from cloud-based scientific workflows," *J. Inf. Data Manag.*, vol. 2, no. 1, pp. 43–50, 2011.

[52] A. Marinho, C. Werner, S. M. S. Da Cruz, M. Mattoso, V. Braganholo, and L. Murta, "A strategy for provenance gathering in distributed scientific workflows," in *SERVICES 2009 - 5th 2009 World Congress on Services*, 2009, no. PART 1, pp. 344–347.

[53] L. Wang, S. Lu, X. Fei, A. Chebotko, H. Victoria Bryant, and J. L. Ram, "Atomicity and provenance support for pipelined scientific workflows," *Futur. Gener. Comput. Syst.*, vol. 25, no. 5, pp. 568–576, 2009.

[54] M. K. Anand, S. Bowers, I. Altintas, and B. Ludäscher, "Approaches for exploring and querying scientific workflow provenance graphs," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6378 LNCS, pp. 17–26.

[55] M. K. Anand, S. Bowers, and B. Ludäscher, "A navigation model for exploring scientific workflow provenance graphs," in *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, WORKS '09, in Conjunction with SC 2009*, 2009.

[56] U. Acar, P. Buneman, J. Cheney, J. Van Den Bussche, N. Kwasnikowska, and S. Vansummeren, "A graph model of data and workflow provenance," *Procs. TAPP'10 Work. (Theory Pract. Provenance)*, p. 8, 2010.

[57] O. Biton, S. Cohen-Boulakia, S. B. Davidson, and C. S. Hara, "Querying and managing provenance through user views in scientific workflows," in *Proceedings - International Conference on Data Engineering*, 2008, pp. 1072–1081.

[58] S. Bowers and B. Ludascher, "An ontology-driven framework for data transformation in scientific workflows," *Data Integr. Life Sci. Proc.*, vol. 2994, pp. 1–16, 2004.

[59] J. Kim, Y. Gil, and V. Ratnakar, "Semantic Metadata Generation for

Large Scientific Workflows," in *Proceedings of the Fifth International Semantic Web Conference*, 2006, pp. 357–370.

[60] L. Ding, J. Michaelis, J. McCusker, and D. L. McGuinness, "Linked provenance data: A semantic Web-based approach to interoperable workflow traces," in *Future Generation Computer Systems*, 2011, vol. 27, no. 6, pp. 797–805.

[61] S. S. Sahoo, A. Sheth, and C. Henson, "Semantic provenance for eScience: Managing the deluge of scientific data," *IEEE Internet Comput.*, vol. 12, no. 4, pp. 46–54, 2008.

[62] S. Bowers, T. Mcphillips, B. Ludascher, S. Cohen, S. B. Davidson, and B. Ludäscher, "A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows," *Lect. Notes Comput. Sci.*, vol. 4145, no. 4145, pp. 133–147, 2006.

[63] L. Finlay, "'Outing' the researcher: the provenance, process, and practice of reflexivity.," *Qual. Health Res.*, vol. 12, no. 4, pp. 531–545, 2002.

[64] R. de Paula, M. Holanda, L. S. A. Gomes, S. Lifschitz, and M. E. M. T. Walter, "Provenance in bioinformatics workflows.," *BMC Bioinformatics*, vol. 14 Suppl 1, no. Suppl 11, p. S6, 2013.

[65] P. Buneman, A. Chapman, and J. Cheney, "Provenance management in curated databases," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, 2006, pp. 539–550.

[66] R. De Paula, M. T. Holanda, M. E. M. T. Walter, and S. Lifschitz, "Managing data provenance in genome project workflows," in *Proceedings - 2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops, BIBMW 2012*, 2012.

[67] F. Chirigati and J. Freire, "Towards integrating workflow and database provenance," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012, vol. 7525 LNCS, pp. 11–23.

[68] O. Hartig and J. Zhao, "Publishing and consuming provenance metadata on the web of linked data," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6378 LNCS, pp. 78–90.

[69] L. Moreau, "The Foundations for Provenance on the Web," *Found. Trends Web Sci.*, vol. 2, no. 2–3, pp. 99–241, 2010.

[70] L. Moreau, "Provenance-based reproducibility in the semantic web," *J. Web Semant.*, vol. 9, no. 2, pp. 202–221, 2011.

[71] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson, "The Open Provenance Model," *Futur. Gener. Comput. Syst.*, vol. 27, no. 6, pp. 743–756, 2011.

[72] M. K. Anand, S. Bowers, and B. Ludäscher, "Techniques for efficiently querying scientific workflow provenance graphs," pp. 287–298, 2010.

[73] T. Stöhr, R. Müller, and E. Rahm, "An integrative and uniform model for metadata management in data warehousing environments," in *Proceedings of the International Workshop on Design and Management*

of Data Warehouses DMDW99, 1999, vol. 1999, pp. 1–16.

[74] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos, “A generic and customizable framework for the design of ETL scenarios,” *Inf. Syst.*, vol. 30, no. 7, pp. 492–525, Nov. 2005.

[75] M. Jäger, T. N. Phan, C. Huber, and J. Küng, “Incorporating Trust, Certainty and Importance of Information into Knowledge Processing Systems -- An Approach,” in *Future Data and Security Engineering: Third International Conference, FDSE 2016, Can Tho City, Vietnam, November 23-25, 2016, Proceedings*, T. K. Dang, R. Wagner, J. Küng, N. Thoai, M. Takizawa, and E. Neuhold, Eds. Cham: Springer International Publishing, 2016, pp. 3–19.

[76] A. Jøsang, S. Marsh, and S. Pope, “Exploring Different Types of Trust Propagation,” in *Trust Management*, 2006, vol. 3986, no. May, pp. 179–192.

KOKKUVÕTE

Käesoleva doktoritöö teema on andmevoogude ja neid realiseerivate komponentide analüüs ning selle protsessi automatiseerimine ettevõtte andmelao keskkonnas. Töö eesmärgiks on luua universaalne metoodika, algoritmid ja tarkvaraline lahendus, mida saab vähese vaevaga rakendada juba olemasoleva keskkonna andmevoogude ja mõjuanalüüsi automatiseerimiseks. Metoodilise lähenemise aluspõhimõteteks on töötava andmelao keskkonna kaardistamine selle tööd mõjutamata ning andmelao süsteemides töödeldavaid andmeid kasutamata. Selline lähenemisviis eeldab andmelao struktuuride, programmide ja raportite metaandmete kogumist ja töötlust ning võimaldab lahendust rakendada võimalikult väikeste kulutustega juba töötavas keskkonnas, selle tööd mõjutamata ja tundlikke andmeid vajamata.

Loodud süsteemi arhitektuur sisaldab dünaamilise ja paindliku struktuuriga andmebaasi kasutamist erinevate metaandmete salvestamiseks, modulaarsetel ja korduvakasutatavatel komponentidel baseeruvat metaandmete kogumis- ning töötlusprogrammide loomist ning veebipõhiseid rakendusi erinevatele kasutajagruppidele, analüüsi teostamiseks ja andmete visualiseerimiseks. Kirjeldatud semantilised meetodid ning reeglipõhine ja tõenäosuslik järeldussüsteem aitavad konstrueerida struktuuride ja programmide sisendite-väljundite baasil suunatud graafi, mis võimaldab andmestruktuuride ja -voogude analüüsiülesanded teisendada alamgraafide läbimise- ja arvutusülesanneteks.

Töös kirjeldatud tarkvara on testitud kümnete rahvusvaheliste ettevõttete andmeladude analüüsiks ja visualiseerimiseks. Ülevaade andmekogudest ning süsteemi jõudlusest on toodud töö viimases peatükis. Kokkuvõttes näitame, et valitud süsteemi arhitektuur, algoritmid ja meetodid on sobivad väga erinevate valdkondade, suuruse ja sisuga andmeladude analüüsiks metaandmete baasil ning kirjeldatud süsteemi komponentide jõudlus skaleerub lineaarselt lähteandmete mahuga.

Publication A

Tomingas, K.; Kliimask, M.; Tammet, T. Data Integration Patterns for Data Warehouse Automation. In: New Trends in Database and Information Systems II: 18th East European Conference on Advances in Databases and Information Systems (ADBIS 2014). Springer, 2014.

Data Integration Patterns for Data Warehouse Automation

Kalle Tomingas[a], Margus Kliimask[b] and Tanel Tammet[a]

[a] *Tallinn University of Technology, Ehitajate tee 5, Tallinn 19086 Estonia*
[b] *Eliko Competence Center, Teaduspargi 6/2, Tallinn 12618 Estonia*

Abstract. The paper presents a mapping-based and metadata-driven modular data transformation framework designed to solve extract-transform-load (ETL) automation, impact analysis, data quality and integration problems in data warehouse environments. We introduce a declarative mapping formalization technique, an abstract expression pattern concept and a related template engine technology for flexible ETL code generation and execution. The feasibility and efficiency of the approach is demonstrated on the pattern detection and data lineage analysis case studies using large real life SQL corpuses.

Keywords. data warehouse, etl, data mappings, template based sql generation, abstract syntax patterns, metadata management.

1. Introduction

The delivery of a successful Data Warehouse (DW) project in a heterogeneous landscape of various data sources, limited resources, and lack of requirements, an unstable focus and budgeting constraints is always challenging and risky. Many long-term DW project failures are related to the requirements and reality mismatch between available data, defined needs and information requirements for decision making [5]. Extract, transform and load (ETL) is a database usage process widely used in the data warehouse field. ETL involves extracting data from outside sources, transforming it to fit operational needs and loading it into the end target: a database or a Data Warehouse.

Mappings between source and target data structures or schemas are the basic specifications of data transformations. Mappings can be viewed as metadata capturing the relationships between information sources and targets. Mappings document the decisions for information structuring and modeling [12]. They are used for several different goals in DW processes: writing a specification for ETL programmers, generating a transformation query or program that uses the semantics of the mapping specification (e.g., a SQL query that populates target tables from source tables), providing metadata about relationships between structures or schemas, providing metadata about data flows and origin sources [4].

Programming mappings in the ETL environment involves writing special database loading scripts (e.g. Oracle Sql*Loader, MSSQL Bulkload, Teradata Fastload, Postgresql Copy etc.) and SQL queries (i.e. select, insert, update and delete statements) which are incremental, iterative, time consuming and routine activities. The manual programming of the data loadings is test-and-error based and not too efficient in case there is no support from the environment and no methodology. Manual scripting and coding of SQL gives high flexibility but backfires in terms of efficiency, complexity, reusability and maintenance of data loadings [7]. The execution and optimization of existing loading programs can be a very complex and challenging task without access to the full dependencies and intelligent machinery to generate optimized workflows [3], [1].

The processes of creation, integration, management, change, reuse, and discovery of data integration programs are not especially efficient without the dependencies and

semantics of data structures, mappings and data flows. The creation and management of human- and machine readable documentation, impact analysis (IA) and data lineage (DL) capabilities has become critical for maintaining complex sequences of data transformations. We can control the risks and reduce the costs of dynamic DW processes by making the data flows and dependencies available to developers, managers and end-users.

The paper describes a methodology for formalizing data transformations to an extent that allows us to decouple unique mapping instances from reusable transformation patterns. We demonstrate handling and storing declarative column mappings join and filter predicates in a reusable expression pattern form. We use the Apache Velocity template engine and predefined scenario templates to handle reusable procedural parts of data transformations. We show how the combination of those techniques allows us to effectively construct executable SQL queries and generate utility loading scripts. We also take a look at what has been previously done in the ETL and DW automation field.

In the third chapter we present our open-source architecture of knowledge and metadata repository (MMX[1]) with the related data transformation language and runtime environment (XDTL[2]) which is used as the technology stack for our metadata-driven Data Warehousing process. In the fourth chapter we describe the decoupling of procedural and declarative parts of data mappings and the template-based SQL construction technique. We introduce a case study of Abstract Syntax Pattern (ASP) discovery from a real life DW environment in the fifth chapter and two data lineage analysis case studies in the sixth chapter.

2. Related Work

The roles and functions of general programming and ETL tools as well as the relations between manual scripting and script generation are discussed in [7]. The first generation ETL tools were similar to procedural programming or scripting tools, allowing a user to program specific data transformations. The concept of mapping was used for initial specification purposes only.

Modern ETL tools - e.g. Informatica PowerCentre, IBM WebSphere DataStage or Oracle Data Integrator - exploit the internal mapping structure for transformation design and script or query generation purposes. In addition to specific ETL technologies there exist the general purpose schema mapping tools that allow discovery and support documenting the transformations or generating transformation scripts (XSLT transformation between XML-schemas, XQuery or SQL DML statements etc.). The meaning and purpose of mappings and general application areas, tools and technologies is discussed by Roth et al. with the generic usage scenarios in different enterprise architecture environments [12]. The declarative mappings designed for ETL program generation and vice versa are discussed in [4] and [6].

In addition to mappings with program generation instructions and data transformation semantics, there exist relations and dependencies between mappings and source or target schema objects. The declarative representation of the dependencies allows us to generate, optimize and execute data transformations workflows effectively. We can find different optimization approaches described in papers [1],[2] and [3]. An extensive study of common models for ETL and ETL job optimization is published by Simitsis et al. [13], [14] and Patil et al. [9]. The dynamic changes of data structures,

[1] www.mmxframework.org
[2] www.xdtl.org

connections between mapping and jobs and a rule-based ETL graph optimization approach is discussed in [8].

An effective ETL job optimization is related to data mappings, dependencies between mappings, dynamics and changes of source structures and the data quality (DQ). All of those aspects can be formalized and taken into account by ETL job automation where timing, the right order and data volumes are always important issues. Estimation and evaluation of data structures, quality of data and discovery of rules can be automated and integrated into ETL processes [11]. By adding rule-based DQ into ETL process, we can automate the mapping generation and improve the success rate of data loadings. Rodiç et al. demonstrated that most of the integration rules can be generated automatically using the source and target schema descriptions [11].

3. System Architecture

The modern enterprise data transformation systems are built according to the model-driven architecture principles, including internal metadata about the source and target models, mappings, transformations and dependencies between models. We introduce a new architectural concept, based on open source java and xml technologies that can be used in lightweight scripting configuration, mixed configuration with partial mapping formalization and full model- and metadata driven knowledge base implementations. When the first lightweight configuration gives you a quick start, low-cost and small technology track and metadata-driven approach gives a knowledge base with different new possibilities (e.g. mapping generation and management, dynamic dependency management and job automation, impact analysis, data quality integration etc.), then both exploit the template-based code construction and automation principles. Our design goals of the new ETL architecture were an open and flexible environment, extensible and reusable programming techniques with moderate formalization and decoupling of declarative knowledge from procedural parts of executable code.

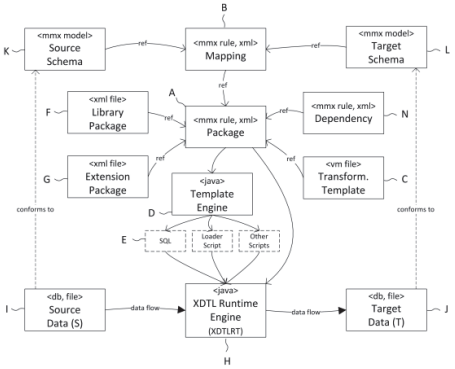


Figure 1. System architecture components.

The general system architecture with main building blocks is drawn in Figure 1. The main system components are ETL Package (A) that can be written in XDTL language or represented as Tasks and Rules and Dependencies (N) in MMX repository. The mapping (B) is a formalized representation of source schema objects (K), target (L), column transformations and patterns, join and filter conditions that can be again be a part of an XDTL package or stored in MMX repository. The transformation Template (C) is a reusable and repeating part of SQL query patterns or some other scripting

executable language scenarios that are written in the Apache Velocity macro language (or any other language of template engine). The template Engine (D) is a configurable java code that is responsible for runtime code construction using Mappings (B) and Templates (C). Examples of mappings and templates are discussed in more detail level in Chapter 4. The tasks in Package (A) can be created using previously prepared Library Packages (F) or managed modularly, reused and published as Extension (G) modules. The XDTL Runtime Engine (H) is a preconfigured environment and java package, able to interpret packages written in the XDTL language, execute those and deliver actual data transformations from the source (I) to the target (J).

3.1. Data Transformation Language (XDTL)

The Extensible Data Transformation Language (XDTL) is an XML based descriptive language designed for specifying data transformations between different data formats, locations and storage mechanisms. XDTL is created as a Domain Specific Language (DSL) for the ETL domain and is designed by focusing on the following principles: modular and extensible, re-usable, decoupled declarative (unique) and procedural (repeated) patterns. The XDTL syntax is defined in an XML Schema document. The wildcard elements of an XML Schema enable extending the syntax of core language with a new functionality implemented in other programming languages or in XDTL itself. The XDTL scripts are built as reusable components with the clearly defined interfaces via parameter sets. The components can be serialized and deserilized between the XML and database representations, thus making XDTL scripts suitable for storing and managing in a data repository. XDTL provides the functionality to use data mappings stored independently of the scripts, being efficiently decoupled from the scripts. Therefore the mappings stored in a repository can exist as objects independent from the transformation process and be reused by several different processes. XDTL acts as a container for a process that often has to use facilities not present in XDTL itself (e.g. SQL, SAS language etc.).

3.2. Knowledge Repository Structure (MMX)

The MMX metadata framework is a general purpose integrative metadata repository built on the relational database technology for different knowledge management (KM) and rule-based analytical applications. The MMX repository is designed according to the OMG Metadata Object Facility (MOF) idea with separate abstraction and modeling layers (M0-M3). The MMX physical data model (schema) is based on principles and guidelines of EAV (Entity-Attribute-Value) or EAV/CR (Entity-Attribute-Value with Classes and Relationships) modeling technique suitable for modeling highly heterogeneous data with very dynamic nature. The metadata model and schema definition in EAV is separated from physical storage and therefore it is easy to modifications to schema on 'data' without changing the DB structures: by just modifying the corresponding metadata. The approach chosen is suitable for open-schema implementations (similar to key-value stores) where the model is dynamic and semantics is applied in query time, as well as model-driven implementations with a formal, well defined schema, structure and semantics.

The MMX physical schema (Figure 2) provides a storage mechanism for various knowledge- or meta-models (M2) and corresponding data or metadata (M1). Three physical tables - object, property and relation - follow the subject-predicate-object or object-property-value representation schemes, where object_type, property_type and relation_type tables are like advanced coding or dictionary tables for object, property and value types. The separate dictionary tables give us an advanced schema representation functionality using special attributes and relational database foreign keys

(FK) mechanism. The formalized schema description and relations between different schemas make our metadata understandable and exchangeable between the other system components or external agents. The URI reference mechanism used and the resource storage schema makes an MMX repository a semantic data store, comparable to Resource Description Framework (RDF), serializable in different semantic formats or notations (e.g. RDF/XML , N3 , N-Triples , XMI etc.) using XML or RDF APIs.

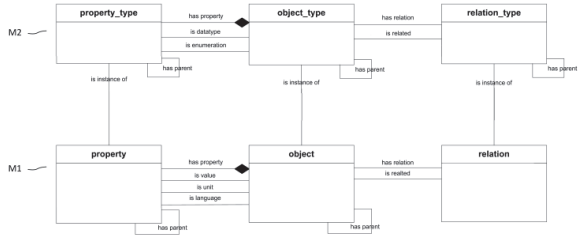


Figure 2. MMX physical schema design.

The MMX physical schema can be seen as a general-purpose, multi-level and hierarchical storage mechanism for different knowledge models, but also as communication medium or information integration and exchange platform for different software agents or applications (e.g. metadata scanners, metadata consumers etc.). Built in limited reasoning capability based on recursive SQL technique and it is captured to data and metadata APIs to implement inheritance and model validation functions. Semantic representation of data allows extend functionality with predicate calculus or apply other external rule-based reasoners (e.g. Jena) for more complicated reasoning tasks, like deduction of new knowledge.

Repository contains integrated object level security mechanism and different data access APIs (e.g. data API, metadata API, XML API, RDF API etc.) that are implemented as relational database procedures or functions. We have live implementations on PostgeSql, Oracle and MsSql platforms and differences in database SQL dialects and functionality are hidden and captured into API packages. Using common and documented API-s or an Object Relational Mapper (ORM) technology (e.g. Hibernate) we can choose and change repository DB technology without touching related applications.

An arbitrary number of different data models can exist inside MMX Metadata Model simultaneously with relationships between them. Each of these data models constitutes a hierarchy of classes where the hierarchy might denote an instance relationship, a whole-part relationship or some other form of generic relationship between hierarchy members. We have several predefined metadata models in MMX repository, e.g.

- terminology (ontology) and classification (based on ISO/IEC11179 [18]);
- relational database (based on Eclipse SQL Model [19]);
- abstract mappings and general ETL models;
- role-based access control model (based on NIST RBAC [20]).

In addition to existing models we can implement any other type of data mode for specific needs, like business process management (business rules, mappings, transformations, computational methods); data processing events (schedule, batch and task); data demographics, statistics and quality measures, etc.

The purpose of MMX repository depends on system configuration and desired functionality. In current paper we handle MMX repository as persistent storage mechanism for ETL metadata and we discuss about relational database and abstracted mapping knowledge models (KM) to store required data and relations. In addition to

repository storage and access technologies MMX Framework has web-based navigation and administration tools, semantic-wiki like content management application and different scanner agents written in XDTL (e.g. DB dictionary scanner) to feel and detect surrounding environment and context. Due to space limitation we do not discuss all those topics in this paper.

4. Template Based SQL Construction

SQL is and probably remains the main workforce behind any ETL (and especially ELT flavor of ETL) tool. Automating SQL generation has arguably always been the biggest obstacle in building an ideal ETL tool (i.e. completely metadata-driven), with small foot-print, multiple platform support on single code base. While SQL stands for Structured Query Language, ironically the language itself is not too well 'structured', and the abundance of vendor dialects and extensions does not help either. Attempts to build an SQL generator supporting full feature list of SQL language have generally fallen into one of the two camps: one of them trying to create a graphical click-and-pick interface that would encompass the syntax of every single SQL construct, another one designing an even more high-level language or model to describe SQL itself, a kind of meta-SQL. The first approach would usually be limited to simple SQL statements, be appropriate mostly for SELECT statements only and struggle with UPDATEs and INSERTs, and be limited to a single vendor dialect.

4.1. Mappings, Patterns and Templates

Based on our experience we have extracted a set of SQL 'patterns' common to practical ETL (ELT) tasks. The patterns are converted into templates for processing by a template engine (e.g. Apache Velocity), each one realizing a separate SQL fragment, a full SQL statement or a complete sequence of commands implementing a complex process. Template engine merges patterns and mappings into executable SQL statements so instead of going as deep as full decomposition we only separate and extract mappings (structure) and template (process) parts of SQL. This limits us to only a set of predefined templates, but anyone can add new or customize the existing ones. Templates are generic and can be used with multiple different mappings/data structures. The mappings are generic as well and can be used in multiple different patterns/templates. Template engine instantiates mappings and templates to create executable SQL code which brings us closer to OO mind-set. The number of tables joined, the number of columns selected, the number of WHERE conditions etc. is arbitrary and is affected by and driven by the contents of the mappings only, i.e. well-designed templates are transparent to the level of complexity of the mappings. The same template would produce quite different SQL statements driven by minor changes in mappings. We have built a series of template libraries to capture the syntax of basic SQL constructs that are used to build complex statements. XDTL Basic SQL Template Library is a set of Apache Velocity templates that implements ‘atomic’ SQL constructs (INSERT, SELECT, UPDATE, FROM, WHERE etc.) as a series of Velocity macros. Each macro is built to expand into a single SQL construct utilizing the mappings in the form of predefined collections (targets, sources, columns, conditions). On top of Basic SQL library one or more higher-level layers can be built to realize more specific or more complex concepts, e.g. loading patterns, scenarios or process flows, as well as specifics of various SQL dialects.

It appears that, by use of Abstract Syntax Patterns, the same principle of reducing a disparate and seemingly diffuse set of all possible transformations in SQL statements to a limited set of patterns applies here as well. Abstract Syntax Pattern (ASP) is a

reappearing code fragment that, similarly to Abstract Syntax Tree, has all the references to concrete data items removed. Thus, mappings between different data domains can be reduced to ASPs to be later processed synchronously with the process template by the same template engine turning them into executable code. Identifying and building a library of common syntax patterns enables creation of a user interface to generate a focused (limited) set of SQL statements without coding or even automatic SQL generation, validation of existing SQL statements [10]. More detailed ASP discovery case study can be found on chapter 5.

Various ETL metamodels discussed in previous works of [15],[16] and [17], but we decided to use pragmatic approach with ASP idea instead of complex and expressive modeling. We had modeling goals like: minimum footprint and complexity, effective code generation for different languages (e.g. SQL, SAS, R etc.), efficient storage and serialization, decomposition to the level where interesting parts would be identified and exposed with clear semantics (i.e. database objects, vendor specific terms and keywords, generic and reusable expressions etc.). In Figure 3 we have implemented mappings knowledge model with four basic classes which are designed as derivation type of rules in MMX repository. Mapping Group (B) is collection of Mappings (C) which used on multiple mapping cascade definition that produces single SQL statement with sub-queries or temporary table implementation. Each Transformation class (D) instance represents one column transformation in SQL select, insert-select or update statement with required source, target and pattern attribute definitions. Condition class (E) represents join and filter predicate conditions that required constructing data set from defined source variables (tables).

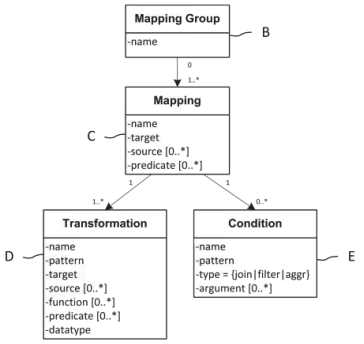


Figure 3. Knowledge model (schema) for mappings representation and storage.

Mapping model (Figure 3) implementation in MMX physical schema (Figure 2) is straightforward transformation where each class implemented as one row in object_type table, each class attribute implemented as one row in property_type table, and each association implemented as one row in relation_type table. Instances of mapping model stored as corresponding rows in object, property and relation tables.

Described method and model for constructing SQL statements complies with the following criteria:

- construction of all significant DML statements (INSERT, UPDATE, DELETE) based on a single mapping;
- construction of SQL statements from a single mapping for several different SQL dialects;
- construction of SQL statements covering different loading scenarios and performance considerations;
- construction of SQL statements based on multiple mappings (mapping groups);

- minimum footprint and complexity of the processing environment.

The next chapter example gives the basic idea of one mapping implementation and usage scenarios for SQL code generation in ETL process.

4.2. Mapping Example

Following simple and generic insert-select SQL statement represents one very basic transformation code for everyday data transformation inside DW from multiple staging tables to target table.

Example 1. Insert-select SQL DML statement:

```
INSERT INTO person t0 (id, name, sex, age)
SELECT t1.cust_id,
       t1.firstname || ' ' || t1.lastname,
       DECODE(t1.sex, 'M', 1, 'N', 2),
       ssn_to_age(t2.ssn)
FROM customer t1
JOIN document t2 ON t2.cust_id = t1.cust_id
WHERE t1.cust_id IS NOT NULL;
```

Same query contains declarative mapping part formalized and represented by following objects and collections in MMX repository tables:

Table 1. Mapping objects source and target properties.

Target	isVirtual	Source	isQuery
t0:person	f (false)	t1:customer	f (false)
		t2:document	f (false)

Table 2. Column Transformation object(s) properties and values.

Pattern	Target	Source	Function	Key	Upd
%c1	%c0:t0.id	%c1:t1.cust_id	null	t	f
%c1||' '||%c2	%c0:t0.name	%c1:t1.firstname; %c2:t1.lastname	null	f	t
decode(%c1,'M',1,'F',2)	%c0:t0.sex	%c1:t1.sex	null	f	f
%f1(%c1)	%c0:t0.age	%c1:t2.ssn	%f1:ssn_to_age	f	t

Table 3. Condition objects properties(s) and their values.

Pattern	Source	Function	Condition Type	Join Type
%a2 = %a1	%a2:t2.cust_id; %a1:t1.cust_id	null	join	inner
%a1 IS NOT NULL	%a1:t1.cust_id	null	filter	null

Simple insert-select template produces initial SQL statement from given mapping (Table 4). When using the same mapping with the different template(s) we can generate update statement or series of different statements.

Table 4. Insert-Select template and result query.

Template	SQL Statement
`#foreach($tgt in $Targets)`	`INSERT INTO person (id, name, sex, age)`
`#if("$tgtmap" == "0")`	`SELECT`
`#INSERT($tgt $Columns)`	`  t1.cust_id`
`#SELECT($Columns)`	`, t1.firstname || ' ' || t1.lastname`
`#FROM($Sources $Conditions)`	`, DECODE(t1.sex, 'M', 1, 'F', 2)`
`#WHERE($Conditions)`	`, ssn_to_age(t2.ssn)`
`#GROUPBY($Columns $Conditions)`	`FROM customer t1`
`#HAVING($Conditions)`	`INNER JOIN document t2`
`#end`	`ON t2.cust_id = t1.cust_id`
`#end;`	`WHERE t1.cust_id IS NOT NULL;`

The described example gives very basic idea and method how to formalize column mappings with reusable patterns and how to construct source and target data sets applying join and filter conditions that described again with reusable expression

patterns. Using one single mapping together with limited number of scenario templates (e.g. full load, initial load, incremental load, insert only, 'upsert' (with or without deletion), versioned insert (history tables), slowly changing dimensions etc.) we can generate long SQL statement batches, that are adapted for specific dialect (when needed), validated, robust and working with expected performance. The main idea here is to formulate and describe as less as possible and reuse and generate as much as possible.

By using described set of methods we have effectively decomposed SQL statement into mappings and patterns. The same method can be applied to any SQL data manipulation statement (e.g. insert, update and delete) of reasonable complexity and we have used same approach, same mappings and different templates to generate code for different execution engines (e.g. SAS script). More expressive examples can be presented when to take transformations with more than 5-10 source tables and targets with 20-100 columns (common in analytical DW environment) then propositions of generated, defined and reused code parts change dramatically.

To conclude the mapping example we can say that presented methodical approach allows us to:

- migrate and translate vendor-specific (SQL) pattern dialects between different platforms or use same mapping code with different transformation scenarios or generate code for different execution engines;
- construct data transformation flows from sources to targets with column level transformation semantics for Impact Analysis and Audit Trail applications;
- construct system component dependency graphs for better management and automation of development and operation processes;
- automate change management and deployment of new functionality between different environments (e.g. development, test, production).

5. Experimental Abstract Syntax Pattern Case Study

Abstract Syntax Pattern (ASP) is the practical idea to narrow down expressiveness of SQL Data Manipulation Language (DML) to allow formalized descriptions of reusable patterns, decoupled data structures and functions. Decomposition of patterns and data structure instances are the central idea of the XDTL environment and the code construction capability, which gives additional flexibility and ergonomics in data transformation design and allows impact analysis capability for maintenance of complex Data Warehouse environment (described in chapter 4). We used existing SQL statements corpus (used for real life data transformations) containing about 26 thousand SQL statements to find hard evidences for existing patterns and we narrowed down the used corpus to 12 thousand DML statement to find specific column, join and where expressions.

We used open source GoldParser[3] library and developed our own custom SQL grammar in EBNF format for SQL text corpus parsing. We developed custom parser program for ASP pattern extraction from SQL corpus, we imported all parsed patterns to database and evaluated and analyzed SQL patterns and "life forms" writing new SQL queries. Implemented parsing program is tuned to recognize column construction patterns, join, where and having condition predicate patterns from SQL DML statements, replacing specific database structure identifiers and constants with `%a` and functions with `%f` pattern.

[3] www.goldparser.org

Example 3. The parsing program detects pattern `%f(%a,%a)` from the original column expression `COALESCE(Table1.Column1,0)` and assigns operator and operand values to replaced variables: `%f = {'COALESCE'}` and `%a = {'Table1.Column1','0'}`

Very general metrics about SQL parsing work can be described with total figures of 8,380 input SQL DML statements (select, insert, update) and 92,347 parsed expressions that group to 2,671 abstract patterns. It gives us 97/3 percentage division between expressions and patterns. Those figures can be improved by hand tuning of pattern detection technique and SQL grammar that is not currently covering all the aspects of used SQL dialect.

Table 6. Insert-Select template and result query.

Statement Type	Patterns Count	Statements Count	Expression Count	Expressions in Top Pattern	Top Pattern Coverage %
Insert	1 890	4 965	61 899	37 924	61
Update	836	2 240	25 361	12 997	51
Select	224	1 175	5 087	3 175	62
All	2 950	8 380	92 347	54 096	59

Table 7. Discovered patterns by pattern types.

Pattern Type	Pattern Count	Statement Count	Expression Count	Expressions in Top Pattern	Top Patten Coverage %
Column	1 897	10 631	78 264	55 921	71
Join	526	4 172	11 684	2 273	19
Filter	323	1 505	2 399	359	14

Based on current results we can conclude that top 10 patterns will cover 83% and top 100 patterns will cover 93% of all expressions that used in SQL DML corpus. Those metrics does not count the fact that most of the patterns that are not in top list are constructed from patterns that are in patterns top list.

To conclude this case-study we can say that actual expressiveness of formalized patterns and mappings will be comparable with expressiveness of real life usage of SQL DML in data transformations. A small set of meaningful patterns (about 100 different patterns) with defined semantics and experimental impact weights will direct us to automated and probabilistic impact analysis calculations that are one of the main applications for SQL formalization technique. We also got the confirmation that SQL parsing technique can be used for data transformation extraction, mapping formalization and future analysis.

6. Case Studies for Automating Data Lineage Analysis

The previously described architecture and algorithms form a basis for an integrated data lineage analysis toolset dLineage (http://dlineage.com). dLineage has been tested in large real-life projects and environments supporting several popular DW database platforms (e.g. Oracle, Greenplum, Teradata, Vertica, PostgreSQL, MsSQL, Sybase) and BI tools (e.g. SAP Business Objects, Microstrategy).

We have conducted two main case studies involving a thorough analysis of large international companies in the financial and the energy sectors. Both case studies involved an automated analysis of thousands of database tables and views, tens of thousands of data loading scripts and BI reports. Those figures are far over the capacity

limits of human analysts not assisted by the special tools and technologies. The automation tools described in the paper enabled us to set up and conduct the analysis project in a few days by just two developers.

The following example graph from the case study maps DW tables to views and user reports: it is generated automatically from about 5 000 nodes (tables, views, reports) and 20 000 links (data transformations mappings form views and queries).

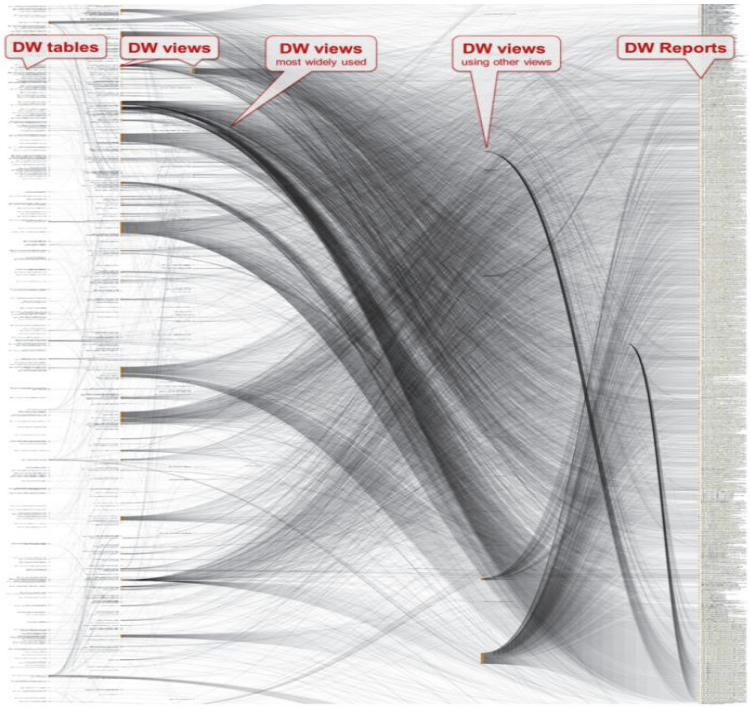


Figure 4. Data lineage graph with dependencies between DW tables, views and reports.

7. Conclusions and Future Work

We have presented a formalized mapping and abstract pattern methodology supporting template-based program construction. The technique is a development upon the ETL language runtime environment (XDTL) and metadata repository (MMX) designed earlier by the authors. We have introduced the technology and presented working samples motivated by real-life challenges and problems discussed in the first chapter. The described architecture and mapping concept have been used to implement an integrated toolset dLineage (http://dlineage.com) to solve data integration and dataflow visualization problems.

We have used our metadata-based ETL technology in the Department of Statistics of Estonian state to implement a system for automated, data-driven statistics production for the whole country. We have also tested our mapping methods and technology for data flow analysis and visualization in large international companies in the financial

and the energy sectors. Both case studies contained thousands of database tables and views along with tens of thousands of data loading scripts and BI reports. The analysis of the large SQL data transformation corpus (see chapter 5) gave us taxonomy of reusable transformation patterns and demonstrated the two-way methodology approach from code to mappings and patterns.

The future work involves refining current implementation details, adding semantics to mappings and patterns, constructing dependency graphs of mappings, data structures and data flows and developing aggregation algorithms for different personalized user profiles and their interests (e.g. business user interest in data structures, flows and availability is different from that of a developer or system operator) as well as using those techniques for solving problems described in the first chapter.

Acknowledgments

This research has been supported by European Union through European Regional Development Fund.

References

[1] Behrend,A. and Jörg,T. (2010). Optimized Incremental ETL Jobs for Maintaining Data Warehouses.
[2] Boehm,M., Habich,D., Lehner,W. and Wloka,U. (2009). GCIP: Exploiting the Generation and Optimization of Integration Processes.
[3] Böhm,M., Habich,D., Lehner,W. and Wloka,U. (2008). Model-driven generation and optimization of complex integration processes. ICEIS.
[4] Dessloch,S., Hernández,M.A., Wisnesky,R., Radwan,A., Zhou J. (2008). Orchid: Integrating Schema Mapping and ETL, IEEE 24th International Conference on Data Engineering.
[5] Giorgini,P., Rizzi,S., Garzetti,M. (2008). GRAnD: A Goal-Oriented Approach to Requirement Analysis in Data Warehouses. DSS 45(1), 4–21.
[6] Haas,L.M., Hernández,M.A., Ho,H., Popa,L. and Roth M. (2005). Clio Grows Up: From Research Prototype to Industrial Tool, in SIGMOD, p.805-810.
[7] Jun,T., Kai,C., Yu,F., Gang,T. (2009). The Research & Application of ETL Tool in Business Intelligence Project, International Forum on Information Technology and Applications, FITA'09, p.620-623.
[8] Papastefanatos,G., Vassiliadis,P., Simitsis,A., Sellis,T. and Vassiliou,Y. (2010). Rule-based Management of Schema Changes at ETL sources, Advances in Databases and Information Systems Associated Workshops and Doctoral Consortium of the 13th East European Conference ADBIS.
[9] Patil, P.S., Rao, S. and Patil,S.B. (2011). Data Integration Problem of structural and semantic heterogeneity: Data Warehousing Framework models for the optimization of the ETL processes.
[10] Reiss,S.P. (2007). Finding Unusual Code, 2007 IEEE International Conference on Software Maintenance, p.34-43.
[11] Rodiç,J. and Baranoviç,M. (2009). Generating Data Quality Rules and Integration into ETL Process.
[12] Roth,M., Hernández,M.A., Coulthard,P., Yan,L., Popa, L., Ho,H.C.T., Salter,C.C. (2006). XML mapping technology: Making connections in an XML-centric world, IBM Systems Journal.
[13] Simitsis,A., Vassiliadis, P. and Sellis,T. K. (2005). Optimizing ETL Processes in Data Warehouses, ICDE, p.564-575.
[14] Simitsis,A., Wilkinson,K., Dayal,U., Castellanos,M. (2010). Optimizing ETL workflows for fault-tolerance, International Conference on Data Engineering (ICDE), p.385-396.
[15] Song,X., Yan,X., Yang,L. (2009). Design ETL Metamodel Based on UML Profile, Knowledge Acquisition and Modeling, KAM '09. p.69-72.
[16] Stöhr,T., Müller,R., Rahm,E. (1999). An Integrative and Uniform Model for Metadata Management in Data Warehousing Environment, Workshop on Design and Management of Data Warehouses (DMDW).
[17] Vassiliadis,P., Simitsis,A., Georgantas,P., Terrovitis,M. (2003). A Framework for the Design of ETL Scenarios, In Proc. of CAiSE'03.
[18] ISO/IEC 11179 Metadata Registry (MDR) standard, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=35343.
[19] Eclipse DB Definition Model, http://www.eclipse.org/webtools/wst/components/rdb/WebPublished DBDefinitionModel/DBDefinition.htm.
[20] NIST Role Based Access Control (RBAC) Standard, http://csrc.nist.gov/groups/SNS/rbac.

Publication B

Tomingas, K.; Tammet, T.; Kliimask, M. Rule-Based Impact Analysis for Enterprise Business Intelligence. In: Artificial Intelligence Applications and Innovations (AIAI 2014), IFIP Advances in Information and Communication Technology. Springer, 2014.

Rule-based Impact Analysis for Enterprise Business Intelligence

Kalle Tomingas[1], Tanel Tammet[1], Margus Kliimask[2]

[1] Tallinn University of Technology, Ehitajate tee 5, Tallinn 19086 Estonia
[2] Eliko Competence Center, Teaduspargi 6/2, Tallinn 12618 Estonia

Abstract. We address several common problems in the field of Business Intelligence, Data Warehousing and Decision Support Systems: the complexity to manage, track and understand data lineage and system component dependencies in long series of data transformation chains. The paper presents practical methods to calculate meaningful data transformation and component dependency paths, based on program parsing, heuristic impact analysis, probabilistic rules and semantic technologies. Case studies are employed to explain further data aggregation and visualization of the results to address different planning and decision support problems for various user profiles like business users, managers, data stewards, system analysts, designers and developers.

Keywords: impact analysis, data lineage, data warehouse, rule-based reasoning, probabilistic reasoning, semantics

1 Introduction

Developers and managers are facing similar Data Lineage (DL) and Impact Analysis (IA) problems in complex data integration (DI), business intelligence (BI) and Data Warehouse (DW) environments where the chains of data transformations are long and the complexity of structural changes is high. The management of data integration processes becomes unpredictable and the costs of changes can be very high due to the lack of information about data flows and internal relations of system components. The amount of different data flows and system component dependencies in a traditional data warehouse environment is large. Important contextual relations are coded into data transformation queries and programs (e.g. SQL queries, data loading scripts, open or closed DI system components etc.). Data lineage dependencies are spread between different systems and frequently exist only in program code or SQL queries. This leads to unmanageable complexity, lack of knowledge and a large

amount of technical work with uncomfortable consequences like unpredictable results, wrong estimations, rigid administrative and development processes, high cost, lack of flexibility and lack of trust. We point out some of the most important and common questions for large DW environments (see Fig.1) which usually become a topic of research for system analysts and administrators:

- Where does the data come or go to in a specific column, table, view or report?
- Which components (reports, queries, loadings and structures) are impacted when other components are changed?
- Which data, structure or report is used by whom and when?
- What is the cost of making changes?
- What will break when we change something?

Fig. 1. General scheme of DW/BI data flows.

The ability to find ad-hoc answers to many day-to-day questions determines not only the management capabilities and the cost of the system, but also the price and flexibility of making changes. The dynamics in business, environment and requirements ensure that regular changes are a necessity for every living organization. Due to its reflective nature, the business intelligence is often the most fluid and unsteady part of enterprise information systems.

Obviously, the most promising way to tackle the challenges in such a rapidly growing, changing and labor-intensive field is automation. We claim that efficient automatization in this particular field requires the use of semantic and probabilistic technologies. Our goal is to aid the analysts with tools which can reduce several hard tasks from weeks to minutes, with better precision and smaller costs.

2 Related Work

Impact analysis, traceability and data lineage issues are not new. A good overview of the research activities of the last decade is presented in an article by Priebe et al. [9]. We can find various research approaches and published papers from the early 1990s with methodologies for software traceability [10]. The problem of data lineage tracing in data warehousing environments has been formally founded by Cui and Widom [2]

and [3]. Our recent paper builds upon this theory by introducing the Abstract Mapping representation of data transformations [14].

Other theoretical works for data lineage tracing can be found in [5] and [6]. Fan and Poulovassilis developed algorithms for deriving affected data items along the transformation pathway [5]. These approaches formalize a way to trace tuples (resp. attribute values) through rather complex transformations, given that the transformations are known on a schema level. This assumption does not often hold in practice. Transformations may be documented in source-to-target matrices (specification lineage) and implemented in ETL tools (implementation lineage).

Other practical works that based on conceptual models, ontologies and graphs for data quality and data lineage tracking can be found in [15] and [12]. De Santana proposes the integrated metadatada and the CWM metamodel based data lineage documentation approach [4]. The workflows and the manual annotations based solution proposed by Missier et al. [8].

Priebe et al. [9] concentrates on proper handling of specification lineage, a huge problem in large-scale DWH projects, especially in case different sources have to be consistently mapped to the same target. They propose a business information model (or conceptual business glossary) as the solution and a central mapping point to overcome those issues.

Our approach to Impact Analysis and Data Lineage differs from previous work in several aspects. Our aim is to merge technical data lineage [3] with semantic integration approaches [9], [11], using grammar based methods for metadata extraction from program texts and a probabilistic rule-based inference engine for weight calculations and reasoning approaches [7]. We also use the novel and powerful web based data flow and the graph visualization techniques with the multiple view approach [16] to deliver the extraction and the calculation of the result to the end-users.

3 System Architecture

We present a working Impact Analysis solution which can be adopted and implemented in an enterprise environment or provided as a service (SaaS) to manage organization information assets, analyze data flows and system component dependencies. The solution is modular, scalable and extendable. The core functions of our system architecture are built upon the following components presented in the Fig.2.

1. Scanners collect metadata from different systems that are part of DW data flows (DI/ETL processes, data structures, queries, reports etc.). We build scanners using our xml-based data transformation language and runtime engine XDTL [18].
2. The SQL parser is based on customized grammars, GoldParser parsing engine [1] and the Java-based XDTL engine.

3. The rule-based parse tree mapper extracts and collects meaningful expressions from the parsed text, using declared combinations of grammar rules and parsed text tokens.
4. The query resolver applies additional rules to expand and resolve all the variables, aliases, sub-query expressions and other SQL syntax structures which encode crucial information for data flow construction.
5. The expression weight calculator applies rules to calculate the meaning of data transformation, join and filter expressions for impact analysis and data flow construction.
6. The probabilistic rule-based reasoning engine propagates and aggregates weighted dependencies.
7. The directed and weighted sub-graph calculations, visualization and web based UI for data lineage and impact analysis applications.
8. The MMX open-schema relational database using PostgreSQL or Oracle for storing and sharing scanned, calculated and derived metadata [17].

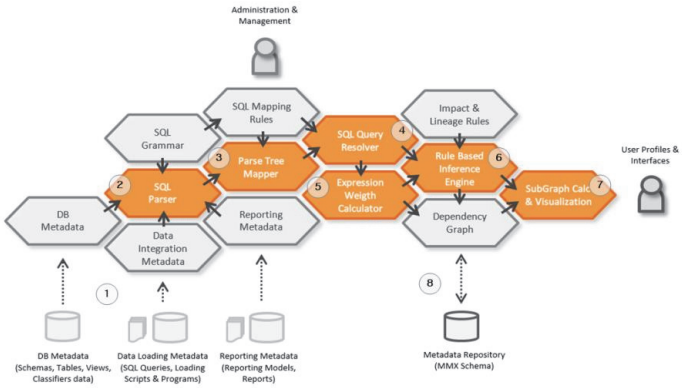


Fig. 2. Impact Analysis system architecture components.

4 Query Parsing and Metadata Extraction

Scanners (No1 in Fig.2) collect metadata from external systems: database data dictionary structures, ETL system scripts and queries, reporting system query models and reports. All the structural information extracted is stored to the metadata database. The scanned objects and their properties are extracted and stored according to the meta-models we have designed: relational database, data integration, reporting and

business terminology models. Meta-models contain ontological knowledge about the metadata and relations collected across different domains and models. The scanner technology (XDTL) and the open-schema metadata database (MMX) design have been described at a more detailed level in our previous work [13],[14].

In order to construct the data flows from the very beginning of the data sources (e.g. the accounting system) to the end points (e.g. the reporting system) we have to be able to find and connect both the identical and the related objects in different systems. In order to connect the objects we have to understand and extract the relations from the SQL queries (e.g. ETL tasks, database views, database procedures), scripts (e.g. loader utility scripts) and expressions (e.g. report structure) collected and stored by scanners. In order to understand the data transformation semantics encoded in the query language statements (e.g. insert, update, select and delete queries) and expressions, we have to involve external knowledge about the syntax and grammatical structure of the query language. Grammar-based parsing functionality is built into the scanner technology. A configurable "parse" command brings semi-structured text parsing and information extraction into the XDTL data integration environment. As the result of SQL parsing step (No2 in Fig.2) we get a large parse tree with every SQL query token assigned a special disambiguated meaning by the grammar.

In order to convert different texts into the tree structure, to reduce tokens and to convert the tree back to the meaningful expressions (depending on search goals), we use a declarative rule set presented in the Json format, combining token and grammar rules. A configurable grammar and a synchronized reduction rule set makes the XDTL parse command suitable for general purpose information extraction and captures the resource hungry computation steps into one single parse-and-map step with the flat table outcome. The Parse Tree Mapper (No3 in Fig.2) uses three different rule sets with more than eighty rules to map the parse tree to data transformation expressions.

After extraction and mapping of each SQL query statement into a series of expressions we execute the SQL Query Resolver (No4 in Fig.2) which contains a series of functions to resolve the SQL query structure:

- Solve source and target object aliases to full qualified object names;
- Solve sub-query aliases to context specific source and target object names;
- Solve sub-query expressions and identifiers to expand all the query level expressions and identifiers with fully qualified and functional ones;
- Solve syntactic dissymmetry in different data transformation expressions (e.g. insert statement column lists, select statement column lists and update statement assign list etc.);
- Extract quantitative metrics from data transformation, filter and join expressions to calculate expression weights (e.g. number of columns in expression, functions, predicates, string constants, number constants etc.).

5 Data Transformation Weight Calculation

Data structure transformations are parsed, extracted from queries and stored as formalized, declarative mappings in the system. To add additional quantitative measures to each column transformation or column usage in the join and filter conditions we evaluate each expression and calculate transformation and filter weights for those.

Expression Weight Calculation (No5 in Fig.2) is based on the idea that we can evaluate column data "transformation rate" and column data "filtering rate" using data structure and structure transformation information captured from SQL queries. Such a heuristic evaluation enables us to distinguish columns and structures used in the transformation expressions or in filtering conditions or both, and gives probabilistic weights to expressions without the need to understand the full semantics of each expression. We have defined two measures that we further use in our probabilistic rule system for deriving new facts:

Definition 1. A primitive data transformation operation $O(X,Y,M1,F1,W_t)$ is a data transformation between a source column X and a target column Y in a transformation set M (mapping or query) having expression similarity weight W_t and having conditions set F1.

Definition 2. Column transformation weight W_t is based on the similarity of each source column and column transformation expression: the calculated weight expresses the source column transfer rate or strength. The weight is calculated on scale [0,1] where 0 means that the source data are not transformed to target (e.g. constant assignment in a query) and 1 means that the source is directly copied to the target (no additional column transformations).

Definition 3. Column filter weight W_f is based on the similarity of each filter column in the filter expression and the calculated weight expresses the column filtering rate or strength. The weight is calculated on scale [0,1] where 0 means that the column is not used in the filter and 1 means that the column is directly used in the filter predicate (no additional expressions).

The general column weight W algorithm in each expression for W_t and W_f components is calculated as a column count ratio over all the expression component counts (e.g. column count, constant count, function count, predicate count):

```
W=IdCount/IdCount+FncCount+StrCount+NbrCount+PrdCount.
```

5.1 Rule System

The defined figures, operations and weights are used in combination with the declarative probabilistic inference rules to calculate the possible relations and dependencies between data structures and software components. Applying the rule system to the extracted query graphs we calculate and produce a full dependency graph that is used for data lineage and impact analysis.

The basic operations used in the rules for the dependency graph calculations are the following:

- The primitive data transformation is the elementary operation between the source column X and the target column Y in the query mapping id set M1 with the filter condition set F1 and the transformation weight W_t (see Definition 1). This is represented by the predicate `O(X,Y,M1,F1,Wt);`
- The predicate `member(X,F1,Wt)` is used in the filter impact calculation rule to detect that the column X is a member of the filter id set F1 with the filter weight Wt;
- The predicate `disjoint(M1,M2)` is used in the impact aggregation rule to detect that two query mapping id sets M1 and M2 are disjoint. The disjointness condition is necessary for aggregating the data transformation relations and weights in case more than one path from different queries connects the same column pairs;
- The predicate `parent(X0,X1)` is used in the parent aggregation rule to detect that the table X0 is the owner or parent object of the column X1. This condition is necessary for aggregating all the column level relations and the weights between two tables for the table level impact relation;
- The function `union(M1,M2)` is used to calculate the impact relations over two disjoint query id sets M1 and M2: it returns the distinct id lists of two sets M1 and M2;
- The function `sum(W1,W2)` is used to calculate the aggregated impact relation weight in case the basic operations are disjoint, i.e. stem from independent queries. The weight calculation is based on non-mutually-exclusive event probabilities (two independent queries means there could be an overlap between two events) and is calculated as the sum of probabilities of W1 and W2: `sum(W1,W2)=(W1+W2)-(W1*W2);`
- The function `avg(W1,W2)` is used to calculate the parent impact weight when the basic operations have the same parent structures. The weight is calculated as the arithmetic mean of W1 and W2: `avg(W1,W2)=(W1+W2)/2;`

The inference rules with the basic operations and the weighs for the dependency graph calculations are the following:

- The basic impact calculation rule for the operation O with no additional filters produces the impact predicate I:
  `O(X,Y,M1,F1,Wt) => I(X,Y,M1,F1,Wt);`
- The basic impact calculation rule for the operation O with a related filter condition produces the impact predicate I with multiplied weights:
  `O(X,Y,M1,F1,Wt)& member(X,F1,Wf) => I(X,Y,M1,F1,Wt*Wf);`
- The transitivity rule is used to calculate the sequences of the consecutive impact relations:

```
I(X,Y,M1,F1,W1) & I(Y,Z,M2,F2,W2) & disjoint(M1,M2) =>
          I(X,Z,union(M1,M2),union(F1,F2),W1*W2);
```

- The column aggregation rule is used when multiple different paths from the different queries connect the same columns: calculate the impact relations with aggregated query id-s and the aggregated weights:

```
I(X,Z,M1,F1,W1) & I(X,Z,M2,F2,W2) & disjoint(M1,M2) =>
          I(X,Z,union(M1,M2),union(F1,F2),sum(W1,W2));
```

- The parent aggregation rule is used when multiple different impact relations connect the column pairs of the same tables: calculate the table level impact relations with aggregated query id-s and aggregated weights:

```
I(X1,Y1,M1,F1,W1) & I(X2,Y2,M2,F2,W2) & parent(X0,X1)& par-
ent(X0,X2) & parent(Y0,Y1) & parent(Y0,Y2) =>
          I(X0,Y0,union(M1,M2),union(F1,F2),avg(W1,W2)).
```

5.2 Dependency Score Calculation

We can use the derived dependency graph to solve different business tasks by calculating the selected component(s) lineage or impact over available layers and chosen details. Business questions like: “What reports are using my data?”, “Which components should be changed or tested?” or “What is the time and cost of change?” are converted to directed sub-graph navigation and calculation tasks. The following definitions add new quantitative measures to each component or node (e.g. table, view, column, etl task, report etc.) in the calculation. We use those measures in the UI to sort and select the right components for specific tasks.

Definition 4. Local Lineage Dependency % (LLD) is calculated as the ratio over the sum of the local source and target Lineage weights W_t:

$LLD = SUM(source(W_t)/source(W_t)+target(W_t))$.

Local Lineage Dependency 0 % means that there are no data sources detected for the object. Local Lineage Dependency 100 % means that there are no data consumers (targets) detected for the object. Local Lineage Dependency about 50 % means that there are equal numbers of weighted sources and consumers (targets) detected for the object.

Definition 5. Local Impact Dependency % (LID) is calculated as the ratio over the sum of local source and target impact weights $W(W_t,W_f)$:

$LID=SUM(source(W) /source(W)+target(W))$.

Local Impact Dependency 0 % means that there are no dependent data sources detected for the object. Local Dependency 100 % means that there are no dependent data consumers (targets) detected for the object. Local Impact Dependency about 50 % means that there are equal numbers of weighted dependent sources and consumers (targets) detected for the object.

Definition 6. Global Dependency Count (GDC) is the sum of all source and target Lineage and Impact relations counts: `GDC=GSC+GTC`.

The Global Dependency Count is a good differential metric that allows us to see clear distinctions in the dependencies of each object. We can take the GDC metric as a sort of "gravity" of the object that can be used to develop new rules, to infer the time and cost of changes of object(s) (e.g. database table, view, data loading programs or report).

6 Conclusions and Future Work

The previously described architecture and algorithms have been used to implement an integrated toolset dLineage (http://dlineage.com). Both the scanners and web-based tools of dLineage have been enhanced and tested in real-life projects and environments to support several popular DW database platforms (e.g. Oracle, Greenplum, Teradata, Vertica, PostgreSQL, MsSQL, Sybase), ETL tools (e.g. Pentaho, Oracle Data Integrator, SQL scripts and different data loading utilities) and BI tools (e.g. SAP Business Objects, Microstrategy). The dLineage dynamic visualization and graph navigation tools are implemented in Javascript using the d3.js graphics libraries.

We have tested our solution during two main case studies involving a thorough analysis of large international companies in the financial and the energy sectors. Both case studies analyzed thousands of database tables and views, tens of thousands of data loading scripts and BI reports. Those figures are far over the capacity limits of human analysts not assisted by the special tools and technologies.

We have presented several algorithms and techniques for quantitative impact analysis, data lineage and change management. The focus of these methods is on automated analysis of the semantics of data conversion systems followed by employing probabilistic rules for calculating chains and sums of impact estimations. The algorithms and techniques have been successfully employed in two large case studies, leading to practical data lineage and component dependency visualizations.

We are planning to continue this research by considering a more abstract, conceptual/business level in addition to the current physical/technical level of data representation.

Acknowledgments

This research has been supported by European Union through European Regional Development Fund.

References

1. Cook,D. (2010). Gold parsing system. URL: http://www.goldparser.org.
2. Cui, Y., Widom, J., & Wiener, J. L. (2000). Tracing the lineage of view data in a warehousing environment. ACM Transactions on Database Systems (TODS), 25(2), 179-227.
3. Cui, Y., & Widom, J. (2003). Lineage tracing for general data warehouse transformations. The VLDB Journal—The International Journal on Very Large Data Bases, 12(1), 41-58.
4. de Santana, A. S., & de Carvalho Moura, A. M. (2004). Metadata to support transformations and data & metadata lineage in a warehousing environment. In Data Warehousing and Knowledge Discovery (pp. 249-258). Springer Berlin Heidelberg.
5. Fan, H., & Poulovassilis, A. (2003, November). Using AutoMed metadata in data warehousing environments. In Proceedings of the 6th ACM international workshop on Data warehousing and OLAP (pp. 86-93). ACM.
6. Giorgini, P., Rizzi, S., & Garzetti, M. (2008). GRAnD: A goal-oriented approach to requirement analysis in data warehouses. Decision Support Systems, 45(1), 4-21.
7. Luberg, A., Tammet, T., & Järv, P. (2011). Smart City: A Rule-based Tourist Recommendation System. In Information and Communication Technologies in Tourism 2011 (pp. 51-62). Springer Vienna.
8. Missier, P., Belhajjame, K., Zhao, J., Roos, M., & Goble, C. (2008). Data lineage model for Taverna workflows with lightweight annotation requirements. In Provenance and Annotation of Data and Processes (pp. 17-30). Springer Berlin Heidelberg.
9. Priebe, T., Reisser, A., & Hoang, D. T. A. (2011). Reinventing the Wheel?! Why Harmonization and Reuse Fail in Complex Data Warehouse Environments and a Proposed Solution to the Problem.
10. Ramesh, B., & Jarke, M. (2001). Toward reference models for requirements traceability. Software Engineering, IEEE Transactions on, 27(1), 58-93.
11. Reisser, A., & Priebe, T. (2009, August). Utilizing Semantic Web Technologies for Efficient Data Lineage and Impact Analyses in Data Warehouse Environments. In Database and Expert Systems Application, 2009. DEXA'09, pp. 59-63.
12. Skoutas, D., & Simitsis, A. (2007). Ontology-based conceptual design of ETL processes for both structured and semi-structured data. International Journal on Semantic Web and Information Systems (IJSWIS), 3(4), 1-24.
13. Tomingas,K., Kliimask,M., Tammet,T. (2014). Mappings, Rules and Patterns in Template Based ETL Construction. The 11th International Baltic DB&IS2014 Conference.
14. Tomingas,K., Kliimask,M., Tammet,T. (2014). Data Integration Patterns for Data Warehouse Automation. The 18th East-European ADBIS2014 Conference.
15. Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002, November). Conceptual modeling for ETL processes. In Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP (pp. 14-21). ACM.
16. Wang Baldonado, M. Q., Woodruff, A., & Kuchinsky, A. (2000, May). Guidelines for using multiple views in information visualization. In Proceedings of the working conference on Advanced visual interfaces (pp. 110-119). ACM.
17. MMX Metadata Framework, URL: http://mmxframework.org
18. XDTL Data Transformation Language, URL: http://xdtl.org

Publication C

Tomingas, K.; Tammet, T.; Kliimask, M.; Järv, P. Automating Component Dependency Analysis for Enterprise Business Intelligence. In: 2014 International Conference on Information Systems (ICIS 2014).

Automating Component Dependency Analysis for Enterprise Business Intelligence

Completed Research Paper

Kalle Tomingas
Tallinn Uni of Technology
Ehitajate tee 5
Tallinn, Estonia
kalle.tomingas@gmail.com

Tanel Tammet
Tallinn Uni of Technology
Ehitajate tee 5
Tallinn, Estonia
tanel.tammet@ttu.ee

Margus Kliimask
Eliko Competence Center
Teaduspargi 6/2
Tallinn, Estonia
margus.kliimask@gmail.com

Priit Järv
Eliko Competence Center
Teaduspargi 6/2
Tallinn, Estonia
priit.jarv@gmail.com

Abstract

We address common problems in the field of Business Intelligence, Data Warehousing and Decision Support Systems: the complexity to manage, track and understand data lineage and system component dependencies in long series of data transformation chains. The paper presents practical methods to calculate meaningful data transformation and component dependency paths, based on program parsing, heuristic impact analysis, probabilistic rules and semantic technologies. Case studies are employed to explain further data aggregation and visualization of the results to address different planning and decision support problems for various groups of technical and business users.

Keywords: Component dependency analysis, impact analysis, data lineage, data warehouse, rule-based reasoning.

Introduction

Developers and managers are facing similar Data Lineage (DL) and Impact Analysis (IA) problems in complex data integration (DI), business intelligence (BI) and Data Warehouse (DW) environments where the chains of data transformations are long and the complexity of structural changes is high. The management of data integration processes becomes unpredictable and the costs of changes can be very high due to the lack of information about data flows and internal relations of system components. The amount of different data flows and system component dependencies in a traditional data warehouse environment is large. Important contextual relations are coded into data transformation queries and programs (e.g. SQL queries, data loading scripts, open or closed DI system components etc.). Data lineage dependencies are spread between different systems and frequently exist only in program code or SQL queries. This leads to unmanageable complexity, lack of knowledge and a large amount of technical work with uncomfortable consequences like unpredictable results, wrong estimations, rigid administrative and development processes, high cost, lack of flexibility and lack of trust.

We point out some of the most important and common questions for large DW environments (see Figure 1) which usually become a topic of research for system analysts and administrators:

- Where does the data come or go to in/from a specific column, table, view or report?
- When was the data loaded, updated or calculated in a specific column, table, view or report?
- Which components (reports, queries, loadings and structures) are impacted when other components are changed?
- Which data, structure or report is used by whom and when?
- What is the cost of making changes?
- What will break when we change something?

Figure 1. General scheme of DW/BI data flows.

The ability to find ad-hoc answers to many day-to-day questions determines not only the management capabilities and the cost of the system, but also the price and flexibility of making changes. The dynamics in business, environment and requirements ensure that regular changes are a necessity for every living organization. Due to its reflective nature, the business intelligence is often the most fluid and unsteady part of enterprise information systems.

Obviously, the most promising way to tackle the challenges in such a rapidly growing, changing and labor-intensive field is automation. We claim that efficient automatization in this particular field requires the use of semantic and probabilistic technologies. Our goal is to aid the analysts with tools which can reduce several hard tasks from weeks to minutes, with better precision and smaller costs.

We will draw a short overview of the previous body of research and related problems in the next chapter. We will continue by describing the components of the presented system in the chapter 3. Chapter 4 will describe query parsing, analysis, resolving and semantics extraction techniques. These and the probabilistic rule-based reasoning techniques described in the chapters 5 and 6 forms the backbone of the presented automated solution. We will present illustrative examples in the chapter 7 and chapter 8 presents real life case studies with a functional analytical application. Chapter 9 concludes by presenting the current status of the work and describing further research.

Related Work

Impact analysis, traceability and data lineage issues are not new. A good overview of the research activities of the last decade is presented in an article by (Priebe et al. 2011). We can find various research approaches and published papers from the early 1990's with methodologies for software traceability (Ramesh and Jarke 2001). The problem of data lineage tracing in data warehousing environments has been formally founded by Cui and Widom (2000, 2003). Our recent papers build background to the theory by introducing the Abstract Mapping representation of data transformations and rule-based impact analysis (Tomingas et.al. 1014, 2015).

Other theoretical works for data lineage tracing can be found in (Fan and Poulovassilis 2003) and (Giorgini et al. 2005). Fan and Poulovassilis developed algorithms for deriving affected data items along the transformation pathway (Fan and Poulovassilis 2003). These approaches formalize a way to trace tuples (resp. attribute values) through rather complex transformations, given that the transformations are known on a schema level. This assumption does not often hold in practice. Transformations may be documented in source-to-target matrices (specification lineage) and implemented in ETL tools (implementation lineage). Woodruff and Stonebraker create solid base for the data-level and the operators processing based the fine-grained lineage in contrast to the metadata based lineage calculation in their research paper (Woodruff and Stonebraker 1997).

Other practical works that based on conceptual models, ontologies and graphs for data quality and data lineage tracking can be found in (Vassiliadis et.al. 2002), (Skoutas and Simitsis 2007) and (Widom 2004). De Santana proposes the integrated metadatada and the CWM metamodel based data lineage documentation approach (de Santana and de Carvalho Moura 2004). The workflows and the manual annotations based solution proposed by Missier et al. (2008).

Priebe et al. (2011) concentrates on proper handling of specification lineage, a huge problem in large-scale DWH projects, especially in case different sources have to be consistently mapped to the same target. They propose a business information model (or conceptual business glossary) as the solution and a central mapping point to overcome those issues.

Several commercial ETL products are addressing the impact analysis and data lineage problems to some extent (e.g. Oracle Data Integrator, Informatica PowerCenter, IBM DataStage or Microsoft SQL Server Integration Services), but those tools and the dependency analysis performed is often limited to the basic functions of the particular system. Another group of commercial tools is formed by the specialized metadata integration products not related to a particular ETL tool, offering a more sophisticated suite of dependency analysis functionality. The examples are ASG Rochade, Adaptive Metadata Manager, Troux Enterprise Architecture Solution or Teradata Metadata Services (MDS): all of those have their own limitations in terms of available functionality and adapters to other products (Priebe et al. 2011).

Our approach to Impact Analysis and Data Lineage differs from previous work in several aspects. Our aim is to merge technical data lineage (Cui and Widom 2003) with semantic integration approaches (Priebe et al. 2011, Reisser and Priebe 2009), using grammar based methods for metadata extraction from program texts and a probabilistic rule-based inference engine for weight calculations and reasoning approaches (Tammet et al. 2010). We also use the novel and powerful web based data flow and the graph visualization techniques with the multiple view approach (Wang Baldonado et.al. 2000) to deliver the extraction and the calculation of the result to the end-users.

System Architecture

We present a working Impact Analysis solution which can be adopted and implemented in an enterprise environment or provided as a service (SaaS) to manage organization information assets, analyse data flows and system component dependencies. The solution is modular, scalable and extendable. The core functions of our system architecture are built upon the following components presented in the Figure 2:

1) Scanners collect metadata from different systems that are part of DW data flows (DI/ETL processes, data structures, queries, reports etc.). We build scanners using our xml-based data transformation language and runtime engine XDTL[1] written in Java.
2) The SQL parser is based on customized grammars, GoldParser[2] parsing engine and the Java-based XDTL engine.
3) The rule-based parse tree mapper extracts and collects meaningful expressions from the parsed text, using declared combinations of grammar rules and parsed text tokens.
4) The query resolver applies additional rules to expand and resolve all the variables, aliases, sub-query expressions and other SQL syntax structures which encode crucial information for data flow construction.
5) The expression weight calculator applies rules to calculate the meaning of data transformation, join and filter expressions for impact analysis and data flow construction.
6) The probabilistic rule-based reasoning engine propagates and aggregates weighted dependencies.
7) The directed and weighted sub-graph calculations, visualization and web based UI for data lineage and impact analysis applications.
8) The MMX[3] open-schema relational database using PostgreSQL or Oracle for storing and sharing scanned, calculated and derived metadata.

[1] http://xdtl.org

[2] http://goldparser.org

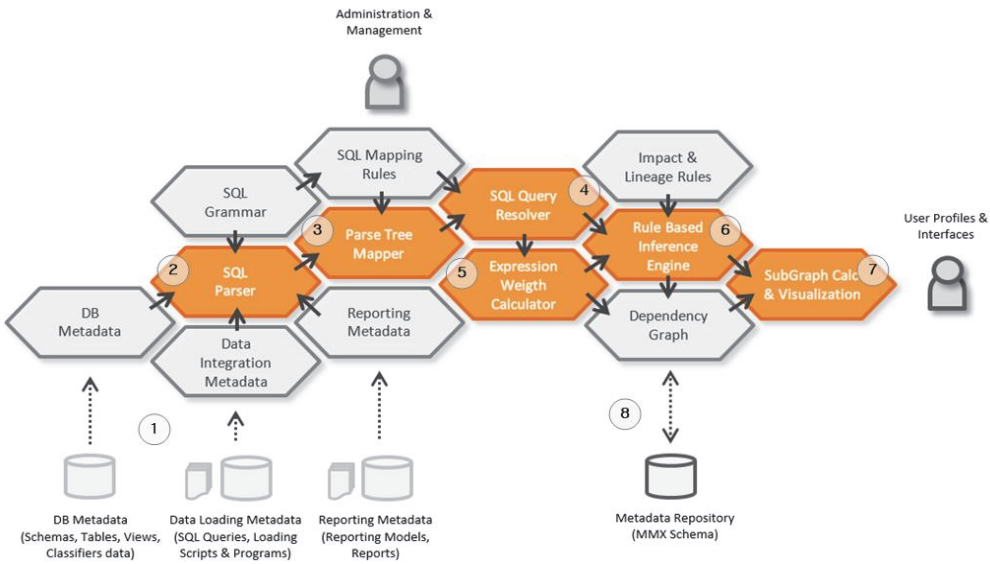


Figure 2. Impact Analysis system architecture components.

Query Parsing and Metadata Extraction

In order to construct the data flows from the very beginning of the data sources (e.g. the accounting system) to the end points (e.g. the reporting system) we have to be able to find and connect both the identical and the related objects in different systems. In order to connect the objects we have to understand and extract the relations from the SQL queries (e.g. ETL tasks, database views, database procedures), scripts (e.g. loader utility scripts) and expressions (e.g. report structure) collected and stored by scanners. In order to understand the data transformation semantics encoded in the query language statements (e.g. insert, update, select and delete queries) and expressions, we have to involve external knowledge about the syntax and grammatical structure of the query language. We use a general purpose Java-based GoldParser engine (Cook 2010) and we have developed a custom SQL grammar written in the Extended Backus-Naur Form (EBNF). Our grammar is based on the ANSI/SQL syntax, but it also contains a large set of dialect-specific notations, syntactical constructions and functions that are developed and trained using large real life SQL query corpuses from the field of data warehousing. The current version of our grammar supports also Teradata, Oracle, Greenplum, Vertica, Postgres and MsSQL dialects.

Grammar-based parsing functionality is built into the scanner technology. A configurable "parse" command brings semi-structured text parsing and information extraction into the XDTL data integration environment. As the result of SQL parsing step (No2 in Figure 2) we get a large parse tree with every SQL query token assigned a special disambiguated meaning by the grammar.

In order to convert different texts into the tree structure, to reduce tokens and to convert the tree back to the meaningful expressions (depending on search goals), we use a declarative rule set presented in the Json format, combining token and grammar rules. A configurable grammar and a synchronized reduction

[3] http://mmxframework.org

rule set makes the XDTL parse command suitable for general purpose information extraction and captures the resource hungry computation steps into one single parse-and-map step with the flat table outcome. The Parse Tree Mapper (No3 in Figure 2) uses three different rule sets with more than eighty rules to map the parse tree to data transformation expressions:

- Stopword list and grammar rules are used to flush the buffer and start the token collection to construct a new expression;
- Mapword list and grammar rules are used to map the collected expressions to meaningful items (e.g. sources, targets, data transformations, joins and filters);
- Tagword list and grammar rules are used to tag special meaningful tokens in expressions to identify all the database object references (e.g. tables, views, columns, functions and constants).

After extraction and mapping of each SQL query statement into a series of expressions we execute the SQL Query Resolver (No4 in Figure 2) which contains a series of functions to resolve the SQL query structure:

- Solve source and target object aliases to full qualified (schema name + object name) object names;
- Solve sub-query aliases to context specific source and target object names;
- Solve sub-query expressions and identifiers to expand all the query level expressions and identifiers with fully qualified and functional ones;
- Solve syntactic dissymmetry in different data transformation expressions (e.g. insert statement column lists, select statement column lists and update statement assign list etc.);

The following list describes the fields and measures of the parse results, which are sources for the following calculation steps. We also use the defined field names in rules and metrics definitions in the next chapters:

- *StrList* - String constant list used in each expression;
- *NbrList* - Number constant list used in each expression;
- *FncList* - Function list used in each expression;
- *IdCount* - Column identifiers count in each expression;
- *StrCount* - String constant count in each expression;
- *NbrCount* - Number constant count in each expression;
- *FncCount* - Functions count in each expression;
- *PrdCount*- Predicate operators count in each expression.

Data Transformation Weight Calculation

Data structure transformations are parsed, extracted from queries and stored as formalized, declarative mappings in the system. To add additional quantitative measures to each column transformation or column usage in the join and filter conditions we evaluate each expression and calculate transformation and filter weights for those.

Expression Weight Calculation (No5 in Figure 2) is based on the idea that we can evaluate column data "transformation rate" and column data "filtering rate" using data structure and structure transformation information captured from SQL queries. Such a heuristic evaluation enables us to distinguish columns and structures used in the transformation expressions or in filtering conditions or both, and gives probabilistic weights to expressions without the need to understand the full semantics of each expression. We have defined two measures that we further use in our probabilistic rule system for deriving new facts:

Definition 1. A primitive data transformation operation O(X,Y,M1,F1,Wt) is a data transformation between a source column X and a target column Y in a transformation set M (mapping or query) having expression similarity weight Wt and having conditions set F1.

Definition 2. Column transformation weight Wt is based on the similarity of each source column and column transformation expression: the calculated weight expresses the source column transfer rate or strength. The weight is calculated on scale [0,1] where 0 means that the data is not transformed form

source (e.g. constant assignment in a query) and 1 means that the source is directly copied to the target (no additional column transformations).

Definition 3. Column filter weight Wf is based on the similarity of each filter column in the filter expression and the calculated weight expresses the column filtering rate or strength. The weight is calculated on scale [0,1] where 0 means that the column is not used in the filter and 1 means that the column is directly used in the filter predicate (no additional expressions).

The general column weight W algorithm in each expression for Wt and Wf components is calculated as a column count ratio over all the expression component counts (e.g. column count, constant count, function count, predicate count).

$$W = \frac{IdCount}{IdCount + FncCount + StrCount + NbrCount + PrdCount}$$

All the used expression column weight calculation figures are listed and defined in the previous chapter. The counts are normalized using the *FncList* evaluation over a positive function list (e.g. CAST, ROUND, COALESCE, TRIM etc.). If *FncList* member is in a positive function list then the normalization function reduces the according component count by 1 to "pay a smaller price" in case the function used does not have a significant impact to column data.

Example 1. Column transformation weight calculation using expression measures:

- When a column data is copied directly from the source column to the target column in the SQL DML statement (e.g. insert-select, update) then the data transformation weight is 1. The following simple query q1: "`INSERT INTO T2 (c1) SELECT c1 FROM T1`" interpreted as the data transformation operation O with the weight 1: O(T1.c1,T2.c1,q1,null,1).
- When a source column is not defined and a data (e.g. constant or function) is assigned to the target column in the SQL DML statement then the data transformation weight is 0 and the direct relation between the source and the target columns does not exist. The following simple query q2: "`INSERT INTO T2 (c1) SELECT '10'`" interpreted as the data transformation operation O with the weight 0: O(null,T2.c1,q2,null,0).
- When a column data is mapped from the source column to the target column in the SQL DML statement column expression then the data transformation weight depends on complexity of expression and the weight is between 0 and 1.
- The following expression samples and the calculated weights for each source-target column pair illustrates the variation of the data transformations:

```
q3: CAST(T1.LogDate AS DATE) as Request_Date       => 1.0
q4: T1.First_Name||' '||T1.Last_Name as Full_Name  => 0.67
q5: MIN(T1.Balance_Amt) as Min_Balance_Amt         => 0.5
q6: SUM(ZEROIFNULL(T1.Payment_Amt)) as Sales_Amt   => 0.33
q7: CASE WHEN T1.Feature_Id is not null THEN 'Y' ELSE 'N' END as Dynamic_Ind
                                                   => 0.17
```

- The previous expression q7 contains parts and measures like *IdCount:* 1 (T1.Feature_Id), *FncCount:* 2 (Case,WhenThen) and *StrCount:* 3 (null,Y,N). When using those values and the weight W definition then we can calculate the column pair operation O(T1.Feature_Id,T2. Dynamic_Ind,q7,null,0.17) weight in the expression q8 like this:

$$W = \frac{1}{1 + 2 + 3 + 0 + 0} = \frac{1}{6} = 0.16667 \cong 0.17$$

Rule System and Dependency Calculation

Rule System

The defined figures, operations and weights are used with combinations of declarative inference rules with probabilistic reasoning to calculate the possible relations and dependencies between data structures and software components. Applying the rule system to extracted query graphs we calculate and produce a full dependency graph that is used for data lineage or impact analysis.

The basic operations used in the rules for the dependency graph calculations are following:

- The primitive data transformation is the elementary operation between the source column X and the target column Y in the query mapping id set M1 with the filter condition set F1 and the transformation weight W_t (see Definition 1) - `O(X,Y,M1,F1,Wt);`
- The function `member(X,F1,Wt)` used in the filter impact calculation rule to detect that if column X is the member of the filter id set F1 with the filter weight Wt;
- The function `disjoint(M1,M2)` used in the impact aggregation rule to detect that two query mapping id sets M1 and M2 are the disjoint sets. The `disjoint` function is needed to aggregate the data transformation relations and the weights when more than one path from the different queries connects the same column pairs;
- The function `parent(X0,X1)` used in the parent aggregation rule to detect that table X0 is the owner or parent object for column X1. The `parent` function is needed to aggregate all the column level relations and the weights between two tables to the table level impact relation and the weight;
- The function `union(M1,M2)` used to calculate the impact relations over two disjoint query mapping id sets M1 and M2, and the function returns the distinct id lists of two sets (M1 and M2);
- The function `sum(W1,W2)` used to calculate the aggregated impact relations weight when the basic operations are the disjoint sets and the part of the independent queries. The weight calculation based on non-mutually exclusive event probabilities (two independent queries means possible overlap between two events) and calculated as probability sum of W1 and W2: `sum(W1,W2)=(W1+W2)-(W1*W2);`
- The function `avg(W1,W2)` used to calculate the parent impact weight when the basic operations having the same parent structures. The weight calculation based on the average sum and calculated as the arithmetic mean of W1 and W2: `avg(W1,W2)=(W1+W2)/2;`

The main inference rules with the basic operations and the weighs for the dependency graph calculations are following:

- The basic impact calculation rule for the operation O with no additional filters produces the impact predicate I (R1):

```
O(X,Y,M1,F1,Wt) => I(X,Y,M1,F1,Wt);
```

- The basic impact calculation rule for the operation O with a related filter condition produces the impact predicate I with multiplied weights (R2):

```
O(X,Y,M1,F1,Wt)& member(X,F1,Wf) => I(X,Y,M1,F1,Wt*Wf);
```

- The transitivity rule is used to calculate the sequences of the consecutive impact relations (R3):

```
I(X,Y,M1,F1,W1) & I(Y,Z,M2,F2,W2) & disjoint(M1,M2) =>
    I(X,Z,union(M1,M2),union(F1,F2),W1*W2);
```

- The column aggregation rule is used when multiple different paths from the different queries connect the same columns: calculate the impact relations with aggregated query id-s and the aggregated weights (R4):

```
I(X,Z,M1,F1,W1) & I(X,Z,M2,F2,W2) & disjoint(M1,M2) =>
    I(X,Z,union(M1,M2),union(F1,F2),sum(W1,W2));
```

- The parent aggregation rule is used when multiple different impact relations connect the column pairs of the same tables: calculate the table level impact relations with aggregated query id-s and aggregated weights (R5):

```
I(X1,Y1,M1,F1,W1) & I(X2,Y2,M2,F2,W2) & parent(X0,X1)& parent(X0,X2) & parent(Y0,Y1)
& parent(Y0,Y2) =>
    I(X0,Y0,union(M1,M2),union(F1,F2),avg(W1,W2)).
```

Fact Inference

Building the knowledge base of object relations is done by applying the inference rules to the existing relations iteratively. Rules R1 and R2 apply to the facts that are prior knowledge and remain unchanged during the inference step, therefore R1 and R2 need to be evaluated only once to perform a conversion from 'O()' facts to 'I()' facts. Rules R3, R4 and R5 apply to the predicates created during the inference process and therefore need to be iterated over, until no new predicates are created.

We build an in-memory data structure for efficient application of iterative inference from the 'I()'-facts. Each fact is kept in a record, with several tree indexes for record lookup by object identifier. The relations contain sets of data flow mappings that the fact is based on, which are stored in variable length arrays ('M-set' records). We also store the set of siblings for the objects in the relation to efficiently evaluate expressions like parent(X0, X1) & parent(X0, X2) ('C-set' records).

An 'M-set' record is a variable sized array containing all the mapping identifiers originating from 'O()' facts. The inference procedure is as follows:

```
derive I() facts using rules R1 and R2 (generation 0)
i := 0
repeat
  infer I() facts using R3, R4 and R5 (generation i+1)
  i := i + 1
until no more new facts were generated
```

Each iteration step takes the set $\{x|x.\text{generation} = i\} \times \{y.\text{generation} \leq i\}$as input and produces the set $\{z|z.\text{generation} = i+1\}$, where x, y, z denote 'I()' facts.

As an example that makes use of most of the features of the in-memory knowledge base representation, consider the algorithm for the parent aggregation rule (R5):

```
i := current generation
Seen := ∅
for each I() predicate where I.generation = i and I ∉Seen
  M := I.M
  W := I.W
  c := 1
  for each I'() predicate where I'.X is a sibling of I.X
    if I'X ≠ I.X and I'.Y is a sibling of I.Y
      M := M ∪ I'.M
      W := W + I.W
      Seen := Seen ∪ I'
      c := c + 1
  if c > 1
    create the aggregate Ip(X.parent, Y.parent, M, W/c)
    Ip.generation := i+1
    Seen := Seen ∪ I'
```

Note that the algorithm skips pairwise examination of predicates and immediately groups all facts where the source and target objects respectively have common parents to produce the aggregate fact. The tree index is used to retrieve predicates with given object identifiers, while the sibling sets are directly linked to the predicate records. The amortized time complexity of this algorithm is in $O(n^2)$if we consider set membership tests to be in $O(1)$and union operation in $O(n)$; here $n = |\{x|x.\text{generation} \leq i\}|$

The prototype implementation inferred 32900 'I()'-predicates from 24671 data flow mappings and 7484 filter mappings in 96 seconds on a desktop-class computer. This problem size corresponds to a real-life data-warehouse setting, making the chosen approach applicable in practice.

Dependency Score Calculation

We can use the derived dependency graph to solve different business tasks by calculating the selected component(s) lineage or impact over available layers and chosen details. Business questions like: "What reports are using my data?", "Which components should be changed or tested?" or "What is the time and cost of change?" are converted to directed sub-graph navigation and calculation tasks. The following definitions add new quantitative measures to each component or node in the calculation. We use those measures in the UI to sort and select the right components for specific tasks.

Definition 4. Local Lineage Dependency % (LLD) is calculated as the ratio over the sum of the local source and target Lineage weights W_t.

$$LLD = \sum \frac{source(W_t)}{source(W_t) + target(W_t)}$$

Local Lineage Dependency 0 % means that there are no data sources detected for the object. Local Lineage Dependency 100 % means that there are no data consumers (targets) detected for the object. Local Lineage Dependency about 50 % means that there are equal numbers of weighted sources and consumers (targets) detected for the object.

Definition 5. Local Impact Dependency % (LID) is calculated as the ratio over the sum of local source and target impact weights $W(W_t,W_t)$.

$$LID = \sum \frac{source(W)}{source(W) + target(W)}$$

Local Impact Dependency 0 % means that there are no dependent data sources detected for the object. Local Dependency 100 % means that there are no dependent data consumers (targets) detected for the object. Local Impact Dependency about 50 % means that there are equal numbers of weighted dependent sources and consumers (targets) detected for the object.

Definition 6. Global Dependency Count (GDC) is the sum of all source and target Lineage and Impact relations counts (GDC=GSC+GTC).

The Global Dependency Count is a good differential metric that allows us to see clear distinctions in the dependencies of each object. We can take the GDC metric as a sort of "gravity" of the object that can be used to develop new rules, to infer the time and cost of changes of object(s) (e.g. database table, view, data loading programs or report).

A Motivating Example

The table 1 below presents four SQL DML queries as an example. The queries are parsed to abstract mappings (M1..M4) with all the available source and target tables (T1..T9). Each mapping has data transformation elements (m1,1.. m4.2), joins (j1.1..j4.1) and filter conditions (f1.1..f4.1) according to the query structure and expressions. All the source and target tables have columns (t1.1..t9.2) according the usage in query expressions. Additional transformation key-value constraints and conditions (c1, c2. c3) are extracted from the query expressions when possible. The result of the parsed and processed query text is the directed query graph.

SQL query parsed to mapping M1	SQL query parsed to mapping M2

`INSERT INTO T4(t4.1, t4.2, t4.3)`
`SELECT T1.t1.1, coalesce(T1.t1.2, '10'),`
`T1.t1.3`
`FROM T1`
`JOIN T2 ON T2.t2.1 = T1.t1.2`
`WHERE T2.t2.2 = '10'`
`AND T1.t1.2 = 'A'`
	`INSERT INTO T5 (t5.1, t5.2)`
	`SELECT T4.t4.1,`
	`coalesce(T1.t4.2, '10')`
	`FROM T4`
	`JOIN T3 ON T4.t4.1 = T3.t3.1`
	`WHERE T3.t3.2 = '10'`
	`AND (T1.t4.2 = '10' OR T1.t4.2 is null)`

SQL query parsed to mapping M3	**SQL query parsed to mapping M4**

`INSERT INTO T4(t4.1, t4.2, t4.3)`
`SELECT T6.t6.1, '20',`
`case when T6.t6.2 = 'B'`
`then 20 else 0 end`
`FROM T6`
`JOIN T7 ON T6.t6.3 = T7.t7.1`
`WHERE T6.t6.2 = 'B'`
`AND T7.t7.2 = 'B'`
	`INSERT INTO T9 (t9.1, t9.2)`
	`SELECT T4.t4.2,`
	`coalesce(T4.t4.3*100, 100)`
	`FROM T4`
	`JOIN T8 ON T8.t8.1 = T4.t4.1`
	`WHERE T4.t4.2 = '20'`

Table 1. SQL query examples that parsed to mapping M1..M4.

The following Figure 3 presents the query transformation graph with all the source and target data structures at column and table level, obtained by parsing and resolving the queries above:

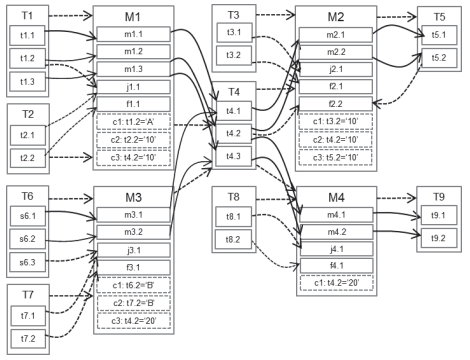


Figure. 3. Parsed query graph.

The expression weight calculation step (No 5 in Figure 2) produces probabilistic weights on scale of [0,1] to all the graph relations that can be derived from the formalized mapping structures.

From the result of rule-based reasoning (No 6 in Figure 2) we calculate the full dependency graph with all the possible inferred relations and their weights. The following component impact analysis and data lineage tasks will be handled as sub-graph navigation and calculation problems. Finally the constructed dependency graph is stored in an open schema metadata database MMX (No 8 in Figure 2) for different applications.

The component impact graph (Figure 4) solves several kinds of component and data structure dependency problems like „What happens when changed?“ or „How many reports will be broken when changed?“. The Data Lineage Graph (Figure 5) solves other types of data lineage and data flow management problems like „Where does the data come from? “, „What reports are using the data?“ or „How are the report column values combined and calculated?“.

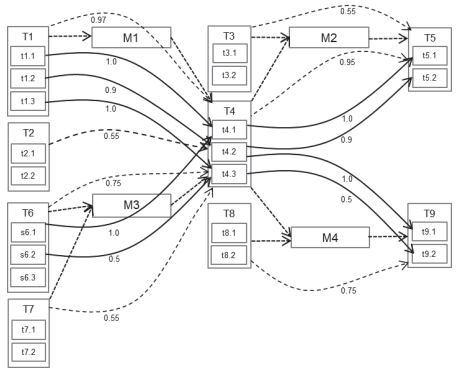


Figure 4. Components impact graph.

The component impact graph (Figure 4) illustrates the impact dependencies, directions and weights calculated by our formulas and rules for the example queries in the Table 1. The solid lines stem from the O relations, i.e. those derived directly from the query expressions. The dashed lines are based on the parent aggregation rules with the weights calculated as averages over the column level weights. In order to maintain the readability of the diagram we do not show the results of the transitivity rule applications.

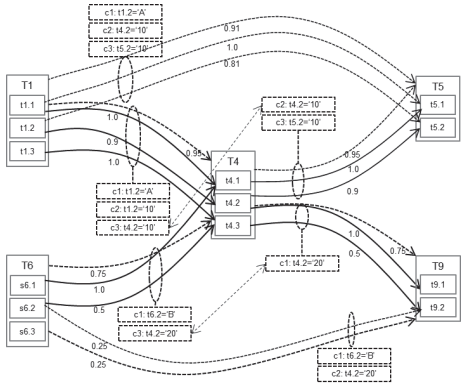


Figure 5. Data lineage graph.

The data lineage graph (Figure 5) illustrates the data lineage and flow dependencies, directions and weights calculated by the previously defined formulas and rules. The solid lines stem from the data transformation operations and weights calculated from the query expressions. The dashed lines represent transitive and aggregate relations on the column and the table level. The weights are calculated as an average (parent aggregation rule) or multiplication (transitivity) over the query expression weights. The condition notes on the figure (c1..c3) are derived from the query filters or the constant assignment list in order to add additional semantics to the transitive dependency calculation. When we connect the condition pairs from different queries that share the same meaning (e.g. c2:t.4.2='10' and c3:t.4.2='10') then we can calculate the conditional transitivity relations (from T1 to T5 and T6 to T9) that reflect the real data flows more precisely.

Real Life Case Studies

The previously described architecture and algorithms have been used to implement an integrated toolset dLineage (http://dlineage.com). Both the scanners and web-based tools of dLineage have been enhanced and tested in real-life projects and environments to support several popular DW database platforms (e.g. Oracle, Greenplum, Teradata, Vertica, PostgreSQL, MsSQL, Sybase), ETL tools (e.g. Pentaho, Oracle Data Integrator, SQL scripts and different data loading utilities) and BI tools (e.g. SAP Business Objects, Microstrategy). The dLineage dynamic visualization and graph navigation tools are implemented in Javascript using the d3.js graphics libraries.

We have tested our solution during two main case studies involving a thorough analysis of large international companies in the financial and the energy sectors. Both case studies analyzed thousands of database tables and views, tens of thousands of data loading scripts and BI reports. Those figures are far over the capacity limits of human analysts not assisted by the special tools and technologies.

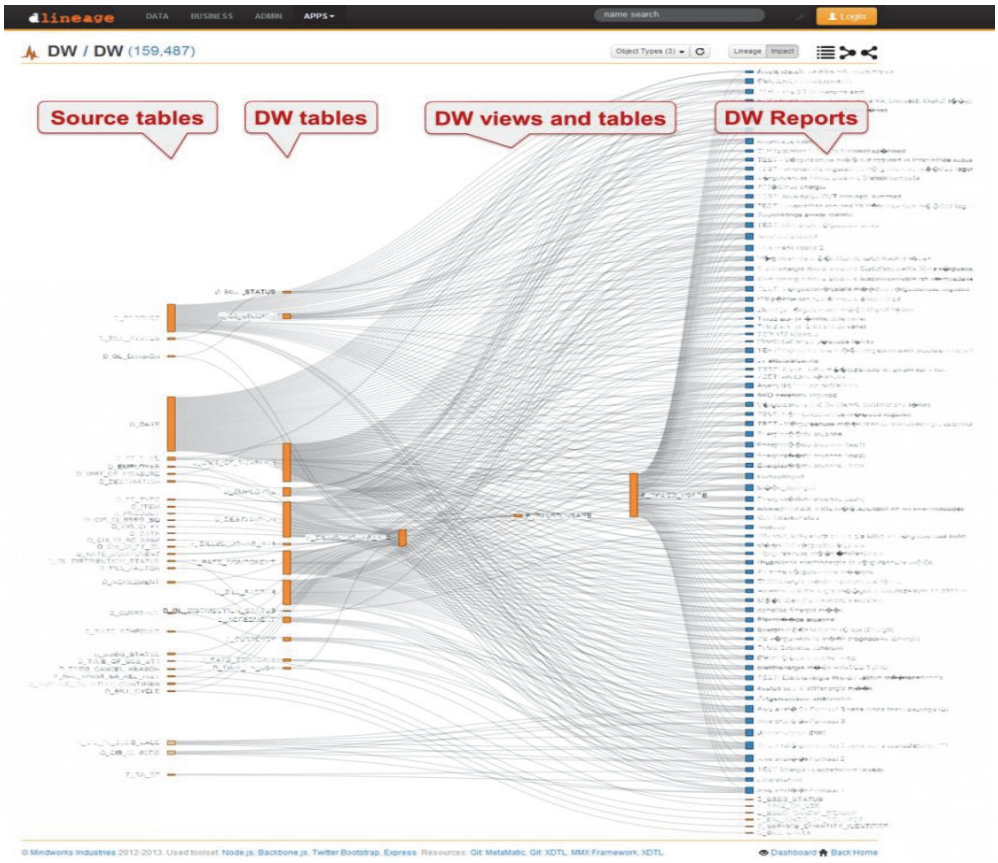


Figure 6. Data lineage graph with dependencies between DW tables, views and reports.

The real-life dependency graph examples (Figure 6 and Figure 7) illustrate automated data collection, parsing and visualization tasks implemented by one-two persons in a few days during the pilot projects. The toolkit requires only the setup and configuration tasks to be performed manually. The rest will be done by the scanners, parsers and the calculation engine. The end result consists of data flows and system component dependencies visualized in the navigable and drillable graph or table form. The result can be

viewed as a single column, table or report dependency network or the full scale overview graph with all the system dependencies - tens of thousands nodes – visible on one screen.

The Enterprise Dependency Graph example (Figure 7) is an illustration of the complex structure of dependencies between the DW storage scheme, access views and user reports. The example is generated using only 3-4 data lineage layers (sources and ETL layers are not present here) and has details at object level (not at column level). At the column and report level the full data lineage graph would be about ten times bigger and too complex to visualize in a single picture. The following graph from DW tables to views and user reports presents about 5 000 nodes (tables, views, reports) and 20 000 links (data transformations in views and queries) on a single image:

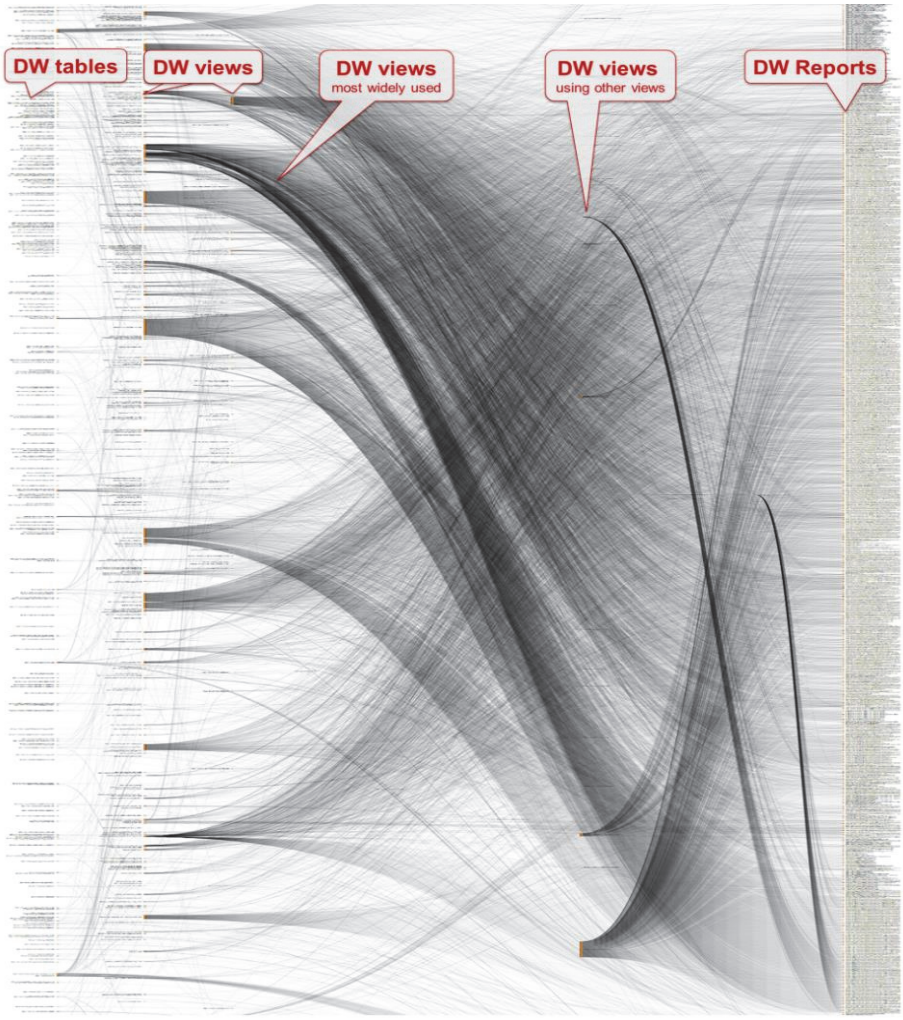


Figure 7. Data lineage graph with dependencies between DW tables, views and reports.

Conclusions and Future Work

We have presented several algorithms and techniques for quantitative impact analysis, data lineage and change management. The focus of these methods is on automated analysis of the semantics of data conversion systems followed by employing probabilistic rules for calculating chains and sums of impact estimations. The algorithms and techniques have been successfully employed in several large case studies, leading to practical data lineage and component dependency visualizations.

We are planning to continue this research by considering a more abstract, conceptual/business level in addition to the current physical/technical level of data representation.

References

Cook,D. (2010). Gold parsing system-a free, multi-programming language, parser. URL: http://www.goldparser.org.

Cui, Y., Widom, J., & Wiener, J. L. (2000). Tracing the lineage of view data in a warehousing environment. ACM Transactions on Database Systems (TODS), 25(2), 179-227.

Cui, Y., & Widom, J. (2003). Lineage tracing for general data warehouse transformations. The VLDB Journal—The International Journal on Very Large Data Bases, 12(1), 41-58.

de Santana, A. S., & de Carvalho Moura, A. M. (2004). Metadata to support transformations and data & metadata lineage in a warehousing environment. In Data Warehousing and Knowledge Discovery (pp. 249-258). Springer Berlin Heidelberg.

Fan, H., & Poulovassilis, A. (2003, November). Using AutoMed metadata in data warehousing environments. In Proceedings of the 6th ACM international workshop on Data warehousing and OLAP (pp. 86-93). ACM.

Giorgini, P., Rizzi, S., & Garzetti, M. (2008). GRAnD: A goal-oriented approach to requirement analysis in data warehouses. Decision Support Systems, 45(1), 4-21.

Luberg, A., Tammet, T., & Järv, P. (2011). Smart City: A Rule-based Tourist Recommendation System. In Information and Communication Technologies in Tourism 2011 (pp. 51-62).

Missier, P., Belhajjame, K., Zhao, J., Roos, M., & Goble, C. (2008). Data lineage model for Taverna workflows with lightweight annotation requirements. In Provenance and Annotation of Data and Processes (pp. 17-30). Springer Berlin Heidelberg.

Priebe, T., Reisser, A., & Hoang, D. T. A. (2011). Reinventing the Wheel?! Why Harmonization and Reuse Fail in Complex Data Warehouse Environments and a Proposed Solution to the Problem.

Ramesh, B., & Jarke, M. (2001). Toward reference models for requirements traceability. Software Engineering, IEEE Transactions on, 27(1), 58-93.

Reisser, A., & Priebe, T. (2009, August). Utilizing Semantic Web Technologies for Efficient Data Lineage and Impact Analyses in Data Warehouse Environments. In Database and Expert Systems Application, 2009. DEXA'09. 20th International Workshop on (pp. 59-63). IEEE.

Skoutas, D., & Simitsis, A. (2007). Ontology-based conceptual design of ETL processes for both structured and semi-structured data. International Journal on Semantic Web and Information Systems (IJSWIS), 3(4), 1-24.

Tomingas, K., Tammet, T., & Kliimask, M. (2014), Rule-Based Impact Analysis for Enterprise Business Intelligence. In Proceedings of the Artificial Intelligence Applications and Innovations (AIAI2014) conference workshop (MT4BD). Series: IFIP Advances in Information and Communication Technology, Vol. 437.

Tomingas, K., Kliimask, M., & Tammet, T. (2015). Data Integration Patterns for Data Warehouse Automation. In New Trends in Database and Information Systems II (pp. 41-55). Springer International Publishing.

Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002, November). Conceptual modeling for ETL processes. In Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP (pp. 14-21). ACM.

Wang Baldonado, M. Q., Woodruff, A., & Kuchinsky, A. (2000, May). Guidelines for using multiple views in information visualization. In Proceedings of the working conference on Advanced visual interfaces (pp. 110-119). ACM.

Widom, J. (2004). Trio: A system for integrated management of data, accuracy, and lineage. Technical Report.

Woodruff, A., & Stonebraker, M. (1997, April). Supporting fine-grained data lineage in a database visualization environment. In Data Engineering, 1997. Proceedings. 13th International Conference on (pp. 91-102). IEEE.

Publication D

Tomingas, K.; Järv, P; Tammet, T. Discovering Data Lineage from Data Warehouse Procedures. In: 8th International Joint Conference on Knowledge Discovery and Information Retrieval (KDIR 2016).

Discovering Data Lineage from Data Warehouse Procedures

Kalle Tomingas[1], Priit Järv[1] and Tanel Tammet[1]
[1] *Tallinn University of Technology, Ehitajate tee 5, Tallinn 19086 Estonia*
{kalle.tomingas, priit.jarv, tanel.tammet}@gmail.com

Keywords: Data Warehouse, Data Lineage, Dependency Analysis, Data Flow Visualization.

Abstract: We present a method to calculate component dependencies and data lineage from the database structure and a large set of associated procedures and queries, independently of actual data in the data warehouse. The method relies on the probabilistic estimation of the impact of data in queries. We present a rule system supporting the efficient calculation of the transitive closure. The dependencies are categorized, aggregated and visualized to address various planning and decision support problems. System performance is evaluated and analysed over several real-life datasets.

1 INTRODUCTION

System developers and managers are facing similar data lineage and impact analysis problems in complex data integration, business intelligence and data warehouse environments where the chains of data transformations are long and the complexity of structural changes is high. The management of data integration processes becomes unpredictable and the costs of changes can be very high due to the lack of information about data flows and the internal relations of system components. Important contextual relations are encoded into data transformation queries and programs (SQL queries, data loading scripts, etc.). Data lineage dependencies are spread between different systems and frequently exist only in program code or SQL queries. This leads to unmanageable complexity, lack of knowledge and a large amount of technical work with uncomfortable consequences like unpredictable results, wrong estimations, rigid administrative and development processes, high cost, lack of flexibility and lack of trust.

We point out some of the most important and common questions for large DW which usually become a topic of research for system analysts and administrators:

- Where does the data come or go to in/from a specific column, table, view or report?
- When was the data loaded, updated or calculated in a specific column, table, view or report?
- Which components (reports, queries, loadings and structures) are impacted when other components are changed?
- Which data, structure or report is used by whom and when?
- What is the cost of making changes?
- What will break when we change something?

The ability to find ad-hoc answers to many day to day questions determines not only the management capabilities and the cost of the system, but also the price and flexibility of making changes.

The goal of our research is to develop reliable and efficient methods for automatic discovery of component dependencies and data lineage from the database schemas, queries and data transformation components by automated analysis of actual program code. This requires probabilistic estimation of the measure of dependencies and the aggregation and visualization of the estimations.

2 RELATED WORK

Impact analysis, traceability and data lineage issues are not new. A good overview of the research activities of the last decade is presented in an article by (Priebe, 2011). We can find various research approaches and published papers from the early 1990's with methodologies for software traceability (Ramesh, 2001). The problem of data lineage tracing in data warehousing environments has been formally

founded by Cui and Widom (Cui, 2000; Cui 2003). Overview of data lineage and data provenance tracing studies can be found in book by Cheney et al. (Cheney, 2009). Data lineage or provenance detail levels (e.g. coarse-grained vs fine-grained), question types (e.g why-provenance, how-provenance and where-provenance) and two different calculation approaches (e.g. eager approach vs lazy approach) discussed in multiple papers (Tan, 2007; Benjelloun, 2006) and formal definitions of why-provenance given by Buneman et al. (Buneman, 2001). Other theoretical works for data lineage tracing can be found in (Fan, 2003; Giorgini, 2008). Fan and Poulovassilis developed algorithms for deriving affected data items along the transformation pathway [6]. These approaches formalize a way to trace tuples (resp. attribute values) through rather complex transformations, given that the transformations are known on a schema level. This assumption does not often hold in practice. Transformations may be documented in source-to-target matrices (specification lineage) and implemented in ETL tools (implementation lineage). Woodruff and Stonebraker create solid base for the data-level and the operators processing based the fine-grained lineage in contrast to the metadata based lineage calculation in their research paper (Woodruff, 1997).

Other practical works that are based on conceptual models, ontologies and graphs for data quality and data lineage tracking can be found in (Skoutas, 2007; Tomingas, 2014; Vassiliadis, 2002; Widom, 2004). De Santana proposes the integrated metadata and the CWM metamodel based data lineage documentation approach (de Santana, 2004). Tomingas et al. employ the Abstract Mapping representation of data transformations and rule-based impact analysis (Tomingas, 2014).

Priebe et al. concentrates on proper handling of specification lineage, a huge problem in large-scale DWH projects, especially in case different sources have to be consistently mapped to the same target (Priebe, 2011). They propose a business information model (or conceptual business glossary) as the solution and a central mapping point to overcome those issues.

Scientific workflow provenance tracking is closely related to data lineage in databases. The distinction is made between coarse-grained, or schema-level, provenance tracking (Heinis, 2008) and fine-grained or data instance level tracking (Missier, 2008). The methods of extracting the lineage are divided to physical (annotation of data by Missier et al.) and logical, where the lineage is derived from the graph of data transformations (Ikeda, 2013).

[1] http://www.goldparser.org/

In the context of our work, efficiently querying of the lineage information after the provenance graph has been captured, is of specific interest. Heinis and Alonso present an encoding method that allows space-efficient storage of transitive closure graphs and enables fast lineage queries over that data (Heinis, 2008). Anand et al. propose a high level language QLP, together with the evaluation techniques that allow storing provenance graphs in a relational database (Anand, 2010).

3 WEIGHT ESTIMATION

The inference method of the data flow and the impact dependencies that presented in this paper is part of a larger framework of a full impact analysis solution. The core functions of the system architecture are built upon the following components presented in the Figure 1 and described in detail in our previous works (Tomingas, 2014; Tomingas, 2015).

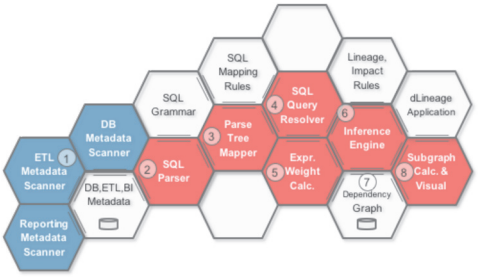


Figure 1: Impact analysis system architecture components.

The core functions of the system architecture are built upon the following components in the Figure 1:

1. Scanners collect metadata from different systems that are part of DW data flows (DI/ETL processes, data structures, queries, reports etc.).
2. The SQL parser is based on customized grammars, GoldParser[1] parsing engine and the Java-based XDTL engine.
3. The rule-based parse tree mapper extracts and collects meaningful expressions from the parsed text, using declared combinations of grammar rules and parsed text tokens.
4. The query resolver applies additional rules to expand and resolve all the variables, aliases, sub-query expressions and other SQL syntax structures which encode crucial information for data flow construction.

5. The expression weight calculator applies rules to calculate the meaning of data transformation, join and filter expressions for impact analysis and data flow construction.

6. The probabilistic rule-based reasoning engine propagates and aggregates weighted dependencies.

7. The open-schema relational database using PostgreSQL for storing and sharing scanned, calculated and derived metadata.

8. The directed and weighted sub-graph calculations, and visualization web based UI for data lineage and impact analysis applications.

In the stages preceding the impact estimation, inference and aggregation the data structure transformations are parsed and extracted from queries and stored as formalized, declarative mappings in the system.

To add additional quantitative measures to each column transformation or column usage in the join and filter conditions we evaluate each expression and calculate the transformation and filter weights for those.

Definition 1. The column transformation weight Wt is based on the similarity of each source column and column transformation expression: the calculated weight expresses the source column transfer rate or strength. The weight is calculated on scale [0,1] where 0 means that the data is not transformed from source (e.g. constant assignment in a query) and 1 means that the source is copied to the target directly, ie. no additional column transformations are detected.

Definition 2. The column filter weight Wf is based on the similarity of each filter column in the filter expression where the calculated weight expresses the column filtering strength. The weight is calculated on the scale [0,1] where 0 means that the column is not used in the filter and 1 means that the column is directly used in the filter predicate, ie. no additional expressions are involved.

The general column weight W algorithm in each expression for Wt and Wf components is calculated as a column count ratio over all the expression component counts (e.g. column count, constant count, function count, predicate count).

$$W = \frac{IdCount}{IdCount + FncCount + StrCount + NbrCount + PrdCount}$$

The counts are normalized using the FncList evaluation over a positive function list (e.g. CAST, ROUND, COALESCE, TRIM etc.). If the FncList member is in a positive function list, then the normalization function reduces the according component count by 1 to pay a smaller price in case the function used does not have a significant impact to column data.

Definition 3. A primitive data transformation operation is a data transformation between a source column X and a target column Y in a transformation set M (mapping or query) having the expression similarity weight Wt.

Definition 4. The column X is a filter condition in a transformation set M with the filter weight Wf if the column is part of a JOIN clause or WHERE clause in the queries corresponding to M.

4 RULE SYSTEM AND DEPENDENCY CALCULATION

The primitive transformations captured from the source databases form a graph G_O with nodes N representing database objects and edges E_O representing primitive transformations (see Definition 3). We define relations $X: E_O \rightarrow N$ and $Y: E_O \rightarrow N$ connecting edges to source nodes and target nodes, respectively. We define label relations $M: E_O \rightarrow \{\{m\} \mid m \; is \; a \; transformation \; identifier\}$ and $W: E_O \rightarrow [0,1]$. Formally, this graph is an edge-labeled directed multigraph.

In the remainder of the article, we will use the following intuitive notation: *e.X* and *e.Y* to denote source and target objects of a transformation (formally, $X(e)$ and $Y(e)$). *e.M* is the set of source transformations ($M(e)$). *e.W* is the weight assigned to the edge ($W(e)$).

The knowledge inferred from the primitive transformations forms a graph $G_L = (N, E_L)$ where E_L is the set of edges e that represent data flow (lineage). We define relations *X*, *Y*, *M* and *W* the same way as with the graph G_O and use the *e.R* notation where *R* is one of the relations {*X*, *Y*, *M*, *W*}.

Additionally, we designate the graph $G_I = (N, E_I \cup E_L)$ to represent the impact relations between database components. It is a superset of G_L where E_L is the set of additional edges inferred from column usage in filter expressions.

4.1 The Propagation Rule System

First, we define the rule to map the primitive data transformations to our knowledge base. This rule includes aggregation of multiple edges between pairs of nodes.

Let $E_{x,y} = \{e \in E_O \mid e.X = x, e.Y = y\}$ be the set of edges connecting nodes *x*, *y* in the graph G_O.

$\forall x, y \in N \; E_{x,y} \neq \emptyset \implies \exists e' \in E_L$ such that (R1),

$e'.X = x \wedge e'.Y = y$ (R1.1)

$e'.M = \cup_{e \in E_{x,y}} e.M$ (R1.2)

$e'.W = max\{e.W \mid e \in E_{x,y}\}$ (R1.3)

An inference using this rule should be understood as ensuring that our knowledge base satisfies the rule. From an algorithmic perspective, we create edges *e'* into the set E_L until R1 is satisfied.

Definition 5. The predicate *Parent(x, p)* is true if node *p* is the parent of node *x* in the database schema.

Filter conditions are mapped to edges in the impact graph G_I.

Let $F_{M,p} = \{x \mid Parent(x,p) \wedge x \text{ is a filter in } M\}$ be the set of nodes that are filter conditions for the mapping *M* with parent *p*. Let $T_{M,p'} = \{x \mid Parent(x,p') \wedge x \text{ is target in } M\}$ be the set of nodes that represent the target columns of mapping *M*. To assign filter weights to columns, we define the function $W_f: N \rightarrow [0,1]$.

$\forall p,p' \in N\ F_{M,p} \neq \emptyset \wedge T_{M,p'} \neq \emptyset \Rightarrow \exists e' \in E_I$ (R2), such that

$e'.X = p \wedge e'.Y = p'$ (R2.1)

$e'.M = M$ (R2.2)

$e'.W = \frac{max\{W_f(x) \mid x \in F_{M,p}\} + max\{W_f(x) \mid x \in T_{M,p'}\}}{2}$ (R2.3)

The primitive transformations mostly represent column-level (or equivalent) objects that are adjacent in the graph (meaning, they appear in the same transformation or query and we have captured the data flow from one to another). The same applies to impact information inferred from filter conditions. From this knowledge, the goal is to additionally:

- propagate information through the database structure upwards, to view data flows on a more abstract level (such as, table or schema level)
- calculate the dependency closure to answer lineage queries

Unless otherwise stated, we treat the graphs G_L and G_I similarly from this point. It is implied that the described computations are performed on both of them. The set *E* refers to the edges of either of those graphs.

Let $E_{p,p'} = \{e \in E \mid Parent(e.X,p) \wedge Parent(e.Y,p')\}$ be the set of edges where the source nodes share a common parent *p* and the target nodes share a common parent *p'*.

$\forall p,p' \in N\ E_{p,p'} \neq \emptyset \Rightarrow \exists e' \in E$ (R3), such that

$e'.X = p \wedge e'.Y = p'$ (R3.1)

$e'.M = \cup_{e \in E_{p,p'}} e.M$ (R3.2)

$e'.W = \frac{\sum_{e \in E_{p,p'}} e.W}{|E_{p,p'}|}$ (R3.3)

4.2 The Dependency Closure

Online queries from the dataset require finding the data lineage of a database item without long computation times. For displaying both the lineage and impact information, we require that all paths through the directed graph that include a selected component are found. These paths form a connected subgraph. Further manipulation (see Section 4.3) and data display is then performed on this subgraph.

There are two principal techniques for retrieving paths through a node (Heinis, 2008):

- connect the edges recursively, forming the paths at query time. This has no additional storage requirements, but is computationally expensive
- store the paths in materialized form. The paths can then be retrieved without recursion, which speeds up the queries, but the materialized transitive closure may be expensive to store.

Several compromise solutions that seek to both efficiently store and query the data have been published (Heinis, 2008; Anand, 2010). In general, the transitive closure is stored in a space efficient encoding that can be expanded quickly at the query time.

We have incorporated elements from the pointer based technique introduced in (Anand, 2010). The paths are stored in three relations: `Node(N1,P_dep,P_depc)`, `Dep(P_dep,N2)` and `DepC(P_depc,P_dep)`. Immediate dependen-cies of a node are stored in the `Dep` relation, with the pointer `P_dep` in the `Node` relation referring to the dependency set. The full transitive dependency closure is stored in the `DepC` relation by storing the union of the pointers to all of the immediate dependency sets of nodes along the paths leading to a selected node.

We can define the dependency closure recursively as follows. Let D^*_k be the dependency closure of node *k*. Let D_k be the set of immediate dependencies such that $D_k = \{j \mid e \in E, e.X = j, e.Y = k\}$.

If $D_k = \emptyset$ then $D^*_k = \emptyset$.

Else if $D_k \neq \emptyset$ then $D^*_k = D_k \cup (\cup_{j \in D_k} D^*_j)$.

The successors S_j (including non-immediate) of a node j are found as follows: $S_j = \{k \mid j \in D^*_k\}$

The materialized storage of the dependency closure allows building the successor set cheaply, so it does not need to be stored in advance. Together with the dependency closure they form the connected maximal subgraph that includes the selected node.

We put the emphasis on the fast computation of the dependency closure with the requirement that the lineage graph is sparse ($|I| \sim |N|$). We have omitted the more time-consuming redundant subset and subsequence detection techniques of Anand et al. (Anand, 2009). The subset reduction has $O(|D|^3)$ time complexity which is prohibitively expensive if the number of initial unique dependency sets $|D|$ is

on the order of 10^5 as is the case in our real world dataset.

The dependency closure is computed by:

1. Creating a partial order L of the nodes in the directed graph G_I. If the graph is cyclic then we need to transform it to a DAG by deleting an edge from each cycle. This approach is viable, if the graph contains relatively few cycles. The information lost by deleting the edges can be restored at a later stage, but this process is more expensive than computing the closure on a DAG.
2. Creating the immediate dependency sets for each node using the duplicate-set reduction algorithm (Anand, 2009).
3. Building the dependency closures for each node using the partial order L, ensuring that the dependency sets are available when they are needed for inclusion in the dependency closures of successor nodes (Algorithm 1).
4. If needed, restoring deleted cyclic edges and incrementally adding dependencies that are carried by those edges using breadth-first search in the direction of the edges.

Algorithm 1. Building the pointer-encoded dependency closure:

```
Input: L  - partial order on GI;
 {Dk|k∈N} - immediate dependency sets
  Output: D*k - dependency closures
                  for each node k∈N
for node k in L:
    D*k = {Dk}
    for j in Dk:
        D*k = D*k ∪ D*j
```

This algorithm has linear time complexity $O(|N| + |E|)$ if we disregard the duplicate set reduction. To reduce the required storage, if $D^*_j = D^*_k$ for any $j \neq k$ then we may replace one of them with a pointer to the other. The set comparison increases the worst case time complexity to $O(|N|^2)$.

To extract the nodes along the paths that go through a selected node *N*, one would use the following queries:

```
select Dep.N2  --predecessor nodes
from Node, DepC, Dep
where Dep.P_dep = DepC.P_dep
and DepC.P_depc = Node.P_depc
and Node.N1 = N

select Node.N1 --successor nodes
from Node, DepC, Dep
where Node.P_depc = DepC.P_depc
and DepC.P_dep = Dep.P_dep
and Dep.N2 = N
```

4.3 Visualization of the Lineage and Impact Graphs

The visualization of the connected subgraph corresponding to a node *j* is created by fetching the path nodes $P_j = D^*_j \cup S_j$ and the edges along those paths $E_j = \{e \in E \mid e.X \in P_j \wedge e.Y \in P_j\}$ from the appropriate dependency graph (impact or lineage). The graphical representation allows filtering a subset of nodes in the application, by node type, although the filtering technique discussed here is generic and permits arbitrary criteria. Any nodes not included in graphical display are replaced by transitive edges bypassing these nodes to maintain the connectivity of the dependencies in the displayed graph.

Let $G_j = (P_j, E_j)$ be the connected sub graph for the selected node *j*. We find the partial transitive graph G_j' that excludes the filtered nodes P_{filt} as follows (Algorithm 2):

Algorithm 2. Building the filtered subgraph with transitive edges.

```
Input: Gj, Pfilt
Output: Gj' = (Pj', Ej')
Ej' = Ej
Pj' = ∅
for node n in Pj:
 if n ∈ Pfilt:
   for e in {e ∈ Ej'| e.Y = n}:
     for e' in {e' ∈ Ej'| e'.X = n}:
      create new edge e'' ( e''.X = e.X,
e''.Y = e'.Y, e''.W = e.W * e''.W)
       Ej' = Ej' ∪ {e''}
     Ej'= Ej' \ {e}
   for e' in {e' ∈ Ej'| e'.X = n}:
     Ej' = Ej' \ {e'}
 else:
   Pj' = Pj' ∪ {n}
```

This algorithm has the time complexity of O(|Pj| + |Ej|) and can be performed on demand when the user changes the filter settings. This extends to large dependency graphs with the assumption that |GJ| << |G|.

4.4 The Semantic Layer Calculation

The semantic layer is a set of visualizations and associated filters to localize the connected subgraph of the expected data flows for the current selected node. All the connected nodes and edges in the semantic layer share the overlapping filter predicate conditions or data production conditions that are extracted during the edge construction to indicate not only possible data flows (based on connections in initial query graph), but only expected and

probabilistic data flows. The main idea of the semantic layer is to narrow down all the possible and expected data flows over all the connected graph nodes by cutting down unlikely or disallowed connections in graph, which is based on the additional query filters and the semantic interpretation of filters and calculated transformation expression weights. The semantic layer of the data lineage graph will hide irrelevant and/or highlight the relevant graph nodes and edges, depending on the user choice and interaction.

This has a significant impact when the underlying data structures are abstract enough and the independent data flows store and use independent horizontal slices of data. The essence of the semantic layer is to use the available query and schema information to estimate the row level data flows without any additional row level lineage information which would be unavailable on schema level and expensive or impossible to collect on the row level.

The visualization of the semantically connected subgraph corresponding to node j is created by fetching the path nodes $P_j = D^*_j \cup S_j$ and the edges along those paths $E_j = \{e \in E | e.X \in P_j \wedge e.Y \in P_j\}$ from the appropriate dependency graph (impact or lineage). Any nodes not included in the semantic layer are removed or visually muted (by changing the color or opacity) and the semantically connected subgraph is returned or visualized by the user interface.

Let $G_j = (P_j, E_j)$ be the connected subgraph for the selected node j where $GD_j = (D_j, ED_j)$ is the predecessor subgraph and $GS_j = (S_j, ES_j)$ is the successor subgraph according to the selected node j. We calculate the data flow graph $G_j{}'$ that is the union of the semantically connected predecessors $GD_j' = (D_j, ED_j)$ and successor subgraphs $GS_j' = (S_j, ES_j)$. The semantic layer calculation is based on the selected node filter set F_j and calculated separately for back (predecessor) and forward (successors) directions by the recursive algorithm (Algorithm 3):

Algorithm 3. Building the semantic layer subgraph using predecessor and successor functions recursively.

```
Function: Predecessors
Input: nj, Fj , GDj, GD'j Wmin
Output: GDj' = (Dj', EDj')
Fn = ∅
if Dj' = ∅ then:
 Dj' = Dj' ∪ nj
for edge e in {e ∈ EDj | e.Y = nj
 Fn = ∅
 if Fj != ∅:
   for filter f in e.{F}:
     for filter fj in Fj:
      if f.Key = fj.Key & f.Val ∩ fj.Val:
           new filter fn (fn.Key=f.Key,
fn.Val=f.Val, fn.Wgt=f.Wgt*fj.Wgt)
         Fn = Fn ∪ fn
 else:
   Fn = Fn ∪ e.{F}
 if Fn != ∅ & e.W >= Wmin :
   Dj' = Dj' ∪ e.X
   EDj' = EDj' ∪ e
   GDj'=Predecessors
(e.X,Fn,GDj,GD'j,Wmin)
return GDj'

Function: Successors
Input: nj, Fj , GSj, GS', Wmin
Output: GSj' = (Sj', ESj')
Fn = ∅
if Sj' = ∅:
 Sj' = Sj' ∪ nj
for edge e in {e ∈ ESj | e.X = nj}:
 Fn = ∅
 if Fj != ∅ then:
   for filter f in e.{F}:
     for filter fj in Fj:
      if f.Key = fj.Key & f.Val ∩ fj.Val:
           new filter fn (fn.Key=f.Key,
fn.Val=f.Val, fn.Wgt=f.Wgt*fj.Wgt)
         Fn = Fn ∪ fn
 else:
   Fn = Fn ∪ e.{F}
 if Fn != ∅ & e.W >= Wmin :
   Sj' = Sj' ∪ e.Y
   ESj' = ESj' ∪ e
   GSj'=Predecessors(e.Y,Fn
,GSj,GSj',Wmin)
return GSj'
```

The final semantic layer subgraph is an union of the recursively constructed predecessor GD_j' and successor GS_j' graphs: `Gj' = GDj' ∪ GSj'`

4.5 Dependency Score Calculation

We use the derived dependency graph to solve different business tasks by calculating the selected component(s) lineage or impact over available layers and chosen details. Business questions like: “What reports are using my data?”, “Which components should be changed or tested?” or “What is the time and cost of change?” are converted to directed subgraph navigation and calculation tasks. The following definitions add new quantitative measures to each component or node in the calculation. We use those measures in the user interface to sort and select the right components for specific tasks.

Definition 6. Local Lineage Dependency % (LLD) is calculated as the ratio over the sum of the local source and target lineage weights W_t.

$$LLD = \frac{\sum \text{source}(W_t)}{\sum \text{source}(W_t) + \sum \text{target}(W_t)}$$

Local Lineage Dependency 0 % means that there are no data sources detected for the object. Local Lineage Dependency 100 % means that there are no data consumers (targets) detected for the object. Local Lineage Dependency about 50 % means that there are equal numbers of weighted sources and consumers (targets) detected for the object.

Definition 7. Local Impact Dependency % (LID) is calculated as the ratio over the sum of local source and target impact weights $W(W_t,W_t)$.

$$LLD = \frac{\sum \text{source}(W)}{\sum \text{source}(W) + \sum \text{target}(W)}$$

5 CASE STUDIES

The previously described algorithms have been used to implement an integrated toolset. Both the scanners and the visualization tools have been enhanced and tested in real-life projects and environments to support several popular data warehouse platforms (e.g. Oracle, Greenplum, Teradata, Vertica, PostgreSQL, MsSQL, Sybase), ETL tools (e.g. Informatica, Pentaho, Oracle Data Integrator, SSIS, SQL scripts and different data loading utilities) and business intelligence tools (e.g. SAP Business Objects, Microstrategy, SSRS). The dynamic visualization and graph navigation tools are implemented in Javascript using the d3.js graphics libraries.

Current implementation has rule system which is implemented in PostgreSQL database using SQL queries for graph calculation (rules 1-3 in section 4.1) and specialized tables for graph storage. The DB and UI interaction tested with the specialized pre-calculated model (see section 4.2) but also with the recursive queries without special storage and pre calculations. The algorithms for interactive transitive calculations (see sections 4.3) and semantic layer calculation (see section 4.4) are implemented in Javascript and works in browser for small and local subgraph optimization or visualization. Due to space limitations we do not stop here for discussion and the details of case studies. Technical details and more information can be found on our dLineage[2] online demo site. We present different datasets processing and performance analysis in the next section and illustrate the application and algorithms with the graph visualizations technique (section 5.2).

5.1 Performance Evaluation

We have tested our solution in several real-life case studies involving a thorough analysis of large international companies in the financial, utilities, governance, telecom and healthcare sectors. The case studies analyzed thousands of database tables and views, tens of thousands of data loading scripts and BI reports. Those figures are far over the capacity limits of human analysts not assisted by the special tools and technologies.

The following six different datasets with varying sizes have been used for our system performance evaluation. The datasets DS1 to DS6 represent data warehouse and business intelligence data from different industry sectors and is aligned according to the dataset size (Table 1). The structure and integrity of the datasets is diverse and complex, hence we have analyzed the results at a more abstract level (e.g. the number of objects and processing time) to evaluate the system performance under different conditions.

Table 1: Evaluation of processed datasets with different size, structure and integrity levels.

	DS1	DS2	DS3	DS4	DS5	DS6
Scanned objects	**1,341,863**	**673,071**	**132,588**	**120,239**	**26,026**	**2,369**
DB objects	43,773	179,365	132,054	120,239	26,026	2,324
ETL objects	1,298,090	361,438	534	0	0	45
BI objects	0	132,268	0	0	0	0
Scan time (min)	**114**	**41**	**17**	**33**	**6**	**0**
Parsed scripts	6,541	8,439	7,996	8,977	1184	495
Parsed queries	48,971	13,946	11,215	14,070	1544	635
Parse success rate (%)	96	98	96	92	88	100
Parse/resolve perform..(queries/sec)	3.6	2.5	26.0	12.1	4.1	6.3
Parse/resolve time (min)	**30**	**57**	**5**	**12**	**5**	**1**
Graph nodes	73,350	192,404	24,878	17,930	360	1,930
Graph links	95,418	357,798	24,823	15,933	330	2,629
Graph processing time (min)	36	62	14	15	6	2
Total processing time (min)	**150**	**103**	**31**	**48**	**12**	**2**

The biggest dataset DS1 contained a big set of Informatica ETL package files, a small set of connected DW database objects and no business intelligence data. The next dataset DS2 contained a data warehouse, SQL scripts for ETL loadings and a SAP Business Object for reporting for business intelligence. The DS3 dataset contained a smaller subset of the DW database (MsSql), SSIS ETL loading packages and SSRS reporting for business intelligence. The DS4 dataset had a subset of the data warehouse (Oracle) and data transformations in stored procedures (Oracle). The DS5 dataset is a

[2] http://www.dlineage.com/

similar but much smaller to DS4 and is based on the Oracle database and stored procedures. The DS6 dataset had a small subset of a data warehouse in Teradata and data loading scripts in the Teradata TPT format.

The datasets size, internal structure and processing time are visible in Figure 2 where longer processing time of DS4 is related to very big Oracle stored procedure texts and loading of those to database.

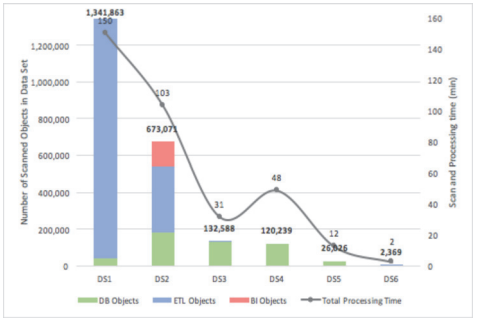


Figure 2: Datasets size and structure compared to overall processing time.

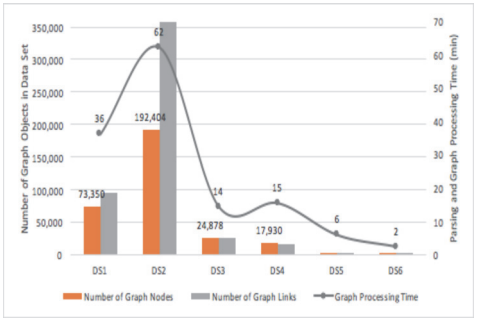


Figure 3: Calculated graph size and structure compared to the graph data processing time.

The initial dataset and the processed data dependency graphs have different graph structures (see Figure 3) that do not correspond necessarily to the initial dataset size. The DS2 has a more integrated graph structure and a higher number of connected objects (Figure 4) than the DS1. At the same time the DS1 has about two times bigger initial row data size than the DS2.

We have additionally analyzed the correlation of the processing time and the dataset size (see Figure 4 and Figure 5) and showed that the growth of the execution time follows the same linear trend as the size and complexity growth. The data scan time is related mostly to the initial dataset size. The query parsing, resolving and graph processing time also depends mainly on the initial data size, but also on the calculated graph size (Figure 4). The linear correlation between the overall system processing time (seconds) and the dataset size (object count) can be seen in Figure 5.

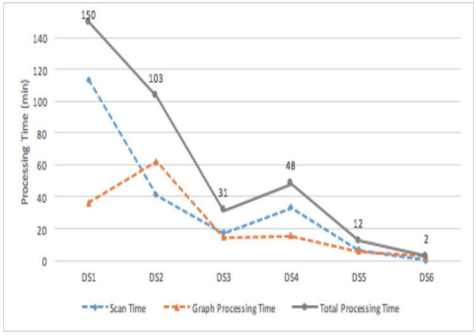


Figure 4: Dataset processing time with two main sub-components.

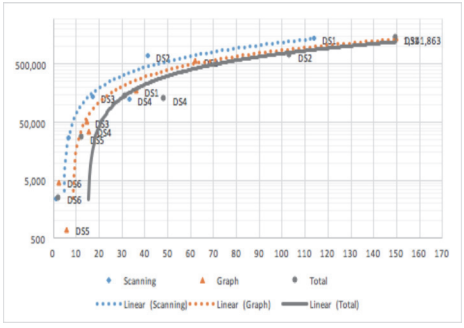


Figure 5: Dataset size and processing time correlation with linear regression (semi-log scale).

5.2 Dataset Visualization

The Enterprise Dependency Graph examples (Figures 6-8) are an illustration of the complex structure of dependencies between the DW storage scheme, access views and user reports. The example is generated using data warehouse and business intelligence lineage layers. The details are at the database and reporting object level, not at column level. At the column and report level the full data lineage graph would be about ten times bigger and too complex to visualize in a single picture. The

following graph from the data warehouse structures and user reports presents about 50,000 nodes (tables, views, scripts, queries, reports) and about 200,000 links (data transformations in views and queries) on a single image (see Figure 6).

The real-life dependency graph examples illustrate the automated data collection, parsing, resolving, graph calculation and visualization tasks implemented in our system. The system requires only the setup and configuration tasks to be performed manually. The rest will be done by the scanners, parsers and the calculation engine.

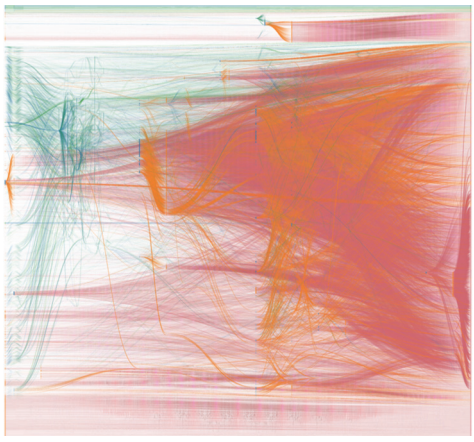


Figure 6: Data flows (blue,red) and control flows (green,yellow) between tables, views and reports.

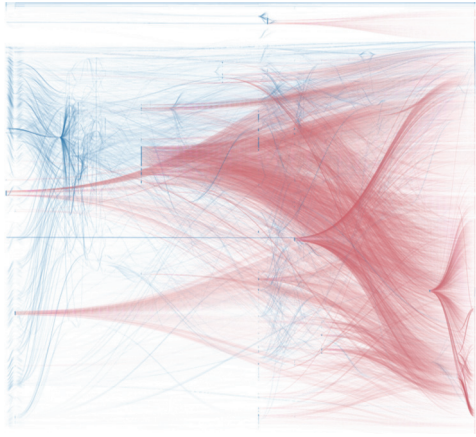


Figure 7: Data flows between tables, views (blue) and reports (red).

Figure 8: Control flows in scripts, queries (green) and reporting queries (yellow) are connecting tables, views and reports.

The end result consists of data flows and system component dependencies visualized in the navigable and drillable graph or table form. The result can be viewed as a local subgraph with fixed focus and suitable filter set to visualize data lineage path from any sources to single report with click and zoom navigation features. The big picture of the dependency network gives the full scale overview graph of the organization's data flows. It allows to see us possible architectural, performance or security problems.

6 CONCLUSIONS

We have presented several algorithms and techniques for quantitative impact analysis, data lineage and change management. The focus of these methods is on automated analysis of the semantics of data conversion systems followed by employing probabilistic rules for calculating chains and sums of impact estimations. The algorithms and techniques have been successfully employed in several large case studies, leading to practical data lineage and component dependency visualizations. We continue this research by performance measurement with the number of different big datasets, to present practical examples and draw conclusion of our approach.

We also considering a more abstract, conceptual and business level approach in addition to the current physical/technical level of data lineage representation and automation.

ACKNOWLEDGEMENTS

The research has been supported by EU through European Regional Development Fund.

REFERENCES

Anand, M. K., Bowers, S., McPhillips, T., & Ludäscher, B. (2009, March). Efficient provenance storage over nested data collections. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (pp. 958-969). ACM.

Anand, M. K., Bowers, S., & Ludäscher, B. (2010, March). Techniques for efficiently querying scientific workflow provenance graphs. In *EDBT* (Vol. 10, pp. 287-298).

Benjelloun, O., Sarma, A. D., Hayworth, C., & Widom, J. (2006). An introduction to ULDBs and the Trio system. IEEE Data Engineering Bulletin, March 2006.

Buneman, P., Khanna, S., & Wang-Chiew, T. (2001). Why and where: A characterization of data provenance. In Database Theory—ICDT 2001 (pp. 316-330). Springer Berlin Heidelberg.

Cheney, J., Chiticariu, L., & Tan, W. C. (2009). Provenance in databases: Why, how, and where. Now Publishers Inc.

Cui, Y., Widom, J., & Wiener, J. L. (2000). Tracing the lineage of view data in a warehousing environment. ACM Transactions on Database Systems (TODS), 25(2), 179-227.

Cui, Y., & Widom, J. (2003). Lineage tracing for general data warehouse transformations. The VLDB Journal—The International Journal on Very Large Data Bases, 12(1), 41-58.

de Santana, A. S., & de Carvalho Moura, A. M. (2004). Metadata to support transformations and data & metadata lineage in a warehousing environment. In Data Warehousing and Knowledge Discovery (pp. 249-258). Springer Berlin Heidelberg.

Fan, H., & Poulovassilis, A. (2003, November). Using AutoMed metadata in data warehousing environments. In Proceedings of the 6th ACM international workshop on Data warehousing and OLAP (pp. 86-93). ACM.

Giorgini, P., Rizzi, S., & Garzetti, M. (2008). GRAnD: A goal-oriented approach to requirement analysis in data warehouses. Decision Support Systems, 45(1), 4-21.

Heinis, T., & Alonso, G. (2008, June). Efficient lineage tracking for scientific workflows. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1007-1018). ACM.

Ikeda, R., Das Sarma, A., & Widom, J. (2013, April). Logical provenance in data-oriented workflows?. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on* (pp. 877-888). IEEE.

Missier, P., Belhajjame, K., Zhao, J., Roos, M., & Goble, C. (2008). Data lineage model for Taverna workflows with lightweight annotation requirements. In Provenance and Annotation of Data and Processes (pp. 17-30). Springer Berlin Heidelberg.

Priebe, T., Reisser, A., & Hoang, D. T. A. (2011). Reinventing the Wheel?! Why Harmonization and Reuse Fail in Complex Data Warehouse Environments and a Proposed Solution to the Problem.

Ramesh, B., & Jarke, M. (2001). Toward reference models for requirements traceability. Software Engineering, IEEE Transactions on, 27(1), 58-93.

Reisser, A., & Priebe, T. (2009, August). Utilizing Semantic Web Technologies for Efficient Data Lineage and Impact Analyses in Data Warehouse Environments. In Database and Expert Systems Application, 2009. DEXA'09. 20th International Workshop on (pp. 59-63). IEEE.

Skoutas, D., & Simitsis, A. (2007). Ontology-based conceptual design of ETL processes for both structured and semi-structured data. International Journal on Semantic Web and Information Systems (IJSWIS), 3(4), 1-24.

Tan, W. C. (2007). Provenance in Databases: Past, Current, and Future. IEEE Data Eng. Bull., 30(4), 3-12.

Tomingas, K., Tammet, T., & Kliimask, M. (2014), Rule-Based Impact Analysis for Enterprise Business Intelligence. In Proceedings of the Artificial Intelligence Applications and Innovations (AIAI2014) conference workshop (MT4BD). Series: IFIP Advances in Information and Communication Technology, Vol. 437.

Tomingas, K., Kliimask, M., & Tammet, T. (2015). Data Integration Patterns for Data Warehouse Automation. In New Trends in Database and Information Systems II (pp. 41-55). Springer International Publishing.

Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002). Conceptual modeling for ETL processes. In Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP (pp. 14-21). ACM.

Widom, J. (2004). Trio: A system for integrated management of data, accuracy, and lineage. Technical Report.

Woodruff, A., & Stonebraker, M. (1997). Supporting fine-grained data lineage in a database visualization environment. In Data Engineering, 1997. Proceedings. 13th International Conference on (pp. 91-102). IEEE.

CURRICULUM VITAE

Personal data

Name: Kalle Tomingas
Date of birth: 22.08.1973
Place of birth: Pärnu, Estonia
Citizenship: Estonia

Contact data

Phone: +372 5040568
E-mail: kalle.tomingas@gmail.com

Education

2008 – 2018 Tallinn University of Technology, PhD
1991 – 2000 Tallinn University of Technology, MSC
1989 – 1991 Pärnu Ülejõe Gymnasium, Highschool

Language competence

English	Fluent
Russian	Communication
Estonian	Native language

Professional employment

2017– … Orion Information Governance, Chief Data Scientist
2005–2017 Mindworks Industries, Consultant
2011–2015 ELIKO Technology and Competence Center, Researcher
2012–2012 Marie Curie Research Fellow in Technical University Graz
1999–2005 Swedbank (Hansabank), Architect
1993–1998 Forexbank (Raebank), Manager, Architect

ELULOOKIRJELDUS

Isikuandmed

Nimi: Kalle Tomingas
Sünniaeg: 22.08.1973
Sünnikoht: Pärnu linn, Eesti
Kodakondsus: Eesti

Kontaktandmed

Telefon: +372 5040568
E-mail: kalle.tomingas@gmail.com

Hariduskäik

2008 – 2018 Tallinna Tehnikaülikool, PhD
1991 – 2000 Tallinna Tehnikaülikool, MSC
1989 – 1991 Pärnu Ülejõe Gümnaasium, keskharidus

Keelteoskus

Inglise keel kõrgtase
Vene keel suhtlustase
Eesti keel emakeel

Teenistuskäik

2017– … Orion Information Governance, teadus- ja arendusjuht
2005–2017 Mindworks Industries, konsultant
2011–2015 ELIKO Tehnoloogia Arenduskeskus, teadur
2012–2012 Marie Curie Research Fellow in Technical University Graz
1999–2005 Swedbank (Hansapank), arhitekt
1993–1998 Forexpank (Raepank), IT juht, arhitekt