METL: Managing and Integrating ETL Processes

Alexander Albrecht
supervised by Felix Naumann
Hasso-Plattner-Institute at the University of Potsdam, Germany
{albrecht,naumann}@hpi.uni-potsdam.de

ABSTRACT

Companies use Extract-Transform-Load (ETL) tools to save time and costs when developing and maintaining data migration tasks. ETL tools allow the definition of often complex processes to extract, transform, and load heterogeneous data into a data warehouse or to perform other data migration tasks. In larger organizations many ETL processes of different data integration and warehouse projects accumulate. Such processes encompass common sub-processes, shared data sources and targets, and same or similar operations. However, there is no common method or approach to systematically manage large collections of ETL processes. With METL (Managing ETL) we present an ETL management approach that supports high-level ETL management. To this end we establish and implement a set of basic management operators on ETL processes, such as MATCH, MERGE or INVERT.

1. INTRODUCTION

A common scenario in data migration is data warehousing, often applied in the areas of business intelligence, customer relationship management, data mining or master data management. In practice, developers often use data integration tools instead of hand-coded scripts for loading data warehouses. Usually one of the various ETL tools is chosen [17]. ETL tools allow the definition of often complex processes to extract, transform, and load heterogeneous data from one or more sources into a data warehouse.

Companies start using ETL tools in order to save time and costs in connection with developing and maintaining data migration tasks: ETL tools support all common databases, file formats and data transformations, simplify the reuse of already created (sub-)processes due to a collaborative development platform and provide central scheduling, logging and monitoring functionality for the execution of all running ETL processes.

Figures 1 to 3 show three simplified ETL processes, which we use as running examples throughout the paper. These ETL processes cover two typical scenarios in the context of data warehouse loading: Augmenting data with additional information and fact loading. The ETL processes were implemented with IBM InfoSphere DataStage, the ETL component of the IBM Information Server[1]. The ETL process in Figure 1 loads daily sales data of a retail company from an FTP server into a database table. Raw sales data consist of sales volume grouped by product key and shop number. At the beginning, data is augmented with additional shop information, such as shop address and the sales district key, using a join transformation over the shop number. Because there are some shops with a missing sales district key, this location information is subsequently consolidated: The data flow is split into two streams of tuples – one stream with records that lack location information, the other with existing sales district keys. A record with a missing location information is assigned a sales district key by a lookup using the zip code of a shop address. Finally, the two split data streams are combined into one, which is in turn loaded into a database table of daily shop sales.

The ETL processes in Figures 2 and 3 load daily sales data of all shops and the online shop into the fact table of a data warehouse. In this sample scenario, both ETL processes were separately developed and the data warehouse provides a single view on sales volume grouped by product key, location, sales week and year. The ETL process in Figure 2 uses a filter transformation in order to extract sales volume for products of the current week from the data warehouse. Weekly sales volume is subsequently aggregated with new daily sales volume from all shops and finally reloaded into the data warehouse. The ETL process in Figure 3 considers only sales volume of products from the data warehouse, that were actually sold over the internet this day. This is achieved by a semi-join over the product key. The aggregation and loading is equivalent to the ETL process in Figure 2.

The continuous use of ETL tools results in a large number of ETL processes. ETL tools store all ETL processes in a repository. The size of a repository may already increase in the course of a complex data integration project up to several hundred ETL processes [1]. The cooperation with our industrial partner shows that over time there are many ETL processes, which may encompass shared data sources, same data targets, common sub-processes and data transformations configured in an equal or similar way. However, we are not aware of a common method, approach, or framework to uniformly manage large repositories of ETL processes.

Due to this fact we presented the notion of ETL man-

VLDB '09, August 24-28, 2009, Lyon, France

[1] www.ibm.com/software/data/integration/info_server/

agement in [2]. Our approach is inspired by the model management research, which defines a set of operators for manipulating models and mappings [10]. In analogy, the main contribution of ETL management is to reduce the effort needed to develop or maintain ETL processes. Although ETL processes are strongly related to schema mappings [7], known model management techniques are not applicable in the context of ETL management: ETL management is based on certain ETL process characteristics that are not represented in schema mappings, such as the order, type, and configuration of data transformation steps. To establish ETL management we develop in the context of this Ph.D. project METL, a prototypical, web-based ETL management platform that supports the following high-level ETL management functionality:

- CREATE – converts non-ETL data transformation steps, such as declarative schema mappings between data sources and targets, into an ETL process.
- IMPORT – creates a tool-independent representation for a product-specific ETL process to support ETL management in a tool-independent manner.
- SEARCH – retrieves all ETL processes that satisfy the specified search query. Search queries may comprise different aspects of an ETL process, such as transformation types or database schemata.
- MATCH – given an ETL (sub-)process, find and rank all corresponding ETL (sub-)processes that extract, transform, or load common data in a similar way.
- INVERT – propagates any data cleansing and consolidating steps applied in an ETL process back to its data sources.
- MERGE – takes two or more ETL processes as input and returns a merged ETL process.
- DEPLOY – generates from METL's tool-independent ETL process representation an ETL process for a specific ETL tool.

Section 3 elaborates our approach to implement ETL management. Related work is surveyed in the following section. Section 4 introduces our prototype system METL, which is currently under development. Finally, Sec. 5 concludes and outlines next steps.

2. RELATED WORK

Although the practical importance of ETL in data integration projects is significant, [17], only little research on ETL at a meta-level has been performed. Most related research results improve ETL process modeling [11, 14, 16], but there is no implementation that supports further processing on such ETL process models.

The problems of setting up and maintaining multiple integration scenarios is addressed in [4]. The author's focus however is at a higher abstraction level namely abstracting different platform-specific integration processes, such as ETL, EAI, or FDBMS. The Orchid approach in [7] uses the work of [14] to convert real, IBM InfoSphere DataStage ETL processes into their abstract OHM model. Moreover, the Orchid approach provides a solution for creating ETL processes only out of declarative schema mappings and vice versa. Furthermore, there is a body of research on ETL process rewriting and optimization by Vassiliadis et al. [15] and Roy [13].

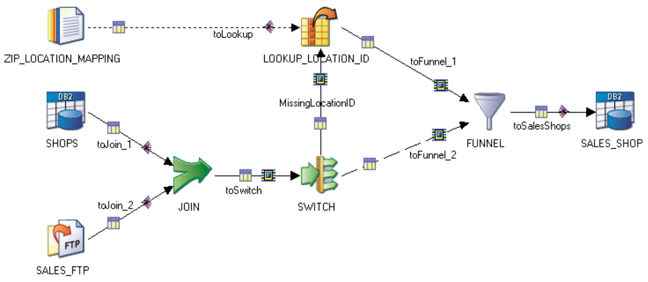


Figure 1: Augmenting sales information

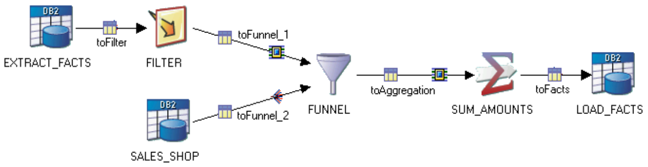


Figure 2: Adding daily to weekly sales volume

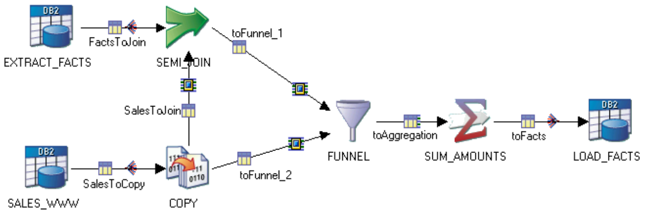


Figure 3: Fact loading based on daily product sales

The work of Kraft et al. describes a coarse-grained optimization (CGO) technique for sequences of SQL statements including creating, loading and dropping tables, inserts and queries [9]. Such statement sequences are similar to ETL processes and are automatically generated by commercial OLAP tools. The introduced rule-based optimization approach allows the merging of similar and dependent statements and is related to techniques we consider for the MERGE operator.

The matching of processes is a basic problem in the field of business process management. Jung and Bae present an approach for workflow clustering and introduced a method how to determine process similarity [8]. This structure-aware similarity measure is related to the one we use for the MATCH operator.

The INVERT operator propagates changes, applied in the ETL process in order to improve data quality, back to the sources. The idea of sending clean data back to the sources is first mentioned in [12] as *backflow of cleaned data*, but still remains to be worked out. Our work on the INVERT operator builds on known methods from materialized view maintenance to detect irrelevant updates [3].

In the context of ETL we also must mention [5] in the area of data lineage. In this work aspects of data that undergoes a sequence of transformations are studied. An introduction to model management, with its inspiring principles of providing generic algebraic operations, can be found in [10].

3. ETL MANAGEMENT

In this section, we provide more details on our ETL management work. We present our approach and research challenges that are targeted in the course of this Ph.D. project. Based on this approach, we expect this project to produce substantial results in the new field of ETL management. The cooperation with our industrial partner and a first analysis of the provided ETL repositories show the importance of ETL management in practice.

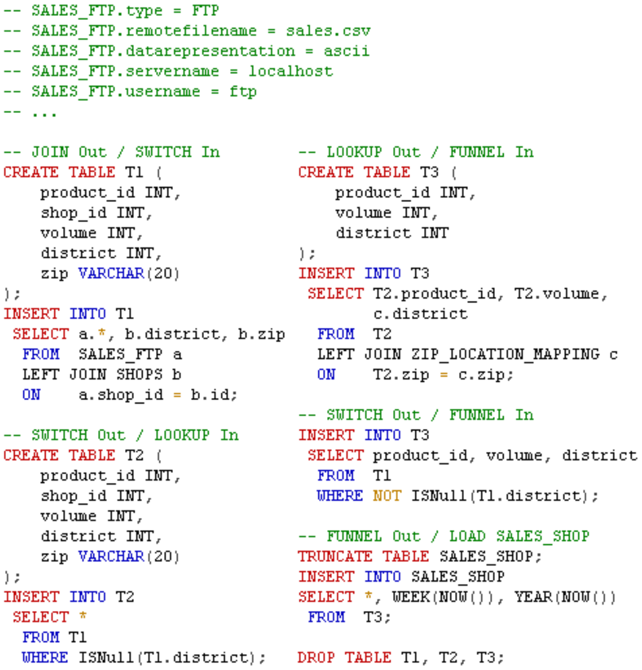
3.1 A general ETL process model

Importing product-specific ETL processes from large ETL repositories and other sources is the first step in ETL management. IMPORT converts a product-specific ETL process specification into a general ETL process representation to enable further processing by ETL management operators in a tool-independent manner. Most commercial and open source ETL tools provide ETL process specifications in some proprietary XML format, which is directly wrapped by IMPORT. Currently there is no standard mechanism or implementation one can use to model ETL processes. With CWM there is a specification for modeling all metadata of a data warehouse project [11]. The contained transformation package is intended to model ETL processes. Unfortunately, there is a lack of documentation and examples in the CWM specification how to actually model ETL processes. In addition, CWM is intended to model ETL processes for exchange, not for further processing like in ETL management. Already the modeling of a single ETL transformation causes excessive main memory consumption due to the elaborate CWM metadata modeling approach.

In our work we model ETL processes as statement sequences based on the Operator Hub Model (OHM) introduced in [7]. OHM covers a dozen ETL transformations available in IBM InfoSphere DataStage and defines their operational semantics as fragments of basic, product-independent operators. For example, the switch transformation in Figure 1 is defined by an OHM fragment consisting of one `SPLIT` operator and sequences of a `FILTER` operator followed by a `BASIC PROJECT` operator for every output stream.

The OHM model is also applicable for ETL process specifications from other ETL tools, such as Informatica PowerCenter[2] or CloverETL[3]. In ETL, transformations are a generalization of relational operators supporting multiple outputs, i.e., filter- or join-like operations, and work with multi-set semantics. In ETL a transformation never produces its own input data. An ETL process is thus a directed acyclic graph or, more precisely, a multidigraph, because in ETL more than one output stream of a transformation may be consumed by one subsequent transformation. The topologically ordered graph structure determines the execution order of the dependent transformations.

In our model, each transformation of an ETL process is described by a statement similar to SQL that reflects the corresponding OHM fragment. The output of each statement is loaded into a temporary table, which is the input to the next transformation. In the final loading step the output is loaded into the data target. Our modeling approach is intended to represent those aspects of ETL processes that are relevant to ETL management. On that level of abstraction, the modeled statement sequences should not be regarded as real execution plans for ETL processes. Figure 4 shows the corresponding statement sequence for the ETL process introduced in Figure 1. The statement sequence always starts with a header of all relevant meta information about each ETL extracting and loading step.



```
-- SALES_FTP.type = FTP
-- SALES_FTP.remotefilename = sales.csv
-- SALES_FTP.datarepresentation = ascii
-- SALES_FTP.servername = localhost
-- SALES_FTP.username = ftp
-- ...

-- JOIN Out / SWITCH In
CREATE TABLE T1 (
    product_id INT,
    shop_id INT,
    volume INT,
    district INT,
    zip VARCHAR(20)
);
INSERT INTO T1
 SELECT a.*, b.district, b.zip
  FROM  SALES_FTP a
  LEFT JOIN SHOPS b
  ON    a.shop_id = b.id;

-- SWITCH Out / LOOKUP In
CREATE TABLE T2 (
    product_id INT,
    shop_id INT,
    volume INT,
    district INT,
    zip VARCHAR(20)
);
INSERT INTO T2
 SELECT *
  FROM T1
  WHERE ISNull(T1.district);

-- LOOKUP Out / FUNNEL In
CREATE TABLE T3 (
    product_id INT,
    volume INT,
    district INT
);
INSERT INTO T3
 SELECT T2.product_id, T2.volume,
        c.district
  FROM  T2
  LEFT JOIN ZIP_LOCATION_MAPPING c
  ON    T2.zip = c.zip;

-- SWITCH Out / FUNNEL In
INSERT INTO T3
 SELECT product_id, volume, district
  FROM  T1
  WHERE NOT ISNull(T1.district);

-- FUNNEL Out / LOAD SALES_SHOP
TRUNCATE TABLE SALES_SHOP;
INSERT INTO SALES_SHOP
SELECT *, WEEK(NOW()), YEAR(NOW())
 FROM  T3;

DROP TABLE T1, T2, T3;
```

Figure 4: ETL statement sequence

3.2 Search

SEARCH returns all ETL processes stored in the repository that satisfy the specified search query. There are many attributes describing an ETL process, such as transformation names, input and output schemata, server addresses, etc. The SEARCH operator supports a query language that is similar to standard search engines: A query consists of search predicates p_1, p_2, ..., p_n. In order to assign a specific ETL context to a predicate, each predicate can optionally be prepended by a label. Each label indicates a certain context, such as transformations, database schemata, or server addresses. Predicates without labels are applied to all contexts. In addition, implemented SEARCH offers an automatic keyword completion feature: While typing the first letters of a query keyword, a list of keywords that start with the entered letters is automatically offered. The proposed keywords are taken from those ETL processes that answer the already typed-in search query. Figure 6 demonstrates the auto-completion feature provided by SEARCH. There may be multiple ETL processes in the repository that answer a query. The resulting ETL processes are therefore ranked by SEARCH based on the relevance to the user. The ranking of the resulting ETL processes is a major challenge, because their degree of relevance to the user has to be inferred. A first ranking approach obtains search terms that appear often within an ETL process but rarely in the entire repository a higher weight. With this TF/IDF-like term weighting approach we create weighted term vectors for the search query and all resulting ETL processes and use the cosine measure

[2] www.informatica.com

[3] www.cloveretl.com

to compute the similarity between each ETL process and the search query. The results of the search query are finally ranked based on the determined numeric similarity values.

A more advanced ranking is achieved by increasing the relevance for those ETL processes where search terms reference same ETL components, such as transformations, source or target databases. The following query example retrieves all sales-related ETL processes that use the term `district` in a input or output schema and contain one or more lookup transformations:

```
sales +schema:district +transformation:lookup
```

In this query example, the relevance of an ETL process increases, if the term `district` is used in the input or output schema of a lookup transformation. Thus the relationship between search terms within each resulting ETL process becomes important in our advanced relevance ranking strategy.

3.3 Match

Given an ETL process, the Match operator allows to find similar ETL processes or sub-processes and therefore provides an easy-to-use method to access all ETL processes stored within the repository. Instead of a query, an ETL process is required as input for Match. In analogy to information retrieval systems, the operator determines a numeric similarity measure on how well each ETL process in the repository matches the given ETL process, and ranks the result according to this value. In general, it is hard to define a suitable similarity measure for ETL processes, because of the presence of semantic or syntactic heterogeneity. Already the definition of input and output schemata for equal transformations causes problems, because column names are defined independently by the ETL developers. Thus in different schemata a variety of abbreviations, synonyms and hypernyms for semantically related attributes occur. In consequence, finding similar transformations at the schema-level becomes difficult. To make matters even more challenging, known schema matching approaches assume that the regarded schemata are at least somehow related to one another. For schemata in different ETL processes we cannot ensure that this assumption holds.

In order to find corresponding ETL processes within the repository, we implemented a basic, structure-aware Match operator. In this approach, we consider transformations to be similar if they belong to the same general ETL component. The structure of an ETL process is described by pairs of transformations: We consider pairs of transformations that are explicitly connected in the ETL process as directly interdependent. Two transformations that are connected by a path of length greater than one, we consider as indirectly interdependent. Each pair of transformations is initially associated with a weight w ∈ [0,1], which determines its dependency.

Match creates for the given ETL process and each ETL process from the repository a weighted multidimensional vector of directly and indirectly interdependent transformation pairs. The cosine measure computes the similarity between the given ETL processes and every ETL process from the repository. Finally, all ETL processes are ranked according to the calculated similarity values.

For the sample scenario from Section 1 Match determines the similarity of the ETL processes from Figures 1 and 2 with 10%, from Figures 1 and 3 with 7% and from Figures 2 and 3 with 77%. There is, of course, a high similarity of both fact loading ETL processes due to the similar structure. In addition, Match identifies the semi-join as a generic filter component an treats it similar to the filter transformation in Figure 2.

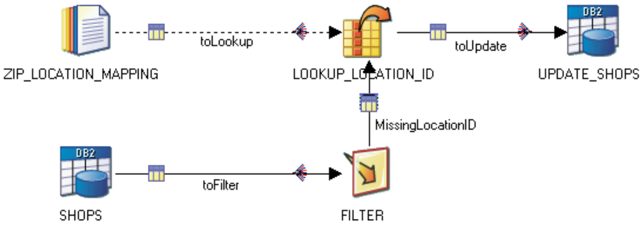


Figure 5: Corresponding data consolidation in ETL

The similarity measure calculated with Match can also be used to automatically organize all ETL processes within a large repository: Similar ETL processes are grouped together, i.e., ETL processes of one group are more related to each other than to ETL processes in other groups. We currently investigate this clustering idea using a large ETL repository of an industrial partner with several hundred ETL processes. Based on this test set we additionally want to infer ETL reference processes for every cluster in order to support and improve ETL process development using the inferred ETL reference processes as best practise patterns. For evaluation we use ETL reference processes manually built by our industrial partner.

3.4 Invert

Data cleansing plays a major role in ETL processes [12]. Especially in the data warehouse context, data with high quality is requested in order to ensure reliable analysis results. Within an ETL process specific data quality problems, such as missing or misspelled values or referential integrity violations, are solved in sequenced transformation steps.

Invert automatically discovers within a given ETL process potential data cleansing and consolidating steps and propagates them back to the sources. This allows performing discovered cleansing steps also on selected data sources. Finally, the user decides out of all discovered repairs which ones to process. In this context, it is important to consider that the application of all discovered cleansing steps is side-effect free, i.e., does not change the result of the ETL process or violate existing constraints on the source data.

The application of Invert improves data quality for applications working on the data sources, avoids multiple fixing of data quality problems in future ETL projects and reduces therefore time in developing new ETL processes. Since master data management (MDM) became an important issue in most companies, the idea of cleansing dirty master data using repair steps from existing ETL processes is a promising approach. Master data describes relevant business objects, such as products, business partners, or customers, in a centralized reference store.

The ETL process of Figure 1 consolidates data by assigning missing district keys to the corresponding values from a lookup table. Invert discovers such lookups as candidate repairs by static analysis of the ETL process model and presents the back-propagated repairs to the user with update statements similar to SQL. For the sample ETL process Invert generates the following update statement.

```
UPDATE SHOPS
SET district = ZIP_LOCATION_MAPPING.district
WHERE ISNULL(district)
AND zip = ZIP_LOCATION_MAPPING.zip
```

As a next step we plan to create a corresponding ETL process based on these update statements in order to allow data consolidation within the original ETL tool. Figure 5 shows the corresponding ETL process of the sample update statement in IBM InfoSphere DataStage.

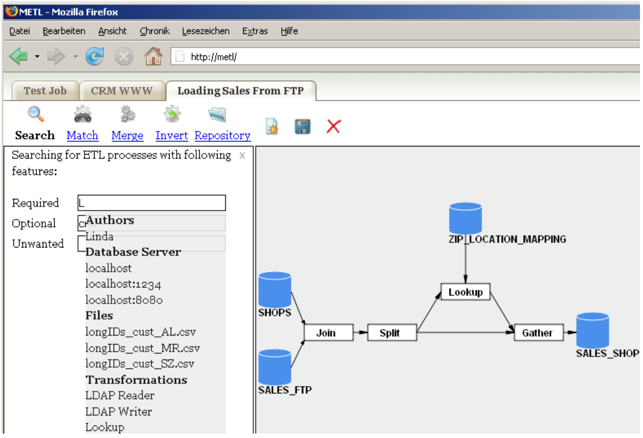
The approach of INVERT can be summarized with the following challenging four points: (i) discover all potential, side-effect free data cleansing and consolidating steps within a given ETL process; (ii) describe each cleansing steps with SQL-like update statements (iii) propagate discovered cleansing steps as updates back to the sources; (iv) convert derived update statements of all selected cleansing steps into an ETL process.

3.5 Merge

The intention of MERGE is to combine two or more ETL processes into one integrated ETL process. In general, company-wide developed ETL processes share common sources and targets. Applying MERGE in such scenarios, achieves a better utilization of shared resources, along with latency improvement, a reduced amount of data transmission and promises an enhancement of performance compared to performing all processes in a separate run. This assumption is motivated by the expectation that in a merged ETL process common data may be processed in an equal way. In addition to optimization, a merged ETL process provides a single view of all information that was originally processed separately. Apart from performance benefits, consolidating large sets of ETL processes promises reduced overhead and maintenance.

Advantages of MERGE can be well demonstrated using the ETL processes introduced in Section 1. It is imaginable, an ETL developer discovers with SEARCH that the database table `SALES_SHOP` is exclusively used by the ETL processes from Figures 1 and 2. The merged ETL process dispenses with materializing the database table and provides an improved performance due to pipelining the data flow during execution. In addition, the merged ETL process reduces scheduling overhead, because only one integrated ETL process instead of two dependent ETL processes has to be administered. In a second sample scenario, an ETL developer wants to merge similar ETL processes from Figures 2 and 3 found by applying MATCH. Because there are different possibilities to merge both ETL processes, MERGE returns a merged ETL process with the least number of transformations. In order to find common transformations, a rewrite of the given ETL processes is necessary. For example, in Figure 2 the filter predicate on *current week* is not optimal: Unnecessary records from the data warehouse are extracted and in turn loaded without modification into the data warehouse, because only sales volume of records are refreshed that have a corresponding product key in the second data source. The optimal filter is therefore a semi-join. Such a filter rewrite would lead to an extra correspondence between both ETL processes and in turn to an additional merged transformation.

The approach of MERGE can also be summarized with four challenging points: (i) creation of equivalent ETL (sub-)processes by applying rewrites to every given ETL process; (ii) identification of combinable transformations among each given ETL process or its equivalent representations; (iii) generation of all possible merged ETL processes (iv) returning the best merged ETL process.



Searching for ETL processes with following features:

Required L
Optional cr
Unwanted

Authors
Linda
Database Server
localhost
localhost:1234
localhost:8080
Files
longIDs_cust_AL.csv
longIDs_cust_MR.csv
longIDs_cust_SZ.csv
Transformations
LDAP Reader
LDAP Writer
Lookup

Figure 6: Autocompletion feature in SEARCH

Like INVERT, MERGE has to work side-effect free, i.e., the result of the merged ETL process is the same as the result of all scheduled ETL processes in a separate run. At the moment our work is focused on developing a set of general rewrite rules, such as filter optimization, and identification rules for combinable transformations.

3.6 Other basic operators

Besides ETL, there are many ways to move data from sources to targets, such as setting up mappings between the data sources or writing SQL scripts. In order to benefit from any data integration effort that is already done, we introduce the CREATE operator, which converts non-ETL data transformation steps, into an ETL representation. Such an ETL process could be enhanced within a specific ETL tool or managed with the introduced ETL management operators in order to make further use of it. In [7] Hernandez et al. introduce Orchid – a prototype system that creates ETL processes from one type of other specification, namely declarative mapping specifications.

The introduced IMPORT operator enables tool-independent ETL management by converting tool-specific ETL process specifications into a generalized ETL process representation. Subsequent ETL management operations are performed on the imported ETL process, which may be deployed in case of INVERT or MERGE as an executable ETL process into the original ETL tool. Therefore we introduce the DEPLOY operator. One major challenge is to support interoperability of different ETL tools based on IMPORT and DEPLOY, i.e., importing an ETL process from one ETL tool and deploying it as an executable ETL process to a second, different ETL tool. This requires exploring different semantics of similar ETL transformations in different ETL tools and describing their expressiveness.

4. METL: A SYSTEM FOR MANAGING ETL

METL is a prototypical, web-based ETL management platform and features higher-level ETL management functionality. The basic software architecture of METL consists of an

Ajax-based GUI using SVG as the underlying ETL process renderer. ETL processes are visualized based on the introduced general ETL process model.

Figure 6 shows METLs browser-based interface and demonstrates the autocompletion feature provided by SEARCH. On the right hand side, ETL processes are imported by clicking the corresponding icon in the toolbar. All ETL processes are imported into a central repository. Each opened ETL process is easily accessible by its tab. A user may explore an ETL process using the graphical interface.

The novel operators for ETL management, such as SEARCH, MATCH, MERGE, and INVERT, appear as buttons on the left hand side. We plan to implement all introduced ETL management operators in course of this Ph.D. project. METL is used as platform to prove the usability and to explain the behavior of all implemented ETL management operators.

5. CONCLUSION & NEXT STEPS

In this paper we present a generic approach for ETL management, meaning that all introduced high-level operators, such as MATCH, INVERT, or MERGE, are applicable to different kinds of ETL processes from different ETL tools. The generic approach is achieved by treating ETL processes in a tool-independent manner. We motivated the usefulness of our approach by examples from the data warehouse context and described the behavior of ETL management operators in more detail. Our suggested approach is intended to support and improve ETL process development by providing ETL management functionality for large ETL process repositories. Our research is orthogonal to other approaches that try to simplify ETL process design by successively converting business models of a company into ETL processes [6].

Our next steps include the implementation and improvement of all introduced ETL management operators, and subsequent case studies. At the moment we are analyzing large ETL repositories of our industrial partner with several hundred ETL processes. The repositories comprise ETL processes for batch integration of financial data. Additionally we plan to build wrappers for common ETL tools, such as IBM InfoSphere DataStage, Informatica PowerCenter, and CloverETL, so that product-specific ETL processes can be imported, and ETL processes generated by ETL management operators can be deployed in different platforms.

Next to facing the challenges described in Section 3, we also have to comprehensibly evaluate our developed approaches within an experimental testbed, such as an ETL benchmark as described by Vassiliadis et al. [17].

6. REFERENCES

[1] Himanshu Agrawal, Girish Chafle, Sunil Goyal, Sumit Mittal, and Sougata Mukherjea. An Enhanced Extract-Transform-Load System for Migrating Data in Telecom Billing. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Cancún, México, 2008.

[2] Alexander Albrecht and Felix Naumann. Managing ETL processes. In *Proceedings of the VLDB International Workshop on New Trends in Information Integration (NTII)*, Auckland, NZ, 2008.

[3] J.A. Blakeley, N. Coburn, and P. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems (TODS)*, 14(3):369–400, 1989.

[4] Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. Model-driven development of complex and data-intensive integration processes. In *Proceedings of the International Workshop on Model-Based Software and Data Integration (MBSDI)*, Berlin, Germany, 2008.

[5] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. *VLDB Journal*, 12(1):41–58, 2003.

[6] Umeshwar Dayal, Malu Castellanos, Alkis Simitsis, and Kevin Wilkinson. Data integration flows for business intelligence. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, Saint Petersburg, Russia, 2009.

[7] Mauricio A. Hernandez, Stefan Dessloch, Ryan Wisnesky, Ahmed Radwan, and Jindan Zhou. Orchid: Integrating schema mapping and ETL. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Cancun, Mexico, 2008.

[8] Jae-Yoon Jung and Joonsoo Bae. Workflow clustering method based on process similarity. In *Proceedings of the International Conference on Computational Science and Its Applications (ICCSA)*, Glasgow, UK, 2006.

[9] Tobias Kraft, Holger Schwarz, Ralf Rantzau, and Bernhard Mitschang. Coarse-Grained Optimization: Techniques for Rewriting SQL Statement Sequences. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 488–499, Berlin, Germany, 2003.

[10] Sergey Melnik. *Generic Model Management: Concepts and Algorithms*. LNCS 2967. Springer Verlag, Berlin – Heidelberg – New York, 2004.

[11] John Poole, Dan Chang, and Douglas Tolbert. *Common Warehouse Metamodel, Developer's Guide (OMG)*. Wiley & Sons, Indianapolis, IN, 2003.

[12] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[13] Prasan Roy. Optimization of DAG-structured Query Evaluation Plans. Master's thesis, Indian Institute of Technology, Bombay, 1998.

[14] Alkis Simitsis. Mapping conceptual to logical models for ETL processes. In *Proceedings of the International Workshop on Data warehousing and OLAP (DOLAP)*, pages 67–76, New York, NY, 2005.

[15] Alkis Simitsis, Panos Vassiliadis, and Timos Sellis. Optimizing ETL processes in data warehouses. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Tokyo, Japan, 2005.

[16] Juan Trujillo and Sergio Luján-Mora. A UML based approach for modeling ETL processes in data warehouses. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, Chicago, IL, 2003.

[17] Panos Vassiliadis, Anastasios Karagiannis, Vasiliki Tziovara, and Alkis Simitsis. Towards a benchmark for ETL workflows. In *Proceedings of the 5th International Workshop on Quality in Databases (QDB)*, Vienna, Austria, 2007.