Automating User-Centered Design of Data-Intensive Processes

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
M.Sc. Vasileios Theodorou
geboren am 21. January 1988 in Athen

Betreuender Hochschullehrer: **Prof. Dr. Wolfgang Lehner**
Technische Universität Dresden
Fakultät Informatik, Institut für Systemarchitektur
Lehrstuhl für Datenbanken
01062 Dresden

Dresden, im Oktober 2016

ABSTRACT

Business Intelligence (BI) enables organizations to collect and analyze internal and external business data to generate knowledge and business value, and provide decision support at the strategic, tactical, and operational levels. The consolidation of data coming from many sources as a result of managerial and operational business processes, usually referred to as Extract-Transform-Load (ETL) is itself a statically defined process and knowledge workers have little to no control over the characteristics of the presentable data to which they have access.

There are two main reasons that dictate the reassessment of this stiff approach in context of modern business environments. The first reason is that the service-oriented nature of today's business combined with the increasing volume of available data make it impossible for an organization to proactively design efficient data management processes. The second reason is that enterprises can benefit significantly from analyzing the behavior of their business processes fostering their optimization. Hence, we took a first step towards quality-aware ETL process design automation by defining through a systematic literature review a set of ETL process quality characteristics and the relationships between them, as well as by providing quantitative measures for each characteristic. Subsequently, we produced a model that represents ETL process quality characteristics and the dependencies among them and we showcased through the application of a Goal Model with quantitative components (i.e., indicators) how our model can provide the basis for subsequent analysis to reason and make informed ETL design decisions.

In addition, we introduced our holistic view for a quality-aware design of ETL processes by presenting a framework for user-centered declarative ETL. This included the definition of an architecture and methodology for the rapid, incremental, qualitative improvement of ETL process models, promoting automation and reducing complexity, as well as a clear separation of business users and IT roles where each user is presented with appropriate views and assigned with fitting tasks. In this direction, we built a tool —POIESIS— which facilitates incremental, quantitative improvement of ETL process models with users being the key participants through well-defined collaborative interfaces.

When it comes to evaluating different quality characteristics of the ETL process design, we proposed an automated data generation framework for evaluating ETL processes (i.e., Bijoux). To this end, we classified the operations based on the part of input data they access for processing, which facilitated Bijoux during data generation processes both for identifying the constraints that specific operation semantics imply over input data, as well as for deciding at which level the data should be generated (e.g., single field, single tuple, complete dataset). Bijoux offers data generation capabilities in a modular and configurable manner, which can be used to evaluate the quality of different parts of an ETL process.

Moreover, we introduced a methodology that can apply to concrete contexts, building a repository of patterns and rules. This generated knowledge base can be used during the design and maintenance phases of ETL processes, automatically exposing understandable conceptual representations of the processes and providing useful insight for design decisions.

Collectively, these contributions have raised the level of abstraction of ETL process components, revealing their quality characteristics in a granular level and allowing for evaluation and automated (re-)design, taking under consideration business users' quality goals.

CONTENTS

INTRODUCTION

Business Intelligence (BI) nowadays involves identifying, extracting, and analyzing large amount of business data coming from diverse, distributed sources. Typically, enterprises rely on complex Information Technology (IT) systems that manage all data coming from operational databases running within the organization and provide fixed user interfaces through which knowledge workers can access information. In order to facilitate decision-makers, these systems are assigned with the task of integrating heterogeneous data deriving from operational activities into Data Warehouses, for the purpose of querying and analysis. This integration requires the extraction of data from internal or external sources such as the Web, their transformation to comply with destination syntax and semantics and the loading of the processed data to Warehouses, in a process known as Extraction Transformation Loading (ETL).

Today, the increasing volume of available data, as well as the requirement for recording and responding to multiple events coming from participants within Big Data ecosystems that are characterized by the 3Vs (volume, variety, velocity) (Russom, 2011), pose a serious challenge for the design of ETL processes. The lack of a common schema for input data that might even be unstructured, makes it even more difficult to produce useful information for end users in real-time. Hence, the existence of diversely structured heterogeneous data deriving from multiple internal and external sources suggest the definition of dynamic models and processes for its collection, cleaning, transformation, integration, analysis, monitoring and so on.

Like every other business process, ETL processes can be examined using models and techniques from the area of Business Process Management (BPM) (Wilkinson et al., 2010; Akkaoui et al., 2013), which is a holistic approach that aims to optimize business processes with respect to effectiveness and efficiency. A central activity of BPM is Business Process Modeling, which concerns representing the structure and workflow of business processes in a way that is understandable both by business users (BU) and IT. One problem that arises with defining appropriate models for ETL processes is relating the process model to fitness to use for the end-user. Analysis of the behavior of business processes is usually conducted by knowledge workers and business analysts who lack knowledge about underlying infrastructure of IT systems and related technologies. Therefore, we argue that the modeling approaches defined in (Wilkinson et al., 2010; Akkaoui et al., 2013) are still too low-level to facilitate the evaluation and incorporation of process enhancements reflecting business requirements. This gap between end-user requirements and the low-level activities performed by ETL tools needs to be addressed using dynamic user-centered tools that can facilitate ad hoc processing of analytical queries with minimal IT intervention, borrowing techniques from the research area of Requirements Engineering.

The ultimate goal of this thesis has been the development of a user-centered framework that automates process model enhancement and selection for BI-related data processing. Some of the key requirements for the design of this framework are usability, efficiency and effectiveness.

The structure of this introductory chapter is as follows: In the next section, we briefly discuss current developments in the area of ETL quality and highlight some emerging opportunities and open issues arising. In addition, we introduce our notion of conflicting quality dimensions in ETL design. In Section 1.2, we identify open challenges in data-intensive process evaluation, stemming from its inherent dynamicity and from the difficulty in finding the right data for simulation. In Section 1.3, we discuss important challenges in ETL automation and we

present a motivating experiment that we conducted, which verifies the problematic nature of manual ETL modifications and the apparent need for further abstractions and mechanizations. Subsequently, in Section 1.4, we present a novel architecture for user-centered declarative ETL that served as the basis for the research presented in this thesis. Finally, we discuss the main contributions of our work and the thesis structure.

1.1 ETL PROCESS QUALITY

ETL requires the execution of real-time, automated, data-centric business processes in a variety of workflow-based tasks. The main challenge is how to turn the integration process design, which has been traditionally predefined for periodic off-line mode execution, into a dynamic, continuous operation that can sufficiently meet end-user needs.

During the past years, there has been considerable research regarding the optimization of ETL flows in terms of functionality and performance (Simitsis et al., 2005; Böhm et al., 2009b). Moreover, in an attempt to manage the complexity of ETL processes on a conceptual level that reflects organizational operations, tools and models from the area of Business Process Management (BPM) have been proposed (Wilkinson et al., 2010; Akkaoui et al., 2012). These approaches make the crucial step of examining data-centric process models in an outlook that brings them closer to the business core and allows their functional validation and performance evaluation based on user input.

However, the dimension of process quality (Sánchez-González et al., 2012) has not yet been adequately examined in a systematic manner. Unlike other business processes, important quality factors for ETL process design are tightly coupled to information quality while depending on the interoperability of distributed engines. Added to that, there is increasing need for process automation in order to become more cost-effective (Simitsis et al., 2009b) and therefore there needs to be a common ground between analysts and IT that would allow the seamless translation of high level quality concerns to design choices.

The identification of different perspectives of ETL processes — i) data-centric view, ii) software view and iii) business process view — attaches to them an interdisciplinary character and enables the consolidation and reuse of tools and practices from multiple well-established research areas for their analysis. Furthermore, recent advancements in the area of Requirements Engineering offer frameworks targeted to the domain of Business Intelligence that can facilitate ETL process analysis on a business level that is backed by measures and runtime behavior characteristics on the technical level. In addition, the emerging Data Warehousing paradigms of agile design (Golfarelli et al., 2012) and self-service BI (Berthold et al., 2010) present new opportunities, denoting the necessity of revisiting existing work in the area, in an angle that can promote ETL design automation, while exposing multiple quality dimensions.

In Figure 1.1, we depict our notion of the space created by different quality dimensions of an ETL process. A set $\mathbb{Q}$ of quality dimensions $\{q1, q2, ..., qn\}$ (e.g., performance, data quality, maintainability) formulate a space where cost is orthogonal to any other dimension; when cost increases (i.e., more available resources), we assume that any quality dimension can show improvement for one specific ETL process. This can happen in the form of modifications on the ETL model during design or management of the deployment configuration (e.g., additional

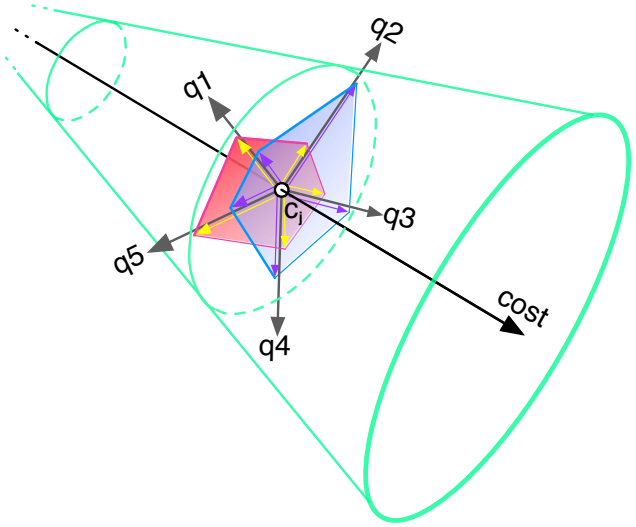


Figure 1.1: Utility cone of ETL quality

CPU/memory, distribution of threads to process activities). Thus, any quality dimension can improve at some expense, but we advocate that when cost is fixed (i.e., for some specific cost value c_j), the improvement of one quality dimension might affect the others either positively or negatively. The trade-offs between quality dimensions and the limitations to the improvement of each of them pose some boundaries on the overall quality, which can be referred to as *utility* of the ETL process, denoting its usefulness to end-users. In Figure 1.1, the boundaries for a specific cost value c_j are sketched as a conic section, only inside which values for the different quality dimensions are permitted. The cone shape is selected to denote that these boundaries become larger as cost increases, for the sake of simplicity of the conceptual reference —it does not signify that such increase is linear. Thus, for a fixed cost c_j, different design and implementation decisions about the ETL process form different radar charts (the angles of the axes are irrelevant) that show how these decisions favor some quality dimension(s) over others and ultimately, the surface that each of them encompasses, provides some indication of the ETL process utility. We call this shape a *utility cone* and although it does not serve as a mathematical model that can define formally the relationships between different quality dimensions, it is a cognitive tool that clarifies our notion about ETL quality improvements.

1.2 DATA-INTENSIVE PROCESS EVALUATION

Data-intensive processes constitute a crucial part of complex business intelligence (BI) systems responsible for delivering information to satisfy the needs of different end-users. Despite growing amounts of data representing hidden treasury assets of an enterprise, due to dynamic business environments, data quickly and unpredictably evolve, possibly making the software that processes them (e.g., ETL) inefficient and obsolete. Besides delivering the right information to end users, data-intensive processes must also satisfy various quality standards to ensure

that the data delivery is done in the most efficient way, whilst the delivered data are of certain quality level. The quality level is usually agreed beforehand in the form of service-level agreements (SLAs) or business-level objects (BLOs).

In order to guarantee the fulfillment of the agreed quality standards (e.g., data quality, performance, reliability, recoverability; see (Barbacci et al., 1995; Simitsis et al., 2009b; Theodorou et al., 2016)), an extensive set of experiments over the designed process must be performed to test the behavior of the process in a plethora of possible execution scenarios. Essentially, the properties of input data (e.g., value distribution, cleanness, consistency) play a major role in evaluating the resulting quality characteristics of a data-intensive process. Furthermore, to obtain the finest level of granularity of process metrics, quantitative analysis techniques for business processes (e.g., (Dumas et al., 2013)) propose analyzing the quality characteristics at the level of individual activities and resources. Moreover, one of the most popular techniques for quantitative analysis of process models is process simulation (Dumas et al., 2013), which assumes creating large number of hypothetical process instances that will simulate the execution of the process flow for different scenarios. In the case of data-intensive processes, the simulation should be additionally accompanied by a sample of input data (i.e., *work item* in the language of (Dumas et al., 2013)) created for simulating a specific scenario.

Nonetheless, obtaining input data for performing such experiments is rather challenging. Sometimes, easy access to the real source data is hard, either due to data confidentiality or high data transfer costs. However, in most cases the complexity comes from the fact that a single instance of available data, usually does not represent the evolution of data throughout the complete process lifespan, and hence it cannot cover the variety of possible test scenarios. At the same time, providing synthetic sets of data is known as a labor intensive task that needs to take various combinations of process parameters into account.

In the field of software testing, many approaches (e.g., (DeMillo and Offutt, 1991)) have tackled the problem of synthetic test data generation. However, the main focus was on testing the correctness of the developed systems, rather than evaluating different data quality characteristics, which are critical when designing data-intensive processes. Moreover, since the execution of data-intensive processes is typically fully automated and time-critical, ensuring their correct, efficient and reliable execution, as well as certain levels of data quality of their produced output is pivotal.

In the data warehousing (DW) context, an example of a complex, data intensive and often error-prone data-intensive process is the ETL process, responsible for periodically populating a data warehouse from the available data sources. The modeling and design of ETL processes is a thoroughly studied area, both in the academia and industry, where many tools available in the market (Pall and Khaira, 2013) often provide overlapping functionalities for the design and execution of ETL processes. Still, however, no particular standard for the modeling and design of ETL processes has been defined, while ETL tools usually use proprietary (platform-specific) languages to represent an ETL process model.

The correct ETL implementation is generally a very costly procedure (Beyer et al., 2016). Moreover, the ETL design tools available in the market (Pall and Khaira, 2013) do not provide any automated support for ensuring the fulfillment of different quality parameters of the process, and still a considerable manual effort is expected from the designer. Thus, we have identified the real need for facilitating the task of testing and evaluating ETL processes in a

configurable manner. In this respect, we need to generate delicately crafted sets of data to test different execution scenarios of an ETL process and detect its behavior (e.g., *performance*) over a variety of changing parameters (e.g., *dataset size*, *process complexity*, *input data quality*).

1.3 CHALLENGES IN ETL AUTOMATION

ETL processes require heavy investments for their design, deployment and maintenance. With data being increasingly recognized as a key asset for the success of any enterprise, interest is growing for the development of more sophisticated models and tools to aid in data process automation and dynamicity. According to a recent Gartner report (Beyer et al., 2016), the data integration tool market is growing with a rate above the average for the enterprise software market as a whole, with an increase of 10.5% from 2014 to 2015 and an expected total market revenue of $4 billion in 2020.

In this context, ETL requirements are becoming more advanced and demanding with expectations such as self-service BI (Abelló et al., 2013) and on-the-fly data processing making ETL projects even more complex. Moreover, ETL users and developers with different backgrounds, using different models and technologies form a confusing landscape of ETL frameworks and processes that is hard to analyze and harness. The reply from academia has been the proposal of models (e.g., using the business process modeling notation (BPMN) (Akkaoui et al., 2013) or the unified modeling language (UML) (Muñoz et al., 2008)) that classify ETL functionalities in different levels of abstraction, creating some common ground for the description of ETL operations and fostering design automation and analysis. On the direction of translating conceptual and logical operations to physical implementations, the design and implementation of large ETL flows is detached from the use of specific technologies and using tested structures and best practices, it can become more reliable, efficient and simple. On the opposite direction, mapping physical to logical and conceptual models allows for the concise representation and reuse of components, as well as different layers of analysis and comparison of ETL flows.

As already mentioned, it has recently been proposed that to tackle complexity, the level of abstraction for ETL processes can be raised. ETL processes have been decomposed to ETL activities (Vassiliadis et al., 2009) and recurring patterns (Castellanos et al., 2009) as the main elements of their workflow representation, making them susceptible to analysis for process evaluation and redesign. The proposed ETL modeling as well as the ETL logical view generated by different open-source and proprietary tools, expose an ETL workflow perspective that opens the door for the identification and specification of ETL patterns. Although there have been some approaches on ETL patterns, considering most used ETL tasks or the morphology of the complete ETL flow, a bottom up methodology that can identify patterns and apply customized analysis on a given generic set of ETL processes is still missing, making the practical exploitation of ETL patterns difficult.

It is apparent that there is a need for an automatized process of ETL quality enhancement, as it would solve many of the above-mentioned issues. Analysts should be in the center of this process, where the large problem space is automatically generated, simulated and displayed in an intuitive representation, allowing for the selection among alternative design choices.

1.3.1 Motivating Experiment

In order to qualitatively assess the complexity and difficulty of improving ETL process quality in a manual fashion, we conducted a controlled experiment focusing on the following issues (i) is it easy using an intuitive GUI, to manually implement improvement actions on an ETL flow that can improve selected quality characteristics? (ii) can this manual process be efficient and effective?, and (iii) what are the most commonly occurring mistakes of this manual approach?

The participants of the study were computer science and information technology students at the Universitat Politécnica de Catalunya, BarcelonaTech, Barcelona, Spain who were enrolled in the basic data warehousing class lasting one semester. The students were taught the basic principles about ETL processes and ETL operations during the lectures and they were additionally supplied with training material, which included detailed examples, illustrating how to implement improvement actions on an ETL flow to enhance its quality characteristics, using the open-source ETL tool called Pentaho Data Integration tool (also known as Kettle)[1]. As part of the practical part of the course, the students had participated in exercises familiarizing with the Pentaho Data Integration tool. The participants belonged to two categories: (i) 21 bachelor students without any prior experience with ETL processes and (ii) 14 masters students with limited experience with ETL. All of the participants were proficient users of computer applications and they had experience with application development.

Regarding the experimental materials, the subjects were provided with one ETL model editable and executable by the Pentaho Data Integration tool, as well as test input data from a relational database and flat files. They were assigned with two tasks and they were given one hour to complete them. The tasks were the following:
T1 — Improve the data quality of the produced output data in terms of data consistency and data completeness
T2 — Improve the performance of the ETL process in terms of time efficiency and the reliability of the ETL process in terms of recoverability and robustness

The initial ETL process included twelve ETL operations of the following types: (i) input source loading, (ii) filter, (iii) attribute alteration, (iv) join, (v) group-by and (vi) extracting to output source. The subjects were asked to complete the tasks without changing the functionality of the ETL process.

	Bachelor students	**Masters students**
AVG mark Task 1 (/10)	7.5	7.7
AVG duration Task 1 (min)	29.5	27.1
AVG mark Task 2 (/10)	6.5	6.2
AVG duration Task 2 (min)	15.9	12.6

Figure 1.2: Experimental results

The results from the students' answers were ETL models that they produced and uploaded to an on-line system. These models were systematically evaluated and graded with regards to

[1] `http://www.pentaho.com/product/data-integration`

their correctness, efficiency, effectiveness and completeness. In addition, the duration of the tasks for each of the subjects was recorded and the results can be seen in Fig. 1.2.

As can be seen from the results, the performance of both bachelor and masters students was relatively low in general. The main mistakes that were identified during marking were the following:

- *Wrong configuration of ETL operations*: The ETL operators usually have complex parameters, describing not only their properties but also their relationship to the flow and the other operations. It was common that these parameters were incompletely or mistakenly configured, resulting to execution issues of the resulting ETL model. For example, in many cases the paths to the input and output sources were missing or wrong, the configurations for the resource allocation did not take any constrains under consideration, etc.

- *Incomplete exploitation of improvement actions*: The improvement actions are configurable and can be deployed on multiple potential locations on the ETL flow. One common mistake made by the students was the placement of improvement steps, only once in the graph. This way many obvious actions that would significantly improve the quality of the ETL were ignored, for example by filtering null values after one input source but not after any of the rest.

- *Wrong placement of improvement steps*: One additional shortcoming that we identified was that students did not find the placement of improvement steps to the right position intuitive. For example, one common mistake was the placement of a recovery point right before the extraction to output source step, which does not produce any actual improvement of the process quality.

The results indicate that manual implementation of improvement actions is error-prone, non-trivial, time-consuming and it suffers from incompleteness, inefficiency, and ineffectiveness. It is apparent that there is a need for an automatized process of ETL quality enhancement, as it would solve many of the above-mentioned issues.

1.4 USER-CENTERED DECLARATIVE ETL

In this section, we introduce our proposed architecture for a quality-aware framework for ETL design. The architecture is depicted in Fig. 1.3, where we also show the relationship between different parts of the architecture and chapters of this thesis.

Our novel methodology for the end-to-end design of ETL processes takes under consideration both functional and non-functional requirements. Based on existing work, we raise the level of abstraction for the conceptual representation of ETL operations and we show how process quality characteristics can generate specific patterns on the process design. The architecture consists of three phases: *functionality-based design*, *quality enhancement* by instillation of user-defined quality characteristics to the process and finally, *deployment and execution*. The main drivers of this proposal are the requirements for automation and user-centricity. In addition, one important dimension is the need for communicating business requirements to the

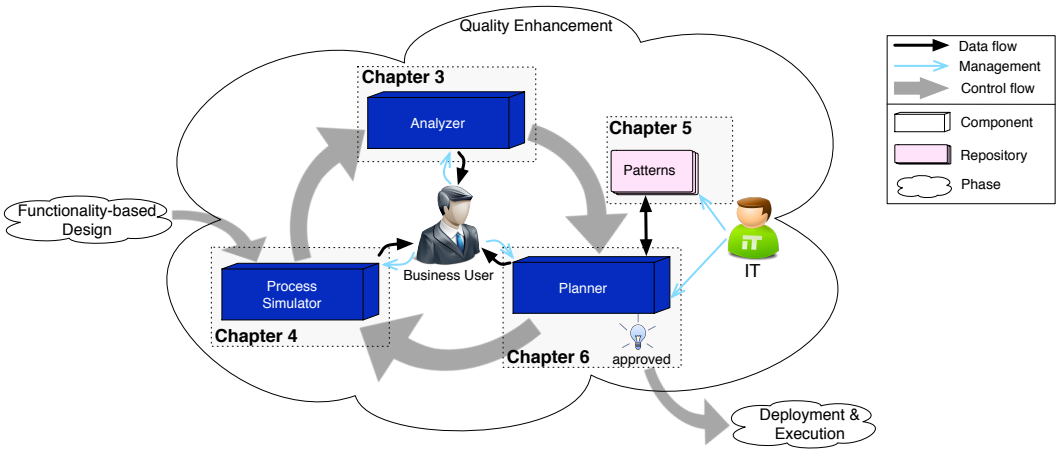


Figure 1.3: Functional architecture

design, coming from BU who lack the background to understand technical details of the process. To provide the means for meeting these requirements, we propose a modular architecture that employs reuse of components and patterns to streamline the design. Furthermore, we apply an iterative model where BU are the key participants through well-defined collaborative interfaces. Following is a description of the different parts of the architecture.

Functionality-Based Design. This part of the architecture only concerns the functional requirements for the ETL process, producing an ETL model that complies with the information requirements and the characteristics of the target repositories (e.g., star schema with specific facts and dimensions). Thus, it is not included in our models about quality (or non-functional) requirements. However, it is presented here to showcase how it guarantees the generation of a functionally correct ETL process, which can subsequently be provided as input to and be improved by a quality-enhancement phase.

Recently, several approaches have been proposed for the automation of this phase. For example, (Romero et al., 2011) uses an ontology of the data sources and their functional dependencies, together with business queries, to semi-automatically generate the ETL steps and the data warehouse multi-dimentional model at a conceptual level. Similarly, (Bellatreche et al., 2013) proposes an approach where the domain model along with user-requirements are modelled on the ontological level and subsequently, an ETL process is produced, also modelled as an implementation-independent ontology. In the same direction, (Jovanovic et al., 2016) takes as input a set of functional requirements about data intensive processes, which the authors call *information requirements* and semi-automatically translate them to concrete implementation steps, with the objective of reusing existing data flows from already implemented data intensive processes as efficiently as possible.

Quality Enhancement. The second phase regards the infusion of quality parameters to the ETL process. Our choice to segregate the functionality of the process from its qualitative perspective in two discrete phases does not only stem from a need for separation of concerns, but also from the fact that BU are the ones qualified to set quality goals and assess process quality. Nevertheless, the role of IT cannot be neglected, since apart from pro-actively designing all the

aspects of this phase, they should constantly oversee the design process and translate technical details to business concepts whenever necessary and vice versa. Our architectural design at this stage is influenced by two paradigms from the areas of Software Development and Business Intelligence: agile methods and self-service BI (Berthold et al., 2010), respectively. The benefits of using agile methods as opposed to the traditional waterfall approach in Data Warehousing activities have recently been recognized (Golfarelli et al., 2012). We identify this stage of the ETL process design as a perfect candidate for the application of agile practices because of the complexity and uncertainty of translating quality requirements to design choices. Thus, we adopt the idea of incremental and iterative design with BU in the center of the process. Likewise, we adopt the concept of strategy-driven business process analysis from the area of self-service BI, where BU make decisions in a declarative fashion based on strategies, goals and measures.

Integrating these ideas, we suggest that BU make decisions in stepwise iterations (sprints) that incrementally improve the quality of the ETL process, until they consider it crosses an acceptable quality threshold. Following is a description of each component that shows the means to facilitate this interaction.

The *Process Simulator* component is assigned with the task of simulating the ETL process and producing primal and complex analytics about both its static structure and its predicted execution behaviour. One decision that has to be made at this point by the BU is the level of detail of the simulation. For example, the analysis can take place on a process level or on a task level, based on a single process run or on aggregated results of multiple process replications. Additionally, the simulation methodology should also be decided, especially for the case of loops, conditions and events (e.g., probabilistic or deterministic).

One obvious issue is the quantification of high level quality characteristics and their mapping to measures on the process. According to our approach, BU have access to high level metrics and statistics about the process quality, where visualization plays a key role. The metrics as well as the BU input should be applicable for different parts and levels of detail of the process, something that is configured by the IT and used as a service by the BU.

Once process measures have been produced, it is time for the *Analyzer* to come into play. The Analyzer takes as input user goals and it is responsible for reasoning about which goals and solution directions are feasible as well as which ones are most fit for use in the specified context. For the first part, it employs goal modeling techniques from the area of Requirements Engineering. Apart from concise visual representation, goal models are used for "what-if" analysis and reasoning. The second process that can be conducted by the Analyzer is the qualitative evaluation of alternative design patterns application, producing patterns prioritization based on user's goals.

The *Planner* is a core component of the quality enhancement phase, responsible for producing patterns on the ETL process that improve its quality, using as input information about the process structure, current estimated metrics and goals and available patterns prioritization. The available patterns toolset can be predefined and extended on a per case basis by the IT and the resulting model after the integration of patterns is a logical model. This model includes a set of configuration and management operations that are not directly related to the functionality of specific flow components, but are rather external to the process (e.g., security configurations). These operations are necessary to complete the palette of available improvement steps for the

satisfaction of quality goals.

Deployment and Execution. Once users observe satisfactory estimations for their measures of interest, they will decide that the quality of the process is acceptable and thus it is ready for deployment and execution.

In Chapter 3, we provide a more detailed view of this architecture, which has been published in (Theodorou et al., 2014b). Furthermore, in the following chapters, we delve deeper into each component of the *Quality Enhancement* phase of our proposed architecture and investigate raised research challenges.

1.5 CONTRIBUTIONS

The main objective of this thesis is to bring end-user requirements closer to ETL process implementation and to promote automation and analysis capabilities as inherent attributes of the process design. In this section, we present the main contributions of our conducted work. The first contribution stems from the literature review that we realized, enabling us to obtain a solid understanding of the research field; identify the issues and recognize the opportunities arising from active research works; and present throughout this document the open challenges and the state of the art of this field. Moreover, addressing the main objective of this thesis, we define a user-centered conceptual modeling method for ETL processes that can facilitate design decisions and runtime evaluation of alternative configurations based on user-defined goals, as presented in Section 1.4. Following, we further describe the more concrete contributions of this thesis mapped to the corresponding facets of data-intensive process design and maintenance.

Analysis and Evaluation. As a first step towards quality-aware ETL, we introduce a sound model for ETL process quality characteristics and the relationships among them, exposing a perspective of trade-offs between different ETL quality dimensions. The model is complete with regards to a systematic literature review that we conducted, through which we collected and classified all the ETL quality aspects that have been proposed in literature. In addition, we provide for each of these quality dimensions, measures and metrics that are backed by literature. Furthermore, we describe how this model can facilitate the use of Goal Modeling techniques for reasoning and making informed decisions about alternative ETL implementations, each of which can provide different fitness-to-use for end-users. We illustrate the latter by employing a goal model that supports the modeling of indicators and by applying it on alternative use-case ETL processes with the same functionality but with different quality characteristics.

Simulation. Regarding ETL process simulation, we present a framework for automatic, flow-aware generation of synthetic data to populate input data sources for supporting ETL process quality testing and design processes. To this end, we present an extensive ETL operations classification based on a review of popular ETL tools and we formalize their semantics for facilitating the automation of data generation. We introduce novel algorithms for ETL flow analysis and operations semantics extraction that largely support our data generation process. Generated data from our methodology guarantee the coverage of the complete ETL flow or parts of it, exposing its functional and non-functional properties. Our methodology supports highly parametrizable data generation and data generation properties (distributions, operations selec-

tivities) can be defined at various levels. To validate our approach, we implement a prototype of our data generation framework following a modular layered architecture that we also define, for parsing, extraction, analysis and data generation. Using the implemented prototype, we present experiments that show a linear trend of performance over ETL complexity, which indicates scalability prospects of our approach.

Maintenance and Redesign. Concerning ETL process maintenance and redesign, we introduce a novel empirical approach for mining ETL structural patterns. We formally define an ETL pattern model and we illustrate how this model is instantiated using a training set of ETL workflows to extract frequently reoccurring structural motifs. We provide a pattern-based analysis of ETL workflows, using the main control flow patterns from the *Workflow Patterns initiative* as a guide and we describe how ETL workflows can be modeled using a graph representation. This modeling enables the use of graph algorithms, such as frequent subgraph discovery algorithms for the mining phase and graph matching algorithms for the recognition phase. For the latter, we adapt the VF2 algorithm (Cordella et al., 2001) with some optimizations and we show through experiments how it can perform very well for ETL workflows. To this end, we implement ETL processes from the TPC-DI framework to perform our experimental evaluation. In addition, we present the most frequent ETL patterns that we identified in a set of 25 of these implemented processes, as well as the results from different configurations of the used algorithms. Moreover, we illustrate how ETL patterns can be used for the translation of logical ETL models to their conceptual representation and for their quality-based evaluation at the granular level of patterns. Finally, we implement a prototype of the *Planner* component of our architecture and showcase how it can be used together with a repository of *quality improvement patterns* to facilitate semi-automated quality-aware ETL redesign in a user-centered, iterative manner.

1.6 THESIS OUTLINE

In Figure 1.3, we illustrate how different chapters of this thesis are related to the different components of the *Quality Enhancement* phase of our architecture. In our view, the *Functionality-Based Design* and the *Deployment and Execution* phases of our architecture have been adequately covered by existing literature and thus extend beyond the scope of the research work presented in this thesis. Moreover, the former has recently been extensively addressed in the PhD thesis of a fellow colleague, Petar Jovanovic (Jovanovic, 2016) and the *ETL Process Engine* component of the latter, has been designed and implemented during our collaborative work with a master student Anton Kartashov, working on his master thesis (Kartashov, 2016).

After presenting the state of the art in our research area in Chapter 2, in Chapter 3, we take a first step towards quality-aware ETL automation by defining a sound model for ETL process quality characteristics and quantitative measures for each characteristic, based on existing literature. Our model shows the dependencies among quality characteristics and can provide the basis for subsequent analysis using Goal Modeling techniques. Hence, in this chapter, we define the conceptual model used in the *Analyzer* component of our architecture, which is subsequently used for our analysis and evaluation of ETL processes. We showcase the use of Goal Modeling for ETL process design through a use case, where we employ the use of a goal model

that includes quantitative components (i.e., indicators) for evaluation and analysis of alternative design decisions.

In Chapter 4, we present our proposed automatic data generator (i.e., *Bijoux*) for ETL process evaluation. Starting from a given ETL process model, *Bijoux* extracts the semantics of data transformations, analyzes the constraints they imply over input data, and automatically generates testing datasets. *Bijoux* is highly modular and configurable to enable end-users to generate datasets for a variety of interesting test scenarios (e.g., evaluating specific parts of an input ETL process design, with different input dataset sizes, different distributions of data, and different operation selectivities). Thus, the analysis presented in this chapter addresses the issues related to the *Process Simulator* component of our architecture. We developed a running prototype that implements the functionality of our data generation framework and we report our experimental findings showing the effectiveness and scalability of our approach.

In Chapter 5, we propose a bottom-up methodology for the identification of ETL structural patterns. We logically model the ETL workflows using labeled graphs and employ graph algorithms to identify candidate patterns and to recognize them on different workflows, covering the creation and utilization of the *Patterns* repository shown in our architecture. We showcase our approach through a use case that is applied on implemented ETL processes from the TPC-DI specification and we present mined ETL patterns. Our results suggest that our approach can be used for the automatic translation of ETL workflows to their conceptual representation and to generate fine-grained cost models at the granularity level of patterns. Given a training set of ETL process models, our methodology can build an ad-hoc repository of patterns and insights on them. Subsequently, this intelligence can lead to informed design decisions about ETL (re-)design, facilitating the evaluation of ETL workflows without the need for their potentially costly execution.

In Chapter 6, we present a tool, called *POIESIS*, for automatic ETL process enhancement. *POIESIS* is the implementation of the *Planner* component of our architecture and thus, it provides a user-centered environment for quality-aware analysis and redesign of ETL flows. It generates thousands of alternative flows by adding flow patterns to the initial flow, in varying positions and combinations, creating alternative design options in a multidimensional space of different quality attributes. Through the presentation of POIESIS we introduce the tool's capabilities and highlight its efficiency, usability and modifiability, thanks to its polymorphic design.

Finally, in Chapter 7, we summarize this thesis and discuss future work directions.

STATE OF THE ART

ETL processes consist of software components and are operated by BU, handling data artifacts and being themselves business processes that are crucial assets for enterprises at operational, tactical and strategic levels. In this chapter, we present the state of the art in our research area, discussing the strengths and limitations of current approaches, as well as opportunities that arise from taking under consideration the multidisciplinary nature of ETL processes.

In Section 2.1, we visit approaches on ETL modeling, that foster ETL design automation and expose quality dimensions, which need to be further investigated. The proposed modeling opens up new paths towards ETL analysis and evaluation and offers the possibility to apply novel optimization techniques to better align ETL processes with business needs. Regarding quality attributes, in Section 2.2 we review literature that has examined various quality concerns in data-intensive activities and data warehousing in specific. We illustrate the contributions of these approaches and we stress some points that need to be extended or examined more carefully. Subsequently, in Section 2.3 we review interesting projects in the area of data generation for testing and benchmarking databases, software and workflow processes and we identify ideas that we can adapt for our case and gaps that have yet to be addressed for ETL process evaluation. Finally, in Section 2.4 we discuss the state of the art in pattern-based analysis of ETL processes and point out the missing angles that our work intends to cover.

2.1 ETL MODELING

Modeling and optimization of ETL processes has been an active research area over the past few years, as an integral part of Data Warehousing and information discovery. In (Vassiliadis et al., 2002), ETL activities are formally defined and classified and in (Castellanos et al., 2009) the foundation for ETL automation is set by recognizing patterns and by defining appropriate generic templates for populating business process data warehouses as a response to business events. The authors abstract the warehousing process and introduce a process-based high-level analysis of ETL design that is independent of concrete implementations. Their comprehensive approach concerns the conceptual representation of ETL processes as well as its mapping to the logical design, in an agile way that can support near real-time monitoring and analysis.

Regarding conceptual representation in particular, in (Simitsis et al., 2005), a workflow paradigm is adopted for the modeling of ETL processes and a searching method is proposed considering a state space where each workflow is a state. Concerning the peculiar nature of ETL workflows, the authors consider alternative configurations of ETL processes and examine their optimization by reassembling the execution of activities within these processes. In the same direction, in (Vassiliadis et al., 2009), a classification of ETL activities is provided, through investigating the particular characteristics of ETL workflows. Thus, the authors introduce a formal representation of workflows and activities based on patterns that they identify, regarding the effect of these activities on data and the dependencies between them. The categorization provided can constitute the basis for ETL process analysis and optimization by exposing the interrelations between different data operators and the contribution of each activity to the outcome of the ETL process. Despite the fact that the above-mentioned work provides a solid conceptual background for ETL processes as workflows that consist of well-defined building blocks, we believe that the level of abstraction should be raised higher in order for ETL mod-

els to be usable to non-technical business users. In this respect, although these models offer the capability of reasoning for low-level, logical optimization, it is important to improve ETL process modeling in order to facilitate their assessment according to end-user requirements.

In an attempt to manage ETL processes on a conceptual level that reflects organizational operations, it has been suggested that tools and models from the area of Business Process Management (BPM) should be used. BPM is the holistic approach that considers business processes as the center of business management in order to improve organization's performance in agile and flexible environments, as explained by (Weske, 2012). BPM researchers and practitioners have developed a wide set of tools and methodologies for business process evaluation, redesign and automation (Dumas et al., 2013). Thus, ETL processes could benefit from the reuse of standards and the accumulated knowledge and heuristics that have been successfully applied and tested in production environments over the past years. One other advantage that BPM can offer to ETL processes is the use of models that can expose warehousing activities as part of business operations and provide a useful perspective both to business managers and IT. Following this concept, in (Wilkinson et al., 2010), business process models for a conceptual view of ETL are proposed that can depict the dynamic nature of such processes in a real-time angle. One important aspect of this work is that it captures business requirements –both functional and non-functional– to drive the design of ETL processes using the Business Process Modeling Notation (BPMN)[1]. To achieve this conceptual modeling it considers BPMN activities as ETL data flow and it identifies other patterns that map to control flow specifically for ETL processes. In addition, it transfers quality objectives through all design levels by annotating the model with XML-like constructs that the authors call *QoX metrics*. Consequently it explains the relationship between different layers of abstraction for the processes and how the conceptual level can be mapped to logical level and tool-specific implementations. This modeling framework can be a stepping stone on our way to embed user goal semantics to ETL processes, but would need to be extended from its current form that focuses on performance metrics.

In a similar manner, (Akkaoui et al., 2013) focuses on the conceptual level and provides a more specific classification of ETL data flow and control flow by presenting a BPMN-based meta-model for ETL processes and describing a systematic approach of code generation from BPMN models. The meta-model in this work derives from a study of various existing industrial ETL tools and their conceptual modeling is illustrated through a running example. These approaches make the crucial step of examining ETL models in an outlook that brings them closer to the business core and allows their validation and evaluation based on user input. However, we believe that these models should be even more flexible and support the definition of quality parameters at more angles. Thus, the mapping of conceptual representation to logical operators is not sufficient – neither for the understanding of ETL dynamics from business users nor for the reasoning for process re-engineering corresponding to requirements.

Several different approaches to ETL modeling and automation have also been proposed. For example, (Muñoz et al., 2008; Muñoz et al., 2009) proposes an MDA[2] approach for the development of ETL processes, where ETL activities constitute the block units of UML[3] activity diagrams among which there is control flow. The proposed framework can enable automatic

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

[2] http://www.omg.org/mda/

[3] http://www.uml.org/

code generation from the conceptual model, using tools and techniques from the area of Model Driven Engineering (MDE). Using UML for modeling and QVT[4] for model transformation, it provides an implementation-hiding meta-model for ETL as well as the required methodology for the development of ETL processes. This approach fosters model checking and automation for ETL processes, but perhaps makes it challenging to connect them with other business processes and organizational resources.

MDA also plays a significant role in (Böhm et al., 2009b,a), which identifies development effort, portability and efficiency as the main issues regarding integration processes. To tackle the first two, it proposes model-driven generation and the distinction of different levels for integration processes– conceptual, logical and physical level, however in a different view compared with (Wilkinson et al., 2010). It considers the generic domain of integration processes and it assumes that the implementation can be realized using different integration systems (ETL, FDBMS and others). In order to cope with performance, it makes an additional abstraction on the logical level that enables platform-independent optimization of integration processes. This approach is introduced as the GCIP framework for which a macro-architecture is provided. The proposed framework enables the use of model-driven techniques for the rule-based and workload-based optimization of complex integration processes, by configuring their internal logical design as well as by selecting the most efficient target integration system to implement the process. This framework can provide the baseline for performance-based optimization of ETL processes on a logical level and propagation of configurations to low level implementation with minimal user interaction.

Another interesting approach that formalizes data operations is the artifact-centric approach to design and development of business processes. (Bhattacharya et al., 2007) studies feasibility issues of artifact-centric models that can couple the data and control flow of business processes. Based on business artifacts, rules and services, a business process model is defined that captures the complete life-cycle and semantics of artifacts that are important for the achievement of business goals. Similarly, (Bagheri Hariri et al., 2011) adopts an artifact-centric approach specifically for relational artifacts, where data are represented by relational databases. It uses μ-calculus to represent properties and actions through which business artifacts transit to different states and it shows how formal analysis concludes about the decidability of reasoning over "weakly" acyclic processes. This approach facilitates the formal representation of rules and states within the business process life-cycle and thus the ground-basis for proving models' feasibility, decidability, integrity and so on. Nonetheless, besides their theoretical value, the difficulty of these models to be comprehended and their limited expressiveness as a price for their completeness, makes it challenging to be used in practice in BI context.

Summing up, it has been suggested in literature that ETL processes can be modeled in various ways, each exposing some perspective that can take advantage of existing work in related research areas. Although it has been shown how such modeling can promote performance optimization and cost-efficiency, a more high-level modeling that can extend these ideas and make the ETL design process more user-centered, focusing on bridging end user goals to ETL activities, is still missing.

[4] http://www.omg.org/spec/QVT/1.1/

2.2 QUALITY OF DATA INTENSIVE PROCESSES

The approaches mentioned so far focus on ETL process executability and performance, but it is apparent that ETL process models need to be enriched with some notion of data utility to the business users, as a result of process characteristics. Current research covers the area of Data Quality, mostly concerning data consistency, data deduplication, data accuracy, information completeness and data freshness. For example (Batini et al., 2009) defines the basic data quality issues that form the basis for data quality assessment and improvement process. Based on that it compares existing methodologies towards that direction and thus presents the main techniques that are available as well as their applicability to different situations. Similarly, (Sadiq et al., 2011) defines and categorizes most important issues of data quality management through extensive literature study, stressing the interdisciplinarity of the field, where business analysts, solution architects and database experts all come up with different approaches towards data quality enforcement. It classifies quality concerns, metrics and locate trends and synergies among different communities in the area over the past two decades.

A more recent approach that focuses on data quality in service-oriented environments (Dustdar et al., 2012) stresses the importance of quality parameters for today's data that is being exchanged between different entities in very high rates. It identifies the challenges of using Service Oriented Architecture (SOA) concepts such as Service Level Agreements (SLAs) within data networks and proposes a new research direction for quality-aware data services. Such an approach can also be followed for ETL processes where the data service is the processing of source data for the destination warehouse. It is obvious that data quality is not the only concern for the optimal design of ETL processes and other, user-centered requirements should be considered as well, such as data availability, reliability, security and process cost-efficiency. Particularly about data quality cost, a comprehensive classification is provided in (Eppler and Helfert, 2004) that points out the considerable losses –direct as well as indirect– deriving from low quality of data.

Quality has always been a central concern for data warehouses, mainly due to the costly resources invested in their development. Focusing on Information Quality, (Naumann, 2002) provides a comprehensive list of criteria for the evaluation of Information Systems for data integration. Likewise, (Jarke et al., 2003) identifies the various stakeholders in Data Warehousing activities and the differences in their roles as well as the importance of reasoning among alternative quality concerns and how that affects design choices. (Jarke et al., 1999) takes the important step of introducing a business perspective to data warehouse quality, by introducing an extended architecture with different conceptual levels, which enables separation of concerns as well as traceability from client intentions to physical data models. It also captures the notion of goal negotiation for conflicting quality goals.

The significance of quality characteristics for the design and evaluation of ETL processes has also gained attention recently. (Simitsis et al., 2009b) recognizes the importance of considering not only process functionality but also quality metrics throughout a systematic ETL process design. Thus, it defines a set of quality characteristics specific to ETL processes (*QoX metrics*) and provides guidelines for reasoning about the degree of their satisfaction over alternative designs and the tradeoffs among them. A more recent work that has also considered the ideas from Simitsis et al. (2009b) is (Pavlov, 2013). Based on well-known standards for

Feature	(Dustdar et al., 2012)	(Naumann, 2002)	(Jarke et al., 2003, 1999)	(Simitsis et al., 2009b)	(Pavlov, 2013)
Highlight data quality	Yes	Yes	Yes	Yes	No
Identification of quality tradeoffs	No	Partially, quality in multi-dimensional space	No	Yes and tentative methodology proposed	No
Connection to design and/or implementation decisions	Information service selection	Data sources selection	Good practices for DW	Improvement steps for specific quality aspect	Only connection to different ETL subsystems

Figure 2.1: Comparison of approaches on ETL quality

software quality, the author maps software quality attributes to ETL specific parameters which he calls *QoX factors*. He defines these factors in ETL context and reasons about the impact that the different ETL subsystems might have on each characteristic.

In the last years, there has been an effort in the area of Business Process Management to quantify process quality characteristics and to empirically validate the use of well-defined metrics for the evaluation of specific quality characteristics. In this respect, (García et al., 2004) proposes a framework for managing, modeling and evaluating software processes; defines and experimentally validates a set of measures to assess, among others understandability and modifiability of process models. Similar empirical validation is provided by (Sánchez-González et al., 2012), which relates understandability and modifiability to innate characteristics of business process models.

To sum up, the area of ETL process quality characteristics has not yet been studied thoroughly enough to take advantage of applicable advances from other research areas. For instance, data quality is a well-studied area and it needs to be highlighted as a core characteristic, that is directly affected by design choices regarding the ETL process. Although some approaches have identified the potential of recognizing different perspectives of ETL processes (e.g., business process perspective), a holistic approach that combines and extends these ideas, making them applicable to practical ETL analysis and design scenarios, is still missing.

In Figure 2.1, we show a comparative summary of the data-intensive process quality approaches mentioned in this section. As we show, the tradeoff analysis of how improving some quality dimension can affect others has not yet been explicitly covered. The same holds for the relationship between quality measures and actionable process design decisions. Our approach differs from the above-mentioned ones in that we specifically focus on the process perspective of ETL processes. Instead of providing some characteristics as examples like (Simitsis et al., 2009b), we propose a comprehensive list of quality characteristics and we adjust them for our case, also describing the tradeoffs among them. In addition, for each of these characteristics we provide quantitative metrics that are backed by literature.

2.3 DATA INTENSIVE PROCESSES TESTING AND EVALUATION

Software development and testing. In the software engineering field, *test-driven development* has studied the problem of software development by creating tests cases in advance for each newly added feature in the current software configuration (Beck, 2003). However, in our work, we do not focus on the design (i.e., development) of ETL processes per se, but on automating the evaluation of quality features of the existing designs. We analyze how the semantics of ETL processes entail the constraints over the input data, and then consequently create the testing data. Similarly, the problem of constraint-guided generation of synthetic data has been also previously studied in the field of software testing (DeMillo and Offutt, 1991). The context of this work is the mutation analysis of software programs, where for a program, there are several "mutants" (i.e., program instances created with small, incorrect modifications from the initial system). The approach analyzes the constraints that "mutants" impose to the program execution and generates data to ensure the incorrectness of modified programs (i.e., "to kill the mutants"). This problem resembles our work in a way that it analyzes both the constraints when the program executes and when it fails to generate data to cover both scenarios. However, this work mostly considered generating data to test the correctness of the program executions and not its quality criteria (e.g., performance, recoverability, reliability, etc.).

Data generation for relational databases. Moving toward the database world, (Zhang et al., 2001) presents a fault-based approach to the generation of database instances for application programs, specifically aiming to the data generation problem in support of white-box testing of embedded SQL programs. Given an SQL statement, the database schema definition and tester requirements, the approach generates a set of constraints, which can be given to existing constraints solvers. If the constraints are satisfiable, a desired database instances are obtained. Similarly, for testing the correctness of relational DB systems, a study in (Chays et al., 2000) proposes a semi-automatic approach for populating the database with meaningful data that satisfy database constraints. Work in (Arasu et al., 2011) focuses on a specific set of constraints (i.e., cardinality constraints) and introduces efficient algorithms for generating synthetic databases that satisfy them. Unlike the previous attempts, in (Arasu et al., 2011), the authors generate synthetic database instance from scratch, rather than by modifying the existing one. Furthermore, (Binnig et al., 2007) proposes a query-aware test database generator called *QAGen*. The generated database satisfies not only constraints of database schemata, table semantics, but also the query along with the set of user-defined constraints on each query operator. Other work (Houkjær et al., 2006) presents a generic graph-based data generation approach, arguing that the graph representation supports the customizable data generation for databases with more complex attribute dependencies. The approach most similar to ours (Lakhotia et al., 2007) proposes a multi-objective test set creation. They tackle the problem of generating "branch-adequate" test sets, which aims at creating test sets to guarantee the execution of each of the *reachable* branches of the program. Moreover, they model the data generation problem as a multi-objective search problem, focusing not only on covering the branch execution, but also on additional goals the tester might require, e.g., memory consumption criterion. However, the above works focus solely on relational data generation by resolving the constraints of the

existing database systems. Our approach follows this line, but in a broader way, given that *Bijoux* is not restricted to relational schema and is able to tackle more complex constraint types, not supported by the SQL semantics (e.g., complex user defined functions, pivot/unpivot). In addition, we do not generate a single database instance, but rather the heterogeneous datasets based on different information (e.g., input schema, data types, distribution, etc.) extracted from the ETL flow.

Benchmarking data integration processes. In a more general context, both research and industry are particularly interested in benchmarking ETL and data integration processes in order to evaluate process designs and compare different integration tools. The vision of invisible deployment of integration processes (Böhm et al., 2009a) could not be realized effectively without the use of defined performance metrics that can expose performance characteristics of processes and constitute concrete toolsets for their evaluation. (Böhm et al., 2008) introduces a scalable framework for benchmarking of possibly heterogeneous integration systems. Inspired by real-life scenarios, it identifies three scale factors– datasize, time and distribution– and provides a platform-independent benchmark that estimates the cost of integration processes. The authors provide a macro- and micro-architecture of their framework and they claim modifiability and usability of the framework in order to be effortlessly tuned and integrated with existing systems for specific use-cases. Plenty other tools have also been proposed for benchmarking of workflow processes and for evaluating integration functionalities. Specifically for ETL processes, (Simitsis et al., 2009a) identifies the most common operational constructs in ETL environments and provides a classification of ETL operations and analysis of how they affect the functionality of process execution. This approach can be seen as a first step towards formalizing common ETL operations that are found in multiple ETL tools as well as their functional implications.Both these works note the lack of a widely accepted standard for evaluating data integration processes. The former work focuses on defining a benchmark at the logical level of data integration processes, meanwhile assessing optimization criteria as configuration parameters. Whereas, the later works at the physical level by providing a multi-layered benchmarking platform called *DIPBench* used for evaluating the performance of data integration systems. These works also note that an important factor in benchmarking data integration systems is defining similar workloads while testing different scenarios to evaluate the process design and measure satisfaction of different quality objectives. These approaches do not provide any automatable means for generating benchmark data loads, while their conclusions do motivate our work in this direction.

General data generators. Other approaches have been working on providing data generators that are able to simulate real-world data sets for the purpose of benchmarking and evaluation. (Gray et al., 1994) presents one of the first attempts of how to generate synthetic data used as input for workloads when testing the performance of database systems. It mainly focuses on the challenges of how to scale up and speed up the data generation process using parallel computer architectures. In (Ming et al., 2014), the authors present a tool called Big Data Generator Suite (BDGS) for generating Big Data meanwhile preserving the 4V characteristics of Big Data[5]. BDGS is part of the BigDataBench benchmark (Luo et al., 2014) and it is used to generate textual, graph and table structured datasets. BDGS uses samples of real world data,

[5] *volume*, *variety*, *velocity* and *veracity*

analyzes and extracts the characteristics of the existing data to generate loads of "self-similar" datasets. In (Rabl et al., 2010), the parallel data generation framework (PDGF) is presented. PDGF generator uses XML configuration files for data description and distribution and generates large-scale data loads. Thus its data generation funcionalities can be used for benchmarking standard DBMSs as well as the large scale platforms (e.g., MapReduce platforms). Other prototypes (e.g., (Hoag and Thompson, 2007)) offer similar data generation functionalities. In general, this prototype allows *inter-rows*, *intra-rows*, and *inter-table* dependencies which are important when generating data for ETL processes as they must ensure the multidimensional integrity constraints of the target data stores. The above mentioned data generators provide powerful capabilities to address the issue of generating data for testing and benchmarking purposes for database systems. However, the data generation is not led by the constraints that the operations entail over the input data, hence they cannot be customized for evaluating different quality features of ETL-like processes.

Process simulation. Lastly, given that the simulation is a technique that imitates the behavior of real-life processes, and hence represents an important means for evaluating processes for different execution scenarios (Paul et al., 1998), we discuss several works in the field of simulating business processes. Simulation models are usually expected to provide a qualitative and quantitative analysis that are useful during the re-engineering phase and generally for understanding the process behavior and reaction due to changes in the process (Law et al., 1991). (Becker et al., 2003) further discusses several quality criteria that should be considered for the successful design of business processes (i.e., *correctness*, *relevance*, *economic efficiency*, *clarity*, *comparability*, *systematic design*). However, as shown in (Jansen-Vullers and Netjes, 2006) most of the business process modeling tools do not provide full support for simulating business process execution and the analysis of the relevant quality objectives. We take the lessons learned from the simulation approaches in the general field of business processes and go a step further focusing our work to data-centric (i.e., ETL) processes and the quality criteria for the design of this kind of processes (Simitsis et al., 2009b; Theodorou et al., 2016).

2.4 ETL PATTERNS

As described above, there has been considerable work in the area of ETL modeling, in an effort to promote automation through the definition of structural abstractions and systematic methodologies for the design and analysis of ETL models. The modeling of ETL processes using well defined, reusable components interacting as workflow activities has set the foundations for their pattern-based analysis and design. In the Business Process Management (BPM) community, such analysis has already taken place, with significant work conducted as part of the Workflow Patterns Initiative (Van Der Aalst et al., 2003; Russell et al., 2005). This work examines workflows and various different Workflow Management Systems (WfMSs) and identifies a set of recurring features (i.e., patterns). It takes under consideration the modeling languages used for the design and the modeling notation of business process models and extracts a number of patterns to describe mostly control-flow and data-flow semantics commonly offered by WfMSs. In (Gschwind et al., 2008), there is a practical illustration of how such patterns can be introduced and integrated into an existing business process model and in (Mussbacher

Feature	(Simitsis et al., 2009a)	(Oliveira and Belo, 2015; Oliveira et al., 2014, 2013)	(Van Der Aalst et al., 2003; Russell et al., 2005)
Application on ETL	Yes	Yes	Not straightforward
Pattern identification method	Expertise	Expertise	Tools examination
Pattern granularity	Complete ETL workflow	ETL tasks	Atomic steps
Pattern reusability	Not supported	High	Needs configurations
Connection with quality characteristics	Only performance but not well-studied	Only correctness	Not addressed

Figure 2.2: Comparison of pattern-based approaches

et al., 2010), the application of patterns on a business process is linked to the strategic decision making level, through its mapping to business goals and non-functional requirements (NFRs).

When it comes to patterns in ETL activities, in (Simitsis et al., 2009a) there is a profiling of ETL workflows with models called *Butterflies*, based on their form with regards to the distribution of activities relatively to the beginning (i.e., data extraction) and the ending (i.e., data loading) part(s) of the ETL process. Such categorization captures the idea of linking discrete ETL components to ETL requirements, e.g., by providing some indication about their computational or memory needs based on their structural morphology. However, it is not described in detail how the example ETL archetypes presented can co-exist as different parts of the same ETL process, nor is any valid methodology for the quantification of the relationships between Butterflies recognition and implications on the workflow proposed.

More recently, the authors in (Oliveira and Belo, 2015) further extend their line of research (Oliveira et al., 2014, 2013) on ETL patterns and propose the grouping of ETL operations to abstract their functionality and form known generic ETL activities. To this end, they formulate a pallet of most used data warehousing tasks in real world and propose the design of ETL processes by using workflows that comprise of customizations of these patterns. Although their work fosters reusability and correctness, one important limitation stems from the definition of universal patterns in a top down approach. In this respect, the pattern-based analysis of random ETL workflows that have been designed or implemented with the use of different technologies might not easily lead to their decomposition in a-priory classified components that are useful for the specific analysis. In other words, this approach assumes that the ETL workflow models comply to some arbitrary-built pattern classification, whereas we advocate that it would make more sense for patterns to be dynamically constructed in an ad-hoc fashion, based on the type of analysis on one hand and the type of examined workflows on the other.

In Figure 2.2, we show a comparison of the main features of the above-mentioned approaches, exposing their strengths and the open challenges. To our knowledge, there is currently no work suggesting the ETL pattern-oriented analysis using a bottom-up approach, where patterns are gathered in an evidence-based manner. Thus, our work is the first one to introduce

the idea of mining frequent ETL components to identify valid patterns for the purpose of the analysis, at an adequate granularity level, instead of relying on experience or expertise to define some stiff, universal motifs. In addition, it aims to open the door to quantitatively relating a number of different ETL quality characteristics to ETL patterns, which is not supported by existing approaches.

QUALITY MEASURES FOR ETL PROCESSES

ETL processes are centred around artifacts with high variability and diverse lifecycles, which correspond to key business entities. The apparent complexity of these activities has been examined through the prism of Business Process Management, mainly focusing on functional requirements and performance optimization. However, the quality dimension has not yet been thoroughly investigated and there is a need for a more human-centric approach to bring them closer to business-users requirements.

In this chapter, we address the challenge of ETL process quality modeling in order to promote automation and dynamicity, as described in Section 1.1. To this end, we identify the opportunities stemming from different ETL perspectives in order to bring a common ground between analysts and IT by connecting end-user requirements to design decisions. We take a first step towards quality-aware design automation by *defining a set of ETL process quality characteristics and the relationships between them, as well as by providing quantitative measures for each characteristic*. For this purpose, we conduct a systematic literature review, extract the relevant quality aspects that have been proposed in literature and adapt them for our case. Subsequently, we produce a model that represents ETL process quality characteristics and the dependencies among them. In addition, we gather from existing literature metrics for monitoring all of these characteristics to quantitatively evaluate ETL processes. Our model can provide the basis for subsequent analysis that will use Goal Modeling techniques (van Lamsweerde, 2001) to reason and make design decisions for specific use cases, as we showcase through the application of a Goal Model with quantitative components (i.e., indicators) to a running example.

The model for ETL quality measures presented in this chapter has been published in (Theodorou et al., 2014a) and an extended journal version of this paper, including the goal modeling techniques presented in this chapter, has been published in (Theodorou et al., 2016).

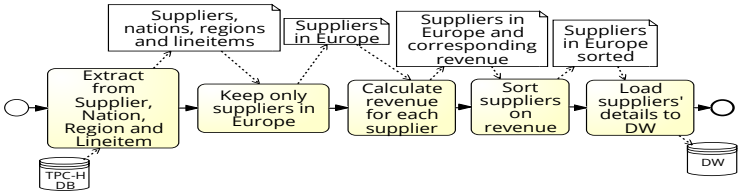


Figure 3.1: CM_A: A simple conceptual model of the running example ETL

Running Example. We illustrate how our model works through a running example, based on the TPC-H benchmark[1]. The running example is an ETL process, which extracts data from a source relational database (TPC-H DB) and after processing, loads data to a data warehouse (DW) and can be described by the following query: *Load in the DW all the suppliers in Europe together with their information (phones, addresses etc.), sorted on their revenue*. The tables that are used from the source database are *Supplier*, *Nation*, *Region* and *Lineitem*. After *Supplier* entries have been filtered to keep only suppliers in Europe, the revenue for each supplier is calculated based on the supplied lineitems and subsequently, they are sorted on revenue and loaded to the DW. One simple conceptual model (CM_A) for this process is depicted in Fig. 3.1, using the Business Process Model and Notation (BPMN[2]).

[1] `http://www.tpc.org/tpch/`
[2] `http://www.bpmn.org`

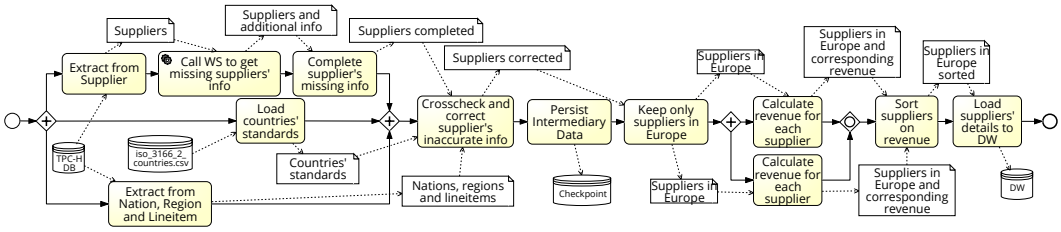


Figure 3.2: CM_B: Conceptual model of the running example ETL, including additional tasks

An ETL process can be designed in more than one way, each design offering different advantages over the other. In Fig. 3.2 we can see an alternative conceptual model (CM_B) for our example ETL process with the same functionality as CM_A (Fig. 3.1), yet including additional tasks. It includes a Web Service (WS) call to complete suppliers' info and improve data completeness; crosschecking with an external data source to correct information and improve data accuracy; replication of the revenue calculation step to improve robustness; and the addition of a recovery point to improve recoverability. In the following sections we will show through our running example how our proposed models can be used as a basis for i) quality-aware evaluation of ETL processes and ii) ETL process re-design that uses as input only the user-defined importance of different quality characteristics. For the former, we show how the two alternative designs of the same ETL process, despite having the same functionality, differ in quality characteristics in a way that can be quantified, using the measures that we have identified. For the latter, we showcase a user-centered method that starts with the design of CM_A; incrementally generates patterns on the process in order to improve specific quality characteristics; and (semi)-automatically produces the design of CM_B and its corresponding implementation.

The chapter is organized as follows. In Section 3.1, we present the extraction of our model from related work. The definitions, measures and dependencies among characteristics are presented in Section 3.2 and Section 3.3, where we distinguish between characteristics with construct implications and those only for design evaluation, respectively. In Section 3.4, we present goal modeling frameworks and in Section 3.5, we validate the usefulness of our models and showcase through examples how they can be used for quality-based ETL process evaluation and redesign. Finally, we provide our conclusions in Section 3.6.

3.1 EXTRACTING QUALITY CHARACTERISTICS

Our model mainly derives from a systematic literature review that we conducted, following the guidelines reported in (Kitchenham et al., 2009). The research questions addressed by this study are the following:
RQ1) What ETL process quality characteristics have been addressed?
RQ2) What is the definition for each quality characteristic?

Our search process used an automated keyword search of SpringerLink[3], ACM Digital Li-

[3] `http://link.springer.com`

	Characteristic	Barbacci et al. (1995)	Simitsis et al. (2009b)	Jarke et al. (2003)	Pavlov (2013)	Naumann (2002)	Dustdar et al. (2012)	kim (cited January 2014)
Characteristics with construct implications	**data quality**	-	data characteristics	quality dimensions	-	relevancy, reputation	data quality	data quality
	data accuracy			data accuracy		accuracy	accuracy	
	data completeness			data completeness		completeness	completeness	
	data freshness		freshness	data freshness, timeliness		timeliness	timeliness	
	data consistency		consistency	data coherence, correctness, minimality		consistent representation	consistency	consistency, deduplication, data conformance
	data interpretability			interpretability		interpretability		
	performance	performance	performance	performance, software efficiency	performance efficiency	-	performance	-
	time efficiency	latency	latency		time behaviour	latency, response time		paralleling & pipelining
	resource utilization				resource utilization			change data capture
	capacity	capacity, throughput				quality of service		
	modes	modes						types of fact tables
	cost efficiency	-	cost, affordability	-	-	value-added, price	pricing	-
	upstream overhead	-	overhead of source systems	-	upstream overhead	-	-	-
	security	security	-	security	-	security	security	security
	confidentiality	confidentiality						compliance management
	integrity	integrity						
	availability	availability	availability	availability			availability	availability
	auditability	-	auditability	-	auditability	-	provenance	lineage & dependency
	traceability		traceability	traceability	traceability	documentation		self-documenting
	reliability	reliability	reliability	reliability	reliability	-	reliability	reliability
	process availability	availability		responsiveness	availability	availability		
	fault tolerance	fault tolerance			fault tolerance			
	robustness	integrity	robustness		robustness			
	recoverability		recoverability		recoverability			recoverability, problem escalation
Characteristics for design evaluation	**adaptability**	-	-	-	adaptability	-	-	-
	scalability		scalability		scalability			scalability
	flexibility		flexibility					
	reusability			portability	reusability			
	usability	-	-	usability	-	-	-	visibility
	understandability					concise representation, understandability		understanding source data
	manageability	-	-	-	modularity, analyzability	-	-	manageability
	maintainability	maintainability	maintainability	maintainability	maintainability			maintainability
	testability			validation	testability	verifiability		
n/a		safety		accessibility, usefulness, believability		customer support, believability, objectivity, amount of data	licencing	

Figure 3.3: ETL Process Characteristics

brary[4], ScienceDirect[5] and IEEE Xplore[6]. Thus, we searched not one, but multiple electronic resources, following the guidelines from (Brereton et al., 2007). These electronic resources were chosen because of their popularity within the software engineering community — and the data warehousing research domain in specific — and because we found during our review planning phase that other indexing services (e.g., Citeseer library, Google scholar) only include for our topic of interest, a subset of the material found in our chosen resources. Our goal was to gather all *peer-reviewed* studies related to our search and therefore we selected a set of resources that is complete with respect to international conference publications, journals and books in the research area of data warehousing.

The search strings were the following:

- (quality attributes OR quality characteristics OR qox) AND ("etl" OR "extraction transformation loading") AND ("information technology" OR "business intelligence")
- (quality attributes OR quality characteristics OR qox) AND ("data integration" OR "information systems integration" OR "data warehouses") AND ("quality aware" OR "quality driven")

The inclusion criterion for the studies was that they should identify a wide range of quality characteristics for data integration processes and thus only studies that mentioned at least 10 different quality characteristics were included. The quality characteristics could refer to any stage of the process as well as to the quality of the target repositories as a result of the process. We should mention at this point that there exists a large number of studies focusing specifically on the quality dimension of Data Quality, but the rationale of our search was to gather all different quality dimensions previously studied, which justifies our inclusion criterion. One exclusion criterion was that studies should be written in English. Whenever multiple studies from same line of work were identified, our approach was to include the most recent or the most extensive version of the study.

The result of our selection process was a final set of 5 studies. Nevertheless, in an attempt to improve the completeness of our sources, we also considered the ETL subsystems as defined in (kim, cited January 2014) for an industry perspective on the area, as well as standards from the field of software quality. Regarding software quality, our approach was to study the work in (Barbacci et al., 1995) and include in our model all the attributes relevant to ETL processes, with the required definition adjustments. This way we reviewed a commonly accepted, generic taxonomy of software quality attributes, while at the same time avoiding the adherence to more recent, strictly defined standards for practical industrial use, which we are nevertheless aware of (Al-Qutaish, 2009). The complete list from the resulting 7 sources, covering the most important characteristics from a process perspective can be seen in Fig. 3.3. In many cases we discovered that the same quality characteristic was referenced using a different term (synonym) and that term is shown at the corresponding table cell under each approach.

Data quality is a prime quality chracteristic of ETL processes. Its significance is recognized by all the approaches presented in our selected sources, except for (Pavlov, 2013) and (Barbacci

[4] `http://dl.acm.org`
[5] `http://www.sciencedirect.com/`
[6] `http://ieeexplore.ieee.org`

et al., 1995), since the factors in their analyses derive directly or indirectly from generic software quality attributes. Our model was enriched with a more clear perspective of data quality in Information Systems and a practical view of how quality criteria can lead to design decisions, after reviewing the work in (Naumann, 2002). Although this framework is intended for interoperation of distributed Information Systems in general, many aspects are clearly applicable in the case of ETL processes where data sources can be found in diverse locations and types.

Performance, was given attention by all presented approaches , which was expected since time behavior and resource efficiency are the main aspects that have traditionally been examined as optimization objectives. On the other hand, (Pavlov, 2013) and (Simitsis et al., 2009b) were the only approaches to include the important characteristic of *upstream overhead*. However, (Pavlov, 2013) does not include *security*, which is discussed in the rest of the works. The same is true for *auditability*, which is absent from (Barbacci et al., 1995) but found in all other works. Reliability on the other hand, is recognized as a crucial quality factor by all approaches. As expected, the more abstract quality characteristics *adaptability* and *usability* are less commonly found in the sources, in contrast with *manageability* which is found in all approaches except for (Dustdar et al., 2012), which does not discuss about intangible characteristics.

Although we include *cost efficiency* in Fig. 3.3, in the remainder of this chapter this characteristic is not examined as the rest. The reason is that we view our quality-based analysis in a similar perspective as (Kazman et al., 2001), who consider cost as a separate concern to the rest of the attributes, according to which any quality attribute can be improved by spending more resources and it is a matter of weighting the benefits of this improvement to the required cost that can lead to rational decisions. In our case, cost is considered to act as a restriction to the search space of alternative process designs. In addition, we regarded *safety* as non-relevant for the case of ETL processes, since these processes are computer-executable, non-critical and hence the occurrence of accidents or mishaps is not a concern. Similarly, we considered that the characteristics of *accessibility*, *usefulness*, *customer support*, *believability*, *amount of data* and *objectivity* found in (Jarke et al., 2003) and (Naumann, 2002) are not relevant for our case, as they refer to the quality of source or target repositories, yet do not depend on the ETL process. Likewise, *licencing* (Dustdar et al., 2012) refers to concrete tools and platforms while our ETL quality analysis is platform independent.

Thus, we regarded for our models only characteristics that are related to the process perspective of the ETL and other aspects, such as characteristics of the source or target repositories, which are independent of the process, were considered to extend beyond the concern of this study. In this respect, we disregarded ETL quality dimensions specific to the target data warehouse, such as the quality of OLAP modeling (e.g., schemata, hierarchies). Nevertheless, our assumption is that the ETL processes evaluated by our models are functionally correct and produce output corresponding to the target repositories' data modeling. Hence, as we will show in the following sections, our method for process redesign receives as input an ETL process that has been designed respecting the *information requirements* of the data warehouse.

Through our study we identified that there are two different types of characteristics — characteristics that can actively drive the generation of patterns in the ETL process design and characteristics that cannot explicitly indicate the use of specific design patterns, but can still be measured and affect the evaluation of and the selection among alternative designs. In the remainder of this chapter we refer to the first category as *characteristics with construct implications*

and to the second as *characteristics for design evaluation*.

3.2 CHARACTERISTICS WITH CONSTRUCT IMPLICATIONS

In this section, we present our model for characteristics with construct implications. Subsequently, we show the relationships between different characteristics and showcase the use of our model through our running example. The proposed list of characteristics and measures can be extended or narrowed down to match the requirements for specific use cases. Thus, our model is extensible and can constitute a template, generically capturing the quality dimensions of ETL processes and their interrelationships, which can be modified and instantiated per case. The definition of each characteristic can be adjusted; the proposed measures are only mentioned as valid examples that can be extended or replaced by other more appropriate ones; users can decide to use only an excerpt of the model; and the quantitative effect that each characteristic can have to another can differ for different cases. The above points also stand for the model of the following section, about characteristics for design evaluation.

3.2.1 Characteristics and Measures

In this subsection, we provide a definition for each characteristic as well as candidate metrics under each definition, based on existing approaches that we discovered coming from literature and practice in the areas of Data Warehousing and Software Engineering. For each metric there is a definition and a symbol, either (+) or (−) denoting whether the maximization or minimization of the metric is desirable, respectively. Similarly to the quality characteristics, the measures come from the areas of Data Warehousing, ETL, Data Integration and Software Engineering and Business Process Management.

1. *data quality (DQ)*: the fitness for use of the data produced as the outcome of the ETL process. It includes:

   (a) *data accuracy*: percentage of data without data errors.

   M1: % of correct values (Batini et al., 2009) (+)

   M2: % of delivered accurate entities (Batini et al., 2009) (+)

   (b) *data completeness*: degree of absence of missing values and entities.

   M1: % of missing entities from their appropriate storage (Simitsis et al., 2009a; Batini et al., 2009) (−)

   M2: % of non-empty values (Batini et al., 2009) (+)

   (c) *data freshness*: indicator of how recent data is with respect to time elapsed since last update of the target repository from the data source.

   M1: Instant when data are stored in the system - Instant when data are updated in the real world (Batini et al., 2009) (−)

   M2: Request time - Time of last update (Batini et al., 2009) (−)

   M3: 1 / (1 - age * Frequency of updates) (Batini et al., 2009) (−)

(d) *data consistency*: degree to which each user sees a consistent view of the data and data integrity is maintained throughout transactions and across data sources.

M1: % of entities that violate business rules (Simitsis et al., 2009a; Batini et al., 2009) (−)

M2: % of duplicates (Batini et al., 2009) (−)

(e) *data interpretability*: degree to which users can understand data that they get.

M1: # of entities with interpretable data (documentation for important values) (Batini et al., 2009) (+)

M2: Score from User Survey (Questionnaire) (Batini et al., 2009) (+)

2. *performance (PF)*: the performance of the ETL process as it is implemented on a system, relative to the amount of resources utilized and the timeliness of the service delivered. It includes:

(a) *time efficiency*: the degree of low response times, low processing times and high throughput rates.

M1: Process cycle time (Majchrzak et al., 2011) (−)

M2: Average latency per entity in regular execution (Simitsis et al., 2009a) (−)

M3: Min/Max/Average number of blocking operations (Simitsis et al., 2009a) (−)

(b) *resource utilization*: the amounts and types of resources used by the ETL process.

M1: CPU load, in percentage of utilization (Majchrzak et al., 2011) (−)

M2: Memory load, in percentage of utilization (Majchrzak et al., 2011) (−)

(c) *capacity*: the demand that can be placed on the system while continuing to meet time and throughput requirements.

M1: Throughput of regular workflow execution (Simitsis et al., 2009a) (+)

(d) *modes*: the support for different modes of the ETL process based on demand and changing requirements, for example batch processing, real-time event-based processing, etc.

M1: Number of supported modes / Number of all possible modes (+)

3. *upstream overhead (UO)*: the degree of additional load that the process causes to the data sources on top of their normal operations.

M1: Min/Max/Average timeline of memory consumed by the ETL process at the source system (Simitsis et al., 2009a) (−)

4. *security (SE)*: the protection of information during data processes and transactions. It includes:

(a) *confidentiality*: the degree to which data and processes are protected from unauthorized disclosure.

M1: % of mobile computers and devices that perform all cryptographic operations using FIPS 140-2 cryptographic modules (Chew et al., 2008) (+)

M2: % of systems (workstations, laptops, servers) with latest antispyware signatures (kpi, cited January 2014) (+)

M3: % of remote access points used to gain unauthorized access (Chew et al., 2008) (−)

M4: % of users with access to shared accounts (Chew et al., 2008) (−)

(b) *integrity*: the degree to which data and processes are protected from unauthorized modification.

M1: % of systems (workstations, laptops, servers) with latest antivirus signatures (kpi, cited January 2014) (+)

(c) *reliability*: the degree to which the ETL process can maintain a specified level of performance for a specified period of time. It includes:

i. *availability*: the degree to which information, communication channels, the system and its security mechanisms are available when needed and functioning correctly.

M1: Mean time between failures (MTBF) (Simitsis et al., 2009a) (+)

M2: Uptime of ETL process (Simitsis et al., 2009a) (+)

ii. *fault tolerance*: the degree to which the process operates as intended despite the presence of faults.

M1: Score representing asynchronous resumption support (Simitsis et al., 2009a) (+)

iii. *robustness*: the degree to which the process operates as intended despite unpredictable or malicious input.

M1: # of replicated processes (Simitsis et al., 2009a) (+)

iv. *recoverability*: the degree to which the process can recover the data directly affected in case of interruption or failure.

M1: # of recovery points used (Simitsis et al., 2009a) (+)

M2: % of successfully resumed workflow executions (Simitsis et al., 2009a) (+)

M3: Mean time to repair (MTTR) (Simitsis et al., 2009a) (−)

5. *auditability (AU)*: the ability of the ETL process to provide data and business rule transparency. It includes:

(a) *traceability*: the ability to trace the history of the ETL process execution steps and the quality of documented information about runtime.

M1: % of KPIs that can be followed, discovered or ascertained by end users (Leite and Cappelli, 2010) (+)

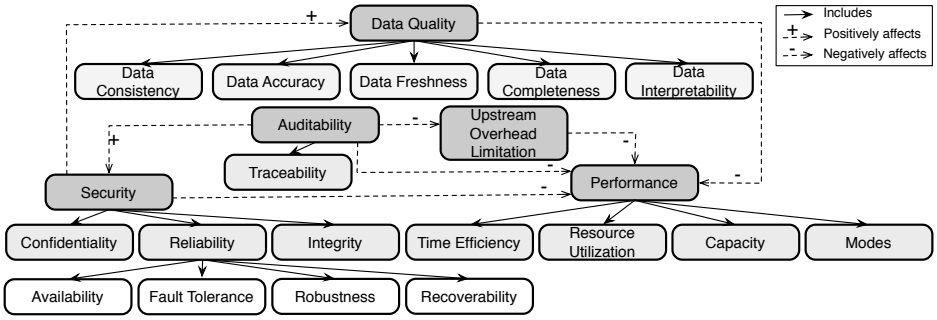


Figure 3.4: Dependencies among process characteristics with construct implications

3.2.2 Characteristics Relationships

In the same direction as (Simitsis et al., 2009b) and (Barbacci et al., 1995) we also recognise that ETL process characteristics are not independent of each other and each time a decision has to be made, the alternative options might affect different characteristics differently, but that this is not realized in completely ad hoc ways. On the contrary, we argue that there is an inherent relationship between characteristics and it can be depicted in a qualitative model that can be instantiated per case for reasoning and automation. Our vision is that this objective model can be generic yet powerful enough in order to aid in a broad range of ETL process design decisions.

Our model for the dependencies among characteristics with construct implications can be seen in Fig. 3.4. In this model we include all the characteristics with construct implications that we have identified and defined in Sec. 3.1. It consists of first-level characteristics and in some cases second- or even third-level sub-characteristics and can be read in a cause-and-effect fashion, i.e., improving one characteristic leads to improvement or deterioration of another characteristic. We should notice that although traditionally availability is classified directly under security, for our case availability is in fact a subgoal of reliability. The reason is that we regard the satisfaction of availability as a necessary but not sufficient condition for reliability. Reliability additionally requires maintaining specified SLAs for the ETL process and therefore in our model we place availability under reliability and reliability under security.

Coming back to our running example from Fig. 3.1 and Fig. 3.2, it is clear that the second design would require more time and more computational resources than the first one in order to perform the additional tasks. The measures of *Process execution time* and *CPU load measured in percentage of utilization* would have higher values indicating worse *time efficiency* and *resource utilization*. Thus, improved *Data Quality* and *Reliability* would have to be considered at the price of decreased *Performance* and whether or not the decision to select the second design would be optimal, would depend on the importance of each of these characteristics for the end-user. We will quantitatively show these effects in the following subsection.

As can be seen in Fig. 3.4 the improvement of any other characteristic negatively affects performance. That is reasonable since such improvements would require the addition of extra complexity to the ETL process, diverging from the optimal simplicity that favours performance. Improving Data Quality would require additional checks, more frequent refreshments, addi-

tional data processing and so on, thus utilizing more resources and imposing a heavier load on the system. In the same manner, improving security would require more complex authentication, authorization and accounting (AAA) mechanisms, encryption, additional recovery points, etc., similarly having negative impact on performance. Likewise, improving auditability would require additional processes for logging, monitoring as well as more resources to constantly provide real-time access to such information to end-users. In a similar fashion, promoting upstream overhead limitation would demand locks and scheduling to minimize impact of ETL processes on competing resources and therefore time and throughput limitations.

On the other hand, improving security positively affects data quality since data becomes more protected against ignorant users and attackers, making it more difficult for data and system processes to be altered, destroyed or corrupted. Therefore, data integrity becomes easier to maintain. In addition, improved system availability and robustness leads to improved data quality in the sense that processes for data refreshing, data cleaning and so on remain undisrupted.

Regarding the impact that improving auditability has to security, it is obvious that keeping track of system's operation traces and producing real-time monitoring analytics foster faster and easier threat detection and mitigation, thus significantly benefiting security. On the contrary, these operations have a negative impact on upstream overhead limitation, following the principle that one system cannot be measured without at the same time being affected.

3.2.3 Calculating the measures

In this subsection, we go through the measures that we have defined and apply them on our running example. We will do the same for the following section about the second category of measures and for both categories, each measure is represented in the form of:

- <Char_Abbr>.<Subchar_No>.<...>.<Measure_Id>(<Model_Id >)

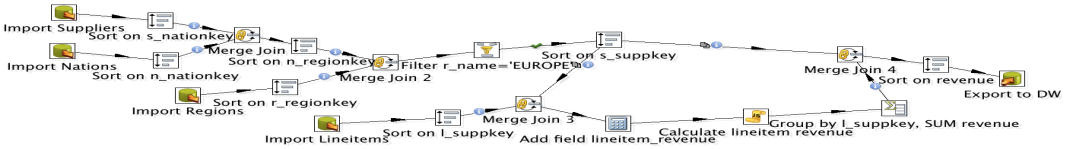
, where *Char_Abbr* is the characteristic abbreviation (e.g., *PF* for performance); *Subchar_No*, *Measure_Id* etc., are the indexes of the measures' enumerations from the related sections; and *Model_Id* can take the values *CM_A* or *CM_B*, for the simple or the more complex ETL flow, or can be missing if we refer to both examples.

Through this example we showcase i) how our proposed measures can be used and ii) how two alternative ETL designs for the same process can differ with regards to different quality dimensions, as is reflected by the corresponding measures, in a way that qualitatively agrees with our proposed model.

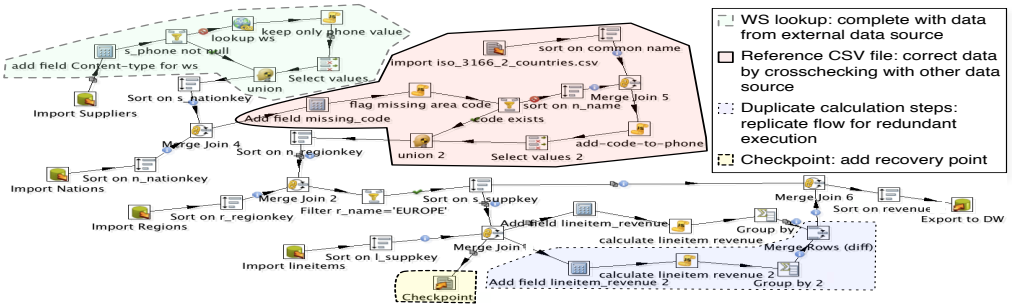
In Fig. 3.5 we can see the corresponding logical representations of the ETL designs from Fig. 3.1 and Fig. 3.2, that were drawn using the Pentaho Data Integration[7] Visual Designer. We conducted our experiments using MySQL database that was populated with data generated by the TPC-H data generator. The number of tuples for each table of the database were as follows:

Data quality can be measured for the data produced and exported to the Data Warehouse. Obviously, the quality of these data depends not only on the ETL process, but also on the data quality at the resources that it utilizes, which in our case is the data quality of the TPC-H

[7] `http://www.pentaho.com/product/data-integration`



(a) Logical representation of initial ETL process that corresponds to CM_A



(b) Logical representation of equivalent ETL process with additional steps that corresponds to CM_B

Figure 3.5: Logical models of alternative ETL processes

Table	Supplier	Nation	Region	Lineitem
# of tuples	10000	25	5	540189

database, as well as the data quality of the data provided by the WS and the text file, for the second ETL flow. For illustration purposes, we artificially dirtied the automatically generated data on the TCP-H database including null values for the phones of suppliers, as well as phone numbers with wrong formatting, not including country codes. We executed this process randomly, but for a specified percentage of records. We have manipulated the data on the TPC-H database so that they contain null address values for 10% of all the suppliers, as well as missing area code at the 10% of non-null phone numbers of the suppliers stored in the database. After executing CM_A or CM_B, the result is the export of 1987 records about suppliers and their revenues to the DW.

To measure *data accuracy*, we count the number of (non-null) exported records that are incorrect, which for our case translates to missing area codes of phone numbers. Then for each of the two ETL flows we get the values for *DQ.a.M1* as shown in Table 3.1.

We notice here that for the second ETL flow, all the incorrect values have been identified and corrected by crosschecking with the text file. That is because we assume that the text file is complete and does not contain any mistakes. This measure could similarly have been used for the incorrect addresses or names if there was a mechanism to identify them or a record could be considered correct only if all the considered attribute values were correct and then the measure could be used in a combined way. The last case would measure the percentage of delivered accurate tuples, as defined in *DQ.a.M2* of data accuracy.

For *data completeness*, we count the number of null values, which for our case can be found

Table 3.1: Data quality measures

Measure	*DQ.a.M1*	*DQ.b.M2*	*DQ.c.M3*
Formula	$\frac{correct\_tupples}{all\_tuples}$	$\frac{non\_null\_tupples}{all\_tuples}$	$\frac{1}{1-age*update\_frequency}$
CM_A	91.5%	90.3%	3
CM_B	100%	95.2%	3

in the suppliers' phone number field. Thus, we get the values for *DQ.b.M2* as shown in Table 3.1.

Again, we observe that data completeness for the second ETL flow is improved, due to the completion of data using the WS. The reason that completeness for the second ETL flow is not 100%, is because we have created the WS registry so that approximately only half of the suppliers can be found there with their details, to resemble the real case situation where improved completeness could come from a more complete registry, of course at a higher cost. When it comes to measure DQ.b.M1 of *data completeness*, incompleteness of that kind could occur when incorrect or missing values would result in tuples of suppliers not being present on the final resultset.

Considering *data freshness*, it would be the same for both ETL flows, unless we assumed that external data sources (WS registry and text file) contain more/less up-to-date information. If we assume for example that the data sources were updated 20 days ago where time units are days, and that they get updated once every month, we can calculate *DQ.c.M3* as shown in Table 3.1.

To estimate *DQ.c.M1*, we would need to know for example when suppliers changed their phone number and when it was updated in the database. For *DQ.c.M2* we would need to know how much time passed between the last update of a record and when it was actually first requested.

When it comes to *data consistency*, *DQ.d.M2* could be measured by the percentage of tuples in the resulting dataset that refer to the same supplier, which nevertheless do not exist in our example since every supplier is uniquely included. In addition, *DQ.d.M1* could have been affected if there were entries with different names for the same country (e.g., *United Kingdom of Great Britain and Northern Ireland* as opposed to *United Kingdom*) or values of different format for the same attribute (i.e., phone number as "424-242-424-242" instead of "424242424242").

For *data interpretability*, to measure *DQ.e.M1* we would take under consideration the existence of comment fields for important attributes and their completeness for the resulting data. We notice that there are plenty such comment fields in our database. For DQ.e.M2, we would conduct a survey where actual/potential users of the ETL process would describe how well they understand the meaning and content of each field of the resulting data.

Performance was measured by executing both of the ETL flows on Kettle Engine, running on Mac OS X, 1.7 GHz Intel Core i5, 4GB DDR3 and kept average values from 10 executions.

In Table 3.2 we can see the calculation of values for measures *PF.a.M1* and *PF.a.M3* for *time efficiency*.

Due to the existence of blocking operators, it is not relevant to calculate measure *PF.a.M2* for latency per tuple. As blocking operations, we consider the steps that have to be completed and all of the values that they process have to be calculated before moving to the succeeding

Table 3.2: Performance measures - Part 1

Measure	*PF.a.M1*	*PF.a.M3*	*PF.b.M1*
Formula	$Process\_cycle\_time$	$No\_of\_aggr + No\_of\_sort$	$\frac{CPU}{No\_of\_logical\_processors}$
CM_A	10.4sec	8	49.25
CM_B	18.9sec	11	55.75

Table 3.3: Performance measures - Part 2

Measure	*PF.b.M2*	*PF.c.M1*	*PF.d.M1*
Formula	$\frac{used\_memory}{total\_memory}$	$\frac{Input\_data\_size}{Process\_cycle\_time}$	$\frac{No\_supported\_modes}{all\_possible\_modes}$
CM_A	0.166	$52906\frac{tuples}{sec}$	50%
CM_B	0.181	$29179\frac{tuples}{sec}$	50%

operators, which for our case are the aggregation and the sorting steps.

For *resource utilization*, the calculation of measures *PF.b.M1* and *PF.b.M2* are shown in Table 3.2 and Table 3.3.

For *capacity*, we consider the size of the input datasets processed by these ETL flows, divided by the time that they need to execute and calculate measures *PF.c.M1* in Table 3.3.

We should notice here that the input size for the two processes slightly differs, because for the second process we also have to take under consideration the input data that come from the data cleaning tasks (WS and text file). A possible explanation for the big difference of the throughput values for the two ETL flows can be that the second flow requires the concurrent execution of a number of steps, which decreases the available processing power that accounts for each.

It is clear from all the performance measures shown above, that the design of CM_B is significantly worse that that of CM_A with respect to performance. In other words, the measures agree with our model, according to which the improvement of some quality characteristic(s) (i.e., data quality and security in our example) negatively affects some other(s) (i.e., performance, if we compare CM_B to CM_A).

When it comes to *modes*, if we consider batch processing and real-time processing as the two possible modes, our ETL flows are only designed to be executed in a batch mode, as calculated in measure *PF.d.M1* (Table 3.3).

Upstream overhead can be considered as the additional load that comes into play because of the use of additional services, which in our case can refer to the WS. Therefore, there would be upstream overhead only for the second ETL flow and it could be measured as such:

- $UO.M1(CM\_B) = AVG\_memory\_consumed\_by\_WS = 426MB$

This memory consumption lasted for as long as the interaction of the ETL with the WS that was an avarage of $12.9sec$, which corresponds to: $\frac{12.9sec}{18.9sec} = 68\%$ of the process cycle time.

Security did not play an important role for setting up our tests, except from *reliability*. Thus, for *confidentiality*, SE.a.M1 would be 0, since we are not using any cryptographic operations and the web service communication is realized over HTTP. SE.a.M2 would refer to the

anti-spyware installed in the operation system of the computer where we run the ETL processes and SE.a.M3 would also be 0 since there was no provisioning for accessing or firing the execution of the ETL processes remotely. SE.a.M4 would be 100%, since all users could have shared access to the terminal of the process execution. Similarly for *integrity*, SE.b.M1 would refer to antivirus installed in the operation system of the terminal where we run the ETL processes.

When it comes to *reliability*, the measures for *availability* and *fault tolerance* can easily be extracted from historic traces of the ETL process execution, monitoring the time of failures and the periods during which the ETL process was available and the periods during which it was not responsive, for example due to memory limitations in high demand. Of course, such analysis presumes that the ETL process has a continuous lifecycle, corresponding to changing data in the data sources. Regarding *robustness*, in the second ETL flow there is a part that duplicates the calculation steps to guarantee that even in the presence of unpredictable or malicious input that could cause the failure of this part of the ETL flow, there will be a duplicate of this part for the resumption of the execution. Thus, the measure for the second ETL flow is:

- $SE.c.iii.M1(CM\_B) = No\_of\_replicated\_processes = 1$

In a similar manner, *recoverability* is improved in the second ETL flow, by the addition of a recovery point. Thus, if for any reason the execution of the ETL flow is interrupted or terminated, already processed data is not lost and execution can continue, starting from the latest checkpoint.

- $SE.c.iv.M1(CM\_B) = No\_of\_recovery\_points = 1$

The next two measures SE.c.iv.M2 and SE.c.iv.M3 for recoverability, can again be extracted from historic traces of the ETL process execution.

Auditability and *traceability* can be measured, arguing that all of the KPIs or measures that have been considered so far can be monitored and followed for both ETL flows. Therefore, assuming that these are all the KPIs that we are interested in, the corresponding measure would be:

- $AU.a.M1 = \%\_of\_observable\_KPIs = \frac{No\_of\_observable\_KPIs}{No\_of\_important\_KPIs} = \frac{31}{31} = 100\%$

3.3 CHARACTERISTICS FOR DESIGN EVALUATION

In this section we show our model about characteristics for design evaluation. These characteristics are the most difficult ones to analyze as they are more abstract and intangible. As mentioned above, the intention of this model is to be used as an extensible, modifiable template that can be adjusted per case. Similarly to Sec. 3.2 we first define the characteristics and then show the relationships among them.

3.3.1 Characteristics and Measures

In this subsection we provide a definition for each characteristic for design evaluation, as well as proposed metrics deriving from literature.

1. *adaptability (AD)*: the degree to which ETL process can effectively and efficiently be adapted for different operational or usage environments. It includes:

   (a) *scalability*: the ability of the ETL process to handle a growing demand, regarding both the size and complexity of input data and the number of concurrent process users.

   M1: Ratio of system's productivity figures at two different scale factors, where productivity figure = throughput * QoS/ cost (Jogalekar and Woodside, 2000) (+)

   M2: # of Work Products of the process model, i.e., documents and models produced during process execution (García et al., 2004) (−)

   (b) *flexibility*: the ability of the ETL flow to provide alternative options and dynamically adjust to environmental changes (e.g., by automatically switching endpoints).

   M1: # of precedence dependences between activities (García et al., 2004) (−)

   (c) *reusability*: the degree to which components of the ETL process can be used for operations of other processes.

   M1: % of reused low level operations in the ETL process (Frakes and Terry, 1996) (+)

   The following measure is valid in the case where there are statistical data about the use of various modules (e.g., transformation or mapping operations) of the ETL process:

   M2: Average of how many times low level operations in the ETL process have been reused per specified time frame (Frakes and Terry, 1996) (+)

2. *usability (US)*: the ease of use and configuration of the implemented ETL process on the system. It includes:

   (a) *understandability*: the clearness and self-descriptiveness of the ETL process model for (non-technical) end users.

   M1: # of activities of the software process model (García et al., 2004) (−)

   M2: # of precedence dependences between activities (García et al., 2004) (−)

3. *manageability (MN)*: the easiness of monitoring, analyzing, testing and tuning the implemented ETL process.

   (a) *maintainability*: the degree of effectiveness and efficiency with which the ETL process can be modified to implement any future changes.

   M1: Length of process workflow's longest path (Simitsis et al., 2009a) (−)

   M2: # of relationships among workflow's components (Simitsis et al., 2009a) (−)

   M3: # of input and output flows in the process model (Muñoz et al., 2009) (−)

   M4: # of output elements in the process model (Muñoz et al., 2009) (−)

   M5: # of merge elements in the process model (Muñoz et al., 2009) (−)

   M6: # of input and output elements in the process model (Muñoz et al., 2009) (−)

(b) *testability*: the degree to which the process can be tested for feasibility, functional correctness and performance prediction.

M1: Cyclomatic Complexity of the ETL process workflow (Gill and Kemerer, 1991) (−)

3.3.2 Characteristics Relationships

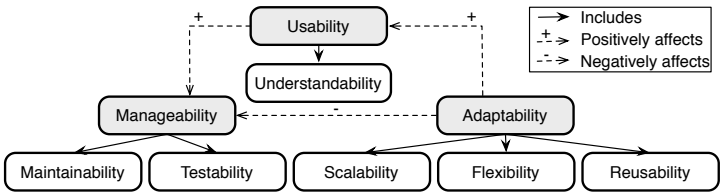


Figure 3.6: Dependencies among characteristics for design evaluation

In Fig. 3.6 we show the dependencies among characteristics for design evaluation. Increased usability favors manageability because a more concise, self-descriptive system is easier to operate and maintain. Similarly, adaptability positively affects usability, since an easily configured system is easier to use and does not require specialized skill-set from the end user. On the other hand, adaptability can be achieved with more complex systems and therefore it negatively affects manageability. This negative relationship might appear counter-intuitive, but it should be noted that our view of adaptability does not refer to autonomic behavior, which would possibly provide self-management capabilities. Instead, we regard manageability from an operator's perspective where control is desirable and the addition of unpredictable, "hidden" mechanisms would make the process more difficult to test and maintain. Regarding the apparent conflict between the negative direct relationship among Adaptability and Manageability and the transitive positive relationship between Adaptability, Usability and Manageability, our qualitative model allows for different direct and indirect influences between characteristics. If the model is instantiated and extended with measurable components to quantitatively specify these influences (i.e., weights on the edges of the Digraph), then the compound effect that the improvement of one characteristic has to another will depend on the elicitation techniques used.

3.3.3 Calculating the measures

Similarly to the previous section and using the same notation, we show in this subsection how our measures can be calculated for the characteristics for design evaluation, for the two alternative designs of Fig. 3.5.

Adaptability was measured as such: for the first measure *AD.a.M1* of *scalability*, we can calculate the throughput, cost and QoS for two different scale factors, for example for different allocation of resources (CPU, memory etc.) to the system for the ETL process execution. As QoS we can consider any of the measures that we have examined so far. The calculation of measure *AD.a.M2* is as follows:

- $AD.a.M2(CM\_A) = No\_of\_work\_products = 1$
- $AD.a.M2(CM\_B) = No\_of\_work\_products = 2$

The difference is due to the fact that the second ETL flow exports data not only to the DW, but also to the checkpoint and so it has 2 work products.

To measure *flexibility*, we have to count the precedence dependences between activities, for example for every *Join* operation, the incoming data streams must be sorted, which dictates precedence dependences between Join and Sort operators. Essentially, the way that both of these ETL flows have been designed, there are precedence dependences between each operation and its predecessor(s), so this measure will equal to the number of edges, if we view the ETL process as a Directed Acyclic Graph (DAG):

- $AD.b.M1(CM\_A) = No\_of\_precedence\_dependencies = 20$
- $AD.b.M1(CM\_B) = No\_of\_precedence\_dependencies = 44$

Regarding *reusability*, we can calculate measures AD.c.M1 and AD.c.M2 with respect to the ability of operations to be reused in other ETL flows. For example, using the GUI for Pentaho Data Integration (PDI), some operators can simply be copied/pasted to/from the clipboard and function normally as part of different flows. These operators could be considered to have locality among the different flows. This is not true for example for Table input/output operations, since the database connectivity and data source to be used must be explicitly defined for every flow. For measure AD.c.M1, we can compare the two ETL flows and identify the operators that have been reused from the first to the second flow, which for our case are all of the operators from the first flow:

- $AD.c.M1(CM\_B) = \%\_of\_reused\_operators = \frac{No\_of\_reused\_operators}{No\_of\_operators} = \frac{20}{20} = 100\%$

Usability and *understandability* were evaluated using the following measures:

- $US.a.M1(CM\_A) = No\_of\_activities = 20$
- $US.a.M2(CM\_A) = No\_of\_precedence\_dependencies = 20$
- $US.a.M1(CM\_B) = No\_of\_activities = 41$
- $US.a.M2(CM\_B) = No\_of\_precedence\_dependencies = 44$

Manageability can be measured, if we consider the ETL flow as a DAG, where each ETL logical operation corresponds to a node and each control flow precedence relationship corresponds to a directed edge. Hence, we have the measures for *maintainability* as shown in Table 3.4 and Table 3.5.

Regarding *testability*, we can measure Cyclomatic Complexity (CC) for each flow:

- $MN.b.M1(CM\_A) = CC = No\_of\_nodes - No\_of\_edges + No\_of\_cycles = 20 - 20 + 0 = 0$

Table 3.4: Maintainability measures - Part 1

Measure	*MN.a.M1*	*MN.a.M2*	*MN.a.M3*
Formula	$Length\_of\_longest\_path$	$No\_of\_activity\_relationships$	No_of_i/o_flows
CM_A	9	20	5
CM_B	23	44	8

Table 3.5: Maintainability measures - Part 2

Measure	*MN.a.M4*	*MN.a.M5*	*MN.a.M6*
Formula	$No\_of\_outputs$	No_of_merge	$No\_of\_i/o\_elements$
CM_A	1	4	5
CM_B	2	8	8

- $MN.b.M1(CM\_B) = CC = 44 - 41 + 0 = 3$

From the above measures, we can clearly see how the design of CM_B is less usable, manageable and adaptable than CM_A. This is not only an intuitive impression from looking at a more complex process model, but can also be quantitatively measured in an automatic and straightforward fashion, thanks to the gathered metrics.

3.4 GOAL MODELING FOR ETL DESIGN

The research area of Requirements Engineering has been active over the past years, offering a plethora of frameworks and methodologies to bridge the gap between stakeholders' (early and late) design goals and specific decisions. The field of Goal-Oriented Requirements Engineering additionally depicts the alignment between requirements and goals and their relationship to strategic decisions.

A *goal* constitutes an objective of a business, for example *Increase revenue*, *Improve customer satisfaction* and so on. A goal can be satisfied or not, depending not only on the success of the corresponding tactical steps undertaken but also on factors external to the goal, such as environmental conditions or the satisfaction or not of other goals that can affect it. Hence, a goal can be decomposed (or *refined* as found in literature) to other more simple goals — its subgoals — in different ways, the most common being AND and OR decomposition. Examples of subgoals for the goal *Increase revenue*, can be the goals *Reduce production costs* and *Increase sales*.

In general, the semantics of an AND-decomposition is that *if all of the subgoals of a goal G that participate in the same AND-decomposition are satisfied, then G is satisfied*. Regarding OR-decomposition, it introduces the possibility of alternative ways to satisfy one goal. That is, *if any of the (groups of AND-)subgoals of a goal G that participate in an OR-decomposition is satisfied, then G is satisfied*. Furthermore, the satisfaction of a goal might be influenced by the satisfaction of another goal, either positively or negatively.

It is natural that goals and their interrelationships are usually depicted diagrammatically using *goal-models*, with goals being represented as boxes and arrows connecting them to show

relationships for goal elicitation. For our case, goals are quality goals about the ETL process (e.g., *Improve performance*, *Improve data quality*) and as we have mentioned above, the satisfaction of one such goal might affect negatively or positively the satisfaction of another, which can be shown in the model using arrows of positive or negative influence. In addition, quality goals can be decomposed to subgoals, for example the goal *Improve reliability* can be satisfied either by satisfying the goal *Improve robustness* OR the goal *Improve recoverability*. These refinements can be directly derived by our model of quality characteristics and subcharacteristics, where the improvement of the latter can play the role of subgoals to the goals of improvement of the former. We can see for our use case, goals, subgoals and influences in Fig. 3.7, but more details about this model will be provided below, where we describe the specific goal modeling paradigm that has been used.

Goal models are used not only to lighten for business users the cognitive load of managing multiple goals but also to aid in their analysis through reasoning. By defining specific rules for satisfaction *propagation* through different parts of the goal model (e.g., if any of the AND-subgoals is denied $\rightarrow$ the goal is denied), users can assume given values for part of the model and test the feasibility of different what-if scenarios; they can see what implications (if any) are applied to the other parts of the model; in some models they can even make decisions about specific actions to be taken, in a top down approach where they select which goal(s) they would like to satisfy. Different goal modeling paradigms offer different levels of expressiveness.

A thorough review of goal-oriented approaches is provided in (van Lamsweerde, 2001, 2003), which presents the basic concepts of goal-oriented requirements engineering and shows through examples how goals can be refined for the selection among alternative architectural designs. The author illustrates the use of the KAOS framework for modeling, specifying and analyzing requirements in order to decide about system architectures. KAOS provides the ability to model business goals and objectives, as well as goal decomposition and hierarchies to reflect the strategic business objectives. Additionally to goal decomposition, contribution links and partial negative or positive contribution among goals, the NFR (non-functional requirement) modeling framework (Chung et al., 2000) also introduces the concept of softgoals: intangible goals without clear-cut criteria. The i* framework (Yu, 1996) integrates these concepts with resources, and dependencies between actors to facilitate the modeling for identifying stakeholders and for examining the problem domain while exposing early phase system design requirements.

In a more recent work, (Horkoff and Yu, 2013) compares several existing approaches on goal oriented requirements analysis and shows how different modeling of the same goals can lead to different evaluation and decisions. To reach that conclusion, the authors use different models for the same goal-analysis procedures and through the comparison they expose different assumptions concerning goal concepts and propagation.

Goal modeling approaches have also been suggested to aid with decision making for Business Intelligence. An application of the Tropos goal modeling methodology (Bresciani et al., 2004) to data warehousing is suggested in (Giorgini et al., 2008), which introduces a technique to map requirements with facts, dimensions and measures to relevant data schemata and decisions. This technique can be seen as a first step towards requirements-oriented data warehousing, concentrating on functionality and early phase design. (Teruel et al., 2014) presents a goal modeling framework to support Collaborative BI systems requirements elicitation and

the business intelligence model (BIM) (Horkoff et al., 2014) is a goal modeling paradigm, specific to BI context that is used to represent, in an object-centered way the interaction among different goals and situations.

3.4.1 Applying BIM to ETL processes

We applied the BIM Model for our running example (Fig. 3.7), in order to aid in ETL Process design decisions. In the BIM model, a *goal* is *an intentional situation that is desired by the (viewpoint) organization* and BIM models the positive or negative relationship among them, their elicitation from other goals (subgoals) and from *tasks* (or *processes* as they are referred in newer versions), which are *processes or set of actions* that are related to some goal and can *achieve* it, to provide a "how" dimension. The satisfaction of a goal can be inferred from the satisfaction level of other goals. For our case, goals are quality goals for the ETL process (e.g., *Improve reliability*) and tasks are some patterns that can be applied to the ETL process in order to improve some quality dimensions (e.g., *Add recovery point* to improve recoverability).

The BIM core also includes *indicators* to *evaluate* the satisfaction of goals and *measure* the use of processes. Indicators act as the measurable component to bridge the gap between business objectives and real, actual data that support or decline their satisfaction. For our case, our defined quality measures (the actual metrics) from Section 3.2 and Section 3.3, can play the role of indicators (e.g., *% of correct values*). It also includes internal and external situations to model state information and at the lowest refinement level. For our case, situations can represent states internal or external to the ETL process, which can be favorable or unfavorable to the goals (e.g., an external situation to the ETL process can be the *Quality of HW/SW resources*).

To put it all together, the main elements of the goal model can be derived in a very straightforward and intuitive manner from the classification and the models that we have introduced in Section 3.2 and Section 3.3.

- The improvement of each quality characteristic constitutes a *goal*.
- The improvement of characteristic a that is a subcharacteristic of characteristic b constitutes an *OR-subgoal* of the goal *Improve b*.
- The *positive/negative influences between goals* can be directly derived from the positive/negative influences that we have shown in the characteristics relationships (Fig. 3.4 and Fig. 3.6).
- The measures (metrics) that we have defined for each quality characteristic can play the role of *indicators*, evaluating the goals related to the corresponding characteristics. It should be noted here that indicators must be unique for each goal, according to the specification of BIM. Thus, in cases where there is more than one measure that can be used for the evaluation of a goal, they should be aggregated (e.g., using simple additive weighting method) into one compound indicator.

Unlike the above-mentioned elements, the *tasks* and the *situations* are not directly derived from our models. Tasks represent an arsenal of possible (reusable) actions that can be taken and their definition as well as their impact on goals require domain knowledge and evidence to

be supported. For our case, the tasks depicted in our model have been collected from literature and validated by experiments that we have conducted with multiple ETL processes. They have also been translated to *patterns* in order to be automatically integrable to any ETL flow using our tool implementation, about which we discuss in the following section. Modeling of situations also requires domain knowledge, as well as contextualization to match the state information of specific use cases. This stands not only for tasks and situations, since for different use cases and contexts, different quality characteristics and metrics can be used, which can be different subsets of the elements of our models, and/or different, new ones defined in a similar fashion, thus resulting in different goal models.

As mentioned above, apart from concise visual representation, goal models are used for what-if analysis and reasoning, which for the case of the BIM model is straightforward, since it can be directly translated to the OWL 2 DL (owl, cited August 2015). As example of translation of (Fig. 3.7) to OWL we show the following facts:

```
\textbf{Class}: ImproveDataQuality SubClassOf: Goal and OR_Thing

Class: ImproveDataCompleteness SubClassOf: (refines some ImproveDataQuality)

Class: ImproveDataCompleteness SubClassOf: Goal and AND_Thing

Class: ReduceTuplesWithNullValues SubClassOf:
(refines some ImproveDataCompleteness)

Class: ImproveDataQuality SubClassOf: (-_influences some ImprovePerformance)

Class: ImprovePerformance SubClassOf: (-_infBy some ImproveDataQuality)

Class: ImproveReliability SubClassOf: (+_influences some ImproveDataQuality)

Class: ImproveDataQuality SubClassOf: (+_infBy some ImproveReliability)

Class: ImproveDataQuality SubClassOf: (refinedBy exactly 2) and
(refines exactly 0) and (influences exactly 1) and (infBy exactly 2)
```

Listing 3.1: OWL facts for running example

For more information about the BIM and its translation to OWL, interested readers are referred to (Horkoff et al., 2012).

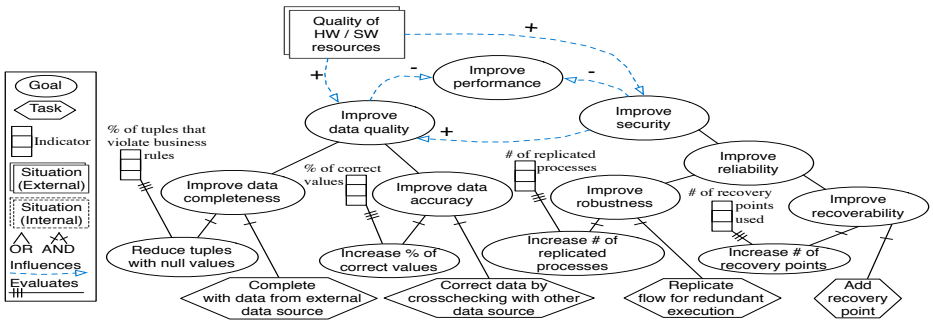


Figure 3.7: Goal modeling for running example

3.5 USER-CENTERED ETL OPTIMIZATION

In this section, we showcase how our model can provide the conceptual and practical base, upon which a user-centered method can be applied to translate user goals about ETL process quality to specific implementation steps. Thus, we present a more detailed view of the architecture that we introduced in Fig. 1.3 and we illustrate how our user-centered declarative ETL approach can be applied to our running example. The description of this architecture has been published in (Theodorou et al., 2014b).

In Fig. 3.8, we can see the architecture for quality-aware end-to-end design of ETL processes. As can be seen from Fig. 3.8, which is labeled with a number for each step, our methodology consists of three phases: design of an ETL process based on functional requirements (steps 1–3); improvement by instillation of user-defined quality characteristics to the process (steps 4–13); and finally deployment and execution (steps 15–17). As mentioned before, in order to promote automation and user-centricity, our architecture is highly modular and is based on patterns to automate the design process. In the following we describe the application of interesting components of the architecture to our running example.

Functionality-Based Design. This part of the method (steps 1–3 in Fig. 3.8) only concerns the functional requirements for the ETL process. Thus, the *ETL Process Designer* component is responsible for the design of an ETL process model that implements the basic ETL functionality: extraction of data from the original data sources, transformation of data to comply

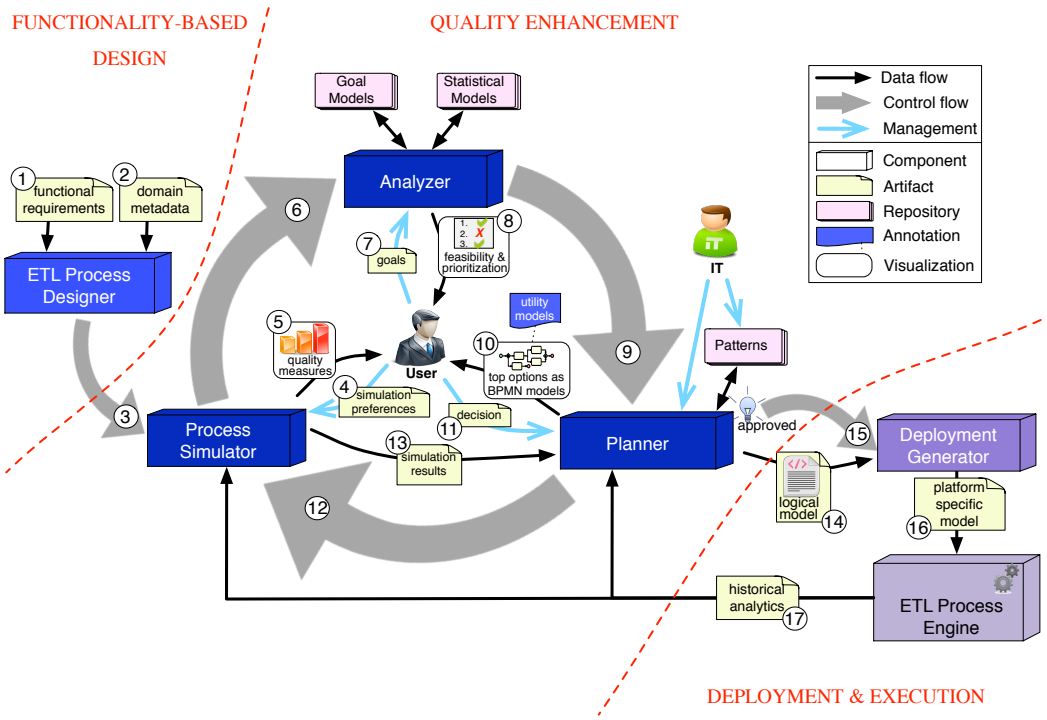


Figure 3.8: Functional architecture

with functional requirements and finally loading into target repositories. The required input at this step is an accurate representation of the domain, including information about business requirements (step 1). Concerning our running example, the input would be a description of the TPC-H relational source tables — schemata and constraints — as well as the appropriate modeling of the query: *Load in the DW all the suppliers in Europe together with their information (phones, addresses etc.), sorted on their revenue*, for example using description logics or a structured description (XML, JSON etc.). We argue that naturally, the former can be modeled by IT with technical competencies for data and knowledge representation, while the latter can be introduced on a high level by BU, since they are the experts for the context of use of the resulting data processes.

The output of this step is a conceptual ETL process model, which is described in a high-level representation. This model must be abstract enough to allow for the incorporation of patterns reflecting user requirements, but at the same time it can be seamlessly translated to a logical, implementation independent model. In addition, this model must be directly translatable to an intuitive visualization for the system user, using for example BPMN. Thus, we suggest that the model at this step is an ETL-specific extension of the Directed Acyclic Graph, where each node is one high-level ETL operator. (Akkaoui et al., 2013) provides such a set of high-level ETL operators as part of their proposed BPMN meta-model. The high level representation output of our running example at this point, is the BPMN process in Fig. 3.1 and the translation to a logical model is the process in Fig. 3.5a. Apart from the process model, domain information about available data sources, entities and their characteristics as well as resource constrains is also passed on to the next phase (step 3) to allow for design alterations, where needed.

Quality Enhancement. The second phase (steps 4–13 in Fig. 3.8) regards the improvement of quality characteristics of the ETL process. As explained before, the *Process Simulator* component is responsible for simulating the ETL process and producing analytics regarding its structure and its expected behaviour. At this point the users provide their simulation preferences (step 4) that reflect quality characteristics of interest and receive as feedback a user-friendly representation of quality measures (step 5). To this end, it is important to use representative, realistic input data to test the process, which can be achieved either by "feeding" the simulator with a sample of real input data or by providing an effective synthetic test data generation mechanism (Nakuçi et al., 2014). Coming back to our running example, the users decide which of the quality characteristics that we have defined in our models they would be interested in, e.g., the quality dimensions of performance, data quality and security. They are also able to browse through and select among predefined metrics that are related to those dimensions and are of interest for their analysis. Such examples are the *indicators* of the goal model in Fig. 3.7 and as explained above, ad-hoc, compound, aggregated measures for specific business needs are also used, with the capability of being broken down to more simple metrics, where visualization plays a key role for their presentation to the user. Calculating the measures as has been showcased in previous sections, happens in a completely automated way, thanks to the machine-readable modeling of the ETL process.

Once the measures calculations about the current version of the process have been presented to the users, they can evaluate how well they align with the strategic goals that have been set and decide which of the quality dimensions should be improved, that can be directly translated to goals in the goal modeling of the following step.

Subsequently (step 6), the *Analyzer* takes as input user goals (step 7) and reasons about goal satisfaction. Selecting which goals are pursued every time, goal models can allow to answer feasibility questions about the set of tasks that can be performed, forming the palette of quality patterns that will be used for the optimization problem (step 8). Considering our running example, the goal model of Fig. 3.7 can be used for such analysis. For instance, the BIM tool[8] can provide "what-if" analysis regarding how given input information about goals, processes, situations, indicators, and domain assumptions propagates to other elements in the model. As an example, it can answer questions such as: *what happens (in other words, what are the possible satisfaction values) with the goal "Improve Performance", if we assume that the goal "Increase # of replicated processes" is satisfied?*. For the model of Fig. 3.7 the answer to such questions might appear rather obvious, but this is not the case with real use cases that can produce a very complex business model with tens of different quality characteristics. The situation can become even more complex if we assume some bigger granularity i) in the influences between goals (e.g., using additionally $++$ and $--$ to denote greater positive and negative influence, respectively) and ii) in the possible satisfiability values (e.g., using additionally *partial satisfaction* and *partial denial* of goals).

A top-down reasoning is also possible, where the user can select which goals need to be satisfied/denied and the tool can provide possible tasks that can be implemented. For example, selecting the satisfaction of the goals *Improve security* and *Improve performance* (at the same iteration) would return no possible tasks, but selecting the satisfaction of the goal *Improve data quality* would return the possible tasks of *Complete with data from external data source* and *Correct data by crosschecking with other data source*.

The second process that can be conducted by the Analyzer is the qualitative evaluation of alternative design patterns application. For this purpose, statistical models can be used that will take as input user goals and quality measures from the simulation of alternative ETL process models and will produce as output (step 8) the (quantitative) relationships between goals and quality patterns, and thus the prioritization of the patterns that should be used, based on user's goals.

The *Planner* is the actuator of the quality enhancement phase, integrating patterns on the ETL process that improve its quality. For our running example, the palette of available improvement steps for the satisfaction of quality goals that are handled by the *Planner*, are the *tasks* of the goal model (Fig. 3.7). Even though the problem space is restricted by estimated (monetary) cost, the optimization problem of selecting an optimal combination of patterns to be applied to the process can be formulated as a multi-objective knapsack problem (Thiele et al., 2009). In order to tackle complexity, we propose the use of goal models and statistical models on the previous step on one hand; and the application of only one pattern during each iteration, on the other. In this direction, after reasoning, the Planner recommends to the user a list of the highest ranked potential patterns (step 10) in a graph-like visualization, together with utility models, which are annotations denoting the estimated affect of each pattern to the quality goals. Judging solely from the BPMN models and the utility models, users make a selection decision (step 11) and the Planner implements this decision by integrating a pattern to the existing process flow. These patterns are in the form of process components and the Planner

[8] `http://www.cs.utoronto.ca/~jm/bim/`

carefully merges them to the existing process (Jovanovic et al., 2012). Subsequently, new iteration cycles commence (step 12), until the users consider that the model adequately satisfies quality goals. The *Planner* receives feedback from the actual runtime of the executed process as well as from their simulation (step 13) in order to adjust its heuristics and increase accuracy when selecting top options. Regarding our running example, after four iterations, where the user would select the goals to *Improve data completeness*, *Improve data accuracy*, *Improve robustness* and *Improve reliability*, the resulting ETL logical model would be the one of Fig. 3.5b, which corresponds to the initial ETL model with added and integrated patterns.

For data completeness, a pattern has been added to complete missing rows and null values from external data sources. We implemented a simple web application that receives a (HTTP) request containing the suppliers' names for suppliers with empty (null) values for their phones. After matching those names to existing records in its registry, if found the application replies with information about the suppliers, which contains their phones. The corresponding logical steps of the ETL process using this service as a client, can be seen in the *WS lookup* part of the ETL flow in Fig. 3.5b. Likewise, for data accuracy, a pattern has been applied to correct data, according to crosschecks with other data sources. We realized this pattern by using a local text file (CSV) that contained the ISO 3166-2 standard information for all countries. This file was crosschecked with the suppliers' records that have missing phone country codes and the corresponding codes for the suppliers' countries were filled in to the telephone numbers. The related logical steps of the ETL process can be seen in the *Reference CSV file* part of the ETL flow in Fig. 3.5b. Similarly, to improve reliability and robustness, corresponding patterns of logical steps of the ETL flow, can be seen in the *Duplicate calculation steps* and the *Checkpoint* parts of the ETL flow in Fig. 3.5b, respectively. It should be noticed that the logical steps of the patterns have been generically defined within our tool, resulting in highly configurable patterns that can be integrated to any ETL process with the appropriate configurations (e.g., URLs of available services, attributes to be joined).

Deployment and Execution. Once users are presented with quality measures' values that they consider satisfactory, they give the green light for deployment and execution of the ETL process (steps 14–17 in Fig. 3.8). The *Deployment Generator* component processes the logical model and translates it to a platform-specific model (step 16). This step can be realized using existing approaches for (semi-)automated transition among different abstraction levels, focusing on cost and performance (Böhm et al., 2009b; Wilkinson et al., 2010). The *ETL Process Engine* executes the ETL process and as mentioned above, keeps traces to provide related historical analytics to the *Planner* and the *Process Simulator* (step 17).

3.6 SUMMARY AND OUTLOOK

In this chapter, we have proposed a model for ETL process quality characteristics that constructively uses concepts from the fields of Data Warehousing, ETL, Data Integration, Software Engineering and Goal Modeling. One important aspect about our model is that for each and every characteristic, there has been suggested measurable indicators that derive solely from existing literature. We have distinguished between characteristics that can provide guidance for pattern generation and others that are more abstract, but, nonetheless can be measured

and evaluated. Our model includes the relationships between different characteristics and can indicate how the improvement of one characteristic by the application of design modifications can affect others. We have shown how our defined models can be used to automate the task of selecting among alternative designs and improving ETL processes according to defined user goals.

4

DATA GENERATOR FOR EVALUATING ETL PROCESS QUALITY

Obtaining the right set of data for evaluating the fulfillment of different quality factors in the ETL process design is rather challenging. First, the real data might be out of reach due to different privacy constraints, while manually providing a synthetic set of data is known as a labor-intensive task that needs to take various combinations of process parameters into account. More importantly, having a single dataset usually does not represent the evolution of data throughout the complete process lifespan, hence missing the plethora of possible test cases.

In this chapter, we revisit the problem of synthetic data generation for the context of ETL processes, for evaluating different quality characteristics of the process design, as explained in Section 1.2. To this end, we propose an automated data generation framework for evaluating ETL processes (i.e., *Bijoux*). For overcoming the complexity and heterogeneity of typical ETL processes, we tackle the problem of formalizing the semantics of ETL operations and classifying the operations based on the part of input data they access for processing. This largely facilitates *Bijoux* during data generation processes both for identifying the constraints that specific operation semantics imply over input data, as well as for deciding at which level the data should be generated (e.g., single field, single tuple, complete dataset).

Furthermore, *Bijoux* offers data generation capabilities in a modular and configurable manner. Instead of relying on the default data generation functionality provided by the tool, more experienced users may also select specific parts of an input ETL process, as well as desired quality characteristics to be evaluated using generated datasets. The work presented in this chapter has been published in (Theodorou et al., 2017).

Running Example. To illustrate the functionality of our data generation framework, we introduce the running toy example that shows an ETL process (see Figure 4.1), which is a simplified implementation of the process defined in the *TPC-DI benchmark*[1] for loading the *DimSecurity* table during the *Historical Load* phase[2]. The ETL process extracts data from a file with fixed-width fields (flat file in the *Staging Area*), which is a merged collection of financial information about companies and securities coming from a financial newswire (FINWIRE) service. The input set is filtered to keep only records about Securities (RecType=='SEC') and then rows are split to two different routes, based on whether or not their values for the field *CoNameOr-*

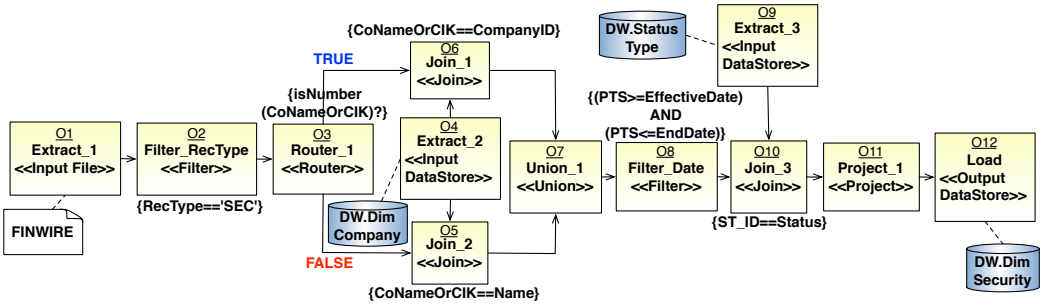


Figure 4.1: ETL flow example: TPC-DI DimSecurity population

[1] `http://www.tpc.org/tpcdi/`

[2] Full implementation available at: `https://github.com/AKartashoff/TPCDI-PDI/`

CIK are numbers (isNumber(CoNameOrCIK)) or not. For the first case, data are matched with data about companies through an equi-join on the company ID number (CoNameOrCIK==CompanyID). On the other hand, for the second case, data are matched with data about companies through an equi-join on the company name (CoNameOrCIK==Name). In both cases, data about companies are extracted from the *DimCompany* table of the data warehouse. Subsequently, after both routes are merged, data are filtered to keep only records for which the posting date and time (PTS) correspond to company data that are current ((PTS>=EffectiveDate) AND (PTS<=EndDate)). Lastly, after data are matched with an equi-join to the data from the StatusType table, to get the corresponding status type for each status id (ST_ID==Status), only the fields of interest are maintained through a projection and then data are loaded to the *DimSecurity* table of the DW.

For the sake of simplicity, in what follows we will refer to the operators of our example ETL, using the label noted for each operator in Figure 4.1 (i.e., `O1` for `Extract_1`, `O2` for `Filter_RecType`, etc.). Given that an ETL process model can be seen as a directed acyclic graph (DAG), *Bijoux* follows a topological order of its nodes, i.e., operations (e.g., `O1`, `O2`, `O3`, `O4`, `O5`, `O6`, `O7`, `O8`, `O9`, `10`, `O11`,and `O12`), and extracts the found flow constraints (e.g., RecType=='SEC' or CoNameOrCIK==Name). Finally, *Bijoux* generates the data that satisfy the given constraints and can be used to simulate the execution of the given ETL process.

Our framework, *Bijoux*, is useful during the early phases of the ETL process design, when the typical time-consuming evaluation tasks are facilitated with automated data generation. Moreover, *Bijoux* can also assist the complete process lifecycle, enabling easier re-evaluation of an ETL process redesigned for new or changed information and quality requirements (e.g., *adding new data sources*, *adding mechanisms for improving data consistency*). Finally, the *Bijoux*'s functionality for automated generation of synthetic data is also relevant during the ETL process deployment. It provides users with the valuable benchmarking support (i.e., synthetic datasets) when selecting the right execution platform for their processes.

The rest of the chapter is structured as follows. Section 4.1 formalizes the notation of ETL processes in the context of data generation and presents a general overview of our approach using an example ETL process. Section 4.2 formally presents *Bijoux*, our framework and its algorithms for the automatic data generation. Section 4.3 introduces modified versions of our example ETL process and showcases the benefits of *Bijoux* for re-evaluating flow changes. In Section 4.4, we introduce the architecture of the prototype system that implements the functionality of the *Bijoux* framework and further report our experimental results. Finally, Section 4.5 concludes the chapter.

4.1 OVERVIEW OF OUR APPROACH

In this section, we present the overview of our data generation framework. We classify the ETL process operations and formalize the ETL process elements in the context of data generation and subsequently, in a nutshell, we present the overview of the data generation process of the *Bijoux* framework.

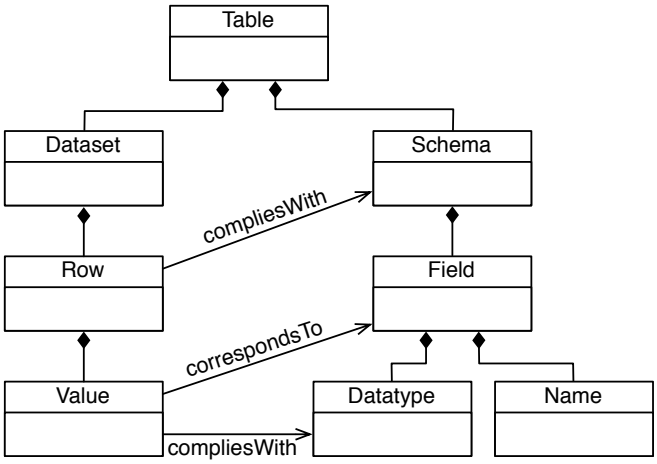


Figure 4.2: Table-access based classification, UML notation

4.1.1 ETL operation classification

To ensure applicability of our approach to ETL processes coming from major ETL design tools and their typical operations, we performed a comparative study of these tools with the goal of producing a common subset of supported ETL operations. To this end, we considered and analyzed four major ETL tools in the market; two commercial, i.e., Microsoft SQL Server Integration Services (SSIS) and Oracle Warehouse Builder (OWB); and two open source tools, i.e., Pentaho Data Integration (PDI) and Talend Open Studio for Data Integration.

We noticed that some of these tools have a very broad palette of specific operations (e.g., PDI has a support for invoking external web services for performing the computations specified by these services). Moreover, some operations can be parametrized to perform different kinds of transformation (e.g., tMap in Talend), while others can have overlapping functionalities, or different implementations for the same functionality (e.g., *FilterRows* and *JavaFilter* in PDI). Tables 4.1 and 4.2 show the resulting classification of the ETL operations from the considered tools.

To generalize such a heterogeneous set of ETL operations from different ETL tools, we considered the common functionalities that are supported by all the analyzed tools. As a result, we produced an extensible list of ETL operations considered by our approach (see Table 4.3). Notice that this list covers all operations of our running example in Figure 4.1, except extraction and loading ones, which are not assumed to carry any specific semantics over input data and thus are not considered operations by our classification.

A similar study of typical ETL operations inside several ETL tools has been performed before in (Vassiliadis et al., 2009). However, this study classifies ETL operations based on the relationship of their input and output (e.g., *unary*, *n-ary* operations). Such operation classification is useful for processing ETL operations (e.g., in the context of ETL process optimization). In this chapter, we further complement such taxonomy for the data generation context. Therefore, we classify ETL operations based on the part of the input table they access when processing the data (i.e., *table*, *dataset*, *row*, *schema*, *field*, or *field value*; see the first column of Table 4.1

Table 4.1: Comparison of ETL operations through selected ETL tools - Part 1

Operation Level	Operation Type	Pentaho PDI	Talend Data Integration	SSIS	Oracle Warehouse Builder
Field	Field Value Alteration	Add constant Formula Number ranges Add sequence Calculator Add a checksum	tMap tConvertType tReplaceList	Character Map Derived Column Copy Column Data Conversion	Constant Operator Expression Operator Data Generator Transformation Mapping Sequence
Dataset	Duplicate Removal	Unique Rows Unique Rows (HashSet)	tUniqRow	Fuzzy Grouping	Deduplicator
	Sort	Sort Rows	tSortRow	Sort	Sorter
	Sampling	Reservoir Sampling Sample Rows	tSampleRow	Percentage Sampling Row Sampling	
	Aggregation	Group by Memory Group by	tAggregateRow tAggregateSortedRow	Aggregate	Aggregator
	Dataset Copy		tReplicate	Multicast	
Row	Duplicate Row	Clone Row	tRowGenerator		
	Filter	Filter Rows Data Validator	tFilterRow tMap tSchemaComplianceCheck	Conditional Split	Filter
	Join	Merge Join Stream Lookup Database lookup Merge Rows Multiway Merge Join Fuzzy Match	tJoin tFuzzyMatch	Merge Join Fuzzy Lookup	Joiner Key Lookup Operator
	Router	Switch/Case	tMap	Conditional Split	Splitter
	Set Operation - Intersect	Merge Rows (diff)	tMap	Merge Join	Set Operation
	Set Operation - Difference	Merge Rows (diff)	tMap		Set Operation
	Set Operation - Union	Sorted MergeAppend streams	tUnite	Merge Union All	Set Operation

Table 4.2: Comparison of ETL operations through selected ETL tools - Part 2

Operation Level	Operation Type	Pentaho PDI	Talend Data Integration	SSIS	Oracle Warehouse Builder
Schema	Field Addition	Set field value Set field value to a constant String operations Strings cut Replace in string Formula Split Fields Concat Fields Add value fields changing sequence Sample rows	tMap tExtractRegexFields tAddCRCRow	Derived Column Character Map Row Count Audit Transformation	Constant Operator Expression Operator Data Generator Mapping Input/Output parameter
	Datatype Conversion	Select Values	tConvertType	Data Conversion	Anydata Cast Operator
	Field Renaming	Select Values	tMap	Derived Column	
	Projection	Select Values	tFilterColumns		
Table	Pivoting	Row Denormalizer	tDenormalize tDenormalizeSortedRow	Pivot	Unpivot
	Unpivoting	Row Normalizer Split field to rows	tNormalize tSplitRow	Unpivot	Pivot
Value	Single Value Alteration	If field value is null Null if Modified Java Script Value SQL Execute	tMap tReplace	Derived Column	Constant Operator Expression Operator Match-Merge Operator Mapping Input/Output parameter
Source Operation	Extraction	CSV file input Microsoft Excel Input Table input Text file input XML Input	tFileInputDelimited tDBInput tFileInputExcel	ADO .NET / DataReader Source Excel Source Flat File Source OLE DB Source XML Source	Table Operator Flat File Operator Dimension Operator Cube Operator
Target Operation	Loading	Text file output Microsoft Excel Output Table output Text file output XML Output	tFileOutpu tDelimited tDBOutput tFileOutputExcel	Dimension Processing Excel Destination Flat File Destination OLE DB Destination SQL Server Destination	Table Operator Flat File Operator Dimension Operator Cube Operator

Table 4.3: List of operations considered in the framework

Considered ETL Operations	
Aggregation	Intersect
Cross Join	Join (Outer)
Dataset Copy	Pivoting
Datatype Conversion	Projection
Difference	Router
Duplicate Removal	Single Value Alteration
Duplicate Row	Sampling
Field Addition	Sort
Field Alteration	Union
Field Renaming	Unpivoting
Filter	

and Table 4.2) in order to assist *Bijoux* when deciding at which level data should be generated. In Figure 4.2, we conceptually depict the relationships between different parts of input data, which forms the basis for our ETL operation classification. In our approach, we consider the *Name* of a *Field* to act as its identifier.

4.1.2 Formalizing ETL processes

The modeling and design of ETL processes is a thoroughly studied area, both in the academia (Vassiliadis et al., 2002; Muñoz et al., 2008; Akkaoui et al., 2013; Wilkinson et al., 2010) and industry, where many tools available in the market often provide overlapping functionalities for the design and execution of ETL processes (Pall and Khaira, 2013). Still, however, no particular standard for the modeling and design of ETL processes has been defined, while ETL tools usually use proprietary (platform-specific) languages to represent an ETL process model. To overcome such heterogeneity, *Bijoux* uses a logical (platform-independent) representation of an ETL process, which in the literature is usually represented as a directed acyclic graph (DAG) (Wilkinson et al., 2010; Jovanovic et al., 2014). We thus formalize an ETL process as a DAG consisting of a set of nodes (**V**), which are either source or target data stores ($\mathbf{DS} = \mathbf{DS}_S \cup \mathbf{DS}_T$) or operations (**O**), while the graph edges (**E**) represent the directed data flow among the nodes of the graph ($v_1 \prec v_2$). Formally:

$ETL = (\mathbf{V}, \mathbf{E})$, such that:
$\mathbf{V} = \mathbf{DS} \cup \mathbf{O}$ and $\forall e \in \mathbf{E} : \exists (v_1, v_2), v_1 \in \mathbf{V} \wedge v_2 \in \mathbf{V} \wedge v_1 \prec v_2$

Data store nodes (**DS**) in an ETL flow graph are defined by a schema (i.e., finite list of fields) and a connection to a source ($\mathbf{DS}_S$) or a target ($\mathbf{DS}_T$) storage for respectively extracting or loading the data processed by the flow.

On the other side, we assume an ETL *operation* to be an atomic processing unit responsible for a single transformation over the input data. Notice that we model input and output data of an ETL process in terms of one or more *tables* (see Figure 4.2).

We formally define an ETL flow *operation* as a quintuple:

$o = (\mathbb{I}, \mathbb{O}, \mathbf{X}, \mathbf{S}, \mathbf{A})$, where:

- $\mathbb{I} = \{I_1, \ldots, I_n\}$ is a finite set of input tables.
- $\mathbb{O} = \{O_1, \ldots, O_m\}$ is a finite set of output tables.
- **X** ($\mathbf{X} \subseteq Attr(\mathbb{I})$) is a subset of fields of the input tables $\mathbb{I}$ required by the operation. Notice that the function $Attr$ for a given set of input or output tables, returns a set of fields (i.e., attributes) that builds the schema of these tables.
- $\mathbf{S} = (\mathbb{P}, \mathbb{F})$ represents ETL operation semantics in terms of:
  - $\mathbb{P}$ = $\{P_1(X_1), \ldots P_p(X_p)\}$: a set of conjunctive predicates over subsets of fields in **X** (e.g., $Age > 25$).
  - $\mathbb{F}$ = $\{F_1(X_1), \ldots F_f(X_f)\}$: a set of functions applied over subsets of fields in **X** (e.g., $Substr(Name, 0, 1)$). The results of these functions are used either to alter the existing fields or to generate new fields in the output table.
- **A** is the subset of fields from the output tables, added or altered during the operation.

Intuitively, the above ETL notation defines a transformation of the input tables ($\mathbb{I}$) into the result tables ($\mathbb{O}$) by evaluating the predicate(s) and function(s) of semantics **S** over the functionality schema **X** and potentially generating or altering fields in **A**.

An ETL operation processes input tables $\mathbb{I}$, hence based on the classification in Figure 4.2, the semantics of an ETL operation should express transformations at (1) the *schema* (i.e., generated/projected-out schema), (2) the *row* (i.e., passed/modified/generated/removed rows), and (3) the *dataset* level (i.e., output cardinality).

In Table 4.4, we formalize the semantics of ETL operations considered by the framework (i.e., operations previously listed in Table 4.3). Notice that some operations are missing from Table 4.4, as they can be derived from the semantics of other listed operations (e.g., Intersection as a special case of Join, Unpivoting as an inverse operation to Pivoting, and Datatype Conversion as a special case of Field Alteration using a specific conversion function).

In our approach, we use such formalization of operation semantics to automatically extract the constraints that an operation implies over the input data, hence to further generate the input data for covering such operations. However, notice that some operations in Table 4.4 may imply specific semantics over input data that are not explicitly expressed in the given formalizations (e.g., *Field Addition/Alteration*, *Single Value Alteration*). Such semantics may span from simple arithmetic expressions (e.g., $yield = divident \div DM\_CLOSE$), to complex user defined functions expressed in terms of an ad hoc script or code snippets. While the former case can be easily tackled by powerful expression parsers (Jovanovic et al., 2014), in the later case the operation's semantics must be carefully analyzed to extract the constraints implied over input data (e.g., by means of the static code analysis, as suggested in (Hueske et al., 2012)).

[3] n is the number of replicas in the Replicate Row operation semantics

Table 4.4: Table of ETL operations semantics

Op. Level	Op. Type	Op. Semantics
Value	Single Value Alteration	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{I}) = Attr(\mathbb{O}) \wedge \lvert\mathbb{I}\rvert = \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}(P_i(t_{in}[\mathbf{X}]) \rightarrow \exists t_{out} \in \mathbb{O}(t_{out}[Attr(\mathbb{O}) \setminus \mathbf{A}] = t_{in}[Attr(\mathbb{I}) \setminus \mathbf{A}] \wedge t_{out}(A) = F_j(t_{in}[\mathbf{X}])))$
Field	Field Alteration	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{I}) = Attr(\mathbb{O}) \wedge \lvert\mathbb{I}\rvert = \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}, \exists t_{out} \in \mathbb{O}(t_{out}[Attr(\mathbb{O}) \setminus A] = t_{in}[Attr(\mathbb{I}) \setminus A] \wedge t_{out}(A) = F_j(t_{in}[\mathbf{X}])))$
Row	Duplicate Row	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{I}) = Attr(\mathbb{O}) \wedge \lvert\mathbb{I}\rvert < \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}, \exists \mathbb{O}\prime \subseteq \mathbb{O}, \lvert\mathbb{O}\prime\rvert = n^{\,3} \wedge \forall t_{out} \in \mathbb{O}\prime, t_{out} = t_{in}$
	Router	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow \forall j(Attr(O_j) = Attr(\mathbb{I}) \wedge \lvert\mathbb{I}\rvert \geq \lvert O_j\rvert))$ $\forall j, \forall t_{in} \in \mathbb{I}(P_j(t_{in}[x_j]) \rightarrow \exists t_{out} \in O_j, (t_{out} = t_{in})$
	Filter	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{O}) = Attr(\mathbb{I}) \wedge \lvert\mathbb{I}\rvert \geq \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}(P_j(t_{in}[\mathbf{X}]) \rightarrow \exists t_{out} \in \mathbb{O}, (t_{out} = t_{in})$
	Join	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{O}) = Attr(I_1) \cup Attr(I_2) \wedge \lvert\mathbb{O}\rvert \leq \lvert I_1 \times I_2\rvert))$ $\forall t_{in_1} \in I_1, t_{in_2} \in I_2, (P(t_{in_1}[x_1], t_{in_2}[x_2]) \rightarrow \exists t_{out} \in \mathbb{O}(t_{out} = t_{in_1} \bullet t_{in_1})$
	Union	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(I_1) = Attr(I_2) \wedge Attr(\mathbb{O}) = Attr(I_1) \wedge \lvert\mathbb{O}\rvert = \lvert I_1\rvert + \lvert I_2\rvert))$ $\forall t_{in} \in (I_1 \cup I_2) \rightarrow \exists t_{out} \in \mathbb{O}(t_{out} = t_{in})$
	Difference	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(I_1) = Attr(I_2) \wedge Attr(\mathbb{O}) = Attr(I_1) \wedge \lvert\mathbb{O}\rvert \leq \lvert I_1\rvert))$ $\forall t_{in}(t_{in} \in I_1 \wedge t_{in} \notin I_2) \rightarrow \exists t_{out} \in \mathbb{O}(t_{out} = t_{in})$
Dataset	Aggregation	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{O}) = \mathbf{X} \cup \mathbf{A} \wedge Attr(\mathbb{O}) \leq Attr(I)))$ $\forall \mathbb{I}\prime \in 2^{\mathbb{I}}(\forall t_{in_1} \in \mathbb{I}\prime(\forall t_{in_2} \in \mathbb{I}\prime(t_{in_1}[\mathbf{X}] = t_{in_2}[\mathbf{X}]) \wedge \forall t_{in_k} \in \mathbb{I} \setminus \mathbb{I}\prime, t_{in_1}[X] \neq t_{in_k}[X])) \rightarrow$ $\rightarrow \exists! t_{out} \in \mathbb{O}(t_{out}[\mathbf{X}] = t_{in_1}[\mathbf{X}] \wedge t_{out}[\mathbf{A}] = F_j(\mathbb{I}\prime))$
	Sort	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{I}) = Attr(\mathbb{O}) \wedge \lvert\mathbb{I}\rvert = \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}, \exists t_{out} \in \mathbb{O}(t_{out} = t_{in})$ $\forall t_{out}, t_{out}\prime \in \mathbb{O}(t_{out}[\mathbf{X}] < t_{out}\prime[\mathbf{X}] \rightarrow t_{out} \prec t_{out}\prime)$
	Duplicate Removal	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{I}) = Attr(\mathbb{O}) \wedge \lvert\mathbb{I}\rvert \geq \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}, \exists! t_{out} \in \mathbb{O}(t_{out} = t_{in})$
	Dataset Copy	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow \forall j(Attr(O_j) = Attr(\mathbb{I}) \wedge \lvert\mathbb{I}\rvert = \lvert O_j\rvert))$ $\forall j, \forall t_{in} \in \mathbb{I}, \exists t_{out} \in O_j, (t_{out} = t_{in})$
Schema	Projection	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{O}) = Attr(\mathbb{I}) \setminus \mathbf{X} \wedge \lvert\mathbb{I}\rvert = \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}, \exists t_{out} \in \mathbb{O}(t_{out}[Attr(\mathbb{O})] = t_{in}[Attr(\mathbb{I}) \setminus \mathbf{X}]))$
	Field Renaming	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{O}) = (Attr(\mathbb{I}) \setminus \mathbf{X}) \cup \mathbf{A}) \wedge \lvert\mathbb{I}\rvert = \lvert\mathbb{O}\rvert))$
	Field Addition	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{O}) = Attr(\mathbb{I}) \cup \mathbf{A} \wedge \lvert\mathbb{I}\rvert = \lvert\mathbb{O}\rvert))$ $\forall t_{in} \in \mathbb{I}, \exists t_{out} \in \mathbb{O}(t_{out}[Attr(\mathbb{O}) \setminus \mathbf{A}] = t_{in}[Attr(\mathbb{I})] \wedge t_{out}[\mathbf{A}] = F(t_{in}[\mathbf{X}]))$
Table	Pivoting	$\forall(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A})(F(\mathbb{I},\mathbb{O},\mathbf{X},\mathbf{S},\mathbf{A}) \rightarrow (Attr(\mathbb{O}) = (Attr(\mathbb{I}) \setminus \mathbf{X}) \cup \mathbf{A} \wedge \lvert\mathbb{O}\rvert = \lvert\mathbb{I}\rvert_a \wedge \lvert\mathbb{I}\rvert = \lvert\mathbb{O}\rvert_a))$ $\forall t_{in} \in \mathbb{I}, \forall a \in Attr(\mathbb{I}), \exists t_{out} \in \mathbb{O}, \exists b \in Attr(\mathbb{O})(t_{out}[b] = t_{in}[a]))$

4.1.3 Bijoux overview

Intuitively, starting from a logical model of an ETL process and the semantics of ETL operations, *Bijoux* analyzes how the fields of input data stores are restricted by the semantics of the ETL process operations (e.g., *filter* or *join* predicates) in order to generate the data that satisfy these restrictions. To this end, *Bijoux* moves iteratively through the topological order of the nodes inside the DAG of an ETL process and extracts the semantics of each ETL operation to analyze the constraints that the operations imply over the input fields. At the same time, *Bijoux* also follows the constraints' dependencies among the operations to simultaneously collect the necessary parameters for generating data for the correlated fields (i.e., *value ranges*, *datatypes*, and *the sizes of generated data*). Using the collected parameters, *Bijoux* then generates input datasets to satisfy all found constrains, i.e., to simulate the execution of selected parts of the data flow. The algorithm can be additionally parametrized to support data generation for different execution scenarios.

Typically, an ETL process should be tested for different sizes of input datasets (i.e., different *scale factors*) to examine its scalability in terms of growing data. Importantly, *Bijoux* is extensible to support data generation for different characteristics of input datasets (e.g., *size*), fields (e.g., *value distribution*) or ETL operations (e.g., *operation selectivity*). We present in more detail the functionality of our data generation algorithm in the following section.

4.2 BIJOUX DATA GENERATION FRAMEWORK

The data generation process includes four main stages (i.e., 1 - *path enumeration*, 2 - *constraints extraction*, 3 - *constraints analysis*, and 4 - *data generation*). In what follows, we first discuss some of the important challenges of generating data for evaluating general ETL flows, as well as the main structures maintained during the data generation process. Subsequently, we present the path enumeration and constraints extraction algorithms and discuss about the data generation stage. Finally, we provide the theoretical validation of our approach.

4.2.1 Preliminaries and Challenges

The workflow-graph structure of the ETL logical model that we adopt for our analysis consists of ETL operations as graph *nodes*, input data stores as graph *sources* and output data stores as graph *sinks*. In particular, input data stores, as well as routing operations (e.g., Routers) that direct rows to different outputs based on specified conditions, introduce alternative directed paths of the input graph (in the rest of the chapter referred to as *paths*), which can be followed by input data. Hence, there are two properties of the generated input data that can be defined:

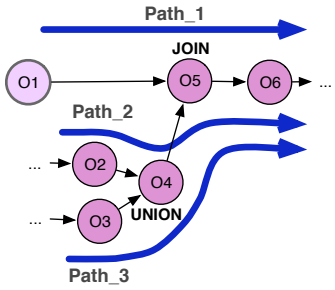
1. *Path Coverage*: Input data are sufficient to "cover" a specific path, i.e., each and every edge (or node) that is on this path is visited by at least one row of data.

2. *Flow Coverage*: Input data are sufficient to "cover" the complete flow graph, i.e., each and every edge (or node) of the flow graph is visited by at least one row of data.

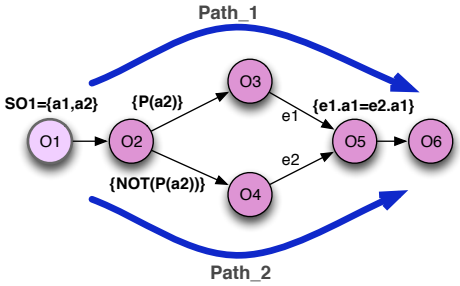
The apparently simple case of *Path Coverage* hides an inherent complexity, deriving from the fact that some joining operations (i.e., *joining nodes*; e.g., Join, Intersection) require the involvement of multiple paths in order to direct data to their output. In addition, new fields are introduced to the flow either through input data stores or Field Addition operations (see Table 4.4), while the fields from different paths are *fused*/joined together through joining operations. This in turn implies two facts: i) *Path Coverage* is not guaranteed by generating the right input data only for the input data store that is involved in a specific path; instead, data generation should be conducted for a combination of paths (i.e., their included input data stores), and ii) during the *Path Coverage* analysis, referring to a field solely by its name is not sufficient; the same field might participate in multiple paths from a combination of paths, in each path holding different properties coming from extracted constraints of different operations. Thus, the *name* of a field should be combined with a *pathid* to identify one distinct entity with specific properties.

In Figure 4.3, we show some notable cases of graph patterns that require special attention during the coverage analysis, as described above.

In Figure 4.3a, we can see how the coverage of `Path_1` (O1→O5→O6...) needs multiple paths to be considered for data generation, because of the joining operation O5 that requires multiple inputs (e.g., a *Join* operation). Thus, coverage can be ensured by using alternative combinations, either `Path_1` in combination with `Path_2` (...O2→O4→O5→O6...), or `Path_1` in combination with `Path_3` (...O2→O4→O5→O6...). It should be mentioned that operation O4 is of a merging type that does not require both of its incoming edges to be crossed in order



(a) Alternative path combinations for coverage of the same path



(b) Multiple rows from same input source required for coverage

Figure 4.3: Notable cases of graph patterns

to pass data to its output (i.e., a *Union* operation) and thus `Path_2` and `Path_3` can be used interchangeably for coverage.

In Figure 4.3b, we show how the coverage of one path might require the generation of multiple rows for the same input source. For example, for the Path Coverage of `Path_1` (O1→O2→O3→O5→O6) it is required to additionally generate data for `Path_2` (O1→O2→O4→O5→O6), because of the existence of the joining operation O5. It should be noticed here that fields *a1* and *a2* in `Path_1` belong to a different instance than in `Path_2`, since the condition of the routing operator O2 imposes different predicates over *a2* for different paths (i.e., P(a2) and NOT(P(a2)), respectively). Hence, at least two different rows from the same input data store are required for *Path Coverage* of `Path_1`.

Example. For illustrating the functionality of our algorithm, we will use the running example introduced in Section 4.1 (see Figure 4.1). For the sake of simplicity, we will not use the complete schemata of the input data stores as specified in the *TPC-DI benchmark*, but instead we assume simplified versions, where the only fields present are the ones that are used in the ETL flow, i.e., taking part in predicates or functions. In this manner, input data stores of the example ETL flow are: $\mathbb{I}$ = {O1, O4, O9}, with schemata SO1 = {PTS, RecType, Status, CoNameOrCIK}, SO4 = {CompanyID, Name, EffectiveDate, EndDate} and SO9 = {ST_ID, ST_NAME}; whilst a topological order of its nodes is: {O1, O2, O3, O4, O5, O6, O7, O8, O9, O10, O11, O12}. Besides this running example, we will also use the auxiliary example graph from Figure 4.4a to support the description of the complete functionality of *Bijoux* □

4.2.2 Data structures

Before going into the details of Algorithms 1 and 2 in Section 4.2.4, we present the main structures maintained by these algorithms.

While analyzing a given ETL graph, in Algorithm 1, *Bijoux* builds the following structures that partially or completely record the path structures of the input ETL graph (i.e., path traces):

- *Path Traces* ($\mathbb{PT}$) collection keeps traces of operations and edges that have been visited, when following a specific path up to a specific node in the ETL graph. Traces of individual paths PT ($PT \in \mathbb{PT}$) are built incrementally and thus, following a specific path on the graph, if a Path Trace PT1 is generated at an earlier point than the generation of a Path Trace PT2, then PT1 will include a subset of the trace of PT2 (i.e., $PT1 \subseteq PT2$). From an implementation point of view, each PT holds a *Signature* as a property, which can be a string concatenation of graph elements that shows which route has been followed in the case of alternative paths. This enables very efficient PT analysis and comparisons by simply applying string operations.
  Example. Referring to our running example in Section 4.1, we can have the following signature of a Path Trace PT1:
  $Sig(PT1) =$ "$I[O1].S[O2, true].S[O3, true].J[O6, e1]$"
  From this signature we can conclude that PT1 starts from I (i.e., Input Source): *O1*; passes through S (i.e., Splitting Operation): *O2* coming from its outgoing edge that corresponds to the evaluation: *true* of its condition; passes through S (i.e., Splitting Operation): *O3* coming from its outgoing edge that corresponds to the evaluation: *true*; passes through J (i.e., Joining Operation): *O6* coming from its incoming edge: *e1*; and so on. For some operations (e.g., Joins) it makes sense to keep track of the incoming edge through which they have been reached in the specific path and for some others (e.g., Routers), it makes sense to keep track of the outgoing edge that was followed for the path.
  Looking at the following signature of Path Trace PT2:
  $Sig(PT2) =$ "$I[O1].S[O2, true].S[O3, true]$" , we can infer that PT1 and PT2 are on the same path of the ETL graph, PT2 being generated at an "earlier" point, since the signature of PT2 is a substring of the signature of PT1. □

- *Tagged Nodes* ($\mathbb{TN}$) structure records, for each node, the set of paths (i.e., operations and edges) reaching that node from the input data store nodes (i.e., source nodes). Thus, each node is "tagged" with a set of Path Traces ($\mathbb{PT}$) which are being built incrementally, as explained above.
  Example. Referring to our running example, within $\mathbb{TN}$ the *O7* operation node will be "tagged" with four different path traces, $PT1$, $PT2$, $PT3$ and $PT4$ with the following signatures:
  - $Sig(PT1) =$ "$I[O1].S[O2, true].S[O3, true].J[O6, e1].J[O7, e1]$"
  - $Sig(PT2) =$ "$I[O1].S[O2, true].S[O3, false].J[O5, e1].J[O7, e2]$"
  - $Sig(PT3) =$ "$I[O4].J[O6, e2].J[O7, e1]$"
  - $Sig(PT4) =$ "$I[O4].J[O5, e2].J[O7, e2]$" □

- *Final path traces* ($\mathbb{FP}$) structure records all the *complete* (i.e., *source-to-sink*) paths from

the input ETL graph, by maintaining all source-to-sink Path Traces (i.e., the union of all Path Traces that tag sink nodes).

When it comes to formalizing the main structure that is being built by Algorithm 2 (i.e., data generation pattern), we define its structure as follows:

- A data generation pattern ($Pattern$) consists of a set of path constraints (i.e., *pathConstr*), where each path constraint is a set of constraints over the input fields introduced by the operations of an individual path. Formally:
  $Pattern = \{pathConstr_i | i = 1, \cdots, pathNum\}$

  Example. In our running example (Figure 4.1), so as to cover the path Path1=(O1→O2→O3→O6→O7→O8→O10→O11→O12), additionally, the path Path2=(O4→O6→O7→O8→O10→O11→O12) and the path Path3=(O9→O10→O11→O12) need to be covered as well, because of the equi-join operators *O6* and *O10*. The *Pattern* would then consist of three constraints sets ($pathConstr1$, $pathConstr2$ and $pathConstr3$), one for each (source-to-sink) path of the flow that has to be covered. □

- A path constraint (i.e., $pathConstr_i$) consists of a set of constraints over individual fields of the given path (i.e., $fieldConstr$). Formally:
  $pathConstr_i = \{fieldConstr_j | j = 1, \cdots, pathFieldNum\}$
  Example. Each constraints set in our example will contain a set of constraints for any of the fields that are involved in imposed predicates of operations on the related path. For example, $pathConstr1$ will contain constraints over the fields: Path1.PTS, Path1.RecType, Path1.Status, Path1.CoNameOrCIK, Path1.CompanyID, Path1.Name, Path1.EffectiveDate, Path1.EndDate, Path1.ST_ID, Path1.ST_name. Notice that each field is also defined by the related path. Respectively, $pathConstr2$ and $pathConstr3$ will contain constraints over the same fields as $pathConstr1$, but with the corresponding path as identifier (e.g., Path2.PTS, Path2.RecType and so on for $pathConstr2$ and Path3.PTS, Path3.RecType and so on for $pathConstr3$). In our example, it does not make any difference maintaining constraints coming from fields of O4 for Path1 (for e.g., CompanyId for Path1), since the flow is not split after it merges, but in the general case they are necessary for cases of indirect implications over fields from one path and for determining the number of rows that need to be generated. □

- A field constraint (i.e., $fieldConstr_j$) is defined as a pair of an input field and an ordered list of constraint predicates over this field. Formally:
  $fieldConstr_j = [field_j, \mathbb{S}_j]$
  Example. An example field constraint that can be found in our running scenario within $pathConstr1$, is:
  $fieldConstr_1 = [Path1.RecType, \{(RecType == `SEC')\}]$ □

- Finally, a constraint predicates list defines the logical predicates over the given field in the topological order they are applied over the field in the given path. Formally:

$\mathbb{S}_j =< P_1(field_j), \cdots, P_{constrNum}(field_j) >$
The list needs to be ordered to respect the order of operations, since in the general case:
$f_1(f_2(field_x)) \neq f_2(f_1(field_x))$

After processing the input ETL graph in Algorithm 1, Algorithm 2 uses the previously generated collection of final path traces (i.e., $\mathbb{FP}$) for traversing a selected *complete* path (i.e., $PT \in \mathbb{FP}$) and constructing a *data generation pattern* used finally for generating data that will guarantee its *coverage*. Thus, Algorithm 2 implements the construction of a data generation pattern for *path coverage* of one specific path. For *flow coverage* we can repeat Algorithm 2, starting every time with a different PT from the set of final path traces $\mathbb{FP}$, until each node of the ETL graph has been visited at least once. We should notice here that an alternative to presenting two algorithms — one for path enumeration and one for pattern construction — would be to present a merged algorithm, which traverses the ETL graph and at the same time extracts constraints and constructs the data generation pattern. However, we decided to keep Algorithm 1 seperate for two reasons: i) this way the space complexity is reduced while computational complexity remains the same and ii) we believe that the path enumeration algorithm extends beyond the scope of ETL flows and can be reused in a general case for implementing a directed path enumeration in polynomial time, while constructing efficient structures for comparison and analysis (i.e., *Path Traces*). A similar approach of using a compact and efficient way to represent ETL workflows using string signatures has been previously introduced in (Tziovara et al., 2007).

4.2.3 Path Enumeration Stage

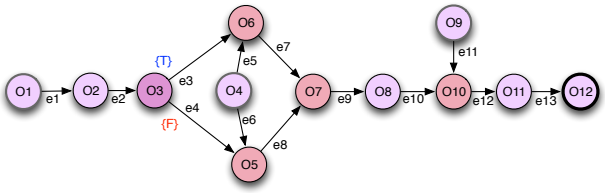
In what follows, we present the path enumeration stage, carried out by Algorithm 1.

In the initial stage of our data generation process, *Bijoux* processes the input ETL process graph in a topological order (Step 2) and for each source node starts a new path trace (Step 5), initialized with the operation represented by a given source node. At the same time, the source node is tagged by the created path trace (Step 6). For other (non-source) nodes, *Bijoux* gathers the path traces from all the previously tagged predecessor nodes (Step 8), extends these path traces with the current operation o_i (Step 9), while o_i is tagged with these updated path traces ($\mathbb{PT}$). Finally, if the visited operation is a sink node, the traces of the paths that reach this node are added to the list of final path traces (i.e., $\mathbb{FP}$). Processing the input ETL process graph in this manner, Algorithm 1 gathers the complete set of final path traces, that potentially can be covered by the generated input data. An example of the execution of Algorithm 1 applied on our running example and the 5 resulting final path traces are shown in Figure 4.

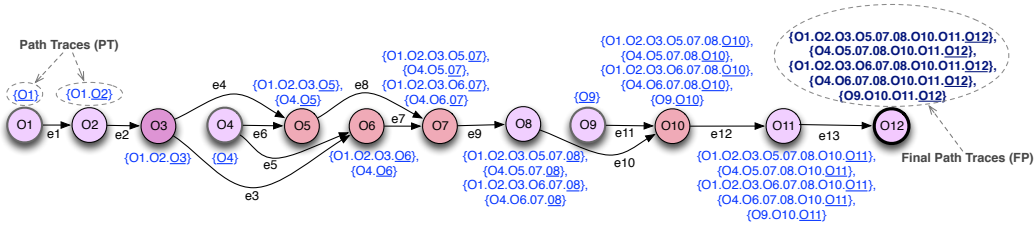
Algorithm 1 Enumerate Paths and Generate Path Traces

Input: ETL
Output: $\mathbb{FP}$

```
1:  TN ← new Tagged Nodes; FP ← ∅;
2:  for each operation o_i ∈ TopOrder(ETL) do
3:      if (o_i is source) then
4:          PT ← ∅;
5:          PT.addElement(new Path Trace(o_i));
6:          TN.addTag(PT, o_i);
7:      else
8:          PT ← TN.UnionOfAll_PTs_forAllPredecessorNodesOf(o_i);
9:          PT.updateBasedOnOperation(o_i);
10:         if (o_i is sink) then
11:             FP.addAllElementsFrom(PT);
12:         else
13:             TN.addTag(PT, o_i);
14: return FP;
```



(a) DAG representation of our running example



(b) Execution of Algorithm 1 for the topological order of the DAG representation of our running example

Figure 4.4: Example of execution of Algorithm 1

4.2.4 Constraints Extraction and Analysis Stage

In what follows, we discuss in detail the constraints extraction and analysis stages of our data generation process, carried out by Algorithm 2.

After all possible *final* paths of input ETL graph are processed and their traces recorded in $\mathbb{FP}$, an end-user may select an individual path she wants to *cover*. To this end, *Bijoux* runs

Algorithm 2 Construct Data Generation Pattern for one Path

Input: ETL, PT, $\mathbb{FP}$
Output: Pattern

$\mathbb{AP} \leftarrow \emptyset$;
for each operation o_i *crossedBy* PT **do**
    if (o_i is of type *joining_node*) **then**
        $AP_i \leftarrow \emptyset$
        for each Path Trace $PT_j \in \mathbb{TN}$.*getAllPathTracesFor*(o_i) **do**
            if (PT_j.*PredecessorOf*(o_i) $\neq$ PT.*PredecessorOf*(o_i)) **then**
                AP_i.*add*(PT_j);
        $\mathbb{AP}$.*add*(AP_i);
$\mathbb{C} \leftarrow$ *allCombinations*(PT, $\mathbb{AP}$);
for each Combination $C \in \mathbb{C}$ **do**
    Pattern $\leftarrow \emptyset$;
    for each Path Trace $PT_i \in C$ **do**
        for each operation o_j *crossedBy* PT_i **do**
            Pattern.*addConstraints*(o_j);
            if (¬Pattern.*isFeasible*) **then**
                abortPatternSearchForC();
    return Pattern;
return $\emptyset$;

Algorithm 2, with a selected path $PT \in \mathbb{FP}$, and builds a data generation $Pattern$ to cover (at least) the given path. Algorithm 2 iterates over all the operation nodes of the selected path (Step 2), and for each *joining node* (i.e., node with multiple incoming edges), it searches in $\mathbb{FP}$ for all paths that reach the same *joining node*, from now on, *incident paths* (Steps 5 - 11). As discussed in Section 4.2.2, routing operations (e.g., *Router*) introduce such paths, and they need to be considered separately when generating data for their coverage (see Figure 4.3). In general, there may be several *joining nodes* on the selected path, hence Algorithm 2 must take into account all possible combinations of the alternative incident paths that reach these nodes (Step 9).

Example. Referring to the DAG of Figure 4.4a, if the path to be covered is (O9→O10→O11 →O12), it would require the coverage of additional path(s) because of the equi-join operator *O10*. In other words, data would also need to be coming from edge *e10* in order to be matched with data from edge *e11*. However, because of the existence of a Union operator (*O7*), there are different alternative combinations of paths that can meet this requirement. The reason is that data coming from either of the incoming edges of a *Union* operator reach its outgoing edge. Hence, data reaching *O10* from edge *e10* could pass through path (O1→O2→O3→O6→O7→O8...) combined with path (O4→O6→O7→O8...) or through path (O1→O2→O3→O5→O7→O8...) combined with (O4→O6→O7→O8...). Thus, we see how two alternative combinations of paths, each containing three different paths, can be used for the coverage of one single path. □

For each combination, Algorithm 2 attempts to build a data generation pattern, as explained above. However, some combination of paths may raise a contradiction between the constraints over an input field, which in fact results in disjoint value ranges for this field and thus makes it unfeasible to cover the combination of these paths using a single instance of the input field (Step 16). In such cases, Algorithm 2 aborts pattern creation for a given combination and tries with the next one.

Example. Referring to the DAG of Figure 4.4a, we can imagine field *f1*, being present in the schema of operation *O6* and field *f2* being present in the schema of operation *O9*. We can also imagine that the datatype of *f1* is *integer* and the datatype of *f2* is *positive integer*. Then, if the joining condition of operation *O10* is ($f1 = f2$) and at the same time, there is a constraint (e.g., in operation *O6*) that ($f1 < 0$), the algorithm will fail to create a feasible data generation pattern for the combination of paths (O1→O2→O3→O5...→O12) and (O9→O10→O11→O12). □

Otherwise, the algorithm updates currently built $Pattern$ with the constraints of the next operation (o_j) found on the path trace.

As soon as it finds a combination that does not raise any contradiction and builds a complete feasible $Pattern$, Algorithm 2 finishes and returns the created data generation pattern (Step 17). Notice that by covering at least one combination (i.e., for each *joining node*, each and every incoming edge is crossed by one selected path), Algorithm 2 can guarantee the coverage of the selected input path PT.

Importantly, if Algorithm 2 does not find a feasible data generation pattern for any of the alternative combinations, it returns an empty pattern (Step 18). This further indicates that the input ETL process model is not correct, i.e., that some of the path branches are not reachable for any combination of input data.

The above description has covered the general case of data generation without considering

	Datatype	Distribution Type	Modification
PTS	Integer	Triangular	-
RecType	String	Uniform Discrete	deform 2%
CoNameOrCIK	String	Complex	-
CompanyID	Long	Uniform	addNullValues 1%
EffectiveDate	Integer	Uniform	setToMaxInt 1%

Field Parameters (FP)

	Selectivity
O2 (Filter_RecType)	0.3
O3 (Router_1)	0.7
O6 (Join_1)	1
O5 (Join_2)	0.95
O8 (Filter_Date)	0.6

Operation Parameters (OP)

Figure 4.5: Data generation parameters (FP and OP)

other generation parameters. However, given that our data generator aims at generating data to satisfy other configurable parameters, we illustrate here as an example the adaptability of our algorithm to the problem of generating data to additionally satisfy *operation selectivity*. To this end, the algorithm now also analyzes the parameters at the operation level (OP) (see Figure 4.5:right). Notice that such parameters can be either obtained by analyzing the input ETL process for a set of previous real executions, or simply provided by the user, for example, for analyzing the flow for a specific set of operation selectivities.

Selectivity of an operation o expresses the ratio of the size of the dataset at the output (i.e., *card(o)*), to the size at the input of an operation (i.e., *input(o)*). Intuitively, for filtering operations, we express selectivity as the percentage of data satisfying the filtering predicate (i.e., $sel(o) = \frac{card(o)}{input(o)}$), while for n-ary (join) operations, for each input e_i, we express it as the percentage of the data coming from this input that will match with other inputs of an operation (i.e., $sel(o, e_i) = \frac{card(o)}{input(o,e_i)}$).

From the OP (see Figure 4.5:right), *Bijoux* finds that operation *O2* (Filter_RecType) has a selectivity of *0.3*. While processing a selected path starting from the operation *O1*, *Bijoux* extracts operation semantics for *O2* and finds that it uses the field *RecType* (*RecType*==‘*SEC*’). With the selectivity factor of *0.3* from OP, *Bijoux* infers that out of all incoming rows for the Filter, 30% should satisfy the constraint that *RecType* should be equal to SEC, while 70% should not. We analyze the selectivity as follows:

- To determine the total number of incoming rows for operation *O8* (Filter_Date), we consider predecessor operations, which in our case come from multiple paths.
- As mentioned above, operation *O2* will allow only 30% of incoming rows to pass. Assuming that the input load size from *FINWIRE* is 1000, this means that in total $0.3 * 1000 = 300$ rows pass the filter condition.
- From these 300 rows only 70%, based on the *O3* (Router_1) selectivity, (i.e., 210 rows) will successfully pass both the filtering (*RecType*==‘*SEC*’) and the router condition (*isNumber(CoNameOrCIK)*) and hence will be routed to the route that evaluates to $true$. The rest ((i.e., $300 - 210 = 90$ rows)) will be routed to the route that evaluates to $false$.
- The 210 rows that pass both previous conditions, will be matched with rows coming from operation *O4* through the join operation *O6* (Join_1). Since the selectivity of operation *O6* is 1, all 210 tuples will be matched with tuples coming from *O4* and meeting the condition *CoNameOrCIK*==*CompanyID* and hence will pass the join condition. On the other

hand, the selectivity of operation *O5* (Join_2), for the input coming from *O3*(Router_1), is 0.95, which means that from the 90 rows that evaluated to $false$ for the routing condition, only 85 will be matched with tuples coming from *O4* and meeting the condition *CoNameOrCIK==Name*. Thus, $210 + 85 = 295$ tuples will reach the union operation *O6* and pass it.

- Finally, from the 295 rows that will reach operation *O8* (Filter_Date) coming from the preceding union operation, only $0.6 * 295 = 177$ will successfully pass the condition *(PTS>=EffectiveDate) AND (PTS<= EndDate)*, as the selectivity of *OP8* is 0.6.

  In order to generate the data that do not pass a specific operation of the flow, a data generate pattern inverse to the initially generated $Pattern$ in Algorithm 2 needs to be created to guarantee the percentage of data that will fail the given predicate.

Similarly, other parameters can be set for the generated input data to evaluate different quality characteristics of the flow, (see Figure 4.5:left). As an example, the percentage of null values or incorrect values (e.g., wrong size of telephone numbers or negative age) can be set for the input data, to evaluate the measured data quality of the flow output, regarding *data completeness* and *data accuracy*, respectively. Other quality characteristics like *reliability* and *recoverability* can be examined as well, by adjusting the distribution of input data that result to exceptions and the selectivity of exception handling operations. Examples of the above will be presented in Section 4.3.

4.2.5 Data Generation Stage

Lastly, after the previous stage builds data generation patterns for covering either a single path, combination of paths, or a complete flow, the last (data generation) stage proceeds with generating data for each input field f. Data are generated within the ranges (i.e., R) defined by the constraints of the provided pattern, using either random numerical values within the interval or dictionaries for selecting correct values for other (textual) fields.

For each field f, data generation starts from the complete domain of the field's datatype $dt(f)$.

Each constraint P, when applied over the an input field f, generates a set of disjoint ranges of values $R_i^{f,init}$ in which the data should be generated, and each range being inside the domain of the field's datatype $dt(f)$. Formally:

$$P(f) = R^{f,init} = \left\{ r^{f,init} | r^{f,init} \subseteq dt(f) \right\} \tag{4.1}$$

For example, depending on the field's datatype, a value range for numeric datatypes is an interval of values (i.e., $[x, y]$), while for other (textual) fields it is a set of possible values a field can take (e.g., *personal names*, *geographical names*).

After applying the first constraint P_1, *Bijoux* generates a set of disjoint, non-empty value ranges R_1^f, each range being an intersection with the domain of the field's datatype.

$$R_1^f = \Big\{ r_1^f | \forall r_1^{f,init} \in R_1^{f,init}, \exists r_1^f, s.t. : \\ (r_1^f = r_1^{f,init} \cap dt(f) \wedge r_1^f \neq \emptyset) \Big\} \tag{4.2}$$

Iteratively, the data generation stage proceeds through all the constraints of the generation pattern. For each constraint P_i it updates the resulting value ranges as an intersection with the ranges produced in the previous step, and produces a new set of ranges R_i^f.

$$R_i^f = \Big\{ r_i^f | \forall r_i^{f,init} \in R_i^{f,init}, \forall r_{i-1}^f \in R_{i-1}^f, \exists r_i^f, s.t. : \\ (r_i^f = r_i^{f,init} \cap r_{i-1}^f \wedge r_i^f \neq \emptyset) \Big\} \tag{4.3}$$

Finally, following the above formalization, for each input field f *Bijoux* produces a final set of disjoint, non-empty value ranges ($R^{f,final}$) and for each range it generates an instance of f inside that interval.

See for example, in Figure 4.6 and Figure 4.7, the generated data sets for covering the ETL process flow of our running example. We should mention at this point, that non conflicting constraints for the same field that is present in different paths and/or path combinations, can be merged and determine a single range (i.e., the intersection of all the ranges resulting from the different paths). This way, under some conditions, the same value within that interval can be used for the coverage of different paths. As an example, in Figure 4.6, the fields *Status* and *ST_ID* that exist in both path combinations, all hold a constraint (ST_ID==Status). These can be merged into one single constraint, allowing for the generation of only one row for the table *StatusType* that can be used for the coverage of both path combinations, as long as both generated values for the field *Status* equal the generated value for the field *ST_ID* (e.g., "ACTV").

Following this idea, it can easily be shown that under specific conditions, the resulting constraints for the different path combinations from the application of our algorithm, can be further reduced, until they can produce a minimal set of datasets for the coverage of the ETL flow.

Data generation patterns must be further combined with other user-defined data generation parameters (e.g., selectivities, value distribution, etc.). We provide more details regarding this within our test case in Section 4.3.

4.2.6 Theoretical validation

We further provide a theoretical validation of our data generation process in terms of: the *correctness* of generated data sets (i.e., *path and flow coverage*).

A theoretical proof of the correctness of the *Bijoux* data generation process is divided into the three following components.

1. *Completeness of path traces.* Following from Algorithm 1, for each ETL graph node (i.e., datastores and operations, see Section 4.1.2) *Bijoux* builds path traces of all the paths

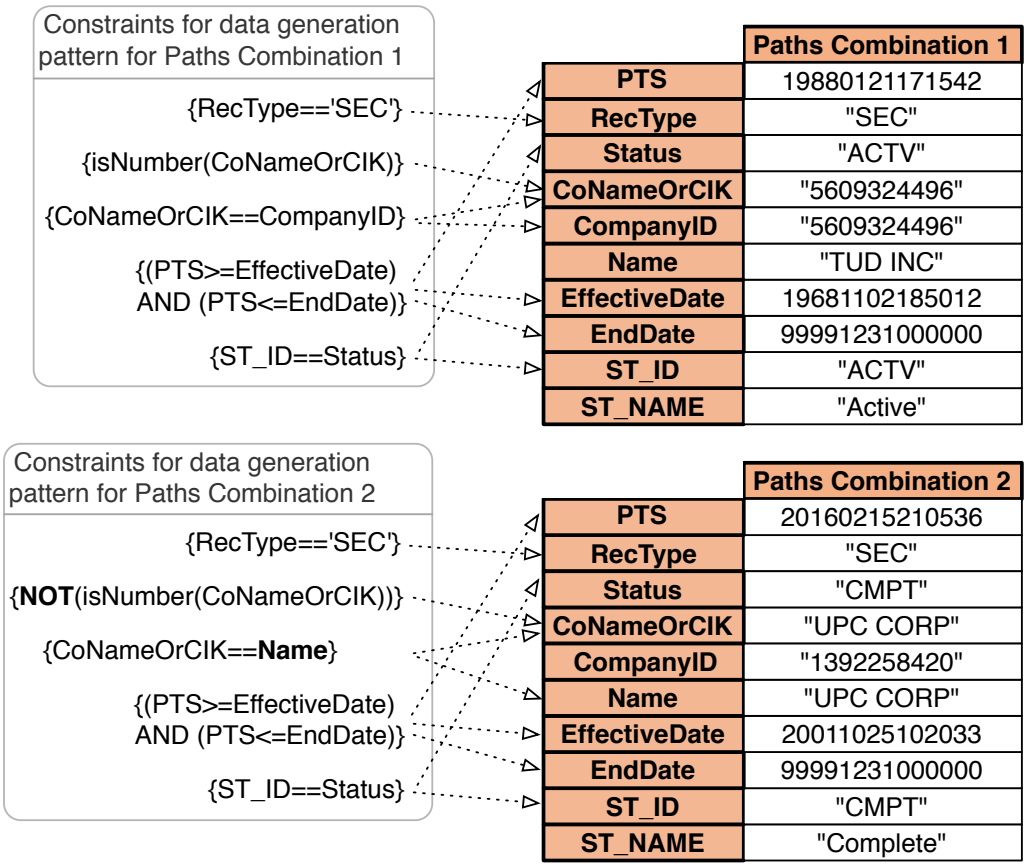


Figure 4.6: Data generated after analyzing all ETL operations

reaching that node (e.g., see Figure 4.4b). Formally, given that an ETL graph node can represent either an operation ($\mathbf{O}$), a source ($\mathbf{DS}_S$), or a target data store ($\mathbf{DS}_T$), we recursively formalize the existence of path traces as follows:

$$\forall v_i \in \mathbf{O} \cup \mathbf{DS}_T, \mathbb{PT}_{v_i} = \bigcup_{j=1}^{|\{v_j | v_j \prec v_i\}|} \left\{ PT_{v_j}^1 \cdot v_i, .., PT_{v_j}^{|\mathbb{PT}_j|} \cdot v_i \right\}. \tag{4.4}$$

$$\forall v_i \in \mathbf{DS}_S, \mathbb{PT}_{v_i} = \{PT_{v_i}\}, PT_{v_i} = v_i. \tag{4.5}$$

Considering that ETL graph nodes are visited in a topological order (see Step 2 in Algorithm 1), the path traces of each ETL graph node are built after visiting all its predeceasing sub-paths. This guarantees that path traces of each node v_i are complete with regard to all its predecessors (i.e., $\{v_j | v_j \prec v_i\}$), hence the final path traces $\mathbb{FP}$ (i.e., path traces of target data store nodes) are also complete.

FINWIRE

PTS	RecType	Status	CoNameOrCIK
19880121171542	"SEC"	"ACTV"	"5609324496"
20160215210536	"SEC"	"CMPT"	"UPC CORP"

StatusType

ST_ID	ST_NAME
"ACTV"	"Active"
"CMPT"	"Complete"

DimCompany

CompanyID	Name	EffectiveDate	EndDate
"5609324496"	"TUD INC"	19681102185012	99991231000000
"1392258420"	"UPC CORP"	20011025102033	99991231000000

Figure 4.7: Generated datasets corresponding to the generated data

2. *Path coverage.* Having the complete path traces recorded in Algorithm 1, Algorithm 2 traverses a selected path (i.e., PT), with all its alternative *incidence paths*, and builds a data generation $Patern$ including a list of constraints over the input fields. Following from 1, this list of constraints is complete. Moreover, as explained in Section 4.2.5, *Bijoux* iteratively applies given constraints, and for each input field f produces a set of value ranges ($R^{f,final}$), within which the field values should be generated.

   Given the statements 4.1 - 4.3 in Section 4.2.5, *Bijoux* guarantees that the data generation stage applies all the constraints over the input fields when generating $R^{f,final}$, thus guaranteeing that the complete selected path will be covered.

   On the other side, if at any step of the data generation stage a result of applying a new constraint P_i leads to an empty set of value ranges, the collected list of constraints must be contradictory. Formally (following from statement 4.3 in Section 4.2.5):
   $(\exists R_i^{f,init}, R_{i-1}^f | R_i^f = \emptyset) \rightarrow \perp$.
   This further implies that the input ETL graph has contradictory path constraints that would lead to an unreachable sub-path, which could never be executed. As an additional functionality, *Bijoux* detects such behavior and accordingly warns the user that the input ETL flow is not correct.

3. *Flow coverage.* Following from 2, Algorithm 2 generates data that guarantee the coverage of a single path from $\mathbb{FP}$. In addition, if Algorithm 2 is executed for each final path $PT_i \in \mathbb{FP}$, it is straightforward that *Bijoux* will produce data that guarantee the coverage of the complete ETL flow (i.e., ETL graph), unless a constraints contradiction for an individual path has been detected.

4.3 TEST CASE

The running example of the ETL flow that we have used so far is expressive enough to illustrate the functionality of our framework, but it appears too simple to showcase the benefits of our

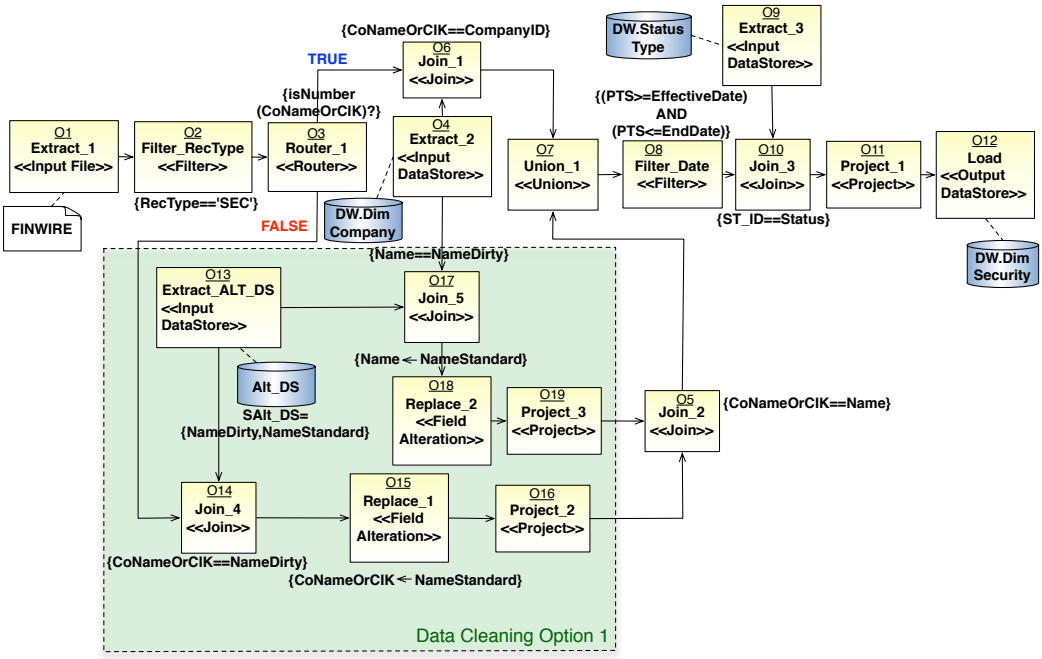


Figure 4.8: ETL flow for data cleaning, using a dictionary

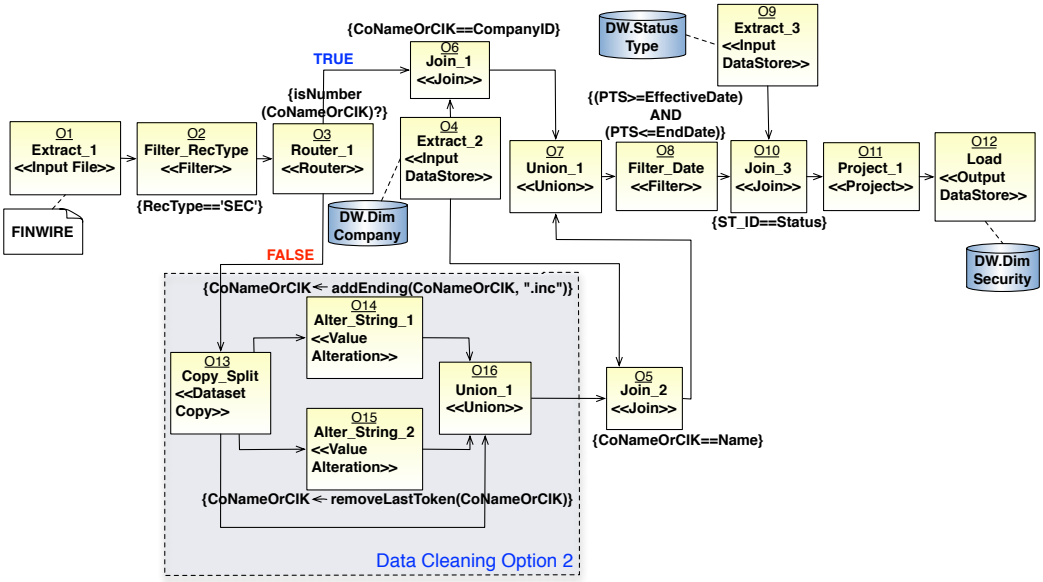


Figure 4.9: ETL flow for data cleaning, trying different string variations for the join key

approach regarding the evaluation of the quality of the flow. In this respect, we present in this section representative examples of how our framework can generate data, not only to enact specific parts of the ETL flow, but also to evaluate the performance and the data quality of these flow parts.

Going back to our running example (Figure 4.1), from now on referred to as *Flow_A*, we can identify a part of the flow that can be the source of data quality issues. That is, rows whose values for the field *CoNameOrCIK* are not numbers are matched with data about companies from the *DimCompany* table, through an equi-join on the company name (CoNameOrCIK==Name). However, company names are typical cases of attributes that can take multiple values in different systems or even within the same system. For example, for a company *Abcd Efgh*, its name might be stored as "Abcd Efgh", or followed by a word indicating its type of business entity (e.g., "Abcd Efgh Incorporated") or its abbreviation with or without a comma (e.g., "Abcd Efgh Inc." or "Abcd Efgh, Inc."). It is also possible that it might be stored using its acronym (e.g., "ABEF") or with a different reordering of the words in its name, especially when the two first words are name and surname of a person (e.g., "Efgh Abcd"). Moreover, there can be different uppercase and lowercase variations of the same string, combinations of the above-mentioned variations or even misspelled values.

Hence, there are many cases that the equi-join (CoNameOrCIK==Name) will fail to match the incoming data from the *FINWIRE* source with the rows from the *DimCompany* table, because they might simply be using a different variation of the company name value. This will have an impact on *data completeness*, since it will result in fewer rows being output to the DimSecurity than there should be.

To this end, we introduce here two more complex ETL flows (Figure 4.8 and Figure 4.9), which perform the same task as the running example, but include additional operations in order to improve the data quality of the output data. The ETL flow in Figure 4.8, from now on referred to as *Flow_B*, uses a dictionary (*Alt_DS*) as an alternative data source. This dictionary is assumed to have a very simple schema of two fields — *NameDirty* and *NameStandard*, to maintain a correspondence between different dirty variations of a company name and its standard name. For simplicity, we assume that for each company name, there is also one row in the dictionary containing the standard name, both as value for the *NameDirty* and the *NameStandard* fields. Operations *O14* and *O17* are used to match both the company names from the *FINWIRE* and the table, to the corresponding dictionary entries and subsequently, rows are matched with the standard name value being the join key, since the values for the join keys are replaced by the standard name values (*(Name←NameStandard)* and *(CoNameOrCIK←NameStandard)*).

Another alternative option for data cleaning is to try different variations of the company name value, by adding to the flow various string operations that alter the value of *CoNameOrCIK*. The ETL flow in Figure 4.9, from now on referred to as *Flow_C*, generates different variations of the value for *CoNameOrCIK* with operations *O14* and *O15*, who concatenate the abbreviation "inc." at the end of the word and remove the last token of the string, respectively. After the rows from these operations are merged through a Union operation (*O16*), together with the original *CoNameOrCIK* value, all these different variations are tried out to match with rows coming from *DimCompany*.

4.3.1 Evaluating the performance overhead of alternative ETL flows

In the first set of experiments, we implemented the three different ETL flows (*Flow_A*, *Flow_B* and *Flow_C*) using Pentaho Data Integration[4] and we measured their time performance by executing them on Kettle Engine, running on Mac OS X, 1.7 GHz Intel Core i5, 4GB DDR3 and keeping average values from 10 executions.

For each flow, we used *Bijoux* to generate data to cover only the part of the flow that was of interest, i.e., to cover the paths from Operations *O1* to *O12* who are covered by the rows that are evaluated as False by operation *O3*. Hence, one important advantage of our tool is that it can generate data to evaluate specific part of the flow, as opposed to random data generators (e.g., the TPC-DI data generator provided on the official website) who can only generate data agnostically of which part of the flow is being covered. This gives *Bijoux* not only a quality advantage, being able to evaluate the flow in greater granularity, but also a practical advantage, since the size of data that need to be generated can be significantly smaller. For instance, the TPC-DI data generator generates data for the FINWIRE file, only around $^1/_3$ of which are evaluated as *true* by the filter *RecType*=='*SEC*' and from them only around $^1/_3$ contains a company name instead of a number.

In order to generate realistic values for the company name fields, we used a catalog of company names that we found online [5] and we used *Bijoux* to generate data not only for the attributes that have been mentioned above, but for all of the attributes of the schemata of the involved data sources as defined in the TPC-DI documentation, so as to measure more accurate time results.

For each flow, we generated data of different size in order to evaluate how their performance can scale with respect to input data size, as shown in the below table, where we can see the number of rows for each data source for the three different scale factors (*SF*).

Data source →	FINWIRE	DimCompany	Alt_DS (for *Flow_B*)
SF_A	4000	4000	60000
SF_B	8000	8000	60000
SF_C	16000	16000	60000

For these experiments, for each flow we assumed selectivities that would guarantee the matching of all the rows in *FINWIRE* with rows in *DimCompany* and the results can be seen in Figure 4.10 For *Flow_C*.

As we expected, the results show an overhead in performance imposed by the data cleaning operations. It was also intuitive to expect that the lookup in the dictionary (*Flow_B*) would impose greater overhead than the string alterations (*Flow_C*). Nevertheless, some interesting finding that was not obvious is that as input data scale in size, the overhed of *Flow_B* appears to come closer and closer to the overhed of (*Flow_C*), which appears to become greater as input data size grows. We should notice at this point that our results regard the performance and

[4] `http://www.pentaho.com/product/data-integration`
[5] `https://www.sec.gov/rules/other/4-460list.htm`

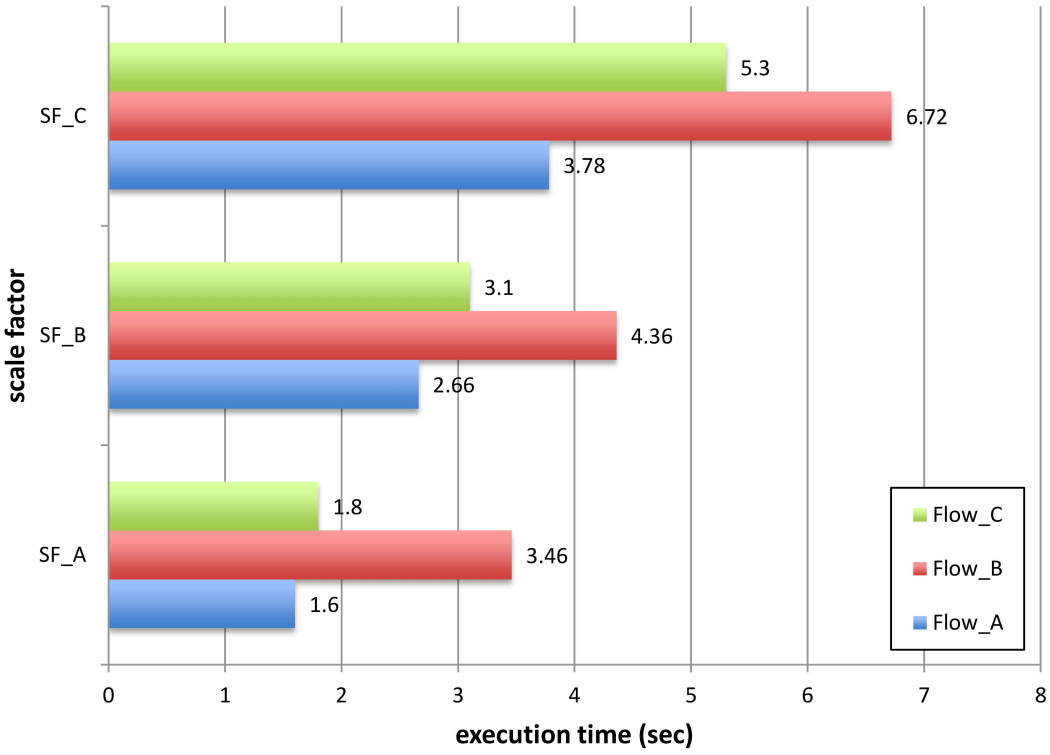


Figure 4.10: Performance evaluation of the flows using different scale factors

scalability of a specific part of the flow – not the complete flow in general – which is a unique advantage of our approach, especially in cases of dealing with bottlenecks.

Consequently, we conducted experiments assuming different levels of input data dirtiness, by setting the selectivity of the different join operations for the different flows. The scenario we intended to simulate was a predefined percentage of different types of data dirtiness. In this respect, we considered four different types of dirtiness:

1. Missing the abbreviation "inc." at the end of the company name (Type_I)

2. A word (e.g., company type abbreviation) exists at the end of the name when it should not (Type_II)

3. The ending of the company name is mistakenly in an extended format (e.g., "incorporated' ' instead of "inc.") (Type_III)

4. Miscellaneous that cannot be predicted (e.g., "corp." instead of "inc." or misspelled names) (Type_IV)

We assumed that *Flow_A* cannot handle any of these cases (i.e., dirty names as an input for the *FINWIRE* source will fail to be matched to data coming from *DimCompany*); that *Flow_B* can solve all the cases for Type_I and Type_III (i.e., there will be entries in the dictionary covering both of these types of dirtiness); and *Flow_C* can cover all the cases for Type_I and Type_II, because of the operation that it performs.

Thus, we generated data that were using real company names from the online catalog; we considered those names as the standard company names versions to generate data for the *DimCompany* source; and we indirectly introduced specified percentages of the different types of dirtiness, by setting a) the selectivities of the join operators and b) by manually generating entries in our dictionary (*Alt_DS*) that included all the names from the catalog together with their corresponding names manually transformed to Type_I and Type_II. The percentages of input data quality (IDQ) that were used for our experiments can be seen in the following table.

Dirtiness Type →	Type_I	Type_II	Type_III	Type_IV
IDQ1	0%	0%	0%	0%
IDQ2	1%	1%	3%	1%
IDQ3	2%	2%	6%	2%

In Figure 4.11, we show how the performance of *Flow_B* scales with respect to different scale factors and data quality of input data. What is interesting about those results, is that the flow appears to be performing better when the levels of dirtiness of the input data are higher. This might appear counter-intuitive, but a possible explanation could be that less data (i.e., fewer rows) actually reach the extraction operation, keeping in mind that read/write operations are very costly for ETL flows.

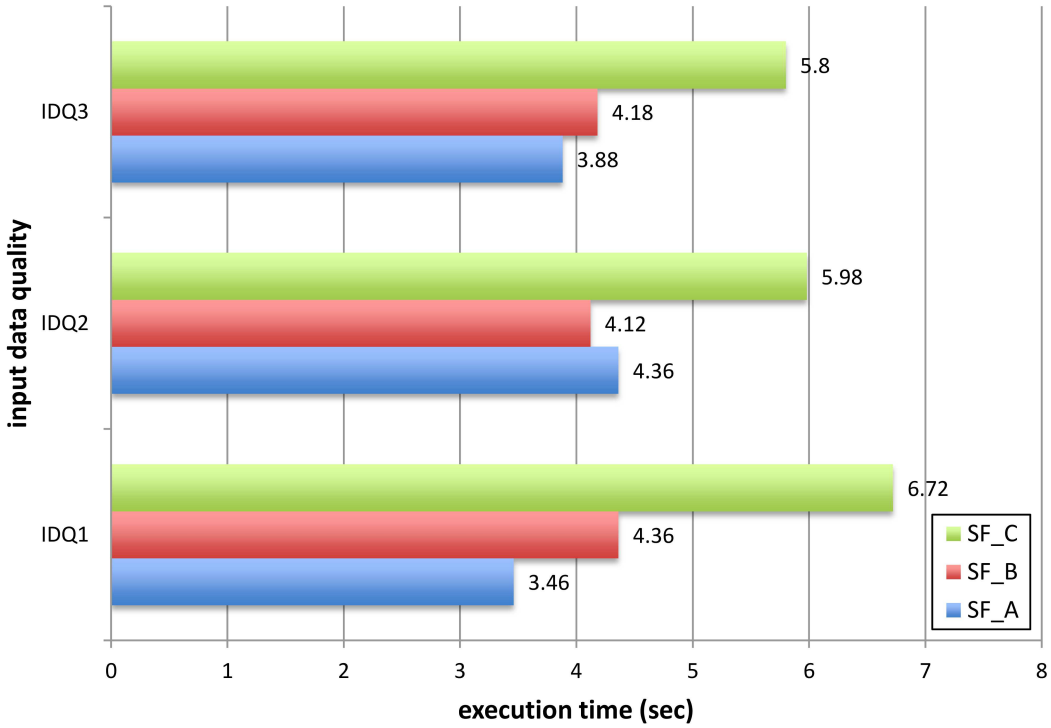


Figure 4.11: Performance evaluation of *Flow_B* using different levels of input data quality

4.3.2 Evaluating the data quality of alternative ETL flows

In the above-mentioned experiments, we evaluated the time performance of different flows, assuming that both data quality levels and data dirtiness characterization were a given. However, in order to evaluate an ETL flow with respect to the quality of the data cleaning that it can provide, it is not sufficient to only evaluate the time performance of different data cleaning options. To this end, in the second set of experiments, our goal was to evaluate which data cleaning option would produce the lowest levels of data incompleteness in the output data of the flow (*DimSecurity* table), using realistic datasets. In this respect, we used the company names from our catalog and for each of them we prepared a query to scrap the Freebase online database[6] and retrieve data about the company name and the known aliases of those names. Consequently, starting from 940 unique company names of our catalog, we were able to construct a dictionary that contained 2520 entries, each containing an alias of a company name and its corresponding standard name. We then used this dictionary as our Alt_DS dictionary; the standard names to populate the *DimCompany* table; and the names as they were on the catalog to populate the FINWIRE file.

Using *Bijoux* , we generated data that used *Flow_A* semantics in order to pass through the part of the flow that was of our interest and the dictionaries as mentioned above to generate realistic data. Despite the fact that it might appear as if the use of dictionaries devalues the use of our algorithm, in fact this is one strength of our approach — that it can be configured to generate data with different degrees of freedom, based on the constraints defined both by the flow semantics and the user. Therefore, it is possible to conduct such analysis, using a hybrid approach and evaluating the flows based on realistic data. The contribution of our algorithm in this case is to generate, on one hand all the data for the different fields of the schemata that are required for the flow execution and to make sure, on the other hand that the generated rows will cover specific parts of the flow.

After executing Flow_B and Flow_C with these input data, we used the following measure for data completeness:
$DI = \%\_of\_missing\_entities\_from\_their\_appropriate\_storage$ (Simitsis et al., 2009a)

The results for the two flows were the following:
$DI_{Flow\_B} = \frac{56}{940} * 100 \approx 6\%$

$DI_{Flow\_B} = \frac{726}{940} * 100 \approx 77\%$

According to these results, we can see a clear advantage of Flow_B regarding the data quality that it provides, suggesting that the performance overhead that it introduces, combined with potential cost of obtaining and maintaining a dictionary, might be worth undertaking, if data completeness is a goal of high priority.

We have explained above how the parametrization of our input data generation enables the evaluation of an ETL process and various design alterations over it, with respect to data quality and performance. Essentially, alternative implementations for the same ETL can be simulated using different variations of the data generation properties and the measured quality characteristics will indicate the best models, as well as how they can scale with respect not only

[6] `https://www.freebase.com/`

to data size but also to data quality of the input data. Similarly, other quality characteristics can be considered, like *reliability* and *recoverability*, by adjusting the percentage of input data that result to exceptions and the selectivity of exception handling operations. In addition, we have shown through our examples how data properties in the input sources can guide the selection between alternative ETL flows during design time.

4.4 BIJOUX PERFORMANCE EVALUATION

In this section, we report the experimental findings, after scrutinizing different performance parameters of *Bijoux*, by using the prototype that implements its functionalities.

We first introduce the architecture of a prototype system that implements the functionality of the *Bijoux* algorithm.

Input. The main input of the *Bijoux* framework is an ETL process. As we previously discussed, we consider that ETL processes are provided in the logical (platform-independent) form, following previously defined formalization (see Section 4.1.2). Users can also provide various parameters (see Figure 4.5) that can lead the process of data generation, which can refer to specific fields (e.g., *field distribution*), operations (e.g., *operation selectivity*) or general data generation parameters (e.g., *scale factors*).

Output. The output of our framework is the collection of datasets generated for each input data store of the ETL process. These datasets are generated to satisfy the constraints extracted from the flow, as well as the parameters provided by the users for the process description (i.e., distribution, operation selectivity, load size).

Bijoux architecture. The *Bijoux*'s prototype is modular and based on a layered architecture, as shown in Figure 4.12. The four main layers implement the core functionality of the *Bijoux* algorithm (i.e., *graph analysis*, *semantics extraction*, *model analysis*, and *data generation*), while the additional bottom layer is responsible for importing ETL flows from corresponding files and can be externally provided and plugged to our framework (e.g., flow import plugin (Jovanovic et al., 2014)). We further discuss all the layers in more detail.

- The bottom layer (*Model Parsing*) of the framework is responsible for parsing the model of the ETL process (*Parser* component) from the given logical representation of the flow (e.g., XML), and importing a DAG representation for the process inside the framework. In general, the *Model Parsing* layer can be extended with external parser plugins for handling different logical representations of an ETL process (e.g., (Wilkinson et al., 2010; Jovanovic et al., 2014)). This layer also includes a *Validator* component to ensure syntactic, schematic and logical (e.g., cycle detection) correctness of the imported models.

- The *Graph Analysis* layer analyzes the DAG representation of the ETL flow model. Thus, it is responsible for identifying and modeling all the ETL flow paths (*Path Enumerator* component; see Algorithm 1), as well as constructing all their possible combinations (*Path Combinator* component).

- The *Semantics Extraction* layer extracts relevant information needed to process the ETL flow. The information extracted in this layer (from the *Constraints Semantics Extractor*

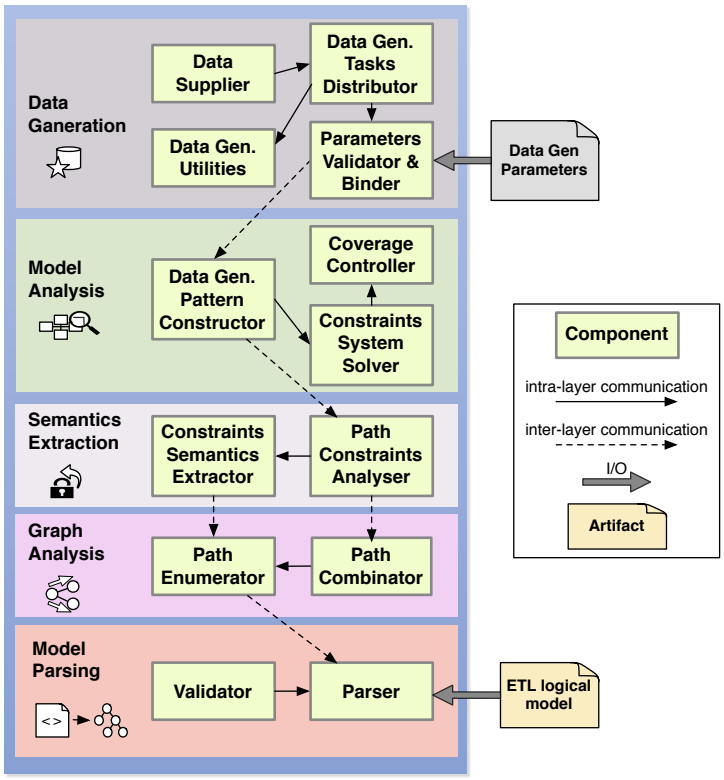


Figure 4.12: Bijoux prototype architecture

component) includes information about input datasets, operation semantics, order of operations, schema changes, and other parameters for data generation. This layer is also responsible for modeling constraints grouped by path (Path Constraints Analyzer; see Algorithm 2) to provide the required constructs for feasibility analysis and the construction of a data generation pattern to the layer above (Model Analysis).

- *Model Analysis* layer realizes the construction of a data generation pattern (*Data Gen. Pattern Constructor* component) that computes for each field (i.e., attribute), in each table, the ranges of values according to the extracted semantics of operations and their positioning within paths and path combinations. To this end, this layer includes the *Coverage Controller* component for implementing such analysis according to the set coverage goal (i.e., path coverage, flow coverage). In addition, it includes the *Constraints System Solver* component, which solves the systems of gathered constraints (e.g., system of logical predicates and equations over specified attributes) and returns the computed restrictions over the ranges.

- *Data Generation* layer controls the data generation stage according to the constraints (i.e., data generation patterns) extracted and analyzed in the previous layer, as well as the Data Gen. Parameters provided externally (e.g., distribution, selectivity). The *Parameters Validator & Binder* component binds the externally provided parameters to the ETL model

and ensures their compliance with the data generation patterns, if it is possible. The *Data Gen. Tasks Distributor* component is responsible for managing the generation of data in a distributed fashion, where different threads can handle the data generation for different (pairs of) attributes, taking as input the computed ranges and properties (e.g., *generate 1000 values of normally distributed integers where 80% of them are lower than "10"*). For that purpose, it utilizes the *Data Gen. Utilities* component, that exploits dictionaries and random number generation methods. Finally, the *Data Supplier* component outputs generated data in the form of files (e.g., CSV files).

4.4.1 Experimental setup

Here, we focused on testing both the functionality and correctness of the *Bijoux* algorithm discussed in Section 4.2, and different quality aspects, i.e., data generation overhead (*performance*) wrt. the growing complexity of the ETL model. The reason that we do not additionally test those quality aspects wrt. input load sizes is that such analysis is irrelevant according to the *Bijoux* algorithm. The output of the analysis phase is a set of ranges and data generation parameters for each attribute. Hence, the actual data generation phase does not depend on the efficiency of the proposed algorithm, but instead can be realized in an obvious and distributed fashion. Thus, we present our results from experiments that span across the phases of the algorithm up until the generation of ranges for each attribute. We performed the performance testing considering several ETL test cases, which we describe in what follows.

Our experiments were carried under an OS X 64-bit machine, Processor Intel Core i5, 1.7 GHz and 4GB of DDR3 RAM. The test cases consider a subset of ETL operations, i.e., *Input DataStore, Join, Filter, Router, UDF, Aggregation* and *Output DataStore*. Based on the TPC-H benchmark[7], our basic scenario is an ETL process, which extracts data from a source relational database (TPC-H DB) and after processing, loads data to a data warehouse (DW) and can be described by the following query: *Load in the DW all the suppliers in Europe together with their information (phones, addresses etc.), sorted on their revenue and separated by their account balance (either low or high)*, as can be seen in Fig. 4.13.

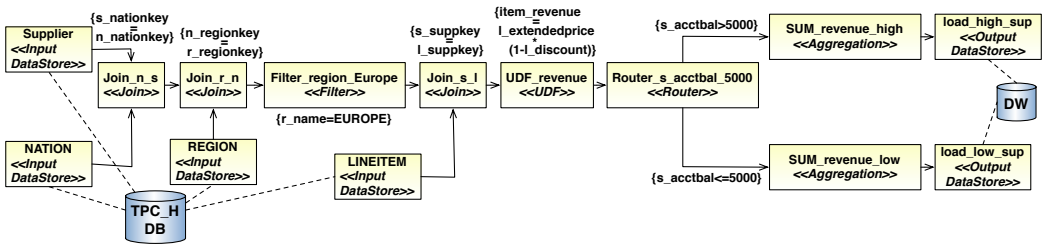


Figure 4.13: Basic scenario ETL process for experiments

The tables that are used from the source database are *Supplier*, *Nation*, *Region* and *Lineitem*. After *Supplier* entries have been filtered to keep only suppliers in Europe, the revenue for each supplier is calculated based on the supplied lineitems and subsequently, they are sorted on

[7]http://www.tpc.org/tpch/

revenue, separated by their account balance and loaded to different tables in the DW. Starting from the basic scenario, we use *POIESIS* (Theodorou et al., 2015), a tool for ETL Process redesign that allows for the automatic addition of flow patterns on an ETL model. Thus, we create other, more complex, synthetic ETL flows. The motivation for using tools for automatic ETL flow generation stems from the fact that obtaining real world ETL flows covering different scenarios with different complexity and load sizes is hard and often impossible.

Scenarios creation. Starting from this basic scenario, we create more complex ETL flows by adding additional operations, i.e., *Join*, *Filter*, *Input DataStore*, *Project* in various (random) positions on the original flow. We add two different Flow Component Patterns (FCP) (Theodorou et al., 2015) on the initial ETL flow in different cardinalities and combinations. The first pattern — *Join* — adds 3 operations every time it is applied on a flow: one Input DataStore, one Join and one Project operation in order to guarantee matching schemata; the second pattern — *Filter* — adds one Filter operation with a random (inequality) condition on a random numerical field (i.e., attribute).

We iteratively create 5 cases of different ETL flow complexities and observe the *Bijoux*'s execution time for these cases, starting from the basic ETL flow:

- *Case 1.* Basic ETL scenario, consisting of twenty-two (22) operations, as described above (before each join operation there exists also one joining key sorting operation which is not shown in Fig. 4.13, so that the flow is executable by most popular ETL engines).
- *Case 2.* ETL scenario consisting of 27 operations, starting from the basic one and adding an additional *Join* FCP and 2 *Filter* FCP to the flow.
- *Case 3.* ETL scenario consisting of 32 operations, starting from the basic one and adding 2 additional *Join* FCP and 4 *Filter* FCP to the flow.
- *Case 4.* ETL scenario consisting of 37 operations, starting from the basic one and adding 3 additional *Join* FCP and 6 *Filter* FCP to the flow.
- *Case 5.* ETL scenario consisting of 42 operations, starting from the basic one and adding 4 additional *Join* FCP and 8 *Filter* FCP to the flow.

4.4.2 Experimental results

We measure the average execution time of the path enumeration, extraction and analysis phase for the above 5 scenarios covering different ETL flow complexities.

Figure 4.14 illustrates the increase of execution time when moving from the simplest ETL scenario to a more complex one. As can be observed, execution time appears to follow a linear trend wrt. the number of operations of the ETL flow (i.e., flow complexity). This can be justified by the efficiency of our graph analysis algorithms and by the extensive use of indexing techniques (e.g., hash tables) to store computed properties for each operation and field, perhaps with a small overhead on memory usage. This result might appear contradictory, regarding the combinatorial part of our algorithm, computing and dealing with all possible path combinations. Despite the fact that it imposes factorial complexity, it is apparent that it does not constitute a performance issue for ETL flows of such complexity. To this end, the solution

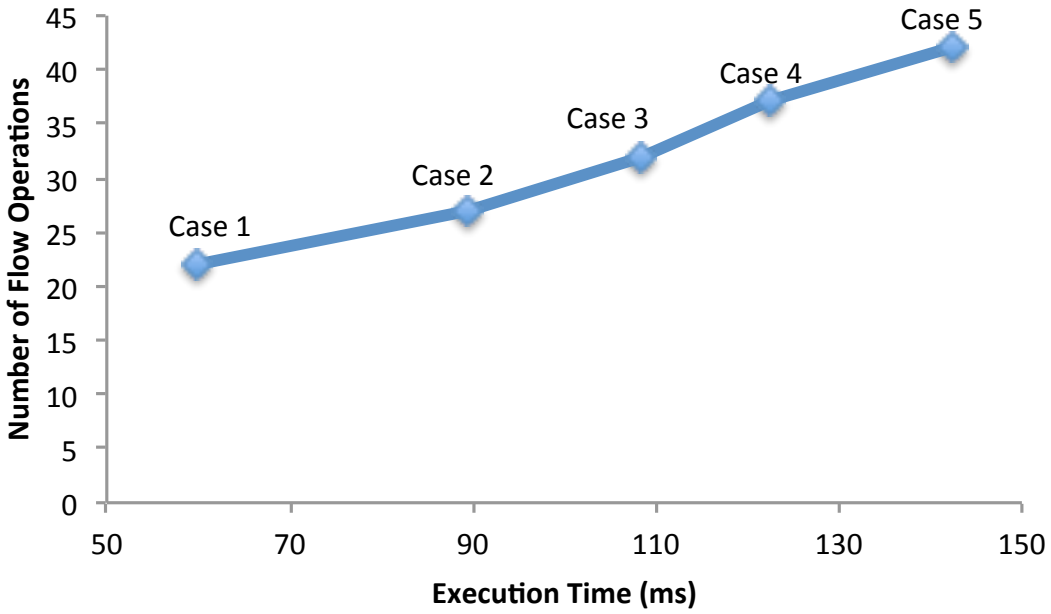


Figure 4.14: Linear trend of constraints extraction time wrt. the increasing number of operations (ETL flow complexity)

space is significantly reduced by i) our proposed greedy evaluation of the feasibility of a pattern every time it is updated and ii) by disregarding path combinations that do not comply to specific rules, e.g., *when considering path coverage, every input of a joining operation involved in any path of a path combination must be flowed (crossed by) at least one other path of that combination*.

4.5 CONCLUSIONS AND FUTURE WORK

In this chapter, we have studied the problem of synthetic data generation in the context of multi-objective evaluation of ETL processes. We proposed an ETL data generation framework (*Bijoux*), which aims at automating the parametrized data generation for evaluating different quality factors of ETL process models (e.g., data completeness, reliability, freshness, etc.), ensuring both accurate and efficient data delivery. In particular, *Bijoux* extracts and analyzes the semantics of data transformations and the constraints they imply over input data, to automatically generate testing datasets that cover the execution of the entire ETL flow. Thus, beside the semantics of ETL operations and the constraints they imply over input data, *Bijoux* takes into account different quality-related parameters, extracted or configured by an end-user, and guarantees that generated datasets fulfill the restrictions implied by these parameters (e.g., operation selectivity).

We have evaluated the feasibility and scalability of our approach by prototyping our data generation framework. The experimental results have shown a linear (but increasing) behavior of *Bijoux*'s overhead, which suggests that the algorithm is potentially scalable to accommodate more intensive tasks. At the same time, we have observed different optimization opportunities

to scale up the performance of *Bijoux*, especially considering larger volumes of generated data.

As an immediate future step, we plan on additionally validating and exploiting the functionality of this approach in the context of quality-driven ETL process design and tuning, as explained in our test case scenario.

FREQUENT PATTERNS IN ETL WORKFLOWS

The increasing dynamicity of Business Intelligence environments has driven the proposal of several approaches for the effective modeling of ETL processes, based on the conceptual abstraction of their operations. Apart from fostering automation and maintainability, such modeling also provides the building blocks to identify and represent frequently recurring patterns. Despite some existing work on classifying ETL components and functionality archetypes, the issue of systematically mining such patterns and their connection to quality attributes such as performance has not yet been addressed.

In this chapter, we address challenges in ETL automation, as described in Section 1.3. We illustrate how our pattern-based approach can address ETL complexity aspects and produce models that enable more granular ETL analysis and the means for automated quality enhancement. In this respect, we introduce our empirical approach for the identification of ETL structural patterns that are significant for different types of analysis. Based on the different types of ETL operations, we logically model the ETL workflows using labeled graphs and employ graph algorithms to identify candidate patterns and to recognize them on different workflows. For the identification phase, we use frequent subgraph discovery techniques and for the recognition phase, we introduce an algorithm that can perform very well for ETL flows and can scale for the cases of multiple and large flows

Our approach can be used for the (pre-)evaluation of alternative ETL workflows at design time, without the need for their execution or simulation, but solely by decomposing them to recognized patterns for which quality characteristics are measured during a learning phase. In addition, it can generate fine-grained cost models at the granularity level of patterns. It can also be used to classify reusable and well-defined ETL steps regardless of the implementation technology in order to i) automatically derive more understandable conceptual modeling and visualization of ETLs and ii) improve the reusability and reliability of ETL components in cases of alternative pattern implementations by exposing their characteristics. To showcase our approach, we implement a set of realistic ETL processes defined in the *TPC-DI benchmark*[1] and we show experimental results from applying our methodology on them.

[1] `http://www.tpc.org/tpcdi/`

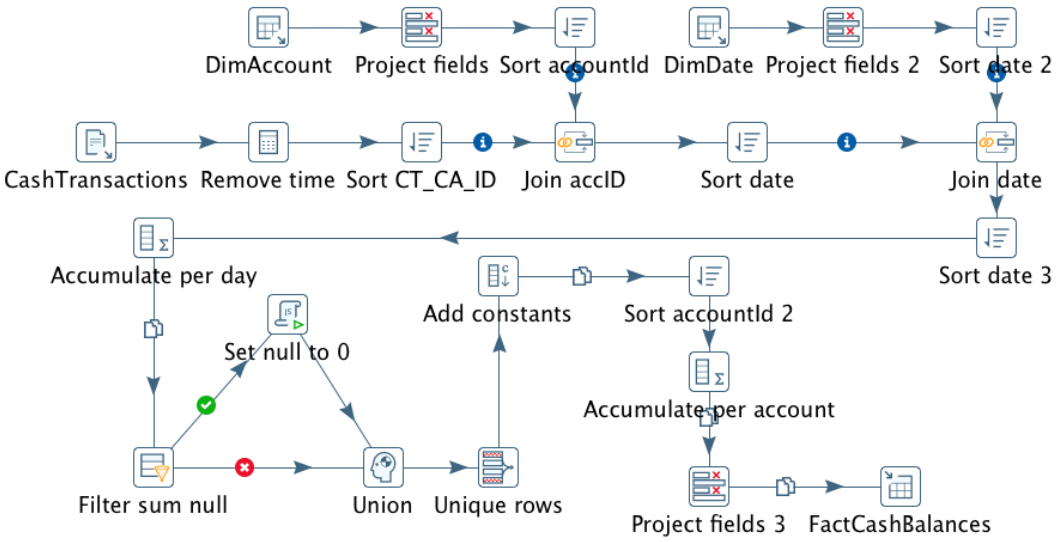


Figure 5.1: ETL flow example: TPC-DI FactCashBalances population

Running Example. From the TPC-DI domain, we adopt as our running toy example an ETL process (see Figure 5.1) that populates the *FactCashBalances* table during the *Historical Load* phase. In Figure 5.1, we show the logical view of the ETL process, as it is viewed from the implementation that we developed using the Pentaho Data Integration (PDI) open source tool[2]. The ETL process extracts data from a plain-text file in the Staging Area (*CashTransaction.txt*) and processes one of its fields to remove *time* and keep only *date* information. Consequently, data are joined with *DimAccount* table to obtain the corresponding keys for customers and accounts, after irrelevant fields from both sources have been projected out and data have been sorted. Similarly, data are joined with the *DimDate* dimension table to obtain corresponding date information and then they are aggregated on a daily basis. After null values have been replaced by zeros —only for the rows that contain null values— distinct rows are kept and constants (e.g., *effectiveDate*) are added to each row. Finally, data are aggregated per account and after keeping only relevant fields, they are loaded to the *FactCashBalances* table of the DW.

The main contributions of this chapter are as follows. We conduct a pattern-based analysis of ETL workflows, using the main control flow patterns from the Workflow Patterns Initiative[3] as a guide. Subsequently, we introduce a novel empirical approach for mining ETL structural patterns, using frequent subgraph discovery algorithms. Furthermore, we present an adaptation of the VF2 graph matching algorithm with optimizations to perform very well on ETL workflows. Finally, we present the most frequent ETL patterns, identified in 25 implemented ETL processes from the TPC-DI framework.

The rest of the chapter is structured as follows. Section 5.1 formalizes the models of ETL frequent patterns and sets the theoretical background for our approach. Section 5.2 presents two interesting use cases of our methodology and Section 5.3 illustrates our architecture and the algorithms that we employ for pattern mining and pattern recognition. Section 5.4 shows

[2]Full implementation available at: `https://github.com/AKartashoff/TPCDI-PDI/`
[3]`http://www.workflowpatterns.com/`

results from applying our methodology on implemented ETLs. Finally, Section 5.5 concludes the chapter.

5.1 ETL PATTERNS

Abstracting ETL processes on a logical level allows for the identification of recurring structures among the produced workflow models, that can indicate patterns. In this respect, we delve into the well-studied area of *workflow patterns* (WP) (Van Der Aalst et al., 2003) and examine their application on ETL workflows, in order to drive insights for the definition of our pattern model, which we subsequently present.

5.1.1 Workflow Patterns for ETL Flows

In this subsection, we present the basic workflow control-flow patterns (Van Der Aalst et al., 2003) that describe control-flow semantics commonly offered by various workflow management systems and we position them in the context of ETL flows.

ETL workflows are data-intensive flows where atomic tasks correspond to ETL operations, for which pipelining plays a crucial role and the smallest unit of data that can flow between them is a tuple. In this regard, we conceptually relate data-flow to control-flow by assuming that the control-flow dependencies refer to processing of data (i.e., tuples) by ETL operations. Of course, we should not exclude the case of blocking operations, where the unit of data that is expected by one operation in order to complete its execution, is a dataset, i.e., a set of tuples that all need to pass from one operation to the other. However, this simply implies some restrictions on the task completion which again produces some tuple(s) as an output to the succeeding operation(s), and thus does not change the generality of our approach. Hence, for one specific tuple, the task (i.e., ETL operation) activation is when this tuple enters this specific task for processing and the task completion is when this task has completed processing this specific tuple or the set of tuples in which it participates, if it is the case of a blocking operation. Following this concept, we perform the analysis below that defines the different workflow patterns in the context of ETL processes:

- Sequence
  Description: A task in a process is enabled after the completion of a preceding task in the same process.
  In ETL context: An ETL operation in a flow begins its execution right after the completion of the execution of a preceding operation in the same flow. The inputs of ETL operations are datasets and thus the smallest token that flows through the ETL is a tuple. In this regard, the Sequence pattern can be regarded in a tuple-by-tuple fashion, translating to: *one tuple will be processed by one ETL operation after its processing by a preceding operator has completed*. This definition of *sequence* is broad enough to cover both the cases i) when one operation does not have to process all tuples of a dataset before their processing by succeeding operations, allowing for pipelining and ii) when the processing semantics of the operation denote a blocking operator (e.g., *sorter* or *aggregator*).

- Parallel Split
  Description: The divergence of a branch into two or more parallel branches each of which execute concurrently.
  In ETL context: Two or more succeeding ETL operations begin their execution right after the completion of the execution of a preceding operation. The inputs of all these succeeding operations are identical datasets, coming as copies of the output of the preceding operation. For example, multiple ETL operations might perform the same processing of the same datasets at the same time implementing redundant execution. This case is useful i) for improving the reliability of the ETL process, so that even if some component fails, there are others executing identical tasks and the process does not need to terminate with errors and ii) for improving the correctness of the process by crosschecking the output results from identical tasks. Another example of parallel split in ETL processes is when (parts of) the same datasets need to be loaded to different output data sources (e.g., for loading surrogate keys correspondence).

- Synchronization
  Description: The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have completed execution.
  In ETL context: Two or more ETL operators are succeeded by the same ETL operator, which requires input from all of them in order to begin its execution. Datasets coming from the preceding operators are thus combined in some way by the succeeding operator. Examples of Synchronization within an ETL flow include different types of Joins where the left and right parts of the join operation come from different incoming flows.

- Exclusive Choice
  Description: The divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on a mechanism that can select one of the outgoing branches.
  In ETL context: Only one of two or more ETL operations that succeed an ETL operation begins its execution right after the completion of the execution of the preceding operation. The output of the preceding operation is directed (routed) to precisely one of the candidate succeeding operations, based on defined conditions and/or policies. As an example, different tuples can be routed to different operations based on condition evaluations or simply in a round robin fashion.

- Simple Merge
  Description: The convergence of two or more branches into a single subsequent branch such that each execution completion of an incoming branch results in the thread of control being passed to the subsequent branch.
  In ETL context: Two or more ETL operators are succeeded by the same ETL operator, which begins its execution every time it receives input from any of the preceding operators. Datasets coming from different operators do not need to be combined with each other, but have to conform to specific constraints for their unified processing by the same

ETL Operator	Related WP
Filter, Single Value Alteration, Project, Field Addition, Aggregation, Sort	Sequence
Router	Exclusive Choice
Splitter	Parallel Split
Join	Synchronization
Union	Simple Merge

Table 5.1: ETL operators

succeeding operation, e.g., common schema. An example of Simple Merge in ETL flows is the union of datasets coming from multiple different operations.

5.1.2 ETL Patterns model

Based on the basic WP for ETL, as they were defined above, we can derive a classification of ETL operators that depends on their control-flow semantics. If we further enrich this classification with the processing semantics of each operator, we obtain the classification of Table 5.1.

This basic set of operators constitutes the building blocks of a vast number of ETL processes logical models, making this number even greater if we consider that operators such as *Single Value Alteration* and *Field Addition* can be based on User Defined Functions (UDF) written in different programming languages. Taking under consideration the topologies that are formed by the way that different operators are connected inside an ETL flow, combined with the type of each operator, we derive structures that are candidate ETL patterns. In order for a candidate pattern to be considered an ETL pattern, it needs to satisfy the following two conditions:

1. It has to occur frequently, i.e., its support with respect to all the (examined) ETL flows has to exceed a threshold value s .

2. It has to be significant for the conducted analysis, i.e., it has to exhibit some important or differentiating behavior that can lead to its characterization, e.g., concerning its functional contribution to the complete workflow or its performance deviation from other parts of the workflow.

An ETL *operation* **o** is an atomic processing unit responsible for a single transformation over the input data, having specific processing semantics **ps** over the input data and a specific branch structure defined as the way it connects with neighboring nodes (i.e., operations). We logically model an ETL flow as a directed acyclic graph (DAG) consisting of a set of nodes, which are ETL operations (**O**), while the graph edges (**E**) represent the directed control flow among the nodes of the graph $(o_1 \prec o_2)$. Formally:
$o = \mathbf{ps}$
$ETL = (\mathbf{O}, \mathbf{E})$, such that:
$\forall e \in \mathbf{E} : \exists (o_1, o_2), o_1 \in \mathbf{O} \wedge o_2 \in \mathbf{O} \wedge o_1 \prec o_2$

This abstract definition of the ETL flow and its operations allows for the analysis of ETLs independent of the technologies that are used for their implementation and thus enables the

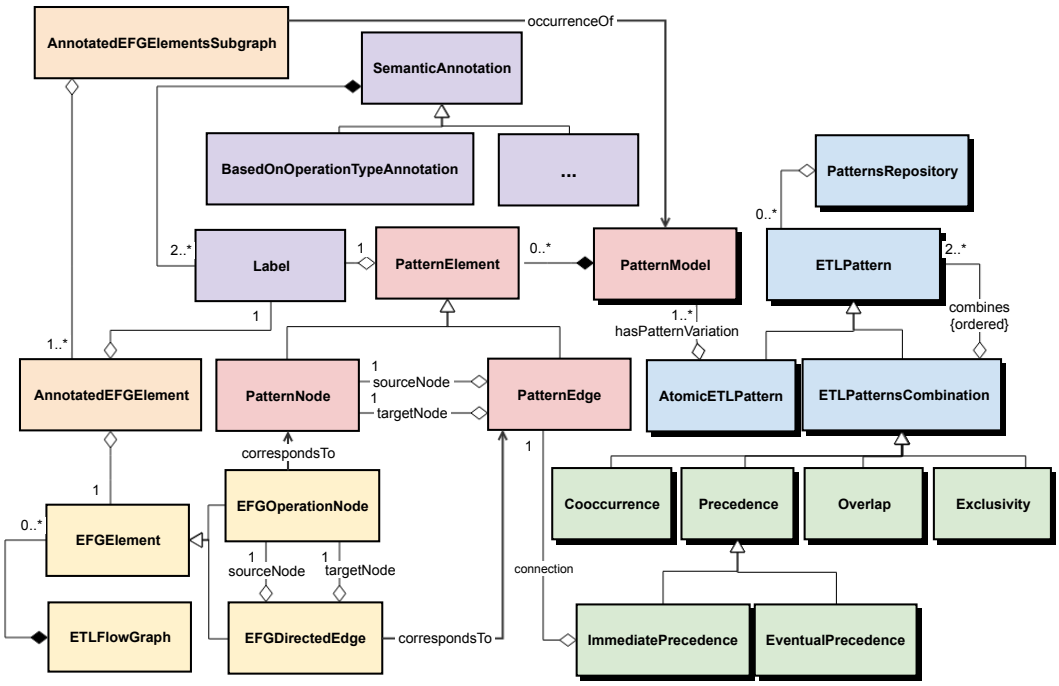


Figure 5.2: ETL Pattern Conceptual Model

mining of ETL patterns from a large number of ETL flows that can easily map to our model. Based on the characteristics of each operator **o**, it can be mapped to one *label* **l** from a predefined set $\mathbb{L}$ through the surjective function *label*.

A *Pattern Model* **PM** would then be a DAG where its nodes **PN** have a specific label **l** and a specific branch structure. Formally:
$pn = \mathbf{l}$
$PM = (\mathbf{PN}, \mathbf{E})$, such that:
$\forall e \in \mathbf{E} : \exists (pn_1, pn_2), pn_1 \in \mathbf{PN} \wedge pn_2 \in \mathbf{PN} \wedge pn_1 \prec pn_2$

We assume that only coherent structures make sense for our analysis and thus pattern models are *connected graphs*, i.e., graphs for which, if we ignore directionality there is a path from any of their nodes to any other node in the graph. We should note here that based on different analysis requirements there can be different definitions of mappings (i.e, mapping functions and sets of symbols), mapping one operator to one label. For instance, an operator can be mapped to a label, based solely on its input and output cardinality or based on its operation type. We have found that the latter case can produce useful results and hence that is the analysis that we use in our work. Thus, the labels that we use for our analysis are within a set $\mathbb{OT}$, where $\mathbb{OT} \subseteq \mathbb{L}$ and each element $ot \in \mathbb{OT}$ refers to the *operation type* of the operation and hence can take values from the classification of ETL operators in Table 5.1.

In Figure 5.2, we present the conceptual model of the pattern model and how it relates to the ETL flow. A more detailed description of Figure 5.2 is as follows: Using a mapping function, we can *semantically annotate* the elements (*EFGElements*) of an ETL Flow Graph, i.e., the edges

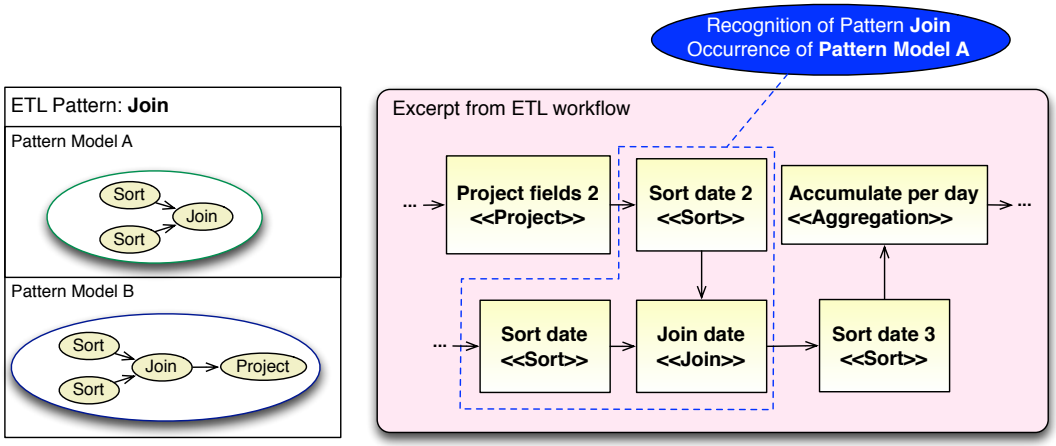


Figure 5.3: Pattern Model and Pattern Occurrence on an ETL workflow

(*EFGDirectedEdges*) and the nodes (*EFGOperationNodes*) by assigning them with corresponding labels and thus producing *Annotated EFG Elements*. A collection of such elements (i.e., a graph containing annotated edges and annotated nodes) can then form an *Annotated EFG Elements Subgraph* **aees** $\in \mathbb{AEES}$, which is an occurrence of a pattern model **pm** $\in \mathbb{PM}$ iff each and every element **e** from the subgraph corresponds to an element **pe** from the pattern model, through a bijective function *occurrenceOf*.

Notice that according to this model, edges between ETL operators can also be mapped to labels. This assumption has been made for completeness and because there can be some practical cases of ETL models where edges can be differentiated according to the manner that datasets flow from the source node to the target node (e.g., *copy-edges* can refer to the case when all outgoing edges from one source node copy the same datasets to all target nodes and *distr-edges* to the case when datasets are distributed among target nodes). However, for our analysis we consider all edges to be of the same type, i.e., that there is only one label to characterize all edges.

As mentioned above, for our analysis we use the semantic annotation (i.e., labeling) based on operation type (*BasedOnOperationTypeAnnotation*), but there can be different kinds of semantic annotation (i.e., subclasses of the class *SemanticAnnotation*), based on the conducted analysis. As is illustrated in Figure 5.2, the same pattern (*AtomicETLPattern*) can have variations, resulting to different corresponding pattern models in Figure 5.3). For instance, two different pattern models can correspond to the same ETL functionality, so if the analysis purpose is the clustering of operations based on their functionality, these two models will constitute variations of the same pattern. An example is illustrated in Figure 5.3), where we show how the ETL pattern *Join* can have two different pattern models (i.e., *Pattern Model A* and *Pattern Model B*) and how by finding the occurrence of one of these models (Pattern Model A) on the excerpt from our example ETL process from Figure 5.1, we can recognize it as an instance of the ETL pattern. Furthermore, a combination of two or more patterns (*ETLPatternsCombination*) can itself be a pattern. In this respect, two (or more) patterns can be combined in the following ways, forming a new pattern:

- *Overlap*: Patterns can overlap, with their pattern models sharing elements or with elements of one pattern model located inside the other. This case also includes *pattern nesting*, where one pattern is located inside the other.

- *Precedence*: One pattern is located (right) after the other.

- *Cooccurrence*: Both patterns occur in the same ETL flow.

- *Exclusivity*: Only one of the patterns can occur in the ETL flow and not the other(s).

It should be noticed that the participation of ETL patterns in ETL Pattern Combinations entails a concrete role for each pattern in the combination and this is denoted by the characterization of the *combines* association as *ordered*. Despite our approach allowing for the occurrence of overlapping patterns and patterns one after the other (i.e., precedence), we do not consider the case of the combinations themselves being patterns (i.e., we only consider *Atomic ETL Patterns*).

5.1.3 Frequent ETL Patterns

As mentioned above, one of the conditions for a candidate pattern to be considered a pattern is that its support has to exceed some predefined value. In other words, its corresponding pattern model(s) have to occur frequently over the entire set of examined ETL workflows. Since both the ETL workflows and the pattern models are represented using graphs, the problem of mining such patterns can be examined under the prism of *frequent subgraph mining*, which is a subclass of *frequent itemset discovery* (Salmenkivi, 2008), where the goal is to discover frequently occuring subgraphs within a set of graphs or a single large graph, with frequency of occurrence above a specified threshold value.

For the purpose of our analysis, we are not interested in frequent subgraphs that *always* appear inside other, bigger frequent subgraphs (i.e., pattern nesting). In this respect, we define a maximality condition: A frequent subgraph (SG1) is *maximal* when there exists no frequent subgraph (SG2) of bigger size, where (SG1) is a proper subgraph of (SG2). Formally:
$isMaximal(SG1) \iff isFrequent(SG1) \wedge \nexists SG2$ such that $isFrequent(SG2) \wedge SG1 \subset SG2$

On the contrary, we define *independent frequent subgraphs* as frequent subgraphs that occur at least once not nested inside the occurrence of another frequent subgraph. Formally:
$isIndependent(SG1) \iff isFrequent(SG1) \wedge \exists occurrence(SG1)$ such that ($\nexists SG2$ such that $isFrequent(SG2) \wedge occurrence(SG1) \subset occurrence(SG2)$)

Figure 5.4 shows the distinction of these two concepts through an example. The two numbers on each frequent subgraph respectively denote the total number of occurrences and the number of independent occurrences (i.e., not inside the occurrence of another frequent subgraph).

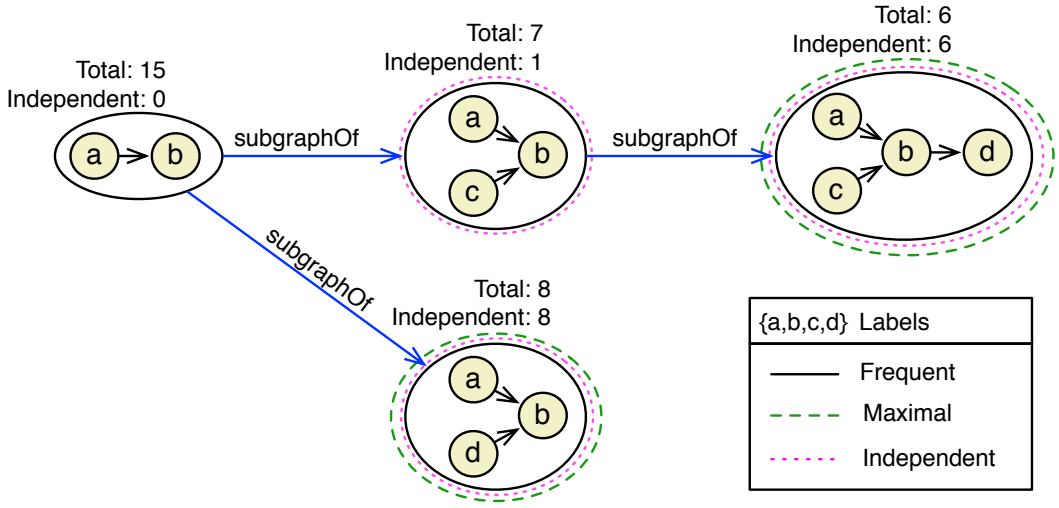


Figure 5.4: Maximal and Independent Frequent Subgraphs

5.2 ETL PATTERNS USE CASES

The identification of patterns within ETL processes and their definition and classification can be used in various ETL projects with contributions spanning from more efficient ETL quality analysis to more usable and reusable ETL models. In this section, we present two use cases that expose the value of using ETL flows patterns that are derived from our empirical approach.

5.2.1 Conceptual Representation of ETL Flows

The value of using conceptual representations of ETL processes has been recognized in several works (Wilkinson et al., 2010; Akkaoui et al., 2013; Oliveira and Belo, 2012), where the proposed modeling notation is BPMN, mainly because of its expressiveness and the support of modeling data artifacts and the data flow of the ETL process. These works have mostly focused on the advantages of a conceptual model during the design phase, for modeling abstractions of process functionalities and automating their translation to concrete implementations. Using our bottom-up approach, it is possible to work the other way round, identifying ad-hoc patterns on arbitrary ETL processes and thus populating libraries of such abstractions in the context of any specific business environment and used ETL technologies. It is then straightforward to conceptually model any ETL process in the same context, by decomposing its different parts to the identified patterns.

In Figure 5.5 we show an example of such a decomposition, translating the logical model of our running example (see Figure 5.1) from its logical model to a conceptual representation in BPMN. The patterns used are mined from ETL processes within the same domain and context, i.e., TPC-DI ETLs implemented using the Pentaho Data Integration tool. The conceptual representation is much more concise and understandable than the logical view and the translation is completely automated.

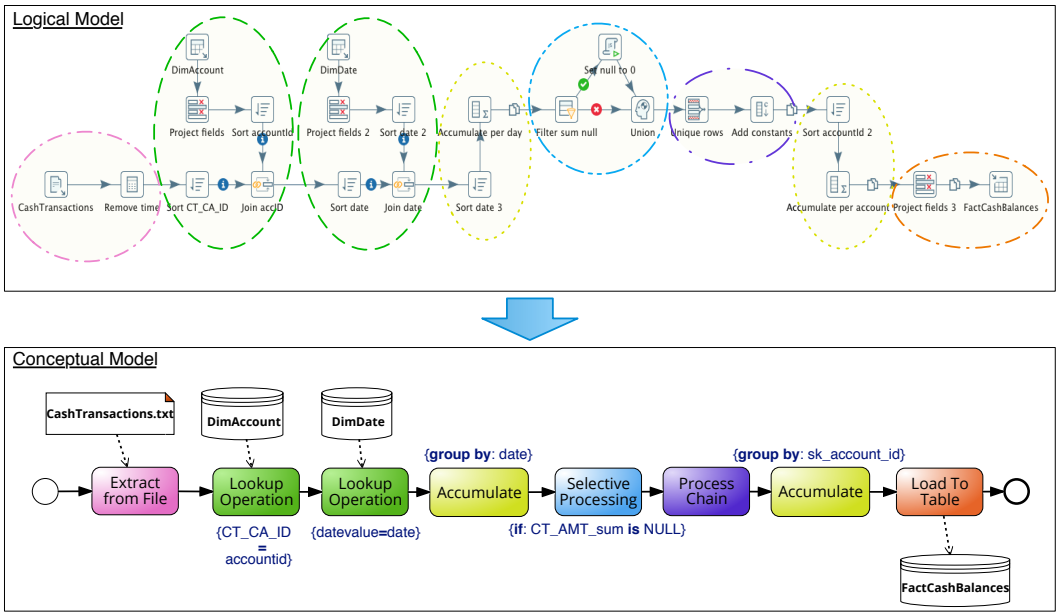


Figure 5.5: Example of translating logical representation of an ETL process to BPMN

5.2.2 Quality-based Analysis of ETL flows

One other interesting application of our bottom-up approach is the more granular evaluation of quality characteristics of ETL processes. In (Theodorou et al., 2016), we have gathered quality measures and metrics from literature and we have illustrated how they can be used to evaluate ETL processes with respect to different quality dimensions. Using our approach, such evaluation can take place in a more granular level than the complete ETL workflow —at the level of patterns. What is more, such evaluation can take place without the need for execution of the ETL workflow under test, but simply by statically examining its logical model and by recognizing pattern occurrences, similarly to Figure 5.5. During a learning phase, the average quality performance of different pattern models can be obtained from the isolated evaluation of their occurrences on a training set of ETLs. Subsequently, after pattern models have been characterized based on their performance, their occurrence on ETL models can signify the existence of parts of the ETL with corresponding performance implications. For instance, different parts of the ETL can be predicted as more, or less costly in terms of consumption of resources, creating a heatmap of the different parts of the ETL. This kind of analysis can also be used for the identification and avoidance of antipatterns (Smith and Williams, 2000) during the ETL design phase.

5.3 ARCHITECTURE

Our general approach architecture is depicted in Figure 5.6 and consists of two phases —the *learning phase* and the *recognition phase*. During the learning phase, a training set of ETL work-

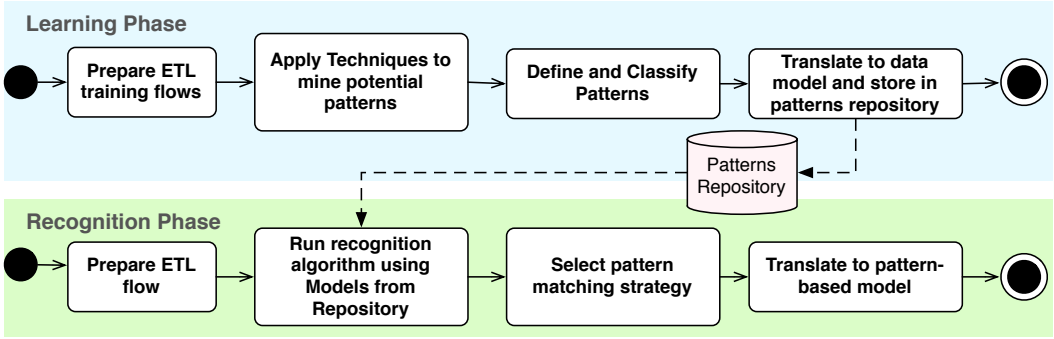


Figure 5.6: Process Architecture of ETL Workflow Patterns Analysis

flows is used to mine ETL patterns and store them in a *patterns repository*. The first step is the modeling of the ETL workflows as the ETL structures defined in Section 5.1. Subsequently, graph algorithms can be applied on the ETL structures to identify reoccurring structures with frequency above a specified support threshold. After the identification of the frequent structures, analysis takes place to define relevant structures as patterns and to classify them according to their functionality. It is during this step that some frequent structures might be dismissed after being considered irrelevant for the conducted analysis (e.g., non-independent patterns). Finally, ETL patterns are stored in a repository, after being translated to an appropriate data model. During the recognition phase, one ETL flow is modeled as an ETL structure and graph matching algorithms are executed to find occurrences of patterns, from the repository to the ETL structure. A pattern matching strategy is selected to disambiguate the cases where different patterns can be recognized on the same part of the ETL and subsequently, the ETL operations are mapped to corresponding pattern occurrences.

5.3.1 Pattern mining

The first step for mining patterns of interest is the identification of reoccurring structures over a set of ETL workflows. As mentioned above, we can view our problem as an application of *frequent itemset discovery*, where the goal is to discover frequently occuring "items" within a set of "baskets". Since we logically model ETL workflows as graphs, it is natural for this stage to use graph mining techniques, i.e., techniques for extracting statistically significant and useful knowledge from graph structures. Given a set of graph representations of ETL workflows, the reoccurring structures of interest are graphs, that occur as subgraphs of the initial graphs more frequently than a specified threshold percentage. Thus, we can employ the use of algorithms from the well studied area of *Frequent Subgraph Mining* (FSM). In (Jiang et al., 2013), there is a detailed review of this mature research area, including the main research challenges and the most interesting proposed solutions.

For our experiments, we decided to use the *FSG* algorithm (Kuramochi and Karypis, 2001), because of 1) its computational efficiency and 2) fast and reliable results from testing that we conducted using its available implementation[4]. This algorithm generates candidate frequent

[4] `http://glaros.dtc.umn.edu/gkhome/pafi/overview`

subgraphs in a bottom-up approach, starting with initial subgraphs of one edge and adding one edge at each step while checking for the frequency criterion. It performs significant pruning to the problem space while searching for patterns, taking advantage of the fact that *if a graph is frequent, then all of its subgraphs are also frequent*.

Two parameters can change the output of the algorithm. Firstly, the support threshold, i.e., the minimum number of ETL flows that need to contain a subgraph for it to be accounted as frequent, can vary. In addition, we can select whether we are interested only in *maximal* subgraphs (see Subsection 5.1.3).

After the identification of frequent patterns, the next step is their filtering in order to maintain only the patterns of some value. In this respect, a first filtering is performed by keeping only *independent* subgraphs (see Subsection 5.1.3). To this end, all the instances of all the frequent subgraphs are recognized within the initial set of ETL workflows, using the pattern recognition algorithm defined below (Subsection 5.3.2) and graph-subgraph relationships among these instances are analyzed. Subsequently, frequent subgraphs are classified as variations of ETL patterns, based on the conducted analysis. For instance, experts can classify these subgraphs according to their conceptual functionality, or performance evaluation can be conducted to classify subgraphs based on their isolated performance as compared to the performance of the complete ETL. The results of such analyses are then stored in a repository of ETL patterns.

5.3.2 Pattern recognition

Once a knowledge base of ETL patterns has been built, occurrences of those patterns can be recognized in any arbitrary ETL workflow. The workflow first needs to be transformed to its graph representation and then the task is reduced to finding a correspondence between the model (i.e., the ETL pattern model) and part(s) of the ETL workflow graph representation. Since both the model and the examined ETL are modeled as graphs, this is a typical use case for a graph matching algorithm. In (Lee et al., 2012), there is an interesting comparison between the implementation in a common codebase of five state-of-the-art subgraph isomorphism algorithms and in (Conte et al., 2004), there is a comprehensive review of different techniques that have been proposed for this NP-complete problem, the most popular being the Ullmann's algorithm (Ullmann, 1976) and the VF2 algorithm (Cordella et al., 2001). After studying these algorithms, we decided to adapt the VF2 algorithm with some optimizations (Algorithm 3), maintaining different data structures while searching for pattern model occurrences. Our algorithm can perform very well for graphs with the cheracteristics of ETL workflows — i) very small branching factors, ii) number of different *labels* comparable to the average graph size.

Taking under consideration the VF2 heuristic of adding to the search space only adjacent nodes while generating candidate matches, our algorithm iterates over the nodes of the pattern model in a breadth-first search (BFS) manner while at the same time matching them with nodes from the annotated EFG, that satisfy certain conditions (see Figure 5.7). The candidate node matches are searched only within adjacent (i.e., neighboring) nodes from the already matched nodes, which we call *frontier nodes* and for a specific candidate pattern match, if there is no adjacent node that satisfies the conditions, it is dismissed. New candidate pattern matches commence after every iteration, until the pattern model has been fully traversed or the set of

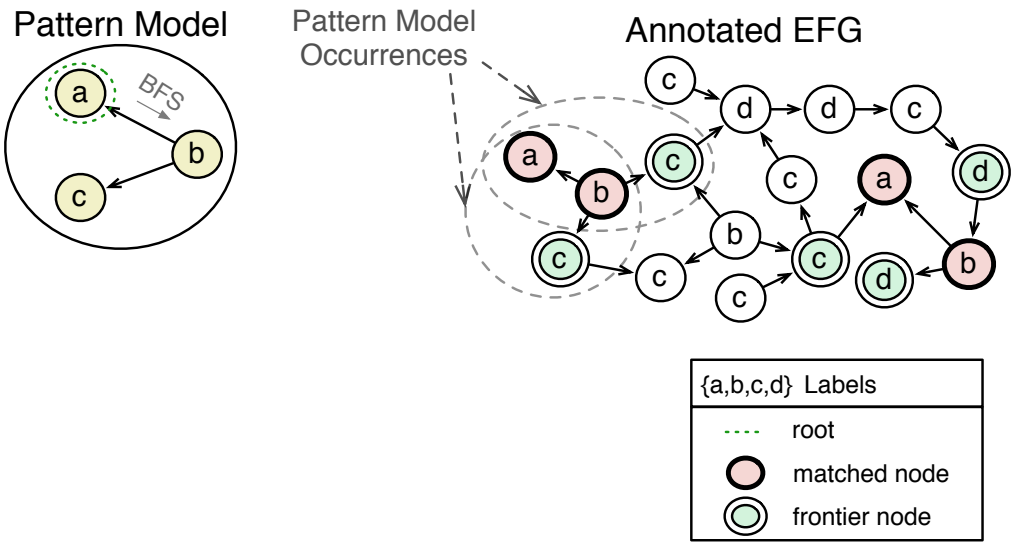


Figure 5.7: Execution of Find Pattern Model Occurrences Algorithm

candidate pattern matches is empty.

One practical heuristic that we use to speed up the execution of the algorithm by keeping the number of initial candidate node matches on the annotated EFG as small as possible, is the execution of preparation steps, through which the root of the BFS, (i.e., the node of the pattern model from which the iteration will start) is selected to be annotated with the label from the pattern model that is least frequently found on the annotated EFG. The EFG is traversed once and the number of occurrences of each label is stored in *count* —a HashMap object that maps labels to integers (i.e., labels' frequencies). One of the nodes of the pattern model that have the label with the minimum frequency in *count*, is selected as the subsequent BFS starting point (i.e., the root).

Given an **ETL**, a pattern model **PM** and its **root** (see for example the node with label *a* from the pattern model in Figure 5.7), Algorithm 3 returns a set $\mathbb{PO}$ of pattern occurrences, that contains all the occurrences of PM in ETL. Each element of this set consists of two parts: i) *m*, which is a set of matches, matching one node from ETL to one node from PM and ii) *f*, which is a set of nodes from the ETL, to maintain all the frontier nodes for one pattern occurrence, i.e., the search space for the subsequent iterations. First, the algorithm iterates over all the nodes of the ETL (Step 2) and finds all the nodes that have same properties as *root* (Step 3). All these nodes then act as the initial node matches around which pattern occurrences are searched (see for example the search space of the annotated EFG in Figure 5.7, where pattern occurrences are searched only around the two nodes with label *a*). The matches are added to the set *m* (Step 4) and the neighbors of these nodes (i.e., their adjacent nodes in the ETL) are added to the set *f* that maintains the *frontier* for the search around each candidate occurrence (Step 5). Candidate pattern occurrences with these characteristics are added to the set $\mathbb{PO}$. Subsequently, the pattern model is traversed in a BFS order (Step 7), ignoring the directionality of the graph and

Algorithm 3 Find All Pattern Model Occurrences

Input: ETL, PM, root ▷ *root* is a node from the PM
Output: $\mathbb{PO}$

1: $\mathbb{PO}$, $\mathbb{PO}_{\not\models}$,m,f $\leftarrow \emptyset$; candOc $\leftarrow$ []; ▷ initializations
2: **for each** $o_i \in$ ETL **do** ▷ iterate nodes of ETL
3: **if** (sameProperties(o_i,root)) **then** ▷ check for match
4: m $\leftarrow$ {[o_i,root]}; ▷ add to set of matches for specific occurrence
5: f $\leftarrow$ *neighbors*(o_i); ▷ add adjacent nodes to the frontier
6: $\mathbb{PO}$.*add*([m,f]); ▷ add candidate occurrence to $\mathbb{PO}$
7: **for each** $pn_n \in$ BFS_order(PM,root) **do** ▷ iterate nodes of PM
8: **for each** $co_l \in \mathbb{PO}$ **do** ▷ iterate candidate occurrences in $\mathbb{PO}$
9: **for each** $o_m \in co_l.f$ **do** ▷ iterate nodes from frontier
10: **if** (sameProperties(o_m,pn_n)) **then** ▷ check for match
11: m $\leftarrow co_l.m \cup \{[o_m, pn_n]\}$ ▷ add to matches
12: f $\leftarrow (co_l.f \setminus \{o_m\}) \cup$ *um-neighbors*(o_m); ▷ update frontier
13: $\mathbb{PO}_{\not\models}$.*add*([m,f]); ▷ add updated occurrence to $\mathbb{PO}_{\not\models}$
14: $\mathbb{PO} \leftarrow \mathbb{PO}_{\not\models}$; $\mathbb{PO}_{\not\models} \leftarrow \emptyset$; ▷ copy and initialize for next iteration
15: **return** $\mathbb{PO}$;

excluding the root since it has already been matched. At this point, we use a second heuristic to speed up the execution of the algorithm: The order by which the pattern model is traversed depends on the frequency of its labels on the ETL, which we have already collected in the *count* object. Thus, when a pattern node has multiple unvisited neighbors (i.e., adjacent nodes that have not yet been visited by the BFS), the order by which they are visited depends on their label —from least to most frequent label in the ETL. For each candidate occurrence, each node from its frontier is checked for having the same properties with the current node from the pattern model (Step 10). If it does, then the matches and the frontier of the current candidate occurrence are updated (Steps 11 and 12). Regarding the frontier update, the *um-neighbors*, i.e., the adjacent nodes to the matched node from the ETL that have not already been matched to a pattern node, are added to the existing nodes in the frontier and the matched node is removed from the frontier (Step 12). Subsequently, a new candidate occurrence with these updated parts is added to a set $\mathbb{PO}_{\not\models}$. In every iteration this set replaces the old set $\mathbb{PO}$ that gets initialized to the empty set (Step 14). We should note that despite the pattern model graph being traversed ignoring its directionality, when the check for same properties between a pattern model node and an ETL node takes place, the directionality is taken under consideration.

Algorithm complexity

Although subgraph isomorphism is well-known NP-hard problem (Shamir and Tsur, 1997), our algorithm can practically execute very fast because of the particularities of ETLs and the heuristics that we use. The adjacent nodes for each node are maintained inside two hashmap objects that map labels to nodes —one hashmap for the *incoming* adjacent nodes and one for the *outgoing* adjacent nodes (Sakr and Pardede, 2011). If n is the size of the ETL graph, then

each of these two objects is of maximum size $(n-1)$ for each ETL node (a node cannot be adjacent to itself and a node cannot be found in two different buckets of the hashmap because it only has one label), thus the space complexity of the algorithm is: $2*(n-1)*n = \mathcal{O}(n^2)$.

When it comes to the time complexity, due to the use of the hashmap objects, the check for same properties between the nodes of the pattern model and the ETL can take place in constant time *t*. Thus, if *m* is the size of the PM, the time complexity is: $t * \sum_{i=1}^{m} N_i$, where N_i is the running number of candidate occurrences (i.e., the size of $\mathbb{PO}$, see Step 8 of Algorithm 3) during each iteration of the BFS. We should note that N_1 is the number of initial candidate occurrences which is determined by the selection of the pattern model *root* element. The growth rate $\frac{N_{k+1}}{N_k}$ of the solution space depends on the fanout of the nodes of the ETL graph on one hand; and on the distribution of the different labels on the ETL nodes, on the other. In other words, the search space grows by being multiplied by the number of neighbors of each node in the candidate occurrence frontier, which is being matched (see Step 12 of Algorithm 3), but it also shrinks at the same time by pruning nodes that do not match the corresponding node from the PM. In the worst case of the ETL being a clique where all the labels of the pattern model and all the labels of the ETL graph are the same one label, there will be no pruning and thus, every time a new node from the pattern model is visited, the number of candidate occurrences will multiply by $(n-p)$ where *p* is the number of already matched nodes, until all *m* nodes are visited. Thus, in the worst case the total number of candidate occurrences during the last iteration will be: $N_m = \prod_{i=1}^{m}(n-i) = \mathcal{O}(\frac{(n-1)!}{(n-1-m)!})$. However, according to our experience with implementing ETL workflows from the TPC-DI benchmark, this is hardly a realistic case for ETL graphs, where the branching factor is close to 1. In addition, the existence of a number of different labels in real ETL graphs, guarantees that a lot of pruning takes place, especially in the common case where no (unmatched) node of specific label is adjacent to a candidate occurrence, which can very easily be checked, with the bucket corresponding to this label being empty. Furthermore, the size *m* of the meaningful pattern models is usually very small (< 10).

5.4 EXPERIMENTAL RESULTS

In this section, we show results obtained from the application of the algorithms to 25 ETL processes from the TPC-DI benchmark that that we implemented using the Pentaho Data Integration open source tool.

5.4.1 Mined ETL Patterns

In this subsection, we present our results from mining frequent patterns during the *Learning Phase* of our approach. To this end, we used the *FSG* algorithm (Kuramochi and Karypis, 2001) on the graph representation of the 25 TPC-DI ETLs. In Figure 5.8, we show the number of frequent patterns of different size (i.e., number of edges) that we obtain, using different values for support. It should be noticed that the FSG algorithm executed in less than 2 $msec$ for all these cases. As expected, since we are not imposing the maximality constraint, as the support increases, the number of identified patterns decreases. In other words, all the patterns that are identified with some support *s* will also be identified with any other smaller support, plus ad-

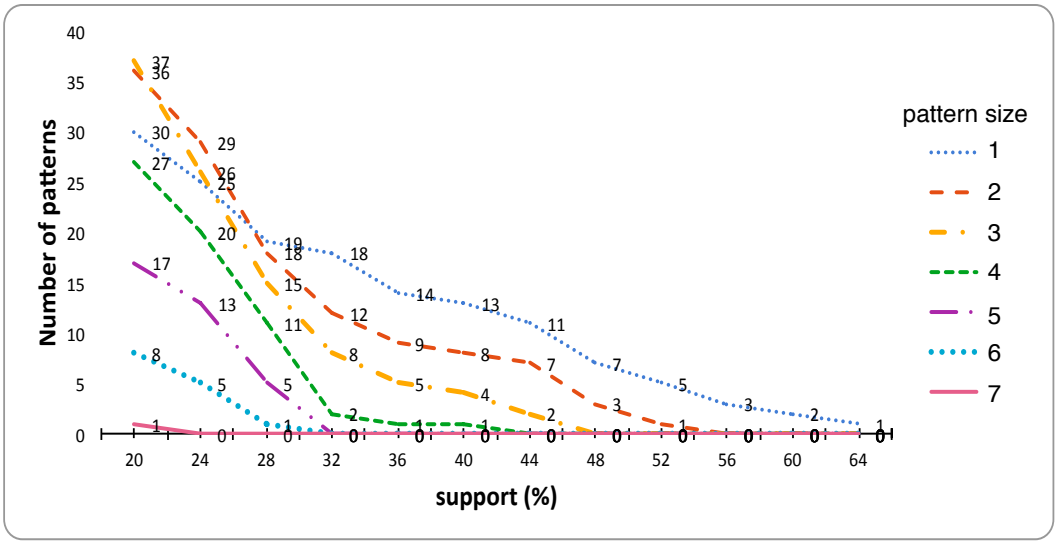


Figure 5.8: Number of (non-maximal) frequent patterns identified for different support values and for different pattern sizes

ditional patterns that do not satisfy the frequency criterion for s. We can also infer from Figure 5.8 that in general, as the size of the patterns decreases, the number of identified patterns increases. However, as can be seen, there are some noticeable exceptions to this rule. The curve for pattern size 1 crosses both with the curve for pattern size 2 and with the curve for pattern size 3. The reason is that the same pattern of size 1 can be a subgraph of two or more patterns of size 2. As an example, let us consider a set of three labels $\{a, b, c\}$ and the identification of the following frequent patterns of size 2: i) $[a - a - b]$, ii) $[a - b - a]$, iii) $[a - a - c]$ and iv) $[a - c - a]$. Each subgraph of these *four* frequent patterns will also be a frequent pattern for the same support s. However, there are only *three* distinct subgraphs of size 1 among these patterns —i) $[a - a]$, ii) $[a - b]$ and iii) $[a - c]$— and it is not necessary that there exist more frequent patterns of size 1 with these labels. The same explanation can be given for the case of the curve for pattern size 2 crossing with the curve for pattern size 3. The reason that the rule holds for greater s values, is that this explained behavior is outgrown by the tendency of larger patterns to be more difficult to find frequently. Finally, we can observe that beyond some support value, there is no frequent pattern identified.

In Figure 5.9, we show for different support values, the coverage of all the ETL workflows from the patterns identified, i.e., the percentage of ETL operations that take part in pattern model occurrences. As we can observe, the coverage decreases as the support value increases, which is an expected behavior since the overall number of identified patterns decreases as well (see Figure 5.8). This decrease appears to be non-linear and especially beyond some value s ($\approx 45\%$) for which coverage is $\approx 80\%$, it appears to decrease faster and faster as support increases. Another interesting observation is that for a small value of s, coverage reaches a very high value, above 93%. This validates our claim that it is possible to automatically translate, if not the complete, at least a very large part of an ETL workflow to its conceptual representation, as we have explained in Subsection 5.2.1.

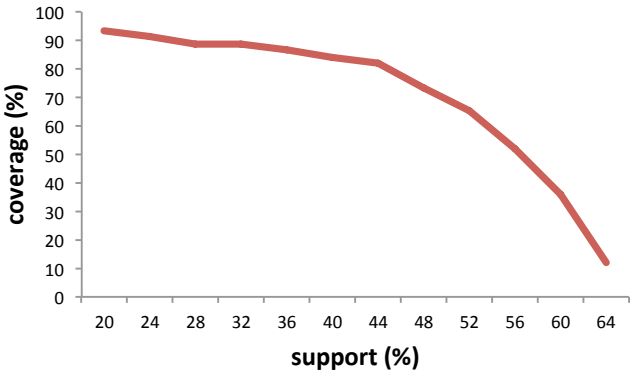


Figure 5.9: Coverage of ETL workflows for different support values

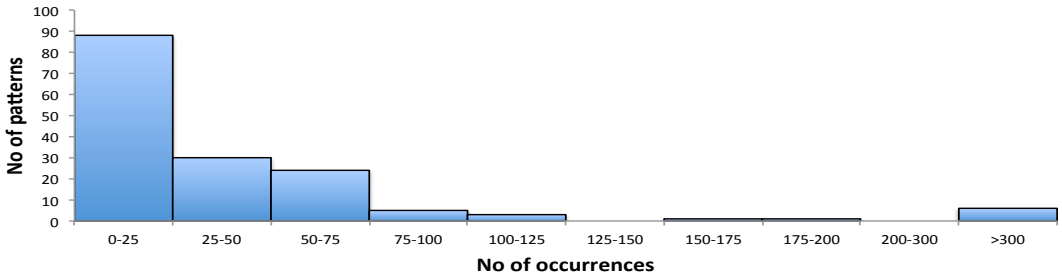


Figure 5.10: Number of patterns w.r.t. their number of occurrences

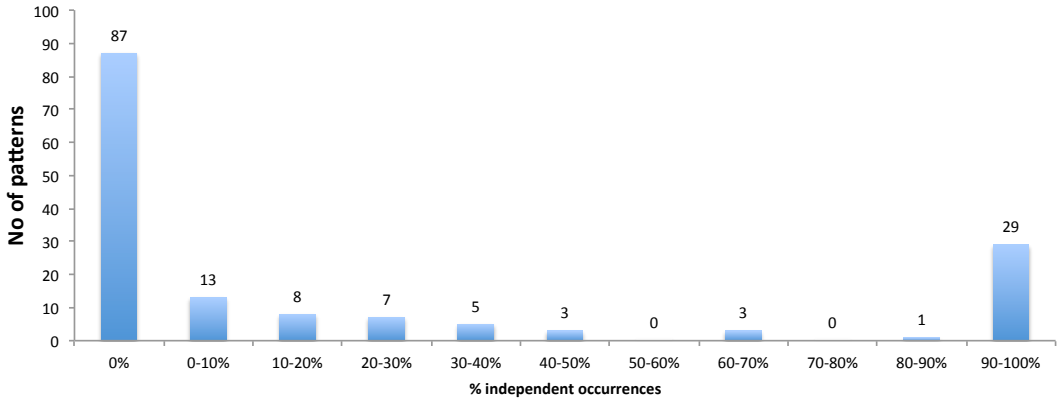


Figure 5.11: Number of patterns w.r.t. their frequency of independent occurrences

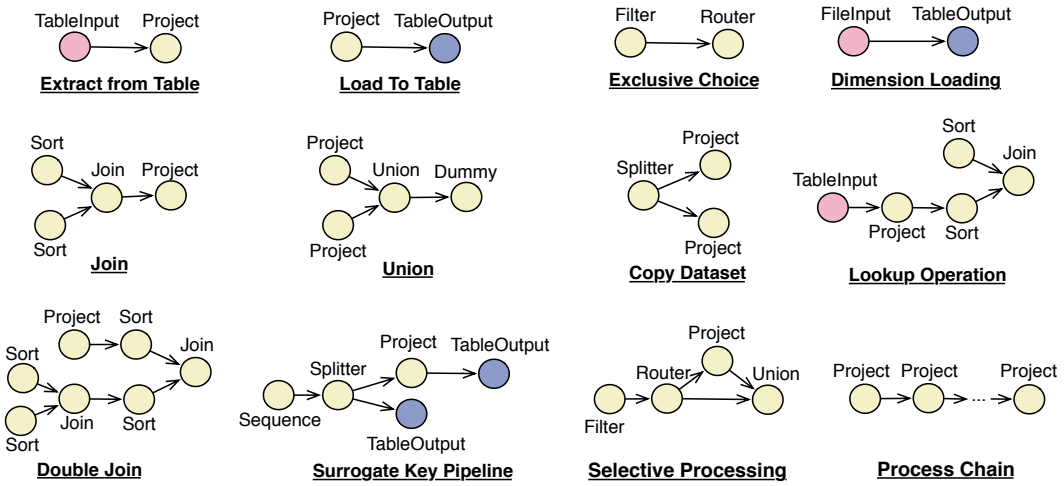


Figure 5.12: Frequent ETL pattern models

Using the support value 20% (i.e., pattern model occurs at least in 5 of the 25 ETLs), we obtained 156 pattern models of different sizes, from 1 to 7 edges. Subsequently, we employed our pattern recognition algorithm to find all the occurrences of each of these pattern models on the 25 ETLs and the results about the number of occurrences for all the pattern models are shown in Figure 5.10. It is clear that some pattern models occur much more frequently than the others, but there is also a big difference between all occurrences and only independent occurrences for each pattern model. This is illustrated in Figure 5.11, where we show the number of pattern models for different ranges of % of independent occurrences. We can see that there are 87 pattern models with 0% independent occurrence i.e., out of all their occurrences, they never occur independently, not nested inside the occurrence of another pattern model. These pattern models are irrelevant for our analysis and thus we only keep the remaining $156 - 87 = 69$ pattern models.

After examining these 69 pattern models playing the role of the ETL expert, we concluded with a set of ETL patterns that we characterized based on their functionality, the most interesting of which are illustrated in Figure 5.12. For each pattern, we show only one pattern model —the one occurring most frequently— but as explained above, each pattern can have two or more variations, as was the case with the patterns that we identified. The variations that we identified for each pattern, only differ in a very small number of model nodes (usually only one node is different) and thus it is possible to implement clustering algorithms based on distance criteria and automatically cluster the different pattern models as pattern variations. We also observed that there are frequent pattern models of small size, which can be combined to synthesize bigger frequent pattern models. As we look from frequent pattern models of smaller size to frequent pattern models of bigger size, the conceptual functionality of the pattern model and its contribution to the ETL process becomes more obvious and concrete.

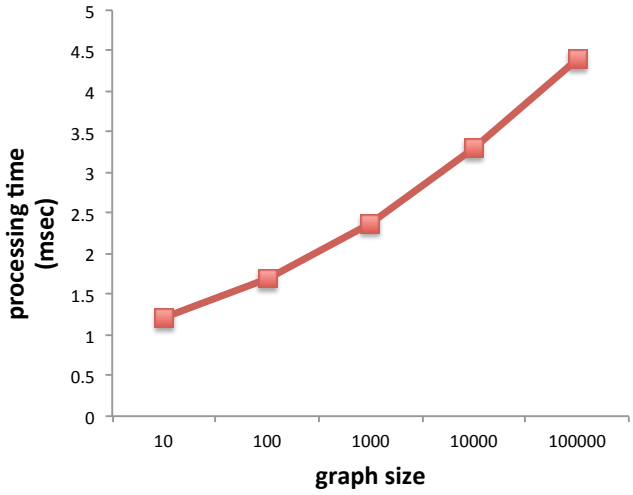


Figure 5.13: Performance of the graph matching algorithm for ETLs of different sizes (Y-axis in log scale)

5.4.2 Performance Evaluation of Graph Matching Algorithm

In this subsection, we present the performance results from running the implementation of our graph matching algorithm (Algorithm 3). To this end, we implemented a synthetic ETL generator that generates ETLs of preferred size and uses statistical characteristics of the 25 TPC-DI ETLs as follows: We parsed the TPC-DI ETLs and stored, for each operation type, i) the average number of succeeding operations and ii) the percentage of succeeding occurrences for each operation type. For example, we found that out of all the 293 succeeding operations of all the operations of operation type *Project*, only 12 operations are of type *TableOutput* and thus the probability of our generator generating an operation of type *TableOutput* after an operation of operation type *Project* is set to: $12/239 = 5\%$. In order to be able to adjust the size of the produced ETL, our generator can modify the percentage of new operations that are of joining type being generated, as opposed to merging parts of the flow with existing operations of joining type. In the same respect, the probability of output operations (i.e., operations that have zero fanout) can be modified dynamically. We generated ETLs of different sizes and for each ETL, we executed our graph matching algorithm for all of the 156 frequent pattern models on a row. The performance results from these experiments, carried under an OS X 64-bit machine, Processor 965 Intel Core i5, 1.7 GHz and 4GB of DDR3 RAM, are shown in Figure 5.13. As can be seen, our algorithm performs very well (almost linearly) for ETLs sharing characteristics with the TPC-DI ETLs, even for graphs of size 10^5.

5.4.3 Granular ETL Performance Evaluation

In subsection 5.2.2, we claimed that with the use of our approach, ETL workflows can be evaluated with regards to their quality characteristics at the granular level of patterns. In this subsection, we show how such an evaluation can take place for the ETL performance. Thus,

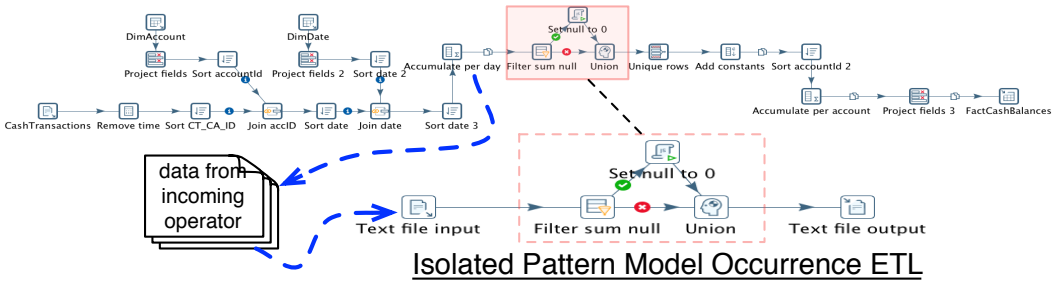


Figure 5.14: Creating an ETL by isolating a pattern model occurrence

we implemented a process that isolates pattern model occurrences (**pmo**) and executes them multiple times to obtain their average execution time, using as input data that are generated from the provided TPC-DI data generator[5] with scale factor 1. This process corresponds to the learning phase that we mentioned in subsection 5.2.2. To this end, the output of each *incoming* operator to the **pmo** (i.e., all the operators that are not part of the **pmo** but are adjacent to at least one node from the **pmo** with direction towards that node) is stored into a text file and an *file input* operator is added to the **pmo**, that reads this text file and passes on its data to the pattern nodes. In addition, we added *file output* operators for each edge for which the source is a node from the **pmo** but the target is a node that is not in the **pmo**, as can be seen in Figure 5.14.

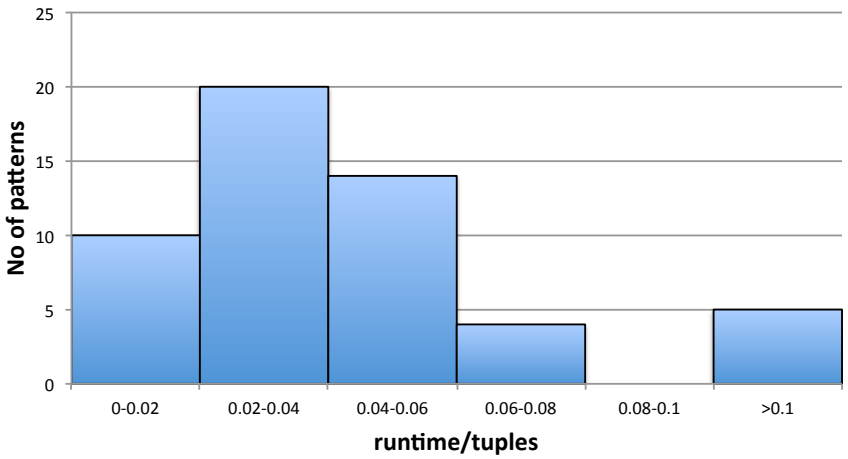


Figure 5.15: Histogram of number of patterns w.r.t. average values of runtime divided by input size

Executing pattern model occurrences from 6 TPC-DI ETLs, we found that the results were adequate to expose the fluctuation in the performance of different pattern models. Results are shown in Figure 5.15, where we show a histogram, the y-axis being the number of patterns and the x-axis being the average value for runtime (in $msec$) divided by input size (i.e., the

[5] `http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp`

sum of all the tuples coming from the generated *file input* files). As we can observe, *five* pattern models have outstanding values for this measure compared to the others. Three of these pattern models are variations of the *Surrogate Key Pipeline* pattern and the other two are variations of the *Union* pattern. Examining the first three, we observed that they all included a sequence generation operator (i.e., for adding to each tuple an integer to act as a surrogate key), right before an output operator for loading data to a database. These results are a clear indication of an anti-pattern, since with this design data cannot be processed in a parallel fashion and can possibly be resolved by pushing back the sequence generation operator earlier in the flow.

5.5 SUMMARY AND OUTLOOK

In this chapter, we introduced a novel empirical approach for pattern-based analysis of ETL workflows in a bottom-up manner. We formally defined an ETL pattern model and we illustrated how it can be instantiated using a training set of ETL workflows to extract frequently reoccurring structural motifs. The graph representation that we adopt enables the use of graph algorithms, such as frequent subgraph discovery algorithms for the mining phase and graph matching algorithms for the recognition phase. For the latter, we adapted the VF2 algorithm with some optimizations and we showed through experiments how it performs very well for ETL workflows. In addition, we presented the most frequent ETL patterns that we identified in implemented processes from the TPC-DI framework, as well as the results from different configurations of the used algorithms. Furthermore, we illustrated the functionality of a tool that we prototyped, for the semi-automatic use of defined patterns for the quality-aware improvement of ETL processes. Results show high efficiency and effectiveness of our approach and future work can delve deeper into the evaluation and pattern-based benchmarking of a larger number of realistic ETL workflows to build a solid Knowledge Base of ETL patterns and their characteristics.

A TOOL FOR QUALITY-AWARE ETL PROCESS REDESIGN

In this chapter, we present our tool **POIESIS**, which stands for **P**rocess **O**ptimization and **I**mprovement for **E**TL **S**ystems and **I**ntegration **S**ervices. This tool is the implementation of the *Planner* component from the architecture in Figure 1.3. Using a process perspective of an ETL activity, our tool can improve the quality of an ETL Process by automatically generating optimization patterns integrated in the ETL flow, resulting to thousands of alternative ETL flows. We apply an iterative model where users are the key participants through well-defined collaborative interfaces and based on estimated measures for different quality characteristics. POIESIS implements a modular architecture that employs reuse of components and patterns to streamline the design. Our tool can be used for incremental, quantitative improvement of ETL process models, promoting automation and reducing complexity. Through the automatic generation of alternative ETL flows, it simplifies the exploration of the problem space and it enables further analysis and identification of correlations among design choices and quality characteristics of the ETL models. A demo of POIESIS has been presented in (Theodorou et al., 2015).

The remainder of this chapter is organized as follows: In Section 6.1, we provide some background for ETL quality analysis and redesign; subsequently, in Section 6.2, we present some information about the design of our tool and in Section 6.3, we provide an overview of the system; finally, in Section 6.4, we showcase an outline of the tool's features.

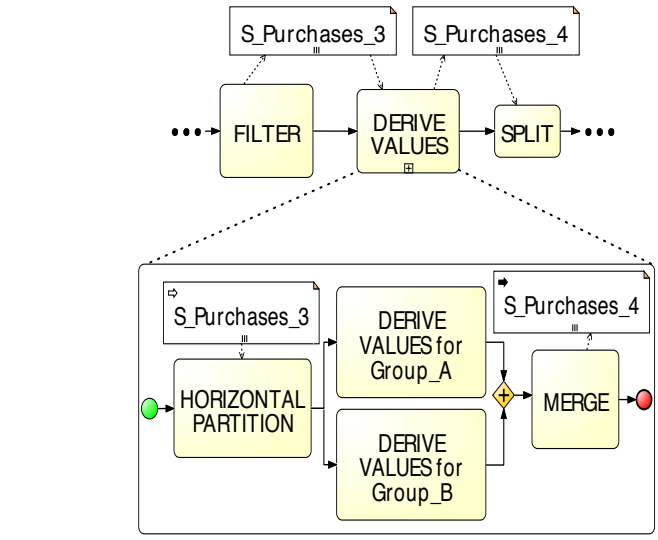
6.1 ADDITION OF FLOW COMPONENT PATTERNS

As has been mentioned above, ETL processes need to be evaluated in a scope that brings them closer to fitness to use for data scientists. Therefore, apart from performance and cost, other quality characteristics, as well as the tradeoffs among them should be taken under consideration during ETL analysis. In Fig. 6.1 we show a subset of ETL process quality characteristics and measures that we have presented in Chapter 3.

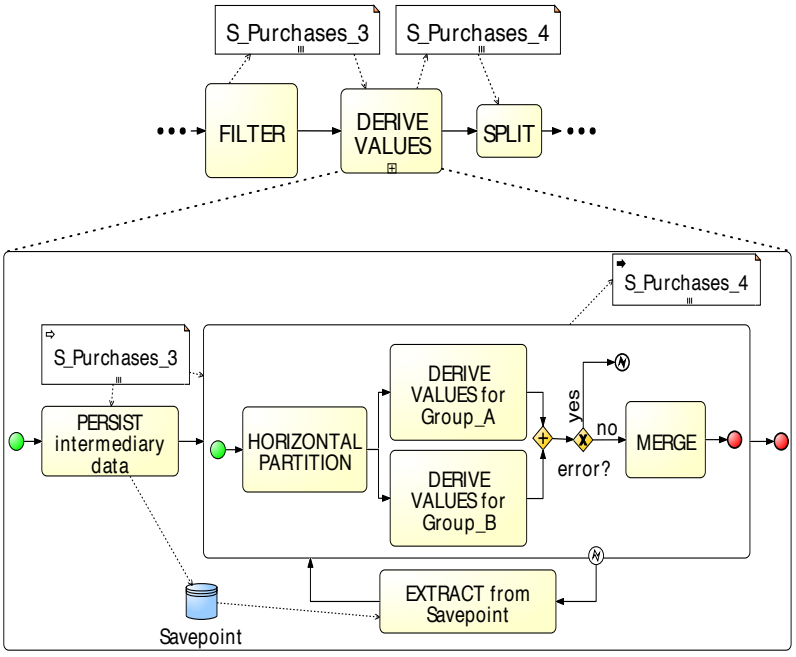
Characteristic	Measure
performance	• Process cycle time • Average latency per tuple
data quality	• Request time - Time of last update • 1 / (1 - age * Frequency of updates)
manageability	• Length of process workflow's longest path • Coupling of process workflow • # of merge elements in the process model

Figure 6.1: Example quality measures for ETL processes

Based on such measures, it is possible to conduct a multi-objective analysis and make design decisions according to user preferences on different quality characteristics, which can often be conflicting.



(a) Improved performance



(b) Improved reliability

Figure 6.2: Generation of FCP on the ETL flow

An initial ETL flow can be modified with the addition of predefined constructs that improve certain quality characteristics, but do not alter its main functionality. We refer to these constructs as Flow Component Patterns (FCP) and their integration can take place on different parts of the initial flow, depending on the flow topology. For example, in Fig. 6.2, we illus-

trate how different quality goals can cause the generation of different FCP on the ETL flow. In the first case, the goal of improving time performance of the process, results in the generation of horizontal partitioning and parallelism within a computational-intensive task and in the second, the goal of improving reliability brings about the addition of a recovery point to the sub-process. Another example would be the goal of improved data quality that would result in crosschecking with alternative data sources.

Central to our implementation is the notion of *application point* of a FCP, which can be either a node (i.e., an ETL flow operation), or an edge or the entire ETL flow graph. As examples, a valid application point for the *ParallelizeTask* pattern is a node that can be replaced by multiple copies of itself and a valid application point for the *FilterNullValues* pattern is an edge on which a filter operation can be added. The entire ETL flow graph as application point serves for the case of process-wide configuration and management operations that are not directly related to the functionality of specific flow components. Examples of the latter include the application of security configurations (encryption, role-based access etc.), management of the quality of Hw/Sw resources, adjusting the frequency of process recurrence etc.

We model the ETL process as one graph G with graph components (V, E), where each node (V) represents an ETL flow operation, and each edge (E) represents a transition from one operation to a successor one. We also assume that there is a set P of available FCP, $P = P_E \cup P_V \cup P_G$, each of which can either be applied on a node, an edge of G, or the entire graph, in order to improve one or more quality characteristics of the ETL flow.

The number Υ of different possible graphs (i.e., ETL processes) that can occur after the application of FCP — assuming that an arbitrary number of available FCP can be applied on each graph component, but each element at most k number of times for every graph component — is the following:

$$\Upsilon = |P_G| * [\sum_{i=0}^{k|P_E|} \binom{k|P_E|}{i}]^{(|E|)} * [\sum_{i=0}^{k|P_V|} \binom{k|P_V|}{i}]^{(|V|)}$$

In the general case, the number of alternative resulting graphs when the complete number of potential FCP applications on edge and node application points is X, is the following:

$$\Upsilon = |P_G| * [\sum_{i=0}^{X} \binom{X}{i}]^{(|E|+|V|)} = |P_G| * 2^{X(|E|+|V|)}$$

After the application of all the FCP, a number of nodes and edges is added to the initial graph and a new Graph is created. This process can be repeated and the question then arises: What is the number of different possible graphs (i.e., ETL processes) that can occur after a number of repetitions of the process? In order to put an upper bound to this question, we assume that after the application of each FCP, ξ new nodes and ψ new edges are added to the initial graph. Then, the number a of graph elements for the resulting graph, after λ repetitions of the process is:

$$a(\lambda) = \begin{cases} |V| + |E| & , \lambda = 0 \\ a(\lambda - 1) * (\xi + \psi) * X & , \lambda > 0 \end{cases}$$

Consequently, the number Υ of different possible graphs (i.e., ETL processes) that can occur after λ repetitions can be recursively calculated as follows:

$$\Upsilon(\lambda) = \begin{cases} |P_G| * [\sum_{i=0}^{X} \binom{X}{i}]^{(a(0))} & , \lambda = 0 \\ \Upsilon(\lambda - 1) * |P_G| * [\sum_{i=0}^{X} \binom{X}{i}]^{(a(\lambda-1))} & , \lambda > 0 \end{cases}$$

It is apparent that the complexity of this analysis is factorial to the size of the graph. Thus, manual configuration of the ETL flow appears inefficient and error-prone, being dependent not only on the users' cognitive abilities but also on characteristics and dynamics of the flow that are hard to predict. Therefore, the need for defining adequate automated mechanisms and heuristics to produce and explore alternative designs and to optimize the ETL flow is evident.

6.2 TOOL DESIGN

Our main drivers throughout the development of this tool have been the objectives of extensibility and efficiency. In this direction, we followed a modular design with clear-cut interfaces and we employed well-known object-oriented design patterns.

The model that was used internally to represent the ETL process flow and allow for its modifications was the ETL flow graph — an extension of the DirectedAcyclicGraph class from the JGraphT free java graph library[1], specific to the ETL flow. Each node of this graph represents an ETL flow operation and each directed edge represents a transition from one operation to a successor one.

Central to our implementation is the notion of *application point* of a quality pattern, which can be either a node (i.e., an ETL flow operation), or an edge or the entire ETL flow graph. As examples, a valid application point for the *ParallelizeTask* pattern is a node that can be replaced by multiple copies of itself using the *Prototype* design pattern and a valid application point for the *FilterNullValues* pattern is an edge on which a filter operation can be added. The entire ETL flow graph as application point serves for the case of process-wide configuration and management operations that are not directly related to the functionality of specific flow components.

Fig. 6.3 depicts the class diagram of the ETL quality patterns. One strong point of our implementation is that it allows for the definition of custom, additional quality patterns during runtime, according to the needs of each specific use case. This is possible, since the *QPattern* abstract class can be sub-classed by any custom class, as long as it implements its abstract methods, i.e., *isApplicable* and *getFitness*, as well as the *deploy* method from the implemented PatternDeployment interface. The former abstract method returns a boolean value, based on the applicability of the pattern on a specific application point on the ETL flow graph and the latter returns a metric to indicate the fitness of the pattern to that specific application point if applicable. The applicability of a pattern is decided based upon specific conditions — instances of the *TopologyCondition* class — that form the applicability prerequisites such as the presence or not of specific data types in the operation schemata (e.g., numeric fields in the output schema of preceding operator). The fitness on the other hand is estimated using heuristics such as the distance of a node from specific type of operations (e.g., distance from input data sources).

[1] `http://jgrapht.org/`

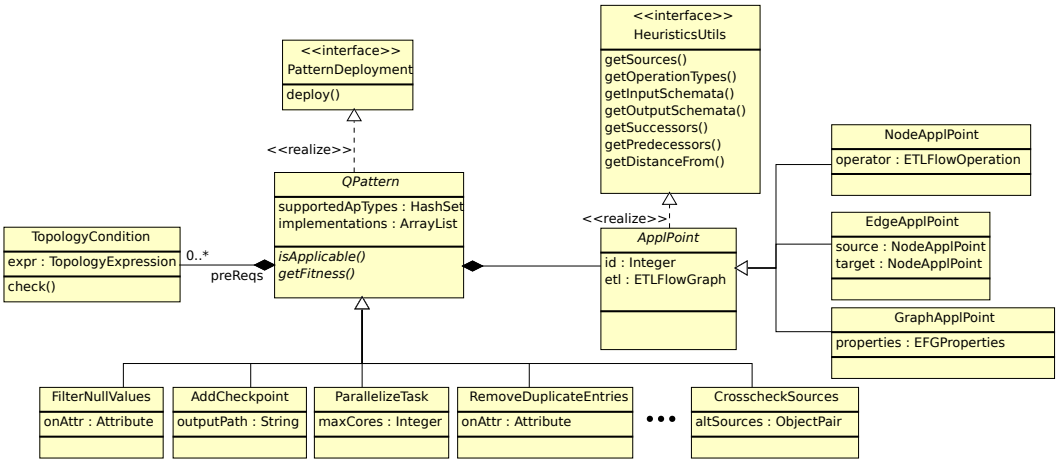


Figure 6.3: ETL Quality Patterns Class Diagram

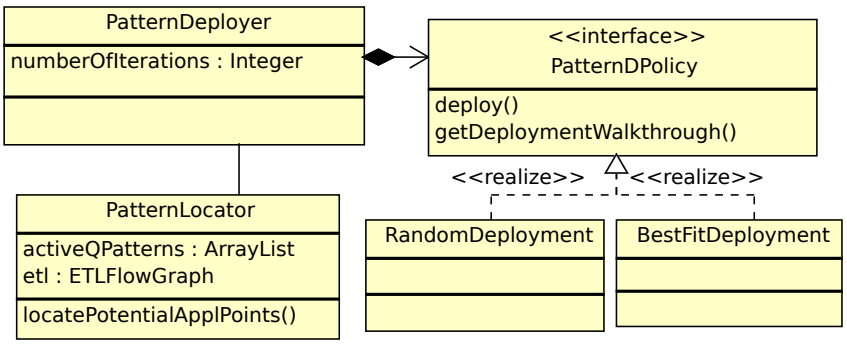


Figure 6.4: ETL Quality Pattern Deployment Class Diagram

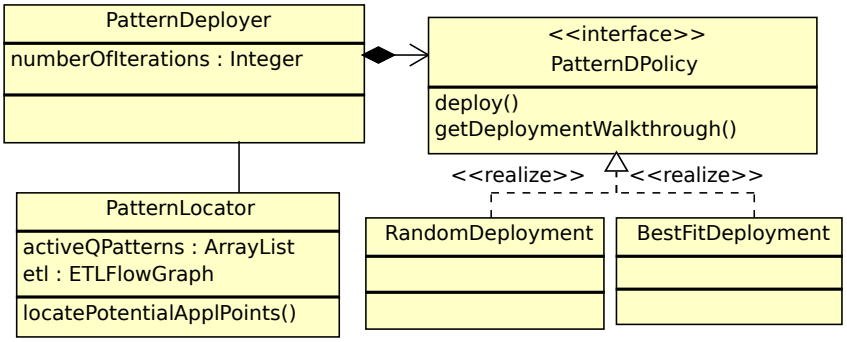


Figure 6.5: ETL Quality Pattern Deployment Class Diagram

The *deploy* method allows for the deployment of any quality pattern on an application point, according to the behavioral design pattern *Strategy*. In essence, the implementation of a quality pattern can be either a setting of configuration parameters or the addition of an ETL flow graph to the existing flow. For the second case, the implementation uses the structural design pattern *Composite* so that it can consist of multiple ETL components but still be manipulated as one object. The *Prototype* design pattern plays an important role at this point, ensuring that the properties of the quality pattern implementation are compatible with the ones of the existing flow.

Regarding the process of pattern deployment, an instance of the *PatternLocator* class traverses the ETL flow graph and identifies the potential application points of the quality patterns that are of interest for each case (i.e., *activeQPatterns*) and their fitness for use for each valid application point. Subsequently, an instance of the *PatternDeployer* class generates quality patterns using the *Factoy method* design pattern, after deciding which combination of patterns should be implemented at that specific step. This decision is implicitly made by the use of a specific realization of the *PatternDPolicy* interface, implementing the *Strategy* design pattern. Thus, the output of this process is a new ETL flow model with the addition of quality pattern(s) and this process can be repeated, while keeping the data sources schemata constant. The class diagram of the above-mentioned classes can be seen in Fig. 6.5.

Our initial testing has shown that our implementation can correctly generate quality patterns on an ETL process flow in just a few milliseconds and it can be used iteratively to incrementally modify an ETL process model. Moreover, as opposed to manual deployment, it guarantees that all of the potential application points on the ETL flow are checked for each quality pattern and it can be customised to select the deployment of patterns based on specific policies that can be harmonized with business requirements.

6.3 POIESIS SYSTEM OVERVIEW

The architecture of our approach can be seen in Fig. 6.6. **POIESIS** takes as input an initial ETL flow and user-defined configurations. Utilizing an existing repository of FCP models, it generates patterns that are specific to the ETL flow on which they are applied. Thus, it pro-

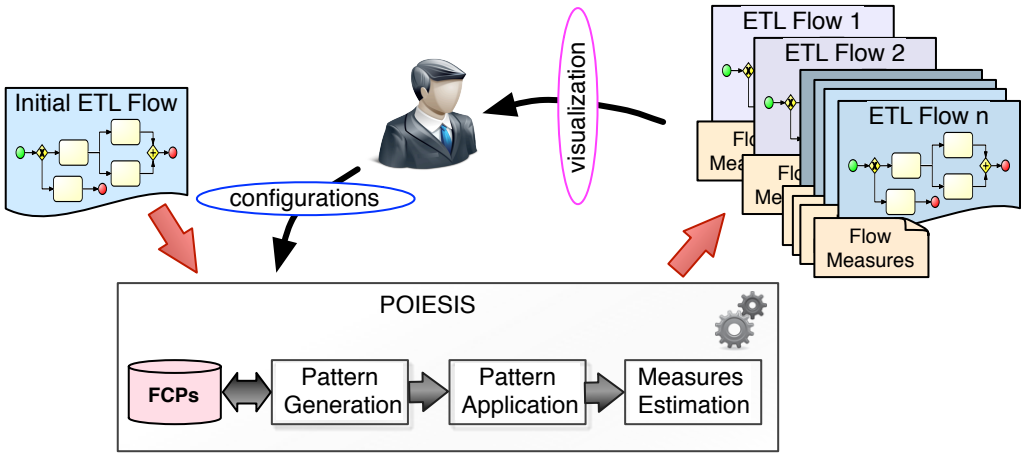


Figure 6.6: POIESIS architecture

duces alternative ETL designs with different FCPs and varying distribution of them on the ETL flow, while keeping the data sources schemata constant. It also estimates defined measures for various quality attributes and illustrates the alternative flows, as well as the corresponding measures to the user through an intuitive visualization.

The internal representation of the FCPs is in the same format as the process flow on which they are deployed. Thus, they can be considered as additional flow components which are positioned at valid application points of the process flow. For example, the *FilterNullValues* pattern is itself an ETL flow consisting of only one operation — a filter that deletes entries with null values from its input. When the *FilterNullValues* pattern is deployed on the initial ETL flow, it is interposed between two consecutive operations. The *FilterNullValues* ETL flow is then configured according to the properties and characteristics of the initial ETL flow as well as the exact application point, ensuring the consistency between data schemata, run-time parameters etc. The same idea is generalized for more complex FCPs or for their more elaborate implementations (e.g., data enrichment additionally to data removal in the described example). In those cases, more detailed configurations might be required to be predefined, such as the access points and data models of additional data sources and processing algorithms of operations.

Our main drivers throughout the development of this component have been the objectives of extensibility and efficiency. In this direction, we followed a modular design with clear-cut interfaces and we employed well-known object-oriented design patterns. The model that was used internally to represent the ETL process flow and allow for its modifications was the ETL flow graph. Each node of this graph represents an ETL flow operation and each directed edge represents a transition from one operation to a successor one.

As a consequence, one strong point of our implementation is that it allows for the definition of custom, additional FCPs, tailored to specific use cases. The applicability of a FCP on the complete ETL flow or some part of it, is decided based upon specific conditions that form the applicability prerequisites, such as the presence or not of specific data types in the operation schemata (e.g., numeric fields in the output schema of preceding operator). Each FCP is related

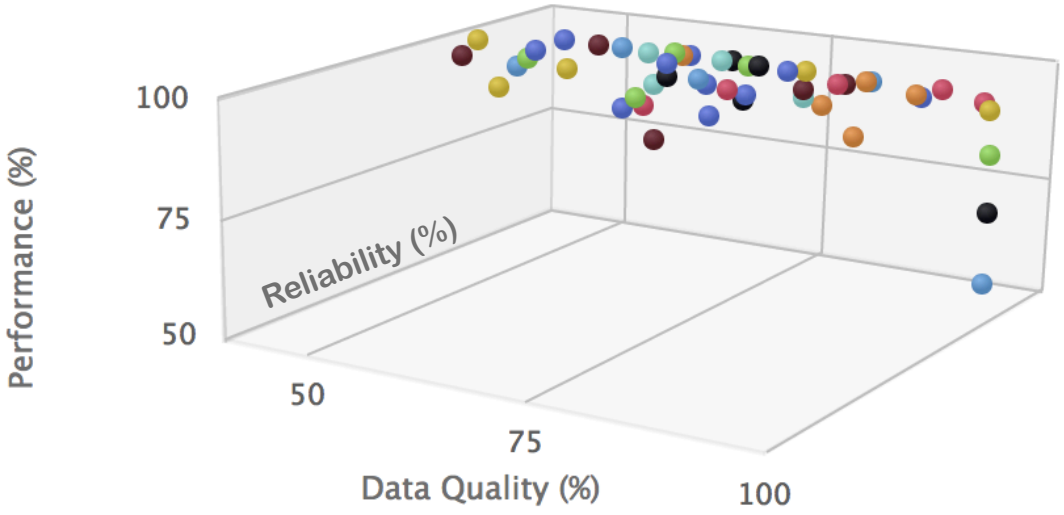


Figure 6.7: Multidimensional scatter-plot of alternative ETL flows

to a particular set of prerequisites that have to be satisfied conjunctively to determine a valid application point. Apart from these strict conditions, there are also *heuristics* to determine the fitness of FCPs for different parts of the ETL flow. For example, according to such heuristics, the addition of a checkpoint is encouraged after the execution of the most complex operations of the ETL fow, in order to avoid the repetition of process-intensive tasks in case of a recovery. Similarly, the application of FCPs related to data cleaning is encouraged as close as possible to the operations for inputing data sources, to prevent cumulative side-effects of reduced data quality. Thus, as opposed to manual deployment, our tool guarantees that all of the potential application points on the ETL flow are checked for each FCP and it can be customized to select the deployment of patterns based on custom policies based on different heuristics.

What is unique about POIESIS is that the redesign process takes place in an iterative, incremental and intuitive fashion. A large number of alternative process designs is automatically generated and these can be instantly evaluated based on quality criteria. Moreover, through a highly interactive UI, the user at any point can interact with a visualization of the ETL process and the estimated measures for each of the alternative designs.

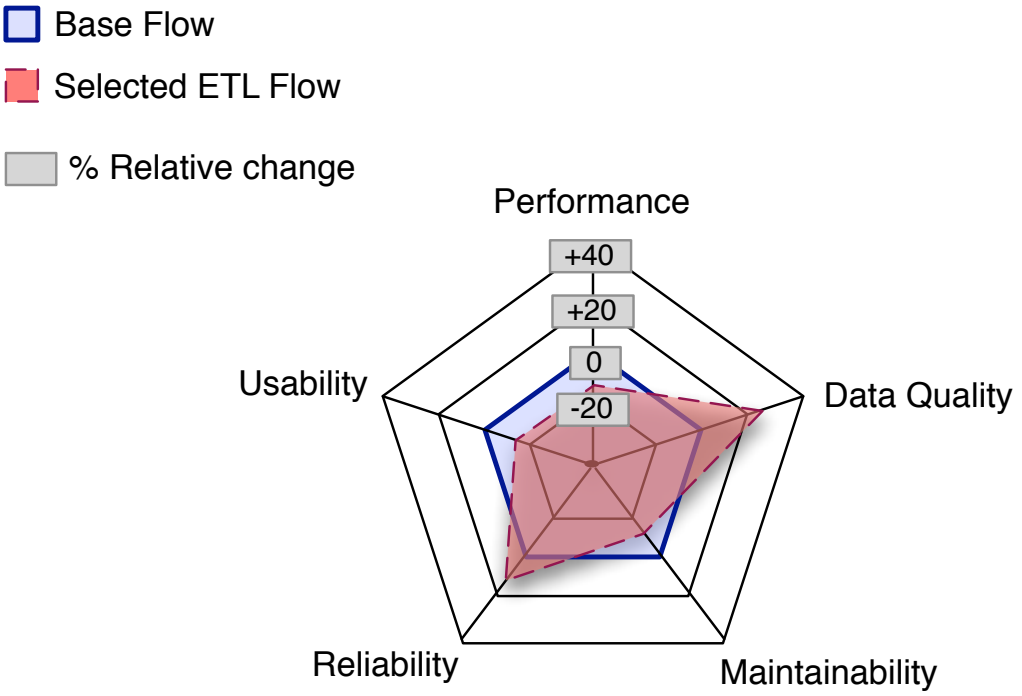


Figure 6.8: Radar chart of relative change in measures for an ETL flow, compared with the initial flow as a baseline

The first step is to import an initial ETL model to the system. This model can be a logical representation of the ETL process and we currently support the loading of xLM (Wilkinson et al., 2010) and PDI[2], but more options will be available in a future version. Subsequently, the user can select the preferred processing parameters, i.e., choose which FCP can be considered in the palette of patterns to be added to the flow, and select the deployment policy for the patterns. It is important to notice at this point that the user can configure the various patterns and even extend them to create custom patterns for future use. The same also stands for the deployment policies, which can be configured according to the user-defined prioritization of goals, as well as the set of constraints based on estimated measures.

Next, after generating and applying relevant FCPs on the ETL flow, the Planner presents to the user a set of potential designs in a multidimensional scatter-plot visualization (see Fig. 6.7), together with quality measures (by clicking on any point on the scatter-plot). The scatter-plot points presented to the user are only the Pareto frontier (skyline) of the complete set of alternative designs, based on their evaluation according to the examined quality dimensions, where larger values are preferred to smaller ones. For example, considering the quality dimensions shown in Fig. 6.7, for one design *ETL1*, if there exists at least one alternative design *ETL2* offering the same or better *performance* and *data quality*, and at the same time better *reliability*, then *ETL1* will not be presented to the user.

[2] `http://community.pentaho.com/projects/data-integration/`

The presented measures (see Fig. 6.8) show on a radar chart the relative change on the metrics for each quality characteristic, denoting the estimated effect of selecting each of the available flows, compared with the initial flow. Apparently, the processing and analysis of the alternative process designs is a process intensive task, mainly due to the large number of alternative flows that have to be concurrently evaluated. Therefore, we employ Amazon Cloud[3] elastic infrastructures, by launching processing nodes that run in the background and enable system responsiveness.

When the user selects (clicks on) any of the different quality dimensions on the measures graph, the corresponding composite measure "expands" to more detailed measures, providing the user with a more in-depth monitoring view. Based on measures and design, the user makes a selection decision and the tool implements this decision by integrating the corresponding patterns to the existing process flow. These patterns are in the form of process components and the Planner carefully merges them to the existing process (Jovanovic et al., 2012). Subsequently, new iteration cycles commence, until the user considers that the flow adequately satisfies quality goals (Theodorou et al., 2014a).

6.4 POIESIS FEATURES

ETL processes can contain tens of operators, extracting data from multiple sources. Their logical representation in *xLM* format can be loaded in the system and the automatic addition of Flow Component Patterns in different positions and combinations on the initial flows, will result in thousands of alternative ETL flows, with different quality characteristics.

Using these processes as input data to our system, we describe here the main capabilities of our tool:

1. Users can interact with the visualizations of our tool's GUI. In particular, they can scroll over/click on any point on the scatterplot that depicts alternative ETL flows on a multi-dimensional space of different quality characteristics. By selecting one point — corresponding to one ETL flow — the process representation and the measures for this flow appear on the screen. Users are then able to view details about the ETL flow, as well as click on any measure so that it expands to more detailed composing metrics.

2. The processing parameters can be configured in order to produce different collections of alternative flows. Thus, users can choose which of the available Flow Component Patterns will be used and which policy will be followed for their deployment.

3. Finally, users can define their own Flow Component Patterns, quality metrics and deployment policies, by extending and pre-configuring the existing ones. They can save their custom processing preferences, adding them to the palette of available patterns for future execution. Examples of the FCPs, which our palette includes, together with the quality attribute that they are intended to improve, are shown in Figure 6.9.

[3] `http://aws.amazon.com/ec2/`

FCP	Related quality attribute
RemoveDuplicateEntries	Data Quality
FilterNullValues	Data Quality
CrosscheckSources	Data Quality
ParallelizeTask	Performance
AddCheckpoint	Reliability

Figure 6.9: Available FCPs

CONCLUSION

7.1 CONCLUSION

In the Big Data era, expectations from data-intensive processes are becoming more and more demanding, pushing for solutions that foster agility. Although information systems are developed by professionals with technical expertise, it is important to align the design of underlying processes with an end-user perspective that reflects business requirements. The automation of ETL processes is a promising direction in order to effectively face emerging challenges in Business Intelligence mainly caused by data volume, velocity and variety. In this direction, several approaches have been proposed for the effective modeling of ETL processes, raising the conceptual level of ETL activities and focusing on the reuse of commonly occurring components during ETL design and their automatic translation to implementation steps. However, the frameworks introduced so far heavily rely on expertise to define some universal abstractions that attempt to be applicable for the analysis of arbitrary ETL workflows. In addition, the level of abstraction does not appear high enough to echo a wide range of quality concerns stemming from the process perspective of ETL processes, other than the traditionally examined dimensions of performance and cost.

In this thesis, we addressed the problem of quality-aware ETL design in multidisciplinary, dynamic business environments. In order to reduce the complexity of deciding optimal process configurations that are in line with business requirements, we raised the level of abstraction of ETL operators and we considered the different roles that BU and IT can play based on their background and goals. Thus, we developed a methodology where BU are assigned with the task of deciding quality goals and evaluating available configurations based on high-level measures. On the other hand, besides development, monitoring and support, IT are responsible for model representation decisions. Based on this concept, we introduced the architecture and methodology of an incremental, iterative framework for end-to-end declarative ETL design and implementation, using goal modeling techniques and keeping BU at the center of quality control.

We have shown how our defined models can be used to automate the task of selecting among alternative designs and improving ETL processes according to defined user goals. To achieve this, we proposed in Chapter 3 a sound model for ETL process quality characteristics that constructively utilizes research work coming from the fields of Data Warehousing, ETL, Data Integration, Software Engineering and Business Process Modeling. In order to adapt models and techniques from all these areas, we based our analysis upon proposed models and abstractions in literature that expose different angles of ETL processes. Instead of conducting isolated analysis about each quality dimension, we adopted a unified view, that implies trade-offs among the improvement of different characteristics and we have qualitatively defined such relationships. One strong point of our model is that it includes measures and quality indicators, which are all backed by existing literature. Thus, users are offered a rich tool-set that steps on well-proven concepts and can aid in decision making during ETL design. In this direction, we have showcased how Goal Modeling techniques can be employed to make informed decisions by assessing the validity of alternative ETL implementations and configurations and by providing the means for their quality-based comparison.

Furthermore, we have studied the problem of generating data for the evaluation of quality characteristics of ETL processes. We identified the need for refined synthetic data generation

because on one hand, real input data for ETL processes are hard to obtain and manipulate and on the other, the evolution of ETL processes is very fast because of continuously changing requirements, making their evaluation with stiff datasets impossible. In this respect, in Chapter 4 we proposed an ETL data generation framework (*Bijoux*), which can generate targeted minimalistic datasets to evaluate specific quality dimensions of the ETL process. This tool is highly parametrizable and can generate datasets to cover the entire ETL workflow, or only specific parts of it, by taking under consideration the semantics of each ETL operation of the workflow. To this end, we have defined novel algorithms, as well as a modular architecture that we also implemented using a prototype. Our experimental results have shown a linear trend of *Bijoux*'s overhead with regards to the complexity of the ETL workflow, which suggests efficiency and scalability of the proposed algorithm.

Moreover, in Chapter 5 we addressed the problem of pattern-based analysis of ETL processes by introducing a novel approach for pattern mining in ETL workflows. We adopted the graph representation of ETL processes and we proposed the use of frequent subgraph discovery algorithms for the identification of frequently recurring structural motifs in any arbitrary set of ETL workflows. In addition, we studied existing graph matching algorithms and we adapted one of them with optimization heuristics to perform very well for ETL workflows, as we validated experimentally. We conducted various experiments to showcase the effectiveness of our approach on implemented ETL processes and we illustrated the most interesting findings, including mined frequent ETL patterns from the TPC-DI benchmark. We also showed how our bottom-up approach can be used both for the automatic translation of an ETL model to its corresponding conceptual representation, using a high-level notation such as BPMN and for the exposure of quality characteristics of ETL processes at the granular level of patterns.

In Chapter 6, we have built a tool that manages a pallet of patterns, each of which is known to have an effect on the improvement of some quality attribute. This tool can semi-automatically integrate these patterns to given ETL workflows in a highly configurable fashion, while at the same time providing intuitive visualizations to end-users about the ETL model and its quality measures estimation.

In conclusion, we have introduced models and frameworks that have brought the costly, time-consuming and error-prone tasks of ETL design and re-design closer to end-users, reducing the gap between IT and BU. We have exposed through our approach a comprehensive view of the quality dimensions of ETL processes, as well as ways to evaluate and to improve them, automating their analysis and making it easier and faster, but at the same time more granular, with the use of patterns. Modifiability has been a central concern during the development of our frameworks —and the tools to validate them— and thus, our methodologies are highly extensible and implementable on a variety of different ETL systems.

7.2 FUTURE WORK

In this thesis, we have introduced a novel approach for user-centered, declarative ETL, we have defined the models that can support it and we have carefully delved into its different aspects. Nonetheless, our work has paved the way for further research, concerning the extension of our frameworks, as well as their adaptation for different areas.

Regarding quality measures of ETL Processes, one interesting research direction can be the quantification of our models, using statistical models. In this respect, various methodologies can be tested, that can take as input a large number of ETL workflows and after estimating for each of them our proposed measures, infer mathematical models about the relationships between different quality attributes and their trade-offs. In this respect, empirical evaluation of the models can take place with the use of extensive controlled experiments involving real ETL users as subjects. The users can state their perceived utility of different ETL processes and through careful evaluation of the results, utility functions of different quality measures can be discovered. In the same direction, future work can define methodologies that use our models as staring points to generate various goal models from different frameworks, for which refinements, satisfiability propagation and elicitation of goals will be straightforward.

When it comes to data generation for ETL process evaluation, the set of ETL operations supported by *Bijoux*can be extended to include operations with semantics that are more difficult to analyze, e.g., groupers and user-defined functions. In addition, it can be modified to cover a broader spectrum of configurable parameters, especially focusing on the ones that enable the evaluation of different quality criteria of an ETL process.

Regarding the use of patterns for the translation of ETL logical models to corresponding conceptual, apart from conducting tests on a larger set of realistic ETL workflows to improve the completeness of identified patterns, it could also be examined how incomplete pattern matching affects the conducted analysis. It is possible that patterns can show great variability or overlap and thus it can be problematic to recognize them on different workflows, but even more problematic to mine them as frequently reoccurring structures. One possible solution that can be examined is the use of approximate matching algorithms or the reordering of ETL operations in the workflow, whenever that is allowed.

Concernig ETL patterns for the generation of more fine-grained cost models, one issue that would be interesting to examine is the relationship between the characteristics of incoming datasets and the quality attributes of different pattern models. To this end, provided that there is a large enough number of ETL workflows to act as the training set, machine learning algorithms can be employed to extract relevant features, as well as regression models. If such analysis can prove valid, then the evaluation of ETL workflows will be made possible without the need of their execution or benchmarking, but simply by examining their static structure and providing as input the right parameters.

However, our work can also be adapted to be applicable for domains other that data warehousing. Many of our proposed models can be adjusted to apply on data-intensive processes in various different areas. As an example, if data processing for big data analysis can be modeled in a process perspective then the main ideas from our approach can be reused. For instance, the processing of operation semantics can suggest the variables for synthetic data generation to evaluate different quality attributes —which of course might be significantly different from the ones for ETL, but the main ideas about trade-offs and their quantification still apply.

BIBLIOGRAPHY

OWL 2 Web Ontology Language Manchester Syntax. http://www.w3.org/TR/owl2-manchester-syntax/, cited August 2015.

The Subsystems of ETL Revisited. http://www.informationweek.com/software/information-management/kimball-university-the-subsystems-of-etl-revisited/d/d-id/1060550, cited January 2014.

KPI Library. http://kpilibrary.com, cited January 2014.

Alberto Abelló, Jérôme Darmont, Lorena Etcheverry, Matteo Golfarelli, Jose-Norberto Mazón, Felix Naumann, Torben Bach Pedersen, Stefano Rizzi, Juan Trujillo, Panos Vassiliadis, and Gottfried Vossen. Fusion Cubes: Towards Self-Service Business Intelligence. *IJDWM*, 9(2): 66–88, 2013.

Zineb Akkaoui, José-Norberto Mazón, Alejandro Vaisman, and Esteban Zimányi. BPMN-Based Conceptual Modeling of ETL Processes. In *DaWaK*, pages 1–14. Springer, 2012.

Zineb Akkaoui, Esteban Zimányi, José-Norberto Mazón, and Juan Trujillo. A BPMN-Based Design and Maintenance Framework for ETL Processes. *IJDWM*, 9(3):46–72, 2013.

R.E. Al-Qutaish. An Investigation of the Weaknesses of the ISO 9126 Intl. Standard. In *ICCEE*, pages 275–279, 2009.

Arvind Arasu, Raghav Kaushik, and Jian Li. Data generation using declarative constraints. In *SIGMOD Conference*, pages 685–696, 2011.

Babak Bagheri Hariri, Diego Calvanese, Giuseppe Giacomo, Riccardo Masellis, and Paolo Felli. Foundations of Relational Artifacts Verification. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management*, volume 6896 of *Lecture Notes in Computer Science*, pages 379–395. Springer Berlin Heidelberg, 2011.

Mario Barbacci, Mark Klein, Thomas Longstaff, and Charles Weinstock. Quality Attributes. Technical report, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1995.

Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM Comput. Surv.*, 41(3):16:1–16:52, July 2009.

Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

Jarg Becker, Martin Kugeler, and Michael Rosemann. *Process Management: a guide for the design of business processes: with 83 figures and 34 tables*. Springer, 2003.

Ladjel Bellatreche, Selma Khouri, and Nabila Berkani. Semantic Data Warehouse Design: From ETL to Deployment á la Carte. In *Database Systems for Advanced Applications*, pages 64–83. Springer, 2013.

Henrike Berthold, Philipp Rösch, Stefan Zöller, Felix Wortmann, Alessio Carenini, Stuart Campbell, Pascal Bisson, and Frank Strohmaier. An Architecture for Ad-hoc and Collaborative Business Intelligence. In *EDBT*, pages 1–6, 2010.

Mark A. Beyer, Eric Thoo, Ehtisham Zaidi, and Rick Greenwald. Magic Quadrant for Data Integration Tools. Technical report, Gartner, August 2016.

Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer Berlin Heidelberg, 2007.

Carsten Binnig, Donald Kossmann, Eric Lo, and M. Tamer Özsu. QAGen: generating query-aware test databases. In *SIGMOD Conference*, pages 341–352, 2007.

Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. DIPBench Toolsuite: A Framework for Benchmarking Integration Systems. In *ICDE*, pages 1596–1599, 2008.

Matthias Böhm, Dirk Habich, Wolfgang Lehner, and Uwe Wloka. Invisible Deployment of Integration Processes. In Joaquim Filipe and José Cordeiro, editors, *Enterprise Information Systems*, volume 24 of *Lecture Notes in Business Information Processing*, pages 53–65. Springer Berlin Heidelberg, 2009.

Matthias Böhm, Uwe Wloka, Dirk Habich, and Wolfgang Lehner. GCIP: Exploiting the Generation and Optimization of Integration Processes. In *EDBT*, pages 1128–1131, 2009.

Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from Applying the Systematic Literature Review Process Within the Software Engineering Domain. *J. Syst. Softw.*, 80(4):571–583, April 2007.

Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

Malu Castellanos, Alkis Simitsis, Kevin Wilkinson, and Umeshwar Dayal. Automating the loading of business process data warehouses. In *EDBT*, pages 612–623, 2009.

David Chays, Saikat Dan, Phyllis G. Frankl, Filippos I. Vokolos, and Elaine J. Weber. A framework for testing database applications. In *ISSTA*, pages 147–157, 2000.

Elizabeth Chew, Marianne Swanson, Kevin M. Stine, Nadya Bartol, Anthony Brown, and Will Robinson. Performance Measurement Guide for Information Security. Technical report, 2008.

Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. Non-Functional Requirements in Software Engineering. *International Series in Software Engineering*, 5, 2000.

D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(03): 265–298, 2004.

L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen*, pages 149–159, 2001.

Richard A. DeMillo and A. Jefferson Offutt. Constraint-Based Automatic Test Data Generation. *IEEE Trans. Software Eng.*, 17(9):900–910, 1991.

Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.

Schahram Dustdar, Reinhard Pichler, Vadim Savenkov, and Hong-Linh Truong. Quality-aware service-oriented data integration: requirements, state of the art and open challenges. *SIGMOD Rec.*, 41(1):11–19, April 2012.

Martin J. Eppler and Markus Helfert. A Framework For The Classification Of Data Quality Costs And An Analysis Of Their Progression. In InduShobha N. Chengalur-Smith, Louiqa Raschid, Jennifer Long, and Craig Seko, editors, *IQ*, pages 311–325. MIT, 2004.

William Frakes and Carol Terry. Software Reuse: Metrics and Models. *ACM Comput. Surv.*, 28 (2):415–435, 1996.

Félix García, Mario Piattini, Francisco Ruiz, Gerardo Canfora, and Corrado A. Visaggio. FMESP: Framework for the Modeling and Evaluation of Software Processes. QUTE-SWAP, pages 5–13. ACM, 2004.

G.K. Gill and C.F. Kemerer. Cyclomatic complexity density and software maintenance productivity. *Soft. Eng., IEEE Trans. on*, 17(12):1284–1288, 1991.

Paolo Giorgini, Stefano Rizzi, and Maddalena Garzetti. GRAnD: A goal-oriented approach to requirement analysis in data warehouses. *Decision Support Systems*, 45(1):4 – 21, 2008.

Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. Sprint Planning Optimization in Agile Data Warehouse Design. In *DaWaK*, pages 30–41, 2012.

Jim Gray, Prakash Sundaresan, Susanne Englert, Kenneth Baclawski, and Peter J. Weinberger. Quickly Generating Billion-Record Synthetic Databases. In *SIGMOD Conference*, pages 243–252, 1994.

Thomas Gschwind, Jana Koehler, and Janette Wong. *Applying Patterns during Business Process Modeling*, pages 4–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

Joseph E. Hoag and Craig W. Thompson. A parallel general-purpose synthetic data generator. *SIGMOD Record*, 36(1):19–24, 2007.

Jennifer Horkoff and Eric S. K. Yu. Comparison and evaluation of goal-oriented satisfaction analysis techniques. *Requir. Eng.*, 18(3):199–222, 2013.

Jennifer Horkoff, Alex Borgida, John Mylopoulos, Daniele Barone, Lei Jiang, Eric Yu, and Daniel Amyot. Making Data Meaningful: The Business Intelligence Model and Its Formal Semantics in Description Logics. In *OTM*, pages 700–717. Springer, 2012.

Jennifer Horkoff, Daniele Barone, Lei Jiang, Eric Yu, Daniel Amyot, Alex Borgida, and John Mylopoulos. Strategic Business Modeling: Representation and Reasoning. *Softw. Syst. Model.*, 13(3):1015–1041, July 2014.

Kenneth Houkjær, Kristian Torp, and Rico Wind. Simple and Realistic Data Generation. In *VLDB*, pages 1243–1246, 2006.

Fabian Hueske, Mathias Peters, Matthias Sax, Astrid Rheinländer, Rico Bergmann, Aljoscha Krettek, and Kostas Tzoumas. Opening the Black Boxes in Data Flow Optimization. *PVLDB*, 5(11):1256–1267, 2012.

Monique Jansen-Vullers and Mariska Netjes. Business process simulation–a tool survey. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume 38, pages –, 2006.

Matthias Jarke, Manfred A. Jeusfeld, Christoph Quix, and Panos Vassiliadis. Architecture and quality in data warehouses: An extended repository approach. *Information Systems*, 24(3): 229 – 253, 1999.

Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis. *Fundamentals of Data Warehouses*. Springer, 2003.

Chuntao Jiang, Frans Coenen, and Michele Zito. A survey of frequent subgraph mining algorithms. *Knowledge Eng. Review*, 28(1):75–105, 2013.

P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *Parallel and Distributed Systems, IEEE Trans. on*, 11(6):589–603, 2000.

P. Jovanovic, A. Simitsis, and K. Wilkinson. Engine independence for logical analytic flows. In *2014 IEEE 30th International Conference on Data Engineering*, pages 1060–1071, March 2014.

P. Jovanovic, O. Romero, A. Simitsis, and A. Abelló. Incremental Consolidation of Data-Intensive Multi-Flows. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1203–1216, May 2016.

Petar Jovanovic. Requirement-driven Design and Optimization of Data-Intensive Flows. *Ph.D. Dissertation*, 2016.

Petar Jovanovic, Oscar Romero, Alkis Simitsis, and Alberto Abelló. Integrating ETL Processes from Information Requirements. In *DaWaK*, pages 65–80, 2012.

Anton Kartashov. Adaptive workflow testing using elastic infrastructures. *Master Thesis*, 2016.

R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *ICSE*, pages 297–306, 2001.

Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Inf. Softw. Technol.*, 51(1):7–15, 2009.

Michihiro Kuramochi and George Karypis. Frequent Subgraph Discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 313–320, Washington, DC, USA, 2001.

Kiran Lakhotia, Mark Harman, and Phil McMinn. A multi-objective approach to search-based test data generation. In *GECCO*, pages 1098–1105, 2007.

Averill M Law, W David Kelton, and W David Kelton. *Simulation modeling and analysis*, volume 2. McGraw-Hill, 1991.

Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. *Proc. VLDB Endow.*, 6(2): 133–144, December 2012.

JulioCesarSampaiodoPrado Leite and Claudia Cappelli. Software Transparency. *Business and Information Systems Engineering*, 2(3):127–139, 2010.

Chunjie Luo, Wanling Gao, Zhen Jia, Rui Han, Jingwei Li, Xinlong Lin, Lei Wang, Yuqing Zhu, and Jianfeng Zhan. Handbook of BigDataBench (Version 3.1) - A Big Data Benchmark Suite, 2014. http://prof.ict.ac.cn/BigDataBench/wp-content/uploads/2014/12/BigDataBench-handbook-6-12-16.pdf. Last accessed: 13/05/2015.

Tim A. Majchrzak, Tobias Jansen, and Herbert Kuchen. Efficiency Evaluation of Open Source ETL Tools. SAC, pages 287–294, New York, NY, USA, 2011. ACM.

Zijian Ming, Chunjie Luo, Wanling Gao, et al. BDGS: A Scalable Big Data Generator Suite in Big Data Benchmarking. *CoRR*, abs/1401.5465, 2014.

Lilia Muñoz, Jose-Norberto Mazón, and Juan Trujillo. Automatic generation of ETL processes from conceptual models. In *DOLAP*, pages 33–40, 2009.

Lilia Muñoz, Jose-Norberto Mazón, and Juan Trujillo. Measures for ETL Processes Models in Data Warehouses. MoSE+DQS, pages 33–36. ACM, 2009.

Lilia Muñoz, Jose-Norberto Mazón, Jesús Pardillo, and Juan Trujillo. Modelling ETL Processes of Data Warehouses with UML Activity Diagrams. In *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*, pages 44–53, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Gunter Mussbacher, Daniel Amyot, and Saeed Ahmadi Behnam. Towards a Pattern-Based Framework for Goal-Driven Business Process Modeling. *Software Engineering Research, Management and Applications, ACIS International Conference on*, 00(undefined):137–145, 2010.

Emona Nakuçi, Vasileios Theodorou, Petar Jovanovic, and Alberto Abelló. Bijoux: Data Generator for Evaluating ETL Process Quality. In *DOLAP*, 2014.

Felix Naumann. *Quality-driven Query Answering for Integrated Information Systems*. Springer-Verlag, 2002.

Bruno Oliveira and Orlando Belo. BPMN Patterns for ETL Conceptual Modelling and Validation. In *ISMIS 2012, Macau, China, December 4-7*, pages 445–454, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Bruno Oliveira and Orlando Belo. Task Clustering on ETL Systems - A Pattern-Oriented Approach. In *Proceedings of 4th International Conference on Data Management Technologies and Applications*, pages 207–214, 2015.

Bruno Oliveira, Vasco Santos, and Orlando Belo. Pattern-Based ETL Conceptual Modelling. In *Model and Data Engineering - Third International Conference, MEDI 2013, Amantea, Italy, September 25-27, 2013. Proceedings*, pages 237–248, 2013.

Bruno Oliveira, Orlando Belo, and Alfredo Cuzzocrea. A Pattern-Oriented Approach for Supporting ETL Conceptual Modelling and its YAWL-based Implementation. In *DATA 2014, Vienna, Austria, 29-31 August, 2014*, pages 408–415, 2014.

Amanpartap Singh Pall and Jaiteg Singh Khaira. A comparative Review of Extraction, Transformation and Loading Tools. *Database Systems Journal*, 4(2):42–51, 2013.

Ray J. Paul, Vlatka Hlupic, and George M. Giaglis. Simulation Modelling Of Business Processes. In *Proceedings of the 3 rd U.K. Academy of Information Systems Conference, McGraw-Hill*, pages 311–320. McGraw-Hill, 1998.

Ivan Pavlov. A QoX Model for ETL Subsystems: Theoretical and Industry Perspectives. CompSysTech, pages 15–21. ACM, 2013.

Tilmann Rabl, Michael Frank, Hatem Mousselly Sergieh, and Harald Kosch. A Data Generator for Cloud-Scale Benchmarking. In *TPCTC*, pages 41–56, 2010.

Oscar Romero, Alkis Simitsis, and Alberto Abelló. GEM: Requirement-Driven Generation of ETL and Multidimensional Conceptual Designs. In *Data Warehousing and Knowledge Discovery*, pages 80–95. Springer, 2011.

Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. *Workflow Data Patterns: Identification, Representation and Tool Support*, pages 353–368. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

Philip Russom. TDWI best practices report: Big data analytics. Technical report, The data Warehousing Institute, 01 2011.

Shazia Sadiq, Naiem Khodabandehloo Yeganeh, and Marta Indulska. 20 Years of Data Quality Research: Themes, Trends and Synergies. In *Proceedings of the Twenty-Second Australasian Database Conference - Volume 115*, ADC '11, pages 153–162, Darlinghurst, Australia, Australia, 2011. Australian Computer Society, Inc.

Sherif Sakr and Eric Pardede. *Graph Data Management: Techniques and Applications*. IGI Publishing, Hershey, PA, 1st edition, 2011.

Marko Salmenkivi. *Frequent Itemset Discovery*, pages 322–323. Springer US, Boston, MA, 2008.

Laura Sánchez-González, Félix García, Francisco Ruiz, and Jan Mendling. Quality Indicators for Business Process Models from a Gateway Complexity Perspective. *Inf. Softw. Technol.*, 54 (11):1159–1174, 2012.

R. Shamir and D. Tsur. Faster subtree isomorphism. In *Theory of Computing and Systems, 1997., Proceedings of the Fifth Israeli Symposium on*, pages 126–131, 1997.

A. Simitsis, P. Vassiliadis, and T. Sellis. Optimizing ETL processes in data warehouses. In *ICDE*, pages 564–575, 2005.

Alkis Simitsis, Panos Vassiliadis, Umeshwar Dayal, Anastasios Karagiannis, and Vasiliki Tziovara. Benchmarking ETL Workflows. In *Performance Evaluation and Benchmarking: First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, pages 199–220, 2009.

Alkis Simitsis, Kevin Wilkinson, Malú Castellanos, and Umeshwar Dayal. QoX-driven ETL design: reducing the cost of ETL consulting engagements. In *SIGMOD Conference*, pages 953–960, 2009.

Connie U. Smith and Lloyd G. Williams. Software Performance Antipatterns. WOSP '00, pages 127–136, New York, NY, USA, 2000. ACM.

Miguel A. Teruel, Roberto Tardío, Elena Navarro, Alejandro Maté, Pascual González, Juan Trujillo, and Rafael Muñoz-Terol. *CSRML4BI: A Goal-Oriented Requirements Approach for Collaborative Business Intelligence*, pages 423–430. Springer International Publishing, Cham, 2014.

Vasileios Theodorou, Alberto Abelló, and Wolfgang Lehner. Quality Measures for ETL Processes. DaWaK, 2014.

Vasileios Theodorou, Alberto Abelló, Maik Thiele, and Wolfgang Lehner. A Framework for User-Centered Declarative ETL. In *DOLAP*, 2014.

Vasileios Theodorou, Alberto Abelló, Maik Thiele, and Wolfgang Lehner. POIESIS: a Tool for Quality-aware ETL Process Redesign. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 545–548, 2015.

Vasileios Theodorou, Alberto Abelló, Wolfgang Lehner, and Maik Thiele. Quality measures for ETL processes: from goals to implementation. *Concurrency and Computation: Practice and Experience*, 28(15):3969–3993, 2016.

Vasileios Theodorou, Petar Jovanovic, Alberto Abellò, and Emona Nakuçi. Data generator for evaluating {ETL} process quality. *Information Systems*, 63:80 – 100, 2017.

Maik Thiele, Andreas Bader, and Wolfgang Lehner. Multi-objective scheduling for real-time data warehouses. *Computer Science - Research and Development*, 24(3):137–151, 2009.

Vasiliki Tziovara, Panos Vassiliadis, and Alkis Simitsis. Deciding the Physical Implementation of ETL Workflows. In *Proceedings of the ACM Tenth International Workshop on Data Warehousing and OLAP*, DOLAP '07, pages 49–56, 2007.

J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42, January 1976.

W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, July 2003.

A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *RE*, pages 249–262, 2001.

A. van Lamsweerde. From System Goals to Software Architecture. In *Formal Methods for Software Architectures. Volume 2804 of Lecture Notes in Computer Science*, pages 25–43. Springer-Verlag, 2003.

Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. Conceptual Modeling for ETL Processes. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP*, DOLAP '02, pages 14–21, New York, NY, USA, 2002. ACM.

Panos Vassiliadis, Alkis Simitsis, and Eftychia Baikousi. A Taxonomy of ETL Activities. In *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, DOLAP '09, pages 25–32, New York, NY, USA, 2009. ACM.

M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2012.

Kevin Wilkinson, Alkis Simitsis, Malu Castellanos, and Umeshwar Dayal. *Leveraging Business Process Models for ETL Design*, pages 15–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

Eric S. K. Yu. Modelling Strategic Relationships for Process Reengineering. *Ph.D. Dissertation*, 1996.

Jian Zhang, Chen Xu, and S. C. Cheung. Automatic Generation of Database Instances for White-box Testing. In *COMPSAC*, pages 161–165, 2001.

List of Figures

CONFIRMATION

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

November 21, 2016