Universidade do Minho
Escola de Engenharia

Bruno Moisés Teixeira de Oliveira

A Pattern-Based Approach for ETL Systems Modelling and Validation

dezembro de 2017

Bruno Moisés Teixeira de Oliveira **A Pattern-Based Approach for ETL Systems Modelling and Validation**

UMinho|2017



Universidade do Minho
Escola de Engenharia

Bruno Moisés Teixeira de Oliveira

A Pattern-Based Approach for ETL Systems Modelling and Validation

Tese de Doutoramento em Informática

Trabalho efetuado sob a orientação do
Professor Doutor Orlando Manuel de Oliveira Belo

dezembro de 2017

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, 20 DE DEZEMBRO DE 2017

Full name:
BRUNO MOISÉS TEIXEIRA DE OLIVEIRA

Signature:
Bruno Moisés Teixeira de Oliveira

Acknowledgments

I would like to express my appreciation to all those that supported and helped me during the years worked on this project, and in particular to:

- Professor Orlando Belo for his supervision, guidance, and friendship during these years.
- Professor Vasco Santos for his guidance and friendship during all my academic/professional life.
- My friends from Polytechnic of Porto and School of Management and Technology (IPP.ESTG) for the help, motivation, and companionship to conclude this project.
- My family, with special attention to my parents and my future wife Ana for all the support and stimulus and patience for the time without my company

A Pattern-Based Approach for ETL Systems Modelling and Validation

Usually, a data warehousing system stores data in an integrated and consistent way, making it an ideal data repository to support decision-making processes. However, to keep this repository properly updated it is necessary to access to a variety of information sources, transform the data gathered according to the established decision-making requirements and load that data into the Data Warehouse System data repository – the data warehouse. All these tasks are done by highly sophisticated programs that together integrates what we use to define as the ETL system. The ETL (*Extract*, *Transform*, *Load*) system is responsible to perform all those tasks, being considered a very time-consuming, error-prone and complex process, involving several participants from different knowledge domains. They are one of the most important components of a data warehousing system, strongly influenced by the complexity of business requirements, their changing, and evolution. These aspects influence not only the structure of a data warehouse itself but also the information sources schemas involved with, since they must handle data with complex data requirements and transformation routines. Moreover, ETL systems are data-oriented processes composed of dozens of granular tasks arranged based on specific languages and architectures, which results in technical and complex terms, difficult to understand and maintain. Despite the efforts done by several researchers for the modelling and implementing them, a solid and simpler approach providing the necessary bridges to create the conceptual and logical models, and validate them before its final implementation is still lacking. However, a specific subset of these tasks can be grouped on a collection together with their relationships to form abstract constructs. Thus, to facilitate the planning and ETL implementation, this work aims to present a set of constructs that represent meta-models (patterns) specially designed to map standard ETL procedures, providing the necessary bridges to represent them at the conceptual level and provide its mapping to execution primitives. Basically, these (ETL) patterns are comprised of a set of abstract components that can be configured to enable its instantiation for specific application

scenarios. With them, generic models can be built, simplifying process views and providing methods for carrying out the acquired expertise to new applications based on well-proven practices that can be used to describe general solutions based on specific skeletons configured and instantiated according to a set of specific integration requirements. The ETL pattern-based approach presented uses BPMN (Business Process Model and Notation) for modelling conceptual ETL workflows, mapping them to real execution primitives using a domain-specific language that allows for the generation of instances that can be executed in an ETL commercial tool. This work demonstrates the feasibility and effectiveness of an ETL pattern-based approach, analysing a test scenario for data integration based on the pattern framework proposed.

Keywords: Data Warehouse, Extract-Transform-Load, ETL conceptual and logical modelling, ETL patterns, Business Process Model and Language (BPMN), pattern configuration and instantiation, domain-specific language and physical model's generation.

Uma Abordagem Orientada a Padrões para a Modelação e Validação de Sistemas de ETL

Os sistemas de *Data Warehousing* suportam o armazenamento de grandes quantidades de dados de forma integrada e consistente, tornando-o num sistema ideal para o suporte de processos de tomada de decisão. De forma a manter os seus repositórios devidamente atualizados, os dados extraídos das fontes de informação utilizadas devem ser transformados de acordo com os requisitos de tomada de decisão para posterior povoamento do seu repositório de armazenamento – o *Data Warehouse*. Todas essas tarefas de gestão e transformação de dados são suportadas por processos de povoamento que com base em sofisticadas estratégias caracterizam o sistema de ETL. Os processos de ETL (*Extract*, *Transform, Load*) são processos bastante específicos, orientados a dados e com uma estrutura que requer a atenção de recursos humanos altamente especializados de diversos domínios. A sua composição é essencialmente baseada em operações muito detalhadas, baseadas em linguagens, metodologias e arquiteturas específicas que originam processos de grandes dimensões, difíceis de desenvolver e manter. Como resultado, o seu desenvolvimento e manutenção consume uma parte substancial dos recursos necessários para a implementação do sistema de *Data Warehousing*. Estas características são essencialmente afetadas pela mudança de requisitos que resulta de processos de negócio evolutivos que afetam não só a estrutura do *Data Warehouse,* mas também a estrutura das fontes de dados utilizadas. Nos últimos anos têm sido realizados desenvolvimentos significativos na área, no entanto, ainda não existe uma abordagem convincente e simples que com base na especificação conceptual e lógica de processos de ETL, proporcione um mapeamento em primitivas de execução tendo por base formalismos sólidos que garantam a noção de consistência do processo. Assim, no sentido de facilitar o seu processo de implementação, um conjunto de meta modelos (padrões) que representam tarefas de ETL tipicamente utilizadas e documentadas, são apresentados. Os padrões representam construtores de alto nível, que só por si permitem desenhar e validar uma primeira versão do sistema de povoamento antes de proceder à sua implementação, simplificando a

representação de modelos mais abstratos (muito úteis para fases iniciais do desenvolvimento do projeto) e ao mesmo tempo disponibilizando os meios necessários para possibilitar o seu mapeamento para primitivas de execução. Desta forma, os recursos direcionados para suportar a fase de planeamento e desenho do projeto podem ser aproveitados em fases de desenvolvimento posteriores (essencialmente mais técnicas), proporcionando uma visão integradora e unificada do processo de desenvolvimento. Com esta abordagem, componentes abstratos podem ser configurados de forma a reorganizar as tarefas que os constituem para proporcionar a geração de instâncias geradas tendo em consideração casos muito particulares. Não só as diversas fases do desenvolvimento do projeto ficam simplificadas, como também é possível encapsular o conhecimento adquirido para outros projetos através da aplicação de práticas conhecidas e validadas, permitindo a construção de sistemas mais fiáveis associados a uma redução do tempo e recursos necessários para a sua implementação. A representação conceptual é suportada pela utilização de BPMN (*Business Process Model and Notation*), utilizando os construtores disponibilizados pela linguagem para o desenvolvimento de fluxos de trabalho bastante detalhados que descrevem os componentes utilizados e como estes se integram com os restantes componentes de ETL existentes. De seguida, os padrões identificados no modelo conceptual são enriquecidos com uma semântica específica, suportada por uma linguagem de domínio especialmente desenvolvida de acordo os requisitos de cada padrão. Estas primitivas lógicas podem ser mais tarde utilizadas para a geração de modelos físicos que podem ser executados utilizando uma ferramenta comercial já existente. Este trabalho apresenta a aplicação dos diversos padrões utilizando para isso um caso de estudo que exemplifica a aplicação de uma abordagem orientada a padrões para o desenvolvimento de sistemas de ETL.

Palavras-Chave: Armazéns de dados, Modelação conceptual e lógica de um processo de ETL, padrões para ETL, Business Process Model and Language (BPMN), linguagem de domínio para configuração e instanciação de padrões e geração de modelos físicos.

Table of contents

Introduction 1

1.1 Context Overview 1

1.2 Motivation and objectives 3

1.3 Thesis and Research Questions 5

1.4 Contributions of the thesis 7

1.5 Document Organization 9

ETL Modelling and Implementation 11

2.1 About ETL Processes 11

2.2 An Architecture for ETL 13

2.2.1 Extraction Processes 13

2.2.2 Transformation Processes 15

2.2.3 Loading Processes 15

2.3 Kimball's ETL Lifecycle Development 16

2.4 ETL Development Approaches 18

2.4.1 Process-Driven Development 20

2.4.2 Model Driven Development 23

2.4.3 Ontology-Driven Development 24

2.4.4 Pattern-Oriented Approaches 25

A Pattern-based Approach for ETL Development 29

3.1 Towards a Pattern-Based ETL Model 29

3.2 ETL Patterns Meta Models 30

3.3 The Core Components of a Pattern 41

3.3.1 The Change Data Capture Pattern 41

3.3.2 The Slowly Changing Dimension Pattern 47

3.3.3 The Data Quality Enhancement Pattern 52

3.3.4 The Surrogate Key Pipelining Pattern 58
3.3.5 The Surrogate Key Generator Pattern 61
3.3.6 The Data Aggregator Pattern 65
3.3.7 Data Conciliation and Integration 67
3.3.8 The Intensive Data Loading Pattern 71
3.4 The Throwable Pattern 74
3.5 The Log Pattern 79
3.6 Patterns Behaviour Validation 82
3.7 The Pattern Language 85

From Conceptual to ETL Physical Implementation 89

4.1 An ETL Case Study 89
4.2 Pattern-oriented methodology for ETL development 93
4.2.1 Conceptual ETL design 94
4.2.2 Logical ETL design 107
4.2.3 Physical ETL design 115

Conclusions and Future Work 123

5.1 Conclusions 123
5.2 Future Work 127

List of figures

Figure 1. A typical ETL system architecture. 12
Figure 2. Vassiliadis and Simitsis high-level model for ETL activities with attribute inclusion – extracted from (Vassiliadis et al., 2002a). 19
Figure 3. Trujillo's proposed UML extension example for ETL conceptual modeling (Trujillo and Luján-Mora, 2003). 21
Figure 4. A BPMN4ETL metamodel excerpt as presented in (El Akkaoui et al., 2011). 24
Figure 5. Graph ontology representation example, describing mapping between source and target DW – extracted from (Skoutas and Simitsis, 2007). 24
Figure 6. An ETL Process composed by Patterns as presented in (Köppen et al., 2011). 26
Figure 7. Representing the structure of a pattern. 31
Figure 8. Representing main pattern components in OWL. 32
Figure 9. Summary of the ontology graph. 35
Figure 10. The basic Alloy specification for defining ETL patterns. 37
Figure 11. Mapping an instance describing the relationship between two "DataObject". 38
Figure 12. The main core of the pattern specification. 39
Figure 13. A CDC pattern and an instance excerpt using Alloy. 43
Figure 14. A CDC instantiation example representation using BPMN. 45
Figure 15. Describing a SCD pattern and an instance excerpt using Alloy 49
Figure 16. a) An ETL scenario representing the data structures involved with a SCD operation; b) The schematic process of a SCD instantiation represented in BPMN. 50
Figure 17. Describing a DQE pattern and an instance excerpt using Alloy 54
Figure 18. a) Data structures used in an DQE pattern application; b) Schematic view of a DQE instantiation process in BPMN. 55
Figure 19. Describing a SKP pattern and an instance excerpt generation using Alloy. 57
Figure 20. A SKP instantiation process represented using BPMN. 59
Figure 21. Alloy specification describing SKG pattern and an instance excerpt generation 63

Figure 22. A BPMN schema for a SKG pattern instantiation. 64
Figure 23. Alloy specification describing Data Aggregator pattern and an instance excerpt generation 65
Figure 24. BPMN example algorithm for a DA instantiation. 66
Figure 25. Describing a DCI pattern and an instance excerpt generation using Alloy. 67
Figure 26. An example of a DCI instantiation process using BPMN. 70
Figure 27. Alloy specification describing IDL pattern and an instance excerpt generation 72
Figure 28. BPMN example algorithm for the IDL instantiation 73
Figure 29. Alloy specification describing "Throwable" pattern and an instance excerpt generation 76
Figure 30. BPMN example algorithm for the Exception handle modules using DQE pattern. 77
Figure 31. The Figure 30 BPMN process variation using a BPMN error event with a rollback configuration 78
Figure 32. Alloy specification describing Log pattern. 80
Figure 33. Excerpt of the SCD pattern specification for dynamic behaviour simulation. 82
Figure 34. Pattern language example for ETL patterns representation. 84
Figure 35. The DW star schema used for the solid urban waste collection case study. 91
Figure 36. BPMN elements that can be used for pattern-based ETL conceptual modelling. 95
Figure 37. ETL extraction phase representation – Layer 1 (a), Layer 2 (b) 96
Figure 38. The ETL extraction process – Layer 3. 97
Figure 39. ETL Transformation layer – Layer 2. 99
Figure 40. ETL transformation layer: Layer 3 subset (part 1) for spreadsheet data conciliation. 100
Figure 41. ETL transformation layer: Layer 3 subset (part 2) for spreadsheet data conciliation. 101
Figure 42. Picking data schema transformation. 102
Figure 43. ETL Transformation layer: Layer 3 for spreadsheet data integration. 104
Figure 44. Examples of a PL4ETL basic pattern configuration syntax (a) and a language instantiation (b). 109
Figure 45. PL4ETL configuration for "Throwable" and "Log" components. 110
Figure 46. Ontology interpretation and DSL specification. 112
Figure 47. Specification of rules using Xtext for a SCD pattern instance. 113
Figure 48. A configuration excerpt for a SCD pattern instance. 114
Figure 49. Physical model generation components overview 116
Figure 50. Excerpt of the several template layers to support ETL patterns physical generation 117
Figure 51. Excerpt of an Apache Velocity template for a SCD transformation. 119
Figure 52. Physical model for a SCD instance for populating the Container dimension 120

List of tables

Table 1. A Simplified View of a Transaction Log File.....44
Table 2. Log table sample data for the DQE pattern.....81
Table 3. A small excerpt of a picking data spreadsheet for a given month and city.....91
Table 4. Spreadsheet-to-Data Mart mappings.....92
Table 5. Patterns description for the Extraction process presented in the Figure 38.....98
Table 6. Patterns description for the Transformation process presented in the Figure 40 and Figure 41.....103
Table 7. Patterns description for the Integration process presented in the Figure 43.....106
Table 8. DSL constructs overview.....108

List of Acronyms

BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
CDC	Change Data Capture
DA	Data Aggregator
DBMS	Database Management System
DCI	Data Conciliation and integration
DQE	Data Quality Enforcement
DM	Data Mart
DS	Data Source
DSA	Data Staging Area
DSL	Domain Specific Language
DSS	Decision Support System
DW	Data Warehouse
DWS	Data Warehousing Systems
ER	Entity-Relationship
EPL	ETL Pattern Layer
ETL	Extract, Transform and Load
IDL	Intensive Data Loading
JSON	JavaScript Object Notation

LM	Log Monitor
MDA	Model-Driven architecture
OLAP	On-line Analytical Processing
OMG	Object Management Group
OWL	Web Ontology Language
PA	Presentation Area
PIM	Platform Independent Model
QVT	Query, View, Transform
RDF	Resource Description Framework
SCD	Slowly Changing Dimension
SOA	Service Oriented Architecture
SK	Surrogate Key
SKG	Surrogate Key Generator
SKP	Surrogate Key Pipelining
SQL	Structured Query Language
UML	Unified Model Language
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Context Overview

Nowadays, software systems applications occupy a very relevant role in all the aspects of our society, helping people performing a lot of activities, ranging from simple tasks to very complex and demanding critical systems. Companies live in an increasingly competitive market that demands new challenges every day. Automation, availability, and development of computer technology to support decision-making processes have been some real major success factors for organizations. The business evolution relates directly to the substantial amounts of data that progressively increase over time to cover several business areas.

Owning substantial amounts of data alone is not enough anymore to get practical results. Data need to make sense and be properly framed and aligned with the organization goals to become useful information for managers, who daily make decisions that affect the organization and consequently the lives of many people. In fact, data are nowadays the basis for growing business activities, providing a way to identify patterns that allow for the improvement of production processes as a way to create value by reducing the resources needed and increasing the quality of products and services.

It was in contexts like this that Decision Support Systems (DSS) emerged for supporting the acquisition, storage, analysis and data sharing processes, using the acquired data to simplify and improve decision-making processes. A DSS system must support multiple types of decisions to meet different requirements based on operational data:

- Operational - decisions that affect the organization in small periods of time.

- Tactical - decisions that affect the business for long periods of time.
- Strategic - decisions that affect the company's market strategy where the system operates.

To have a more concrete view about these types of decisions, we can see an operational decision as something that, for example, requires an analysis regarding stocks, suppliers, and deliveries; a tactical decision that covers, for example, information about sales trends or market research; and a strategic decision that is related to the emergence of new business areas or even to a change of business scope.

The Data Warehousing Systems (DWS) allows for the storage of large amounts of data, using specific systems architecture for advanced data analysis tools to support the requirements of several user profiles. According to Inmon (2005), a Data Warehouse (DW) is a special type of database that holds a large amount of data oriented to some subject matter, rather than being oriented to the management of some area (subject-oriented). These data should not change once entered in the warehouse (non-volatile), they should be time-variant to discover business trends, and be provided from several sources that will be used to feed the DW repository in a consistent format (both in data structure and data instances), and integrated. Every time a data conflict is identified in a data integration process, specific policies should be applied to handle them in the most appropriated way. Any data integration process involves the notion of consistency over current and historical data, enabling deeper analysis and process optimization for supporting decision-making processes. Data integration also provides a unified view to system users, avoiding common inconsistencies that typically are found when different storage formats are used. Problems such as loss of integrity, different data formats and data redundancy are common. Data integration is provided from one of the most important DWS components: The Extract, Transform and Load (ETL) system. These populating systems allow for data extraction from source systems to temporary storages for data processing, ensuring data consistency to support the load of well-formed data - all this preserving the architectural rules defined by DW's requirements.

Usually, ETL systems design and implementation involve the development of complex tasks that interact normally with several other DW components. ETL systems are a very particular kind of software, and like most of software systems, must be planned carefully. Their life cycle covers the most typical phases of common software development process. Software systems modelling (Sommerville, 2004) activities help to produce more abstract models that are very valuable tools, whose utility can be demonstrated in almost all phases of software system development. All the models produced, whether conceptual or logical, provide us a very useful kind of communication that we may have with future users of the system under development. Although being an excellent groundwork for system requirements validation, ETL development addresses very specific needs (Kimball and Caserta, 2004). Each DWS implementation serves its own user community linked to a specific set of business and decision-making processes supported by oriented data models.

Additionally, ETL designers frequently deal with legacy systems that provide limited mechanisms for data extraction and data inconsistencies, resulting from years of business change and evolution. All this requires extreme care and concern from ETL architects and software engineers in all the aspects related to architecture, design, and implementation. Moreover, current commercial tools for supporting ETL implementation processes provide specialized transformation tasks developed based on a specific methodology and architecture. For that reason, ETL development results in very complex data migration processes that require specialized personnel to deal with tool specificities. Additionally, resulted processes are represented using proprietary notations and implemented according to specific architectures and philosophies, making them error-prone by nature and difficult to maintain/migrate when different tools or approaches are needed.

1.2 Motivation and objectives

Today, developing information systems is a very challenge task, often a complexity work that engages high-level personnel with knowledge and skills in different areas (Scacchi, 2002). Usually, to start a software development process, software engineers made some preliminary drafts to obtain a first view of the system and the necessary requirements to be validated. Even in an initial version, software models describe the target system, regardless of the methodology or the technology used for implementing them, revealing the most basic needs of the project in a very clear and precise way (Losavio et al., 2001). Typically, software models are described using well-defined languages, often having their own syntax and requirements, meaning and notation, which are the most suitable for computer interpretation (Kleppe et al., 2003). The initial definition of more detailed software models provides an abstraction that can be useful throughout the development cycle, ranging from design to validation, ensuring clarity and facilitating the communication between technical and non-technical users.

Any ETL development process faces enormous and complex problems, frequently not expected, in all its development stages, not only on the stage of extraction of data but also in their subsequent stages of cleaning, transformation, and loading. ETL systems are considered as a very particular kind of software and its initial definition, with a detailed model, is extremely useful throughout the process development, ensuring to designers, architects and developers a very useful guide. Each ETL system is a specific case, and, for that reason, the process reusability it is quite hard to accomplish. Typically, specific data clients arise different ways of making decisions that usually require different dimensional models and consequently different data sets (Weske et al., 2004). Thus, even if we adopt a same standard solution to implement a DW (which happens often in the retail area), people have diverse ways of thinking, which means that original data models already built in a previous solution should be adjusted accordingly to the requirements of those decision makers. Apart from the difficulties in adopting a standard solution in the data model definition, the implementation of extraction processes

for different operational sources imposes (naturally) different extraction strategies, raising several problems that impose readjustments in the ETL system configuration and tasks involved with.

We believe that the success of a DWS deeply depends on of the adequacy of its populating system, which imposes extreme care and concern in its planning, architecture, design, and implementation. Researchers in the field have been committed to the study and construction of methodologies and languages to support ETL development, mainly in the conceptual and logical modelling activities. All proposals, which were developed till the present, are very interesting and useful. However, they all leave a path quite long and difficult enough for discovering how to generate an ETL physical implementation that can be executed accordingly to the conceptual specifications that were made previously.

The use of software reuse techniques is nowadays a very common and important technique to develop software systems based on the composition of components. These preconfigured components are sometimes based on well-known and validated design patterns that describe abstract solutions for solving recurring problems. Since the ETL development lifecycle shares the main steps of the most typical phases of any software process development, this work proposes the use of patterns (Gamma et al., 1994), as a way to increase reliability, reduce development risks and enhance standards compliance, providing a more flexible approach to support ETL implementation, from early development stages to its physical implementation. The main idea is to organize ETL tasks into clusters of tasks that can be executed as a single block. The set of "standard" clusters are used to represent some of the most common techniques used in real world ETL scenarios (meaning that there can be found commonly in almost ETL systems for real world applications). Based on a set of configuration parameters, the tasks that compose each cluster should produce a specific output. They act as routines that are available to be used, being executed with the aim to simplify ETL design and development, reducing the implementation errors and the developing time that usually occurred in ETL systems development and deployment.

One of the main objectives of this work is to slightly shorten the distance between more abstract models and its real implementation. The designed pattern-based approach maps typical design techniques - e.g. *Surrogate Key Pipelining* (SKP), *Slowly Changing Dimensions* (SCD) and *Change Data Capture* (CDC), to configurable components that can be adapted to specific application scenarios. Using the *Web Ontology Language* (OWL) (McGuinness and van Harmelen, 2004), an ETL pattern-based ontology was designed to support ETL systems operational requirements and to describe each pattern configuration, enabling its use for a more clear ETL modelling and representation. Basically, it was designed as an intermediate layer to separate technical knowledge, typically used in commercial tools, from the domain knowledge used by decision-makers (McGuinness and Wright, 1998). Users should be able to describe what to do without describing how it will be done. At the same time, users should be capable of extending and changing patterns behaviour

without affecting the entire system and by consequence, the remaining patterns. These are two of the most critical issues we had pursuit in this wo7rk.

Based on the pattern ontology specification, we also wrote a *Domain-Specific Language* (DSL) to support the translation of ontology's lexical tokens to a format that can be executed in a specific ETL tool. The proposed DSL provides a way to represent and formalize each pattern configuration, describing patterns instances based on their operational requirements. To cover physical-mapping logic, specific templates were developed to encapsulate the conversion process. Thus, the use of a template engine allows for conceptual and logical primitives mapping to its correspondent physical representation that can be executed using commercial tools (Oliveira and Belo, 2015a).

1.3 Thesis and Research Questions

When dealing with an ETL development, most designers and architects plan the work to be done considering the tool they will use for implementation. They can choose an existing commercial tool, develop their own tool or can simply follow a programmable approach based on ad-hoc procedures. This means that the initial plan and prototypes designed are in most cases algorithms developed based on a technical language or even a specific programming language or architecture. This approach is by nature error-prone, hard to understand and communicate, limiting process reusability and evolution. Additionally, most of the initial efforts done using software modelling language (such as UML) are sometimes ignored by ETL developers because they are un-natural to represent ETL specificities and are viewed as an extra effort that it's hard to reuse due to the disparities between the two representations. All this contributes to the use of ad-hoc strategies for ETL planning that results in complex software, difficult to implement, reuse and maintain.

The presented thesis introduces a pattern-oriented approach for ETL development and implementation. The use of preconfigured components to support the ETL development is useful not only at conceptual levels, but also for the generation of execution primitives with the possibility to use them on powerful frameworks provided by commercial ETL tools. We believe that common ETL development problems can be avoided or at least minimized using more abstract components that represent common operational tasks, hiding their internal states, processes, and variables. That way, a new abstraction layer is created where designers are not describing how a particular task will be implemented but how that task will be used. The development team build the ETL system using selected components and integrate them together as a single unit based on a well-defined architecture.

Although the general idea of using patterns for ETL development is not new, its application to ETL context for a complete development approach delivers many and varied challenges that should be addressed. The ETL conceptual representation is an important issue since provides a high-level

representation of ETL tasks. Many of the main contributions in the area address the use of specific modelling languages to cover ETL specific issues, or the application of existing general-purpose modelling languages with adaptations or even extensions. Working with coarse-grain components leaves the complexity of specific transformation tasks representation outside the process scope. We want to represent patterns as "black boxes" that should be coordinated by a powerful language that should preserve simplicity and ensure familiarity to users, provide expressiveness for representing all coordination aspects, and dispose a solid serialization format to allow its computer interpretation.

With BPMN, the requirements are quite well supported, however, there is still a major important aspect considering the enrichment of the patterns represented at the conceptual level. While it is helpful to represent patterns as abstract components in conceptual models, it is still necessary to provide additional information to each one to allow its mapping to execution primitives, regardless the infrastructure used. For that, we developed an independent domain-specific language covering each pattern specificities as a way for enriching conceptual primitives. After modelling phases, the "real" ETL process should be generated to take advantage of existing tools infrastructure. Since most the existing tools allow for the representation of their processes and configuration using specific formats, this gap can be solved using Model-to-Text transformations that based on a set of specific transformation templates, encapsulate all the logic behind the conversion from the conceptual and logical representations to a target tool representation.

The approach presented in this thesis lead us to several advantages that minimize some of the problems described before, namely:

- A clear distinction and independence between abstract models and their correspondent physical representation abstracts users from implementation details and simplifies initial ETL drafts. Designers are not talking anymore about "join" the table "A" with table "B" and filter the "addresses". Instead, they are describing that two entities will be used to find the "addresses" they need. This contributes to process understandability, better planning and improved communication and productivity.
- The development process is now framed in different phases, contributing to process validation before proceeding to subsequent phases. Additionally, following the "traditional" approaches, the tasks used are highly dependent, which compromises the work distribution between teams and consequently productivity. With the pattern oriented approach, each pattern represents an isolated component, which means that different teams can be working in project different parts (Microsoft Patterns & Practices Team, 2009), keeping a high-level communication.

The framework provided by pattern-oriented approach allow for ETL developers a set of components with clearly defined functionality based on well-known techniques that cover data extraction, transformation and load from information sources to target data warehouse repository. The ETL

patterns are assembled in the context of well-defined architecture using specific interfaces for communication. So, changes can be implemented by replacing or eliminating components without affecting the remaining workflow logic. Thus, most of the development time is spent customizing an already existing component instead of programming it.

All these aspects contribute to the development of more consistent, reliable, flexible and reusable processes, improving productivity and consequently the cost needed for ETL implementation. Moreover, process reusability is now facilitated inside and outside project scope, spreading development costs across projects with some level of similarity. Since we are working with a flexible, reliable and reusable design methodology that guides system implementation in smaller parts, the complexity is significantly decreased and the testing methods are simplified, influencing positively the target system quality.

Despite several positive contributions, the pattern-oriented approach can also face some challenges related to its application:

- Patterns flexibility can be an issue when unexpected scenarios are faced. During ETL development several specific or not planned situations can occur that may require slight modifications to patterns behaviour. This may turn the approach not suitable for a given specific scenario or can promote the use of workarounds that may affect ETL construction. Moreover, since a large variety of possible situations should be planned and supported by each pattern internal configuration, additional process logic and functionalities that are not needed for some cases are embedded in patterns composition, which can make this approach less efficient when physical models are generated.
- The complexity related to each pattern composition and configuration can make this approach painful for some users due to the complexity of patterns configuration due the effort for learning a new language with very specific properties to support pattern generation. This means that errors in pattern configuration can have unpredictable side effects that may be difficult to understand.
- The pattern configuration supported by a domain specific language means that patterns do not exist at the physical level. Thus, its reuse and interchangeability is available by the use of pattern generator engine, implying pattern rebuild when some change is applied (Sametinger, 1997). Otherwise, if alterations are only made at physical level, conceptual and logical representation will be outdated and useless for posterior maintenance phases.

1.4 Contributions of the thesis

The use of a pattern-oriented design covering the ETL conceptual modelling, its enrichment using logical primitives and finally its generation to physical programs, represents the main contribution of

this thesis as a way to fill the gap between conceptual and physical primitives. This methodology is supported by the concept of patterns: a set of configurable components that replaces all the granular tasks typically used. To achieve this, a study of the most cited and relevant ETL approaches covering the three design phases will be conducted.

In addition, the ETL patterns formal specification to build ETL workflow will be studied, revealing the main components and requirements related to their functional properties (Oliveira et al., 2016; Oliveira and Belo, 2017a, 2016). The framework is composed by preselected components that cover the main ETL activities, namely:

- Changing Data Capture (CDC – section 3.3.1) pattern, representing the extraction procedures to carry source data to Data Staging Area.
- Slowly Changing Dimension (SCD – 3.3.2) pattern, encapsulating the logic to preserve historical data.
- Data Quality Enhancement (DQE – 3.3.3) pattern, representing a set of data transformations required to clean, conform and standardize data, considering the target repository constraints.
- Surrogate Key Pipelining (SKP – 3.3.4) pattern, representing the process of loading fact data into the data warehouse based on dimensional surrogate keys.
- Surrogate Key Generator (SKG – 3.3.5) pattern, representing the logic behind the generation of new surrogate keys for source data.
- Data Aggregator (DA – 3.3.6) pattern, representing for the generation of aggregate data to support data warehouse populating processes.
- Data Conciliation and Integration (DCI – 3.3.7) pattern, representing the conciliation and integration logic needed to obtain an integrate view of the data gathered from heterogenic sources.
- Intensive Data Loading (IDL – 3.3.8) pattern, representing efficient strategies to load data from data staging area to the target data warehouse repository.

Based on those specifications, a design methodology is presented covering the main ETL development phases:

- Conceptual modelling (Belo et al., 2014; Oliveira et al., 2014, 2013a; B Oliveira and Belo, 2013; Bruno Oliveira and Belo, 2013a, 2013b, Oliveira and Belo, 2015b, 2014a, 2014b, 2012), representing the initial step for ETL implementation. Models are produced based on the pattern concept, revealing at this point, the main interaction between the selected patterns. The coordination mechanisms are represented and supported by a powerful modelling language to express the workflow constraints.

- Logical modelling (Belo et al., 2015b; Oliveira et al., 2013b; Oliveira and Belo, 2015a), representing the enrichment of the ETL conceptual modelling developed in the previous step using a specific configuration language to provide additional information related to the behaviour of each pattern.
- Physical modelling (Belo et al., 2015a; Oliveira et al., 2015), representing the last ETL development step responsible to enable the generation of physical models based on the previous specifications. We propose a Model-to-Text approach based on specific templates that encapsulate the logic behind the generation of each pattern to target physical architectural philosophy.

To demonstrate the application of all the aspects related to the pattern-oriented approach, an example based on a real ETL scenario is presented, covering the main aspects of the proposal presented in this thesis.

1.5 Document Organization

In addition to this chapter, this thesis has four more chapters describing the different development stages to cover the entire ETL development life-cycle adopting a pattern-oriented approach. Thus, it is organized as follows:

- **Chapter 2 - ETL Modelling and Implementation.**

  The main objective of this chapter is to describe the ETL process life cycle in a detailed manner, as well as to discuss the contributions made by many authors in the field of ETL modelling and implementation.

- **Chapter 3 – A Pattern-Based Approach for ETL Development.**

  In this chapter, it is presented the pattern-based ontology we conceived and developed to support the main operational components of each pattern we designed, covering the main properties and restrictions that can be used to enable its posterior usage and instantiation. Each pattern specification is also formally described using Alloy, allowing its correspondent validation before proceeding to its correspondent physical representation.

- **Chapter 4 – From Conceptual to ETL Physical Implementation.**

  Based on a case study, we present and discuss the application of the pattern-oriented approach presented previously, demonstrating how patterns can be used for the most relevant tasks in an ETL process. For conceptual representation, we used the BPMN language for designing patterns workflows, enabling not only the representation of pattern instances but also to describe the constraints related to its coordination. The logical representation is also presented using a specific configuration language, which was specially built to cover each

pattern operational requirements. Finally, the physical generation of the ETL system is accomplished using specific templates to support the generation of ETL physical instances based on the conceptual and logical specifications defined, which can be executed using a selected ETL commercial tool.

- **Chapter 5 – Conclusions and Future Work.**

  This chapter presents a detailed discussion of the work done in this thesis, approaching the most relevant issues and describing its main advantages and practical utility in the design and implementation of real-world ETL systems. Finally, some pertinent findings are presented and discussed, and pointed out some future directions to extend this work to new application scenarios and ETL use-cases.

Chapter 2

ETL Modelling and Implementation

2.1 About ETL Processes

Usually, a DW stores massive and integrated data representing an organizational view over transactional systems, providing an understandable picture of all business activities. The process to design and implement a DW system represents a complex task that consumes a lot of time and resources. It involves the definition of relevant business requirements, the data that must be needed, as well as the necessary infrastructures to support the implementation of the DW. There are two main approaches for storing all the necessary data to support decision-making processes when designing a DW, namely: the normalized approach (Inmon, 2005) and dimensional approach (Kimball and Ross, 2002).

Using the Inmon approach, data are stored following the most common normalization rules up to third normal form and is typically designed using traditional techniques, such as the Entity-Relationship (ER) diagrams. The normalized structure divides data into entities, which creates several (dozens in complex cases) tables in a database. This approach endorses that a DW should be designed following a top-down methodology to include all organization data, which results in a very complex project with a very broad scope. This means that the time and efforts needed are substantial since the project implementation starts until the point that end users can take advantage of the DW. After the creation of DW main structure, specific and subject oriented "departmental views", also called Data Marts (DM), are built to serve analytic needs from each department involved with. This approach is commonly referred as a top-down approach since global business needs are analysed from very initial phases to support the design and implementation of the DW.

Contrasting to the Inmon's approach, Ralph Kimball (Kimball and Ross, 2002) proposed a dimensional approach for DW building, defending a non-normalized structure supported by a well-organized set of DMs. A DM is a focused database modelled following the dimensional modelling techniques to satisfy specific business needs, often associated with a particular business division on an organization. In Kimball's bottom-up approach, is suggested an incremental developing process, starting first by designing the most critical data marts that serve specific analytic needs, each on created autonomously and later combined into a broader structure: the DW.

Over the years, several works comparing both approaches - normalized and dimensional - were done by several researchers. Some conclusions about the use of each design methodology for implementing a DWS relies primarily on the organization requirements (Breslin, 2004). However, both approaches tend to be similar in mature DW development phases. It is globally accepted that dimensional approach tends to return faster results, allowing incremental evolution and reduce financial and human costs when compared to the normalized approach (Ariyachandra and Watson, 2006). In fact, the failure experiences related to the top-down approach shows that the estimated high costs discourage DW implementation - it is very difficult to identify all requirements for each department involved and the time needed to return some feedback or at least intermediate results is too long. By these reasons, the dimensional approach will be used to show the applicability of the presented ETL pattern-based approach we present and discuss in this work. All the ETL patterns presented were developed to support specific techniques that are typically used jointly with a dimensional approach for the DW development. For example, the Surrogate Key Pipelining (SKP) process is a typical procedure applied in dimensional schemas and it is considered in the pallet of patterns we designed, representing the necessary procedures in populating dimension and fact tables.

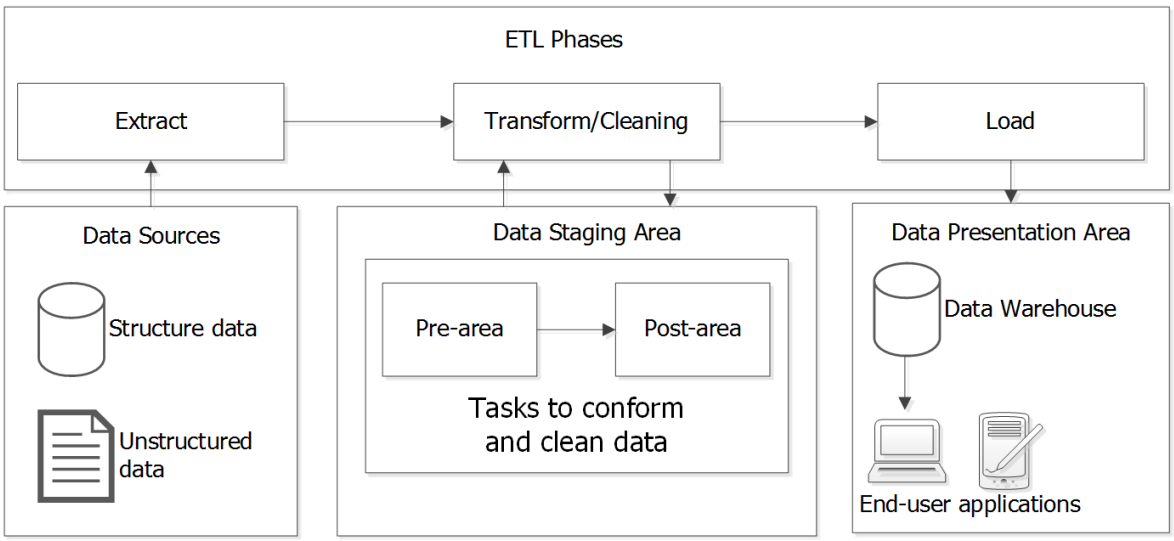


Figure 1. A typical ETL system architecture.

2.2 An Architecture for ETL

According to Kimball and Caserta (2004), an ETL process development consumes about 70% of the resources needed for a typical DWS implementation and maintenance, making it a critical component on the DW reliability. However, the main purpose of an ETL system is quite simple to understand: extracting data from source systems, updating and standardizing data provided from heterogeneous data sources based on a specific set of schema rules, providing its integration and consistency, and finally loading data into the target DW. Typically, an ETL architecture can be organized in three distinct but inter-related parts, namely extract, clean and conform (transforming for short) and load (Kimball and Caserta, 2004), operating over three distinct DW layers: Data Sources (DS), Data Staging Area (DSA) and the Presentation Area (PA) (Kimball and Caserta, 2004) (Figure 1).

2.2.1 Extraction Processes

An organization may have multiple data source systems, some of them providing non-relational storages, that serve all its operational needs. Naturally, the storage structure of each data source may diverge, influencing the entire ETL process, mainly affecting the extraction and transformation phase. Additionally, data must be transformed to match the schema of the DW, serving specific requirements to support decision-making processes (Berson and Smith, 1997). Usually, the ETL process starts with specific data extraction procedure in a specific set of sources, considering their own schema restrictions that need to be carefully managed to extract relevant data. Frequently, the development process deals with distinct operational and database systems or even non-database systems such as text files or spread sheets, just to a few. Particularly, non-database systems provide a less formalized storage structure, and in some cases restricted or obsolete technology, which increases the complexity and the difficulty of the ETL extraction process. When planning the extraction phase, both initial and incremental data extraction (Kimball and Caserta, 2004) processes should be considered.

When the initial extraction is performed, the DW is empty, and the ETL process must do a complete extraction of all data from each data source that are related to the decision processes to be supported. Then, based on that criterion, specific Change Data Capture (CDC) techniques for identifying and tracking data that has been changed in the information sources of the system are used. Due to the much smaller data volume that these methods extract, they have better performance when compared to others that use differential extraction strategies. However, sometimes is too difficult to identify the data that was changed due to the limitations of the technology used the sources' system. For cases like these, some of the following techniques are typically used:

- **Audit columns**, which represent some control attributes that are appended to each table such as date and the operation performed (insert, update or delete) in the source data. For extracting changed or new data, the modified date of each record should be used to identify relevant data since last successful data extraction. Audit columns are usually populated by database triggers, database scripts or specific front-end applications.
- **DBMS (Database Management System) transactional logs**, which store all the transactions performed in source systems using (proprietary) database logs. Based on scheduled points, the database transactional log allows for identifying actions that affected data and need to be preserved in the DW, some of them as event history records.
- **Audit tables**, which are special tables used in transactional databases to store new or modified records along with specific control columns appended for representing metadata records (this alternative is very similar to audit columns). With this technique new or changed data are stored using specific tables, serving as buffers to support ETL extraction procedures that should be cleaned after the extraction of all tagged records to the DSA - for this reason, specific date attributes don't need to be compared since audit tables only hold current and unprocessed data.

The ETL team is responsible for choosing the best suitable CDC strategy for data extraction. Each technique has its own specificities and must choose according to the DW environment requirements. Moreover, the source systems availability can affect the choose CDC process, mainly if we have to respect a specific (extracting) time window. Despite daily or month data extraction cycle being common, sometimes there is a need to make decisions based on a limited time scale that become incompatible with a traditional approach. For example, real-time decisions require real-time ETL systems. These systems usually extract smaller chunks of data, very frequently or even in streaming mode (Vassiliadis and Simitsis, 2009).

At the end of the extraction step, extracted data are stored (ideally) in an intermediate storage area between operational systems and the DW. The Data Staging Area (DSA) is a location (frequently a conventional database system) where data gathered from data sources remains during the execution of a set of very specialized tasks (cleaning, surrogating, conciliating, etc.) before being loaded and integrated into the target DW (Kimball and Caserta, 2004). That way, when transformation step takes place, the operational systems normal functioning is not affected by cleaning and conforming procedures. Figure 1 presents a DSA example representing the "pre" and a "post" staging areas. The *pre-area* represents a repository where extracted data will be firstly and temporarily stored for being iteratively used by transformation and cleaning tasks. After the execution of these tasks, data is stored in the *post-area*, indicating that can be loaded into the DW. To optimize all data transactions, each DSA sub-area is supported by simplified data structures based on relational tables or other simpler technology, such as text, XML or JSON files (if ETL team decided to materialize data).

2.2.2 Transformation Processes

The Data transformation, often known as Cleaning and Conforming (Kimball and Caserta, 2004), is the most important phase of the entire ETL process. At this stage, data that come from information sources is transformed to obtain accurate, unambiguous, consistent and complete data. Inconsistent records are rejected by specific cleaning procedures that only approve data according to the DW's requirements, avoiding data duplication and ensuring that business rules are properly implemented. Handling data coming from several sources, frequently heterogeneous, should also be carefully planned to solve data conflicts between them and to find records with the same meaning, but sometimes having different structures or different data types.

To address such problems, a large variety of procedures are typically applied based on well-known cleaning and conforming techniques (Rahm and Do, 2000) used to align source data structures to the requirements of the target DW schema. Initially, simple attribute decomposition and text transformation tasks (e.g. capitalization of a word or a sentence, or renaming or adding an attribute) are usually applied to enforce domain constraints associated to an attribute or set of attributes. To integrate data, it must be ensured their compatibility to prevent name conflicts between different data sets. For this, data dictionary repositories are frequently used to support these cases, and along to specific transformation rules represent the support meta-data needed to ensure transformation phase. This process may support several integration scenarios involving, for example, handling new data records about the same object but having contradictory values.

Additionally, data changes across time that are applied over each DW's dimension are also an important issue, since attributes with history preservation should be handled according to some SCD policy (Kimball and Ross, 2002) in order to maintain a consistent view over time. SCD procedures can be also considered as a kind of a load task. However, considering the ETL architecture presented, after the application of proper data transformation and conciliation procedures, data is stored to a DSA post-area for posterior loading in the DW.

2.2.3 Loading Processes

The last step is to deliver the data that provided from transformation tasks to the DW. Both dimension and fact tables are filled with transformed data used to serve analytical tools covering specific decision-making requirements. Considering all dimensional schema structural constraints previously defined, fact tables must preserve referential integrity between the primary keys of dimension tables and their corresponding foreign keys in the fact table. Dimensional keys represent an important strategy, mainly when data come from multiple sources and are used for data delivery. Records relating to the same data subject can coexist between different data sources, with different natural keys, of course. For example, a same client can have distinct primary keys on each

operational data source where he is referred. Thus, and based on a set of predefined rules (support meta-data), the ETL process should be able to identify identical records and handle them appropriately (typically in transformation step). For that reason, DWs typically use context independent fields, known as surrogate keys (SK), for each dimension's primary key. Often, these artificial keys are implemented using auto-increment fields based on minimal data structures with small integers types, aiming to improve star joins performance between dimension tables and fact tables.

To ensure the relationship between natural keys and surrogate keys we can use mapping tables or some special columns defined in the structure of dimension tables. Thus, when data need to be loaded into the DW, mapping tables are queried for retrieving the respective SK for each record, while for new records new specific SKs are generated and correspondent mapping tables updated. Finally, fact tables can be loaded using SKP or Intensive Data Load processes (IDL), delivering data to a DW based on efficient algorithms and providing the necessary basis for data browsing and analysis, aimed to support the generation of reports on the presentation area by analytical tools. It is also important to note that when an ETL process is being developed, some procedures may be performed in distinct phases or may spread over other ones. Based on the approach taken, some cleaning and conforming procedures can be embedded in the extraction step or even SCD procedures can be applied in the transformation tasks before they are stored to the DSA *post-area*.

2.3 Kimball's ETL Lifecycle Development

Like any kind of software, the use of standard methodologies, models and notations facilitate and improves ETL system development, providing a very useful guide to any ETL team to reduce problems related to the preliminary system development, and its posterior evolution, implementation and maintenance. Despite representing a critical component when building and designing a DW system, it is a nonsense that currently there still is a lack of a standard methodology for supporting ETL system development phases. Initially, Kimball and Caserta (Kimball and Caserta, 2004) addressed this issue providing a methodology involving a set of steps for supporting and guiding the whole ETL development process lifecycle. In this methodology, the first step considers building a high-level plan in which data sources and a potential target schema for data migration are identified. This first layout is maintained at a higher level of abstraction providing a superficial view encapsulating a few technical specifications. In this way, communication with the development team and other participants who are not familiar with technical details will be facilitated.

When the initial plan is produced, it will be necessary to choose an ETL tool to support process implementation. The generated documentation from such applications is one of the greatest advantages of its use. However, these tools provide its own specific implementation for many of general components used regularly in an ETL process development. The proprietary notations and

methodologies followed by these tools improve communication and representation of processes between the ETL development team, increasing the performance, quality and consistency of the entire development process. Due to their nature, these models are very detailed since they consider many implementation issues. Specifically, at initial design phases, there is often a need to share the initial models with nontechnical users to discuss preliminary system requirements. Naturally, nontechnical users have serious difficulties for understanding the models generated by these tools, especially in what is concerning for understanding the meaning of each operation and the results of the final process. Additionally, if for any reason software engineers change the supporting tool, they will need to migrate current ETL processes to the new tool environment, spending additional efforts to understand the new notation, mechanisms, and methodology, of course. All this implies extra financial and human resources to produce equivalent results. There are several tools on the market, each one with its own architecture and methodology, see for example the cases of Microsoft Integration Services[1], Oracle Warehouse Builder[2], Talent Open Studio[3] or Pentaho Data Integration[4].

After the phases of requirement definition and tool selection, it is necessary to develop and document a set of basic strategies for the most common ETL activities, namely: data extraction methods from each data source, the DSA storage structure, the necessary policies to ensure data quality metrics and a way to handle records without approval by quality control, handle attributes with variation, study the source systems availability and define strategies for data temporary storage on DSA. Posteriorly, the process for populating each dimension and fact tables should be detailed. The data granularity of each dimension hierarchy should be preserved and a plan describing the sequence of tables that will be loaded as well as the sequence of transformations to be applied to data should be described. Initially, transformations are performed based on dimensions storing current data, and then applied to the dimensions that store each row of data related to some point in the past (maintaining a history of changes). At this point of the process, we need to have already guaranteed data compatibility using specific transformations such as data type conversions, 'NULL' values replacement, checking data inconsistencies and validation of data granularity in dimension hierarchies, and others. From the moment that the data are prepared, it can be loaded into the respective dimension tables according to the mapping rules previously defined.

Once dimension tables were properly loaded, fact table records can be loaded. There should be a special attention to ensure that referential integrity rules between fact and dimension tables are preserved. For that, we can use a typical ETL procedure known as surrogate key pipelining for replacing all the natural keys used by surrogate keys for each referred dimension. Finally, the last steps involve the creation of aggregations for each dimension, the optimization of the transformations

[1] http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/integration-services.aspx
[2] http://www.oracle.com/technetwork/developer-tools/warehouse/overview/introduction/index.html
[3] http://www.talend.com/products-data-integration/talend-open-studio.php
[4] http://kettle.pentaho.com/

workflow and the creation and scheduling of tasks for handling unexpected data or error scenarios to solve and prevent data anomalies.

2.4 ETL Development Approaches

An ETL system development plan can start by a brief sketch of its architecture, components, and services, based on some ideas, proposals, or practical needs. From this initial sketch, we can develop some working strategies for requirements gathering and for the specification of system features. Even when they are simple sketches, initial models are a good deal of work and dialogue between development team's members and future users, providing a nice basis to sustain all subsequent work and some confidence about the success of the project under development. In other areas of work, such as Civil Engineering, it is common to design and develop models for projecting a given system. Models are excellent working tools for any construction project since they allow for revealing what the project will entail in a very clear and precise way. However, a model does not provide a detailed view of all project's aspects, nor address all the issues involved with the implementation of its modules. It is a "final product" rather imperfect. Even so, a model is a very useful instrument, especially for end-users.

The utility of a software modelling (Losavio, 2002; Scacchi, 2002) is greater the better the image it provides regarding a possible project outcome. However, its level of detail can never be equal to the final product, since if that happen the sketch would be the final product, obviously. On the other hand, an inaccurate model is not a very useful tool, since it leaves many questions in the air, for discussing and answering. Achieve the best compromise on all these issues is not easy. Considering that the ETL process is a critical component in the successful implementation of any DWS, the initial definition of a detailed model will be extremely useful throughout the entire process development, installation, and validation, providing to its architects and engineers a very useful guide.

The design and implementation of this component involve the development of complex activities that interact with the clear majority of the DWS's components. These activities operate on data that are stored in transactional systems, whose availability is usually constrained and limited. As more business areas are being assimilated in the DWS, the evolution of the system must be considered to follow the natural development and growth of the data stored in the DWS (Michael V. and Zhiping, 2006). The increased costs of hardware and maintenance reflect the growth of DWS, considering not only the amount of the processed data but also the increasing complexity of the operations, which naturally limits the choice of the infrastructure to be adopted by the system. Thus, the infrastructure development for supporting the operations execution should be considered since the planning phase, without conditioning the execution window.

ETL modelling tools and methodologies have been an area of intensive research since the emergence of DWS. Many proposals were done so far, promoting the use of conceptual and logical models for ETL modelling, aiming to reduce systems design complexity, producing detailed documentation, and providing the ability to easily communicate with business and end users. Often, the most accepted proposals for DWS development based their ETL components in ad-hoc approaches with no formalization or standard notation. Therefore, several authors proposed specific modelling approaches based on a clear separation of the different design stages: conceptual, logical and physical. The use of models whether being conceptual or logical has captured the attention of many researchers in the field, which try to develop a representation model and a set of execution mechanisms for reducing the natural complexity of an ETL development and implementation process.

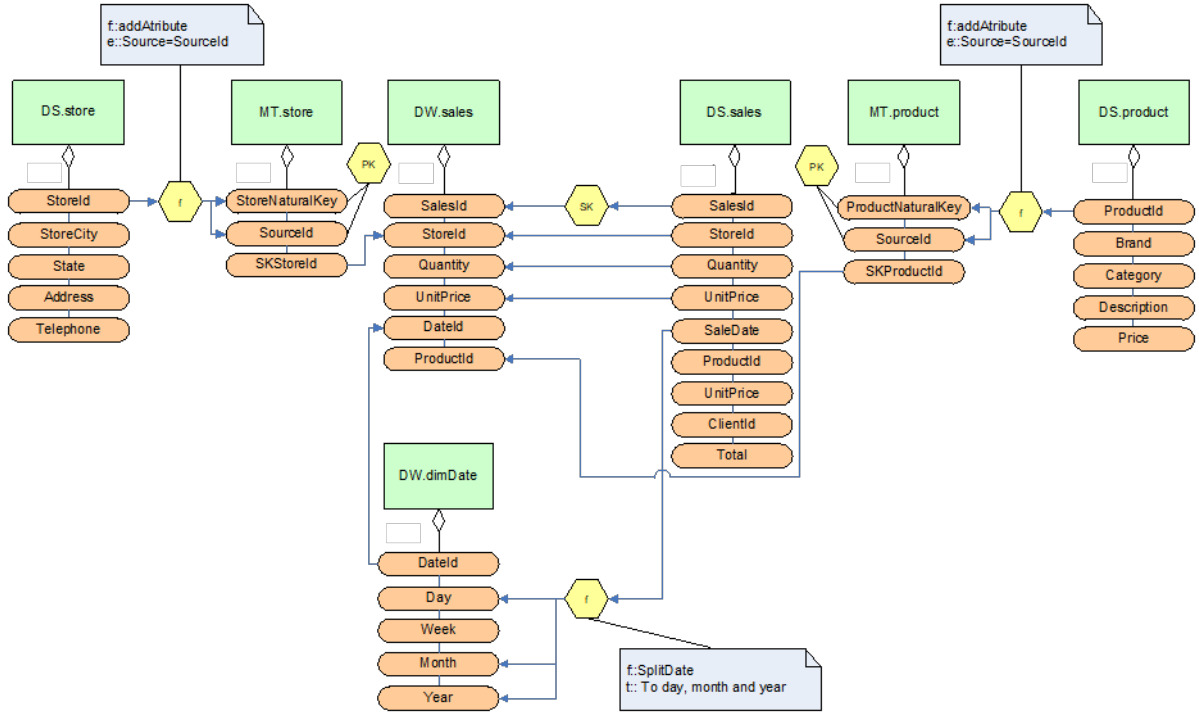


Figure 2. Vassiliadis and Simitsis high-level model for ETL activities with attribute inclusion – extracted from (Vassiliadis et al., 2002a).

In general, most of the common proposals presented till now focus on the ETL processes abstraction, revealing since early design phases the most elementary requirements and needs that will support the system implementation. However, a standard approach to deal with ETL development, covering not only ETL design based on more abstract models but also its subsequent implementation using the efforts done in initial phases, is still missing. In the following sections, we will present some of the most relevant proposals occurred in the area, grouping them according to their development strategy

2.4.1 Process-Driven Development

The first contributions made for supporting ETL development presented some design techniques based on specific languages and notations oriented for ETL specificities or by adapting existing general-purpose languages and notations to ETL particularities. Based on specific meta-models, most approaches focused on ETL documentation and formalization, providing the means to simplify ETL development and extensibility. The proposals presented by Vassiliadis and Simitsis cover several aspects of ETL conceptual specification (Vassiliadis et al., 2002a), its representation using logical views (Vassiliadis et al., 2002b), the mapping between the two perspectives (Simitsis and Vassiliadis, 2008) and posteriorly its implementation using a specific ETL tool (Vassiliadis et al., 2000). These authors proposed a new set of elements oriented for expressing ETL natural features, which are very useful in early stages of DW project. They also presented a methodology (Simitsis and Vassiliadis, 2003) for the design of the ETL conceptual part that is oriented for representing attributes relationships between data repositories used, and the necessary ETL transformations for attribute mapping between data sources and the DW schemas.

After requirements analysis, in conceptual modelling, data repositories along with their interrelationships are identified based on a set of mapping rules and attributes transformations, using a pallet of specific transformations. Usually, other technical staff (such as database administrators) performs this process, providing a better understanding of some database codes, implicit rules and some potential problems associated to the data. Following this methodology for conceptual modeling, the Figure 2 presents a simple ETL representation to support a given SKP process used to replace natural keys by correspondent surrogate keys. In this example, we use three prefixes to identify data repositories: *DS* – data provided from data sources, *MT* – mapping table identifier and *DW* – target data warehouse repository.

Further, at the logical level, a similar graphical notation is used for the specification of execution flows of all the operations involved in the ETL, describing the logical sequence of operations required to implement the transformations presented in the conceptual model for the workflow representation. Tasks are sequenced and may, for optimization reasons, be repositioned during the execution of the flows. Additionally, Vassiliadis et al. (2002c) proposed an architecture graph based approach that is used for workflow modelling, representing a formal logical model for ETL activities. A tool to support all these design concepts, ARKTOS (Vassiliadis et al., 2000), was also developed.

In opposite to Vassiliadis approach, Trujillo and Luján-Mora (2003) proposed an approach for ETL conceptual modelling using UML (Unified Model Language[5]). In their work, UML was extended with some special constructs to represent ETL specifications with more detail. With the use of UML, authors wanted to minimize development efforts and costs associated with the notations and

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

methodology learning curve, proposing the use of a standard and widely accepted object-oriented language. This approach meant to be integrated to the multidimensional modelling approach using UML that was also proposed by the same authors (Luján-Mora and Trujillo, 2002). The Figure 3 shows an example of some elements used by the proposed notation to support the representation of a given SKP process. The *Conversion*, *Join* and *Loader* are just some examples of the extended operations provided by this notation for ETL modelling.

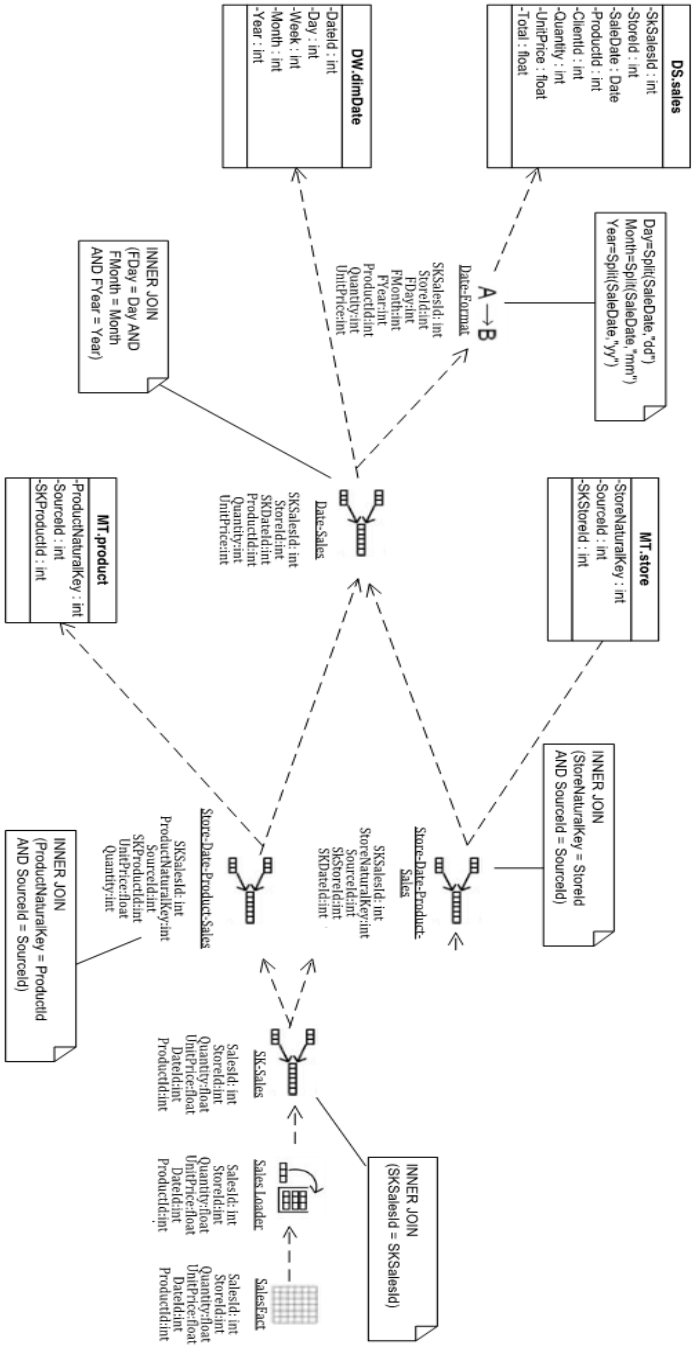


Figure 3. Trujillo's proposed UML extension example for ETL conceptual modeling (Trujillo and Luján-Mora, 2003).

Still using general purpose languages and based on the foundation that an ETL process can be considered a particular type of business process, El Akkaoui and Zimanyi (2009) presented a proposal using standard BPMN (Business Process Model and Language[6]) for ETL design. The authors showed how to use the BPMN notation for developing an ETL conceptual model. They also avoided the major drawback that usually exists between modelling conceptual specifications and their implementation, demonstrating how we can execute them using BPEL (Business Process Execution Language[7]) (Ouyang et al., 2007). The specification process they demonstrated using BPMN during modelling a conceptual perspective for an ETL system revealed that the BPMN notation, originally designed for modelling business processes, can be successfully applied to the DWS populating processes. Their approach shows that it was possible to properly construct a ETL conceptual model using BPMN, easy to read and to understand for all stakeholders involved in the ETL system development. This simplicity is derived from the BPMN freedom for process modelling, providing very expressive constructs that allow for the representation of a same process or task in different ways. The Figure 12 presents the application of BPMN constructs to model a simple ETL process describing a typical CDC procedure.

Some authors focus on the development of ETL processes based on existing business processes already implemented and used it in organization daily operations, taking advantage of existing knowledge and routines. Dayal et al. (2009) introduced a layered methodology for supporting the design of ETL processes for operational business intelligence systems, using a unified formalism driven by a set of quality measures (QoX objectives), represented using BPMN for conceptual representation. Basically, following their methodology, the development process starts by modelling business processes and their services' requirements and objectives, passing by logical design to its physical implementation. Later, using several intermediate layers, the approach was improved (Wilkinson et al., 2010), based on a layered methodology that ranges of organization business processes modelling to physical implementations. The QoX metrics are used to guide all translation processes based on existing business processes for requirements analysis and conceptual modelling. They use BPMN notation (with specific annotations) for ETL conceptual modelling based on the QoX properties, describing the necessary steps for its translation to logical views based on an XML representation that can be used by some commercial tools. The authors argued that its approach provides a unified formalism, both for ETL and organizational processes, enabling the representation of data intermediate states and providing an abstract view over all process, encapsulating technical details from business users.

[6] http://www.bpmn.org
[7] http://www.oasis-open.org/committees/wsbpel/

2.4.2 Model Driven Development

The use of Model Driven Development (Mellor et al., 2003) is increasing in several software development areas. The Model Driven Architecture (MDA) (Kleppe et al., 2003) is an OMG (Object Management Group[8]) standard that supports system development in all its stages, from requirements analysis to its implementation. The modelling activity describes the structure and the semantic to produce final systems guided (automated) from early models, including documentation, system specifications and requirements. All these is supported by solid industry standards that enhance system planning and design, contributing to the final system quality and maintainability.

Muñoz et al. (2009), proposed an MDA approach to derive ETL conceptual models represented using UML activity diagrams to carry out an automatic code generation for ETL tools. These authors proposed a set of QVT (Query, View, Transformation) transformations to provide the generation of code for specific platforms. Based on MDA concepts, a Platform Independent Model (PIM) was developed based on a set of activities identified by these authors in previous works (Luján-Mora et al., 2004). The PIM model can be translated to specific platform models for physical execution, using the QVT language for model conversion. Later, El Akkaoui et al. (2011) also presented a work based on MDA for ETL process development, presenting and discussing ETL processes specification through the use of a specific set of BPMN components that was specifically designed. The authors proposed a vendor-independent model (BPMN4ETL – Figure 4) for unified design and covered its automatic generation from source code to specific computer platforms using the BPMN-based meta-model. They also explored and discussed the bridges for a model-to-text translation, providing its execution using ETL commercial tools. Thus, the model elements are iteratively translated to the selected tool using code templates developed using Acceleo[9]. Posteriorly, El Akkaoui et al. (2012) addressed two important architectural layers related to the specification of ETL processes using BPMN, namely:

1) Process orchestration, which can be accomplished by several BPMN elements, like events and flow control gateways.
2) Data process operations, which are related to some specific operations that allow for the manipulation of data among several data sources coordinated by control process elements.

Using BPMN constructs, these authors presented a specific meta-model covering those layers for ETL specification, addressing also the problematic of ETL maintenance phase on ETL life cycle, particularly when business requirements change. Additionally, they extended the model-driven framework, not only to provide automatic code generation but also to improve ETL maintenance using model-to-text (using the so-called BPMN4ETL meta-model) and model-to-model transformations (to reflect requirements changes in an ETL model). The authors identify structural updates and provide

[8] http://www.omg.org/
[9] http://www.eclipse.org/acceleo/

strategies for handling schema changings. Thus, if data sources or the DW evolve, ETL models can be automatically updated.

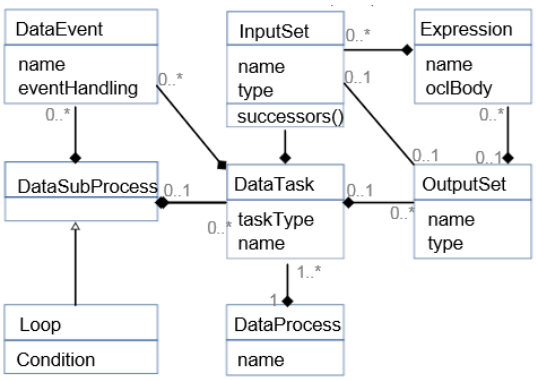


Figure 4. A BPMN4ETL metamodel excerpt as presented in (El Akkaoui et al., 2011).

2.4.3 Ontology-Driven Development

Data sources heterogeneity is one of the main problems we face in the development of an ETL system. Even under the same domain, each data source represents a specific set of data requirements and some different ways to represent data. Ontologies can be used for describing data integration processes as a way of expressing semantics relationships between attributes of the entities used for data loading.

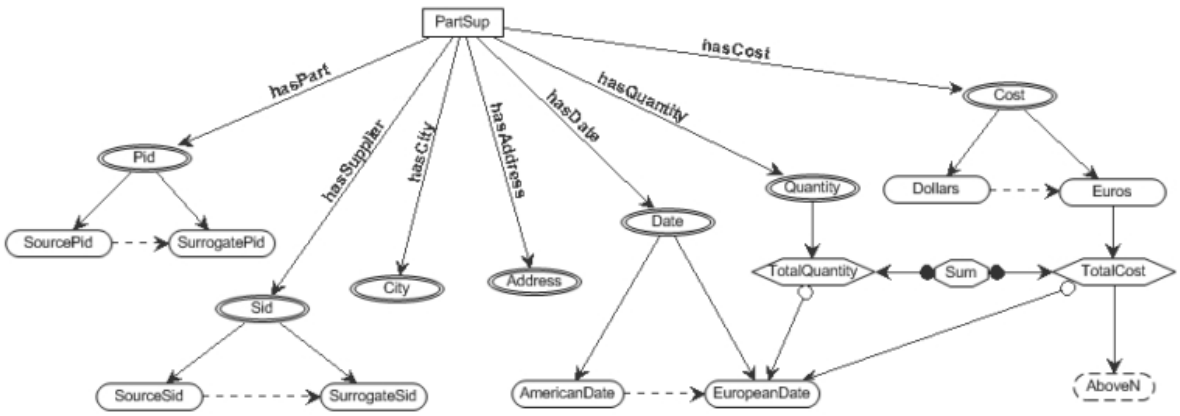


Figure 5. Graph ontology representation example, describing mapping between source and target DW – extracted from (Skoutas and Simitsis, 2007).

Skoutas and Simitsis (2007, 2006) proposed an ontology-based approach to facilitate ETL conceptual design, using a graph-based representation with the ability for supporting structured and semi-structured data sources. Both data sources and DW are semantically annotated to infer their mappings and conflicts. The graphs are presented as a generic data model for schemas representation that using an ontology graph allows for data source annotation into an annotated graph format. In practice, a specific vocabulary is created for annotating each data source elements, resulting in a specific ontology - Figure 5. These elements can be used to reveal relevant concepts

and the specific operations that should be performed to provide data integration. Later, these same authors extended the work presented by Skoutas et al. (2009) with a set of graph transformation rules that determine the order of operations and the evaluation of the approach presented using artificial scenarios.

In a similar line of work, Zhang and Wang (2008) and Jiang et al. (2010) proposed other ontology approaches for supporting the ETL development. As well, Zhang and Wang (2008) proposed a framework model based on semantic web technologies that allow the abstraction of data sources schemas metadata thought the creation of specific ontologies for each source (local ontologies) and for the DW (global ontologies). Both representations are then mapped for the identification of structural and semantic relationships. The next steps involve the mapping of the ontologies produced to find relationships between the concepts and the use of a semantic reasoner to identify ETL support rules. The approach presented by Jiang et al. (2010) shares several lines of research lines with this approach. However, local ontologies are not used since the authors argue that the whole process will be simpler using only the global domain ontology.

2.4.4 Pattern-Oriented Approaches

The general idea of using software patterns for building ETL processes was first explored by Köppen et al. (2011). They proposed a pattern-oriented approach to support ETL development, providing a general description of a set of patterns. The *Converter*, *Separator* or *History* patterns are just some examples presented in Figure 6. They first explored some major aspects for defining composition and relationship properties describing constraints applied to the use of each pattern. Patterns were presented only at the conceptual level, lacking to support their instantiation for execution primitives. The pattern-oriented approach presented here differs from the work presented by Köppen et al. (2011) since a set of configurable components that can be used in different ETL development phases are provided, covering the initial design phases to produce system skeletons that can be enriched to allow for its mapping to a specific set of execution primitives. Fine-grained tasks are encapsulated inside these components, resulting in a new ETL development level defined using an abstraction layer that simplifies and carries the acquired knowledge between projects.

The guidelines provided by Kimball (Kimball and Caserta, 2004) present a set of techniques normally used in ad hoc fashion, which lacks from methodological foundations and notations for ETL visual representation. As a result, the generated documentation can hardly be used for computational interpretation. Compared to the approach presented previously by Kimball (Kimball and Caserta, 2004), the methodologies and notations proposed by more recent works represent a structuring and formalize approach to support ETL design based on the considerations and methods presented by Kimball (Kimball and Caserta, 2004). This improves quality, simplicity and better understanding for all people involved in an ETL process development.

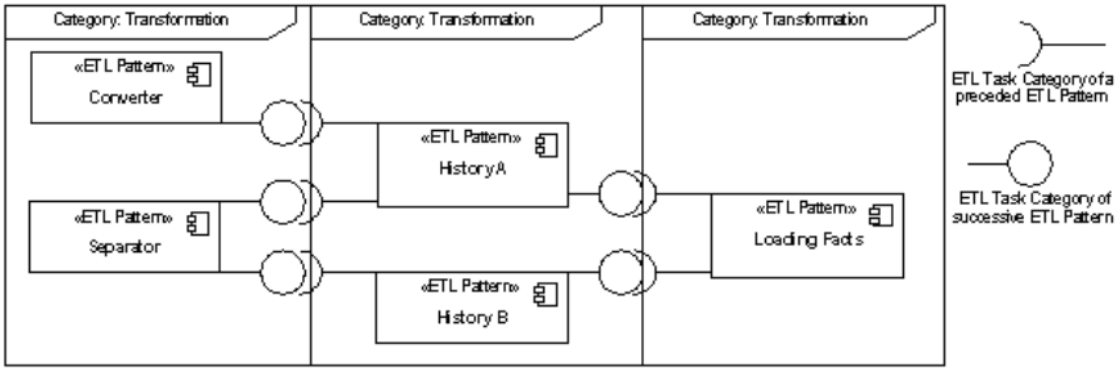


Figure 6. An ETL Process composed by Patterns as presented in (Köppen et al., 2011).

For ETL process representation, and considering the conceptual and logical viewpoints, the works presented by Vassiliadis and Simitsis (Simitsis and Vassiliadis, 2003; Vassiliadis et al., 2002b, 2002c) reveal two main problems, namely:

1) the notation is clearly "ETL-oriented", causing an extra effort to ETL development team for learning the architectural and notation specificities, compromising and limiting communication;
2) because has limited support for being represented in programming primitives.

Vassiliadis et al. (2000) developed the tool Arktos. However, today is an unsupported and limited tool. Moreover, in practice, the use of the proposed notation can become rather confusing since the dimensions and fact tables may have a high number of attributes coming from several sources, leading to an increase of the schemas complexity, resulting in an ETL model difficult to read and understand. Trujillo softened this problem, proposing the use of UML language for ETL representation. UML is a very known and used language for software development, which may improve communication between ETL design team that can be more familiar with their constructs. Thus, the notation proposed by Trujillo (Trujillo and Luján-Mora, 2003) reduced some issues related to the readability of the model and its subsequent support by a large variety of modelling tools available today. However, when more complex ETL process are used (with a greater number of transformations), ETL diagrams become quite difficult to read and to understand, essentially because of the limiting number of transformation tasks proposed. This means that the use of annotation elements for complementing ETL diagrams will be overused, compromising directly the model interpretation. Additionally, in this approach, the translation process to execution primitives is not very natural, since UML is essentially used to describe system requirements and not to support its execution. Despite these problems, in our opinion, all these works represent some quite important contributions in ETL modelling and design.

The use of BPMN for representing ETL processes was revealed in by El Akkaoui and Zimanyi (2009). It was a very interesting approach, showing that it is possible to design and construct a ETL

conceptual model in BPMN, is easy to read and to understand. Using a very practical way, El Akkaoui and Zimanyi (2009) show how the BPMN notation, originally designed for modelling business processes, can be adopted successfully for modelling ETL systems and interpreted by computers using the interchangeable formats available to support BPMN representation[10]. However, due to the construction freedom and redundancy (equivalent processes can be implemented using different representation) that BPMN denotes, some authors (Dijkman et al., 2008; Muehlen and Ho, 2008; Ou-Yang and Lin, 2008) pointed several issues that can compromise process implementation, resulting from misunderstandings and incomplete identification of business requirements, mainly due to the use of methods or validation models that are (typically) manual and error-prone. When patterns are used, BPMN modelling is especially useful to express coordination constraints, since each pattern encapsulates its own internal behaviour, which can be used and changed without affecting the remaining patterns. Thus, the BPMN conceptual models are clearer, leaving patterns configuration aspects to further development phases. Particularly, for these tasks, the works based on MDA approaches and ontologies revealed very important aspects to deal with patterns semantics to support the ETL conceptual and logical modelling, the conceptual to logical mapping primitives and (subsequently) to physical primitives, disconnecting the data transformation logic from physical tool responsible for process execution.

[10] For exemple: http://www.xpdl.org

Chapter 3

A Pattern-based Approach for ETL Development

3.1 Towards a Pattern-Based ETL Model

The use of domain models is an important task throughout all the ETL development process lifecycle, enhancing the application of reusable techniques and knowledge sharing. Domain models support the development of software projects, serving as the base for describing requirements and vocabulary related to domain rules that can be applied to the data involved with. Thus, model representations can be used to improve communication and to develop software reuse strategies. Furthermore, the modern programming languages and development philosophies promote software reuse, providing powerful frameworks with a set of preconfigured components for producing software with faster, better quality, and potentially having lower costs. These preconfigured components are sometimes based on well-known and validated design patterns that describe abstract solutions (enabling its reuse in many different scenarios) to solve recurring problems, providing a way to share the experience and knowledge acquired.

Model-driven techniques are being used to support several software areas in the development of models, covering code generation through a model synchronization process between physical and abstract representations (El Akkaoui et al., 2011). Concerning the ETL nature, whether being conceptual or logical, models covering specific data requirements reveal a symbiotic relationship between business requirements and process implementation. This means that each ETL system is always different even when they are defined in the same application domain, which seriously compromises the reusability outside problem space. To minimize such problems, we propose a specific ontology covering a set of the most common ETL procedures. Based on some well-known

design strategies, these ETL procedures represent reusable components that can be instantiated for covering some particular needs. The proposed ontology serves as a base platform for creating domain specific models that can be enriched by new detail layers (more abstract or more concrete), which can be shared and reused across several projects. Since the most traditional ETL problems and concerns are encapsulated in a single component and thus its complexity avoided by ETL team, the resulting context of each pattern application generates simpler ETL processes, easier to read and use. Considering these assumptions and the patterns we idealized, the physical model generation assumes a crucial step in the entire ETL process development.

In this chapter, we present all the patterns we designed in conceptual terms. At this point, while we do not have a concrete idea of what is the target execution environment, it is not possible to predict how to support patterns in a physical way. In the next chapter, we will focus completely on the ETL development approach, using all the ideas presented in this chapter as a base for creating abstract conceptual models and providing the necessary means to enrich them with specific logical primitives. A model-driven approach will be followed, approaching all code generation aspects and, when necessary, the enrichment of the models created. Based on these two perspectives, we dispose of the mechanisms and structural constraints to generate physical models that can be executed using a specific ETL tool. We will use a specific application scenario for demonstrating how we use a pattern-based approach to develop an ETL system, covering all development stages and showing how specific requirements and business particularities can be modelled and described.

3.2 ETL Patterns Meta Models

The use of software patterns is a well-known reuse-based technique often applied in software engineering in a lot of different domains (Gamma et al., 1994), supporting component reuse techniques and sharing knowledge across applications. Using software patterns, we can preserve the use of well-proven techniques, contributing to the final software quality and reducing the development time and costs (Alexander et al., 1977). Concerning the ETL environment specificities, patterns can be characterized using a set of pre-established tasks grouped together based on a specific configuration related to the context where they are used.

Creating and using reconfigurable components allows for avoiding rewriting some of the most repetitive tasks that are regularly used in ETL process implementation. Several tasks, such as surrogate key process generation, lookup operations, data aggregation processes, data quality filters or slowly changing dimensions are just some few examples of common procedures that are used frequently in any DWS implementation. This way, users can focus on more general requirements, leaving the complexity of its implementation to posterior development steps. Consequently, users only need to provide configuration metadata to the conversion engine responsible for generating the correspondent physical model.

Due to the knowledge complexity involved in the patterns application to specific contexts (Dietrich and Elgar, 2007), the ETL processes can be affected by inconsistencies and misunderstandings related to communication problems that result in different meanings or architectural contradictions. Ontologies can be used for providing contextual data to describe each pattern based on its structural properties (Noy and McGuinness, 2001). At the same time, an ontology can express ETL patterns properties in a formal way that can be interpreted by a computer. Ontologies are being used by many organizations also to encode and share information across multiple systems, providing a way to electronic agents understand and use the information based on a solid and shared formalism.

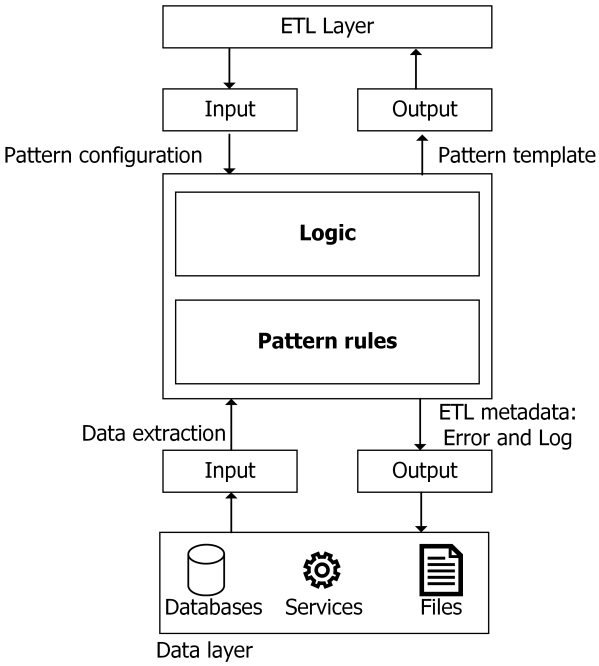


Figure 7. Representing the structure of a pattern.

OWL (Web Ontology Language)[11] is a language based on Web semantic technology that is often used to describe domain-specific meta-models for representing properties and relationships between domain concepts. OWL is a W3C[12] (World Wide Web Consortium) standard that was developed for providing a simple structure to process and use semantic data across applications on the web. With OWL, classes or concepts are described and arranged to form taxonomic hierarchies with properties describing the composition in terms of entity attributes and restrictions over the relationships between concepts. So, ETL patterns can be syntactically expressed using classes, data properties, and object properties, disposing of a basic structure to support the development of a specific language for pattern instantiation. To support this, we designed an intermediate layer to separate technical knowledge, typically applied in commercial tools, from the domain knowledge used by decision-makers (McGuinness and Wright, 1998).

[11] http://www.w3.org/TR/owl-features/
[12] https://www.w3.org/

The pattern structure (Figure 7) represents a pattern core that encapsulates all pattern rules defined for supporting the operational requirements and the logic behind them. Each pattern uses input and output interfaces both for communicating with ETL layer and the data layer that supports pattern application. The construction rules are provided by each pattern to the context it is applied for sustaining well-formed workflows considering the configuration provided to the pattern, while the data from data layer is used both for data extraction and data storage, representing the pattern intermediate or final outputs.

As an integral part of the pattern configuration, additional ETL metadata is used to support the error and log strategies for handling errors and pattern events. Thus, the "Throwable" and "Log" instances are used together with the pattern core, encapsulating all logic behind error and log handling in the respective components.

```
(...)
<owl:Ontology rdf:about="ETLPatterns"/>
  <owl:ObjectProperty rdf:about="ETLPatterns#HasCore">
    <rdfs:domain rdf:resource="ETLPatterns#Pattern"/>
      <rdfs:domain>
        <owl:Restriction>
          <owl:onProperty rdf:resource="ETLPatterns#HasCore"/>
            <owl:qualifiedCardinality rdf:datatype="(...)">1
           </owl:qualifiedCardinality>
            <owl:onClass rdf:resource="ETLPatterns#Pattern"/>
        </owl:Restriction>
      </rdfs:domain>
      <rdfs:range rdf:resource="ETLPatterns#PatternCore"/>
      <rdfs:range> (...)
  </owl:ObjectProperty>
<owl:ObjectProperty rdf:about="ETLPatterns#HasThrowableComponent"> (…)
</owl:ObjectProperty>
(...)
<owl:DatatypeProperty rdf:about="ETLPatterns#Name">
  <rdfs:domain rdf:resource="ETLPatterns#Pattern"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
(...)
<owl:Class rdf:about="ETLPatterns#Pattern"/>
<owl:Class rdf:about="ETLPatterns#PatternCore"/>
(...)
```

Figure 8. Representing main pattern components in OWL.

The "Throwable" pattern uses the input configuration for handling error or exception scenarios through the application of specific recovering strategies, which are preconfigured for each pattern used. The errors that cause critical failure scenarios can be configured to use specific procedures to maintain data consistency. For example, if some error is compromising the ETL execution, the process can be aborted using a rollback procedure, maintaining the data in a consistent state.

The "Log" pattern stores the ETL events timeline data lineage identification, detect bottlenecks, and execution errors, among other things. Thus, the ETL process can be analysed to identify specific error trends and to define strategies for minimizing them, reducing the ETL resources that usually are needed for subsequent data loads. The log structures can vary in granularity and scope, triggering the events associated to their using conditions associated with pattern' internal tasks or by other more general conditions (such as achieved process milestones).

In Figure 8 we can see an ontology excerpt used for describing the main pattern structure using OWL. To represent the main objects for supporting the referred ontology we used the following classes and slots:

- **Classes** – "Pattern", "PatternCore", "Throwable" and "Log". The "Pattern" class contains the basic data properties for each pattern, e.g., the identification number ("Id"), "Name" and "Description". The "PatternCore" encapsulates all the parameters to support the logic behind each pattern's operational requirements. The "Throwable" class describes the exception and error handling processes, while the "Log" class describes the logging processes to track all pattern events.
- **Object Properties** – "HasCoreComponent", "HasThrowableComponent" and "HasLogComponent", describe the relationship between individuals of each class involved and the main relationships between classes to support patterns basic structure. Consistency properties should be preserved to enforce that each pattern has only one core and zero or more "throwable" and "log" components. This means that several "Throwable" and "Log" patterns can be configured for the same pattern, allowing the description of different scenarios for each pattern.

The "PatternCore" class represents the most general element used in a pattern, while the "Extraction", "Transform" and "Load" classes represent the three types of patterns that are intrinsically associated with each typical phase of an ETL process.

The "Extraction" class instances are used to extract data from data sources represented as data objects (e.g. a table or file), embodying typical data extraction processes and algorithms that can be applied to the data structures used. The "Extraction" class can be specialized in two more specific data extractors: "Full extraction" class and "Change Data Capture" class. The "Full extraction" is used to extract data from a given data source without any criteria (all data currently available), while the "Change Data Capture" class can be based on two distinct types of data extraction:

- **Differential**, which is used to identify new data since the last successful extraction and for that reason can be referred as an incremental data extraction process; in this data extraction type, all data records from source and target repositories are compared in order

to identify the data changes; basically, a minus operation is performed between data from each data source and target repository.

- **Incremental**, which is used to extract data from data sources since the last successful extraction operation or based on a definition of a specific schedule or event.

The "Transformation" class represents patterns that are used in the ETL transformation phase for the application of a set of cleaning or conforming tasks (Rahm and Do, 2000) for aligning source data schema requirements to the target DW schema ones. This pattern category represents a large variety of procedures that are often applied in DWS, such as:

- application of well-known data variation policies, known as SCD techniques;
- generation of surrogate keys;
- conciliation and integration of data involving several data sources;
- schema transformation.

Patterns that belong to this class can also be specialized. For example, the "Data Quality Enhancement" (DQE) class can also be specialized as a "Normalization" pattern representing a set of tasks needed whenever it is necessary to standardize or correct data according to a given set of mapping rules. Thus, all the most frequent ETL patterns can be represented along with all its operational stages and using the ontology hierarchy for supporting the ETL patterns' meta-model. We can add new patterns or change the structure of old ones without compromising the pattern global structure.

Finally, the "Load" class represents patterns that are used for loading data into the DW repository, encapsulating efficient algorithms for data loading or index maintenance to support loading procedures. The "Intensive Data Loading" (IDL) class represents a subtype of the "Load" class, embodying all the necessary operational requirements for loading data into the DW's structures, according to the schema restrictions implemented. Each subclass can also include other additional properties or override predefined ones. For example, the "Incremental" class uses a date type property to identify new or changed records.

Considering the ontology definition, each property should be described based on its cardinality, value, domain, and range. The property's domain links it to a class, while the range associates a property to a class or data range. This allows for the association between classes and data types, providing a way for establishing restrictions between concepts. For example, all the patterns should have one (using "Exactly" cardinality restriction) core ("PatternCore" class) but can have zero or more (using "some" existential restriction) log ("PatternLog" class) and "Throwable" ("Throwable" class) individuals. These classes are "Disjoint", i.e. an object cannot be an instance of more than one of these components. The "Throwable" and "PatternLog" patterns are not mandatory. We can define more than one

instance for the same pattern in order to represent different application scenarios. Additionally, pattern subclasses can override inherited constraints, using specific cardinality restrictions based on their own requirements. To do this, we use sub-properties for imposing specific constraints.

After the taxonomy definition, we need to enrich the meta-model to support the basic rules for the development of well-formed ETL patterns. For that, each class should be defined using properties. For example, the "Extraction" class is composed of "DataType" properties, such as pattern "Id" and "Name" (inherited from "Pattern" class), and "Object" properties, such as "PeriodLiterals" that refers to the extraction interval to be used (e.g. Hour, Daily, Month) –, "one of" property, and the metadata related to the repository connection. The relationship between data sources is supported using mappings. The "Mapping" class describes the relationships between data repositories that are used for patterns configuration.

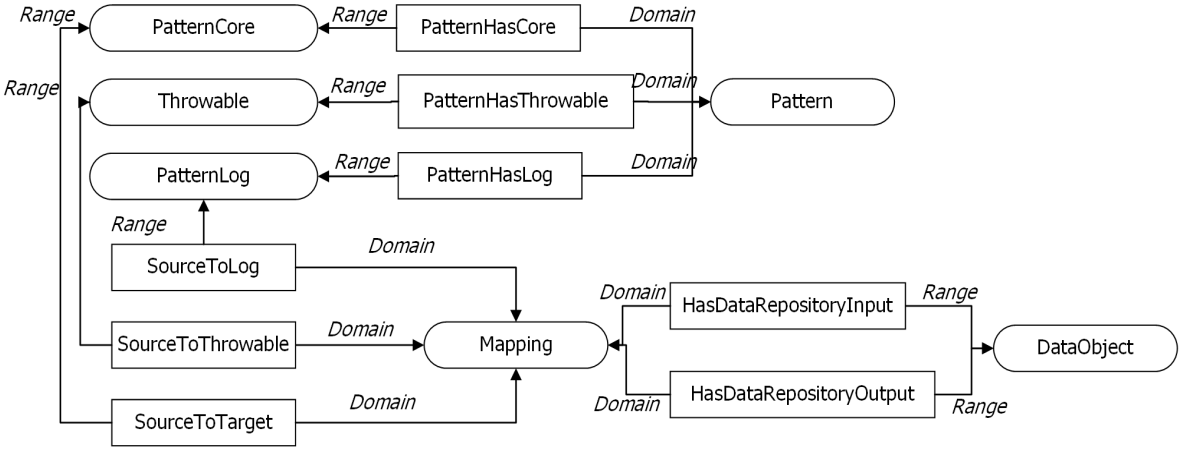


Figure 9. Summary of the ontology graph.

In Figure 9 we can see a summary of a graph ontology representing the main concepts related to the ETL pattern operational requirements. The round corner rectangles are used for identifying classes and rectangles for representing object properties with the associated domain and range properties. The three pattern components are identified along with the object properties to define its instantiation. The "SourceToLog", "SourceToThrowable" and "SourceToTarget" object properties represent each pattern component configuration for specific mappings ("Mapping" class). The "Mapping" class describes the relationship between the input ("HasDataRepositoryInput" object property) and output ("HasDataRepositoryOutput" object property) "DataObject" instances, representing data repositories that hold the data used for the pattern application. This way, it is possible to establish relationships between data repositories' attributes in order to ensure the data migration processes that can be established between them. These data repositories can have a physical representation (e.g. databases and files that require a specific connection protocol) or can be passed as a stream of records. The pattern structure proposed in Aalst et al. (2003) and Appleton

(2000) was adopted for describing a subset of ETL patterns and their own operational requirements and relationships.

The ETL patterns ontology contributes significantly to reduce potential ambiguities related to the application of patterns and describe the involved knowledge, assisting ETL designers in the description of patterns' operational requirements and relationships in a way that can be computer-readable and integrated into specific applications to support ETL design and implementation.

Although the pattern structure can be described in a straightforward manner, several richer structural constraints for specifying pattern integrity within a process context should be considered. While structural constrains can be easily described using modelling languages, its behaviour and mainly its validation imposes several challenges. Thus, to check system correctness and avoid several implementation problems involving patterns structural properties, we can formalize them using the Alloy language (Jackson, 2012). Alloy is a declarative specification language that supports problem structural modelling and validation, helping to avoid process inconsistencies and architectural contradictions, providing automatic analysis mechanisms for searching for false assertions through the generation of counterexamples that represent scenarios in which the specification rules are not true.

With Alloy, models can be created to capture key system aspects, providing an analysable system representation that can be used to evaluate future system implementation. This way, patterns can be checked using a powerful simulation engine before its physical generation. The simulation engine generates instances describing the model structures and behaviours without the need to provide input data. The test cases are generated by the analyser to find well-formed instances or to find scenarios in which the model constraints are not true. The Alloy analyser provides substantial support for automatic assertion checking, within a bounded universe, by relying on off-the-shelf constraint solvers used to check if there exists a model instance that satisfies a given set of constraints. The Alloy language flexibility (and its object-oriented nature) makes it in a very powerful way for specifying and analysing software design models, addressing both complex structural constraints (Anastasakis et al., 2010; Cunha et al., 2013; Kuhlmann and Gogolla, 2012) and behavioural constraints imposed by transformations (Büttner et al., 2012; Macedo and Cunha, 2014). Embedding ETL pattern specifications into Alloy not only provides a formal specification of their structure and behaviour but also allows for their fully automatic verification, preserving system consistency.

For the pattern-oriented framework presented, a set of formal specifications in Alloy allows for expressing patterns structural constraints and behaviour, showing how a pattern configuration can be validated according to the specifications and constraints defined by its correspondent Alloy model. The Alloy representation is built based on relevant parts of the pattern-oriented framework, leaving the operational aspects of patterns execution, in which we can express how new data is obtained from old data to programs and applications that will implement them.

The ontology class hierarchy can be easily described using Alloy signatures, which introduce sets of elements of a certain type in the model. Abstract signatures are used to describe data domains that should be extended to embody proper elements. This is the case of top-level signatures, such as "Field", "DataObject" and "PatternCore". These abstract signatures can then be specialized through the extension of concrete data domains in a way quite similar to the pattern structure described by the ontology.

```
abstract sig Field{}
abstract sig KeyField extends Field{}
sig PKField, SKField, FKField extends KeyField{}
sig ControlField, VariationField, DescriptiveField (...) extends Field{}
sig DateField, OperationField, (...) extends ControlField{}
sig TempObject, OperationalObject, SourceObject, (...) extends
DataObject{(...)}
(...)
fact consistentField{all f:Field | one fields.f (...)}
sig DataObject{
    fields: some Field,
    keys: some PKField & fields
}
fact dataObjectKeyFields {
   all o: DataObject | o.keys in o.fields
   (...)
}
pred consistentDataObject[s:State,o:DataObject] {
   all r1,r2: o.rows.s |
      (all f: o.keys | f.(r1.values) = f.(r2.values)) => r1 = r2
      (...)
}
sig Mapping {
    inData: one DataObject,
    outData: one DataObject - inData,
    association: Field -> Field
}
fact consistentMapping {
    all m: Mapping | m.association in m.inData.fields -> m.outData.fields
    (...)
}
abstract sig PatternComponent{mapping: some Mapping}
abstract sig PatternCore extends PatternComponent{sourceToTarget: some Mapping
& mapping}
abstract sig Throwable extends PatternComponent{sourceToThrowable: some
Mapping & mapping}
abstract sig Log extends PatternComponent{sourceToLog:some Mapping & mapping }
abstract sig Pattern{
       coreComponent:one PatternCore,
       throwableComponent: set Throwable,
       logComponent: set Log}
fact patternConsistency {
   all pattern : Pattern {
      pattern.sourceToLog.outData in LogObject
      pattern.sourceToThrowable.outData in ThrowableObject, (...)
   }
}
abstract sig Extract, Transform, Load extends PatternCore{}
(...)
```

Figure 10. The basic Alloy specification for defining ETL patterns.

For example, the "Field" signature can be specialized in control fields ("ControlField") for describing attributes that represent useful meta data related to each record. This attribute specialization is structured using several types, namely:

- the "KeyField" for identifying attributes that belong to some type of key (e.g. primary key and foreign key);
- the "ControlField", such as the date fields ("DateField") for describing temporal data, the operation fields ("OperationField") that are useful to designate actions performed in source systems, which are quite useful for supporting some ETL procedures like CDC or SCD;
- the throwable fields ("ThrowableField") for representing data structures to support error handing;
- the log fields ("LogField") for characterizing specific meta data describing how to store the occurred events during an ETL execution.

The Figure 10 presents an excerpt from the Alloy specification used to describe the core of our pattern-oriented approach, formalizing the meta-model structure and providing a new degree of detail. Beyond the signatures for representing the "Field" structure, the specification presents the "DataObject" signature for describing the data repositories data structure used by ETL patterns. Alloy signatures may contain fields of arbitrary arity that will embody the associations between the different objects. For example, data objects represent repositories that are used to read and write data, and contain a set of field declarations that is shared by its extensions. Each "DataObject" instance is related to a non-empty set of "Field" elements (imposed by the keyword "some") that represent data fields that compose data records and a set of "SKField" fields used to express the SK fields. These can then be restricted by additional constraints using Alloy facts. For example, the fact "dataObjectKeyFields" enforces that every "SKField" is selected from the data object "Field" elements and the "consistentField" fact enforces that fields belong to a single data object.

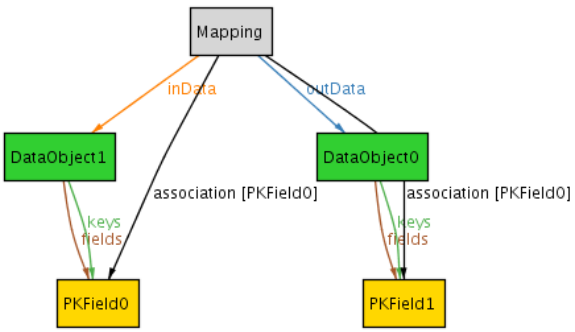


Figure 11. Mapping an instance describing the relationship between two "DataObject".

Additionally, other data object specializations are also represented, either to include new properties or to distinguish them syntactically between data object nature. For example, the "TempObject" represents data objects stored temporarily in the DSA, the "OperationalObject" represents a particular

object with special meaning in the DW context (e.g. Fact tables), and the "SourceObject" denotes a source object that stores raw data waiting for processing.

The "Mapping" signature represents the association between attributes of two data sources, establishing the relationship between them through a binary field association between fields (a->b). With mappings, other constraints related to metadata consistency should be defined. For example, we can impose that a mapping is only valid if the associated fields from the input ("inData") and the output ("outData") data sources ("consistentMapping" fact) are compatible according to model rules. The Figure 11 shows a very simple Mapping instance, describing the mapping between the "PKField0" and the "PKField1" field from the "DataObject1" and the "DataObject0" data objects. The relation "InData" indicates the input data object, the "outData" relation indicates the output data object, the "keys" relation indicates that some field participates in the correspondent data object as key field, the "fields" describes the fields that belong to some data object and the "association[PKField0]" relation describes which fields are related together.

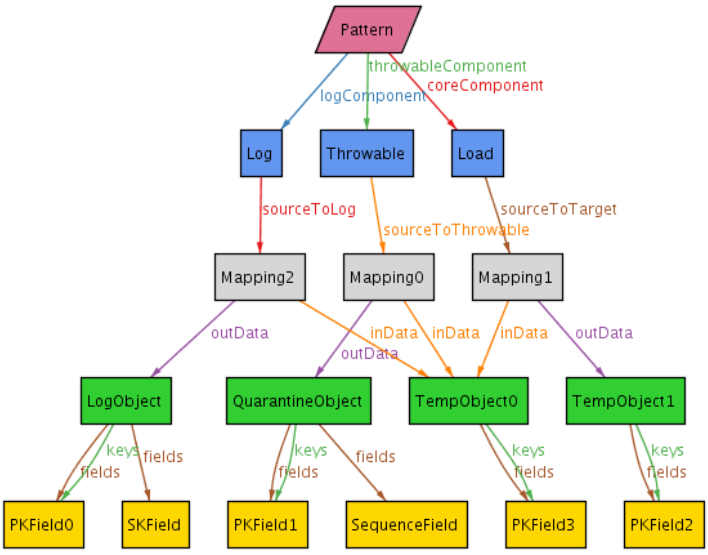


Figure 12. The main core of the pattern specification.

The "PatternComponent" signature represents the pattern components described and the mappings they embody. Based on the ontology, the "PatternCore" signature is composed by the: "sourceToTarget" mapping that represents the set of relationships between source and target fields for the pattern application, the "Throwable" signature represents the "sourceToThrowable" mapping that describes the set of relationships between the source repository' fields and error handling support structures, and the "Log" signature that represents the "sourceToLog" mapping that defines the set of field relationships between source schema and target log structure. According to its own requirements, each "PatternCore" specialization should preserve additional constraints over the arity of these fields according to their own requirements.

The Figure 12 presents a possible pattern instance with the representation of three "PatternComponent". To simplify its representation, the fields associations for "Mapping" signature were omitted. This pattern is composed by one "Log" component that establishes the relationship between source data stored in the "TempObject0" repository and the target "LogObject" repository for log activities, one "Throwable" component describing the relationship between source data stored in the "TempObject0" repository and the target "QuarantineObject" repository for error handling, and the "CoreComponent" specialized by the "Load" signature that represents the class of patterns used for data load between the "TempObject0" repository and the "TempObject1" repository.

All the constraints described until now represent static constraints or states. However, it is also possible to describe dynamic behaviour using Alloy. Basically, an Alloy model can be analysed with specific constraints and checked if it follows the constraints presented by using a set of predicates. Basically, there are two different (and related) perspectives that can be modelled, namely: the pattern structure and the pattern behaviour based on the structural components defined. This means that fields belonging to signatures are used in the pattern behaviour validation, revelling the integration of both perspectives. To support such analysis, a specific "Row" signature was defined for describing the association between a field and a specific value ("Value" signature). This association is included in the "DataObject" signature to describe the data row that data objects can have. Next, the "rows" fact describes all the possible row states, considering that a specific row should have a set of values for the fields defined in the data object configuration.

While facts represent constraints enforced by the system, certain constraints should not be enforced but rather preserved by the model, denoting the notion of correctness in the specification. These predicates include properties, like the uniqueness of the values in SK fields ("consistentDataObject" predicate), and the referential integrity constraints between the SK fields from each data source involved. As defined above, predicates embody the notion of pattern consistency, checking whether the constraints are themselves valid. The model can then be automatically processed by the Alloy Analyser, either for simulating instances according to the specification or for checking the correctness of concrete instances.

In the next sections, we will present the three main pattern components we idealize: the "Core" components in which we describe all the procedures we developed to build the pattern-oriented framework (section 3.3); the "Throwable" component describing the general strategies to handle unexpected situations (section 3.4); and the "Log" component describing ETL events registering strategies (section 3.5). To simplify the representation of each "Core" component logic, we will present excerpts of their Alloy models for describing structural constraints and associated objects. Additionally, we will also use some BPMN diagrams to describe patterns instances, representing their main tasks along with their workflow constraints that should be generated for solving specific application scenarios.

As already presented, the use of BPMN for representing ETL workflows was revealed by El Akkaoui and Zimanyi (2009). BPMN revealed interesting aspects of ETL processes representation, providing a powerful notation and language to express ETL requirements, simple to read, understand and share. The expression capabilities provided by BPMN allows to that two different persons can express the same process using different perspectives or different constructs, which is very useful to understand the description of general strategies we presented to express each pattern instance presented. Since we are essentially describing computational algorithms, we use only a small subset of the BPMN artefacts available (for example, the human tasks are not used). We use BPMN pools and lanes to describe different process branches that involve different entities and specific transformation processes. For each pool representing the main pattern core instance, three lanes are used, each one to each pattern component described before. The BPMN sub-processes are used to distinguish between atomic tasks that represent a single operation and composite activities that represent multiple tasks that work together to perform a job. Sometimes, at this representation level, the choice between these BPMN activities is done based on activity complexity or just by designer preference, since in most cases the activities can involve more than one task to be performed (depending on the detail used for the process description). For some tasks, the BPMN sub-processes are represented as "Transaction sub process" (double line border), meaning that all the individual activities represent a single logical unit of work so that they either succeed or fail together. The "Exclusive Gateways" are used to select the appropriate path that should be followed by the process based on specific conditions. Finally, the BPMN events are used to represent process start, process end or intermediate events that can occur within task scope.

3.3 The Core Components of a Pattern

The core of a pattern represents the logic behind each pattern's operational requirements and provides the communication with specific instances of "Throwable" and "Log" components. Next, we will present a set of specific patterns, which were especially designed for covering some of the most important procedures that can be included in a complete ETL system, according to its main development stages: extraction, transformation, and load, which are represented in Figure 10 as Alloy specializations of the "Pattern" signature. We express patterns composition based on a three-part rule describing the context they are applied: the forces that typically compromise or difficulty the pattern application and how the proposed solution (Gabriel, 1998) solve them (Appleton, 2000).

3.3.1 The Change Data Capture Pattern

According to Inmon (2005), a DW must be a subject, non-volatile and time-variant repository. Once entered in a DW, data should not change in any form and the raw data captured in information source systems should be maintained as when they were gathered, in order to capture business

requirements changes and discover business trends. However, identifying changed data it is not a simple task. An ETL process must be capable of detecting changes in data sources, following the execution of daily business operations in all data procedures that were made - insert, update or delete -, which may affect the DW data representation but also the calculated measures involved with. Change Data Capture (CDC) (Kimball and Ross, 2013) methods are used to detect data changes on information sources since the last successful data extraction. One general strategy for implementing a CDC pattern can be defined as follows:

Name: Change Data Capture (CDC).

Classification: Extraction pattern.

Problem: After the DW initial load, we should use some specific data identification techniques to detect changed data in source systems for reflecting those changes into the DW.

Forces: The following aspects can compromise or difficult the pattern application:

- **Legacy or limited systems**: Dealing with old or very specific technology can bring several problems for implementing mechanisms that allow for the identification of new or changed data.
- **Intrusive techniques and performance bottlenecks**: The CDC techniques can be implemented using built-in applications in information source systems. For example, the use of triggers is one of the most known techniques for implementing CDC strategies. However, these mechanisms can influence source systems daily operations, causing performance bottlenecks or increasing their maintenance costs. Additionally, some CDC techniques can be impractical since demand significant computational resources for the identification of new data changes.
- **Proprietary technology**: Some source systems provide transaction logs having registered all system events in a specific time interval. However, in most cases, transaction logs are specific structures that require specific software for extracting and interpreting data. Thus, to manage this, we need to implement a personalized solution, often using a third-party software, for extracting data we need. Regardless the approach followed, the maintenance process can be quite problematic, since new software updates can influence log structures organization, which may imply additional efforts to maintain extraction log-based applications working.
- **Real data changes**: Some data changes, coming from data corrections processes, for instance, should not be considered for data extraction. Real data changes and data corrections should be properly distinguished appropriately in an ETL extraction process.

As already described, there are two main approaches for implementing a CDC process: the differential extraction and the incremental extraction. Choosing the most suitable one depends on the source

systems requirements and limitations. Using differential extraction, all the data coming from the source will be compared, record by record and field by field, with the one already stored in target repositories. This is necessary to identify new data. However, in the incremental extraction approach, we use special purpose mechanisms for tracking changes (e.g. triggers) that prepare data for extraction, based on insert, update or delete operations that may occur in the data source. Tagging source data using audit columns is also another possible approach.

```
(...)
enum ExtractionInterval{DAILY, HOUR, MONTH, (...)}
sig AuditObject extends SourceObject {
    date_fields: some DateField & fields,
    opr_fields: some OperationField & fields,
    var_fields: some VariationField & fields
}
sig CDC extends Extract{
   lastRecordId:one PkField
   extractionInterval: lone ExtractionInterval
}
{one sourceToTarget}
fact consistentCDC {
   all cdc:CDC {
   cdc.coreComponent sourceToTarget.inData in AuditObject
   cdc.coreComponent sourceToTarget.outData in AuditObject
   let i = (cdc.coreComponent.sourceToTarget.inData),
       o = (cdc.coreComponent sourceToTarget.outData) {
          (cdc.coreComponent sourceToTarget.association).(o.var_fields)
in i.var_fields
          (...)
}
fact consistentPatternForCDC{(…)}
(...)
```

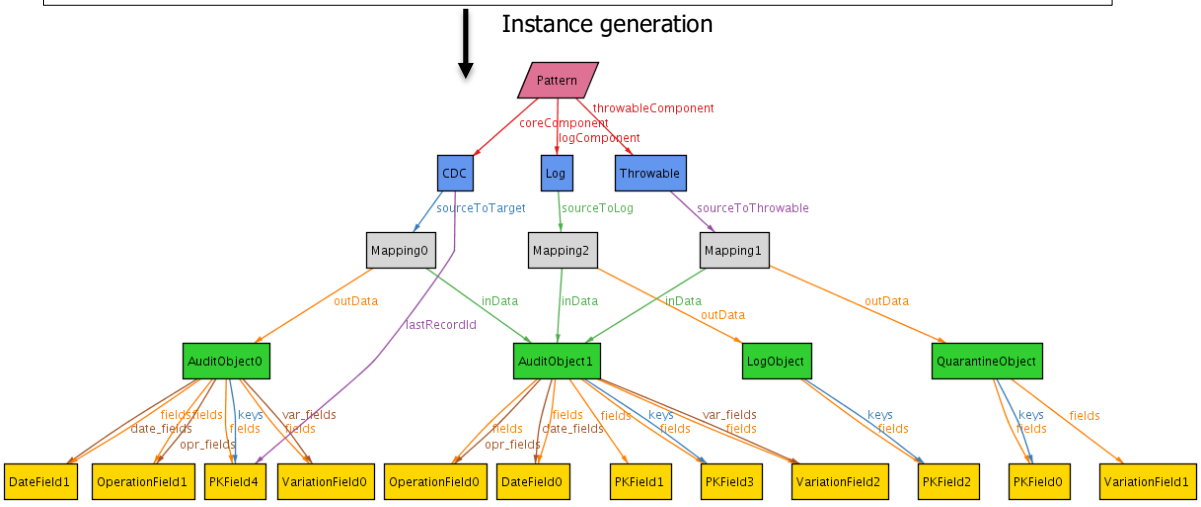


Figure 13. A CDC pattern and an instance excerpt using Alloy.

For example, temporal and specific control fields can be used for identifying changes in source systems. However, sometimes this approach cannot be used because it can affect significantly system performance. Thus, a database policy or some database management systems (DBMS) limitations can be serious obstacles for implementing some event-based mechanisms like triggers. Fortunately, most DBMS use a transaction log for storing all data modifications occurred in the system. Having this,

scanning and processing transaction log contents allow for the identification of such changes. This method does not affect transactional data and does not imply the implementation of additional mechanisms in transactional systems.

Table 1. A Simplified View of a Transaction Log File

C. LSN	T.ID	Operation	Transaction Name	Alloc Unit Name	RowLog Contents
1	1	LOP_BEGIN_XACT	INSERT	*NULL*	*NULL*
2	1	LOP_INSERT_ROWS	*NULL*	dbo.product	0x10…
3	1	LOP_COMMIT_XACT	*NULL*	*NULL*	*NULL*
4	2	LOP_BEGIN_XACT	UPDATE	*NULL*	*NULL*
5	2	LOP_MODIFY_ROW	*NULL*	dbo.product	0x63…
6	2	LOP_MODIFY_ROW	*NULL*	dbo.product	0x63…
7	2	LOP_COMMIT_XACT	*NULL*	*NULL*	*NULL*
…	…	…	…	…	…

Regardless the method used, when some change occurs, the event and the latest version of the record extracted are stored using specific data structures supported by "Audit" objects (Figure 13). Audit tables are a very common approach for representing audit objects, which integrate (at least) some fields (fields) with a variation control status ("var_fields"), the operation ("opr_fields") performed, and a date structure ("date_fields") for representing a set of fields used on a "DataObject" signature. The date fields describe the temporal data for the action performed in the source systems, while the operation fields designate the operation performed, and the variation fields identify new data versions. The Alloy intersection operator (&) helps to enforce all data object constraints, allowing to restrict the nature of the fields inherited by the "DataObject" signature.

The "Extract" signature is specialized for representing the CDC pattern signature, inheriting the "sourceToTarget" mappings from the "PatternComponent" signature. This mapping describes the relationship between the source audit object used by the CDC process and the respective target repository. The "sourceToTarget" mapping describes the use of one audit object as input and one audit data object as output for storing the extracted data temporarily in the DSA. Audit objects differ from temp and source objects because they store data with specific metadata associated (control fields). Some additional facts, such as "consistentCDC", enforce the use of an "AuditObject" signature as input and output for the "sourceToTarget" mapping, as well as the correct association between fields from the same nature for serving the CDC pattern specific needs. Two additional fields are used to represent the last record read and support subsequent data extraction processes ("lastRecordId"), and the extraction interval configuration ("extractionInterval").

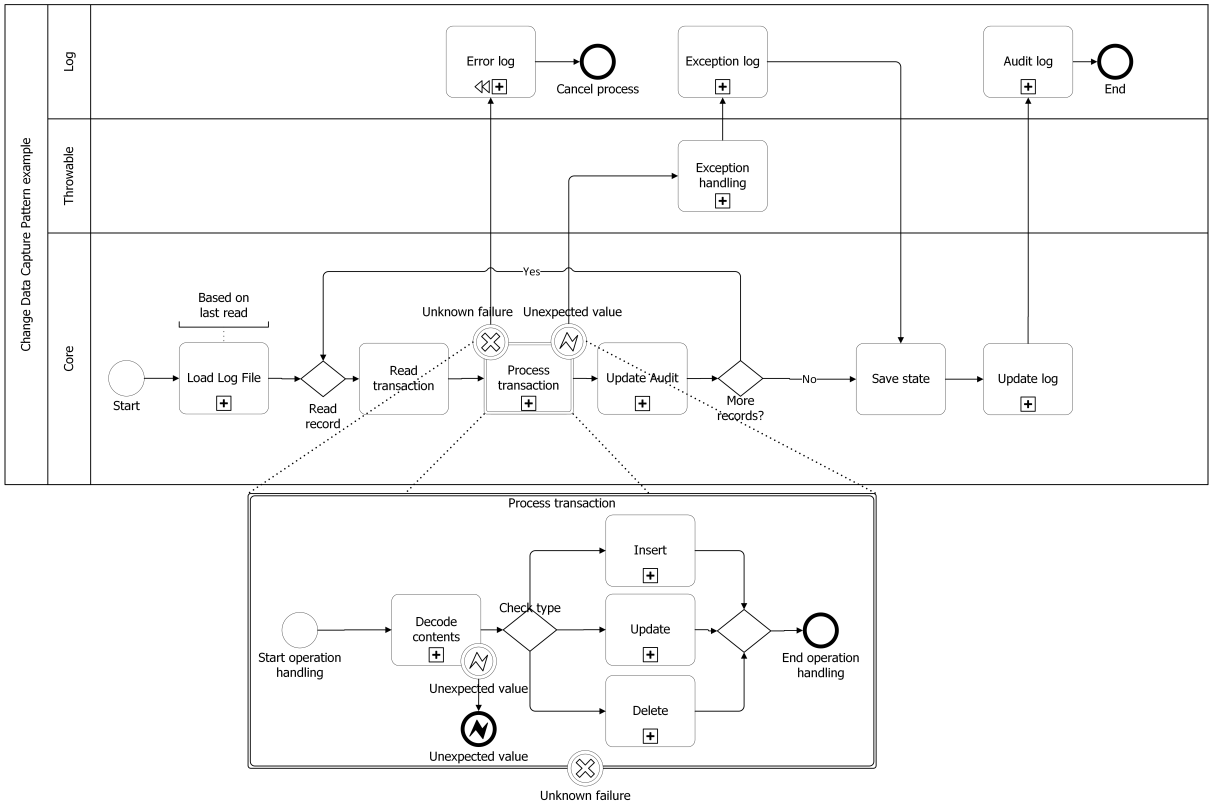


Figure 14. A CDC instantiation example representation using BPMN.

The "sourceToThrowable" and "sourceToLog" mappings from "Throwable" and "Log" components, respectively, impose additional requirements related to the relationship that should be defined using mappings between "AuditObjects" and "Throwable" data objects (for input and output respectively). Those restrictions are imposed by the "consistentPatternCDC" fact for the generation of well-formed "CDC" signature instances. In Figure 13 we can see an excerpt of the Alloy specification representing a subset of the ETL patterns meta-model describing the CDC pattern structure and constraints including some related objects. Additionally, it is also presented an instance that was generated based on a single specification excerpt for a pattern with three "PatternComponent": "Core", "Throwable" and "Log". More specifically, the example shows how the mappings from each component are related to the "DataObjects" used for the CDC "PatternCore" specialization. The "Throwable" and "Log" components use as input the source "AuditObject", storing the output data in "QuarantineObject" and "LogObject" respectively. Since the representation of all pattern elements for this instance could compromise the pattern instance readability, the association between fields from each "DataObject" was omitted from the Figure 13.

Table 1 represents a simplified DBMS log transaction file conceptual view. In this table, we can see "LSN" (Log Sequence Number) that represents a unique identifier for each record inside the transaction log, which is represented only by a simple integer value. The transaction identification ("T.ID") is another element presented in the table. It preserves the modifications that belong to the same transaction ID, providing a way to identify related operations. Each transaction begins with a

'LOP_BEGIN_XACT' operation and ends with a 'LOP_COMMIT_XACT' operation. Between these two rows, many 'LOP_MODIFY_ROW' operations can occur, representing a specific type of modification. The 'LOP_BEGIN_XACT' value represents a specific type of transaction described by a transaction name column: 'INSERT', 'UPDATE' or 'DELETE'. For each operation, the "AllocUnitName" column represents the respective table where the modification was performed. The changes are stored in the "RowLog contents" column, which sustains the data in a hexadecimal format. These contents need to be decoded to interpret and analyse data in a more readable way.

To solve this application scenario using pattern approach, the CDC process begins by loading the log file accordingly to its initial configuration. All mappings should be provided for identifying the source repository (in this case the log file), the variation fields (e.g. "RowLog contents" column), and the control fields (e.g. "Transaction Name"). If available, transaction log must be read based on the previous record read (using a specific control field). That is, instead of reading records from the log file, the process reads the transaction log entries based on the last one that was read.

In Figure 14, we can see a BPMN model representing the main steps that should be generated for handling the described example. The algorithm represents several BPMN lanes within a pool in order to describe different process branches related to each pattern component described before ("Core", "Throwable" and "Log"). Thus, the tasks related to the configuration of each component are grouped and better defined according to their scope, becoming easier to read and follow. The BPMN tasks (e.g. "Read transaction") and (collapsed) sub processes (e.g. "Load Log File") are used to distinguish between atomic tasks and composite activities that represent multiple tasks that work together to perform a job. For the "Process Transaction" sub process, an expanded sub process is also used, showing a possible description of the activities that can be used internally to perform the job in which the sub process was created. This process is configured as "Transaction sub processes" (double line border), meaning that all the individual activities represent a single logical unit of work. The "Exclusive Gateways" are used to select the appropriated path that should be followed by the process (e.g. to select the appropriated task inside "For each operation" expanded sub process). Finally, the BPMN events represent the triggers associated with different scenarios, namely: the "Start" event based on a specific schedule, and the "Cancel" and "Error" events. The "Error" events represent serious events that trigger during an activity execution, while the cancel events cancel the transactional sub process and triggering the correspondent compensation. In the Figure 12 context, the "Error" event is handled by the activity represented in the "Throwable" lane and the "Cancel" event is handled by the "Log" lane to store the event using the log mechanisms before the process abortion.

After reading the log file ("Load Log" File sub process – it is assumed that records are read to memory), we can find an expanded sub process that represents all the tasks need to handle the transaction log event. After reading each entry ("Read Transaction" task) in the log, a decoding

algorithm for extracting meaningful data ("Decode Contents" sub process) is executed. The decoding process can be implemented using external applications that are invoked and coordinated by a CDC instance. Next, the "Process transaction" sub process identifies the type of transaction received: "Insert", "Update" and "Delete", applying all the actions associated to each one of the transactions. The process starts with a "Timer Start Event", which describes a repetitive process execution based on time-based cycles (configured by the "extractionInterval" property). When all the transactions are processed, the last processed record identification is preserved ("Save State" sub process) in order to support future CDC executions. As mentioned, the "Throwable" and "Log" components represent distinct roles and are described using different BPMN lanes. In this case, two "Throwable" events were configured: 1) a BPMN intermediated error event to describe errors that can occur in "Decode Contents" sub process, and 2) a cancellation BPMN event for describing a (unexpected) critical error that can occur during "Process transaction" sub process. The exception scenario throws a specific event when the decoded contents are unexpected ("Exception Handling" sub process) and updates the exception log ("Exception log" sub process) before the iteration ends (a convergent exclusive gateway is used). For the cancellation event, the "Error log" sub process is used to update a specific log file and the CDC process ends, preserving the process current state.

3.3.2 The Slowly Changing Dimension Pattern

Usually, DWSs are built based on the concept of temporal data. The facts stored in fact tables are linked to a specific date that refers the moment in which they occurred. Fact and some dimension tables are strongly related with the time dimension table and they can be affected by time, which means that some elements of data they own can change, being necessary to associate their validity to a certain time interval. To maintain the DW consistency, that data coming from data sources can be modified for reflecting a change in a given point of time that should be preserved.

When a DW is updated, we need to take some decisions in order to maintain a consistent view of its data. The Slowly Changing Dimension (SCD) pattern embodies some well-defined policies representing design patterns to support old and new data over time in the context of a DW's dimension table. Several types of SCD can be applied (Kimball and Caserta, 2004; Kimball and Ross, 2002), each one considering specific application scenarios. The SCD pattern embodies a specific SCD strategy for preserving the dimension historical data using a secondary table. Thus, this approach involves the use of two distinct tables: one stores up-to-date dimension's data (the dimension table) and the other one stores the previous version of the record values (the historical table). With this strategy, all current and historical records are stored with no limitations, being easy to compute when compared to other similar approaches (Santos and Belo, 2011). Next, we can find the SCD pattern generic description:

Name: Slowly Changing Dimension (SCD).

Classification: Transformation pattern.

Problem: Due to several reasons, the dimensional data of a DW can change over time, imposing in some application cases the preservation of older versions of data for supporting historical data analysis and reporting.

Forces: The following aspects can compromise or difficult the application of the pattern:

- **Audit data preservation**: The change data capture processes are responsible for storing all the details related to the changes occurred in the information source systems. The audit data are used for triggering the correspondent activity for each predefined case, e.g. insert, update and delete. However, the occurrence of some data errors or other technological aspects limitations can cause inconsistencies, which should be properly handled, obviously. For example, an update operation over a record that does not exist in the target repository causes an exception scenario, which means this record should be handled as new one.
- **Multiple changes for the same record**: Conflicting changes involving the same record (usually between different data sources) can cause some consistency problems. These problems should be manually analysed.
- **Data is valid only in a specific time-interval**: historical data are stored using specific historic dimensions, while current data are stored using the main dimension table. However, current and old data should be somehow related and flagged as valid for a given period.

The process associated with a SCD pattern instance begins every time that is detected a change in a data object that is selected for populating the target system. Typically, these data are stored using some specific DSA audit tables, which purpose is simply to store the records that were changed. Audit tables are used in the DSA, providing to the SCD process the records that need to be processed according to the operation performed in source systems. When records are new ("Insert"), they will be stored in the dimension table as current data, while deleted records will be marked as inactive in target dimension. For updated records, a more complex process should be initiated, since the current versions of them should be transferred to the history table (partially or totally), accordingly to the specification of the dimension table's SCD. All or some of these operations can be registered in the system's log file, identifying the record, the performed operation, the data source origin, and the temporal data.

The quarantine objects also play a significant role for SCD process, since can be used for processing unexpected scenarios (configurable from the "Throwable" component) that do not compromise the process operation. If structural or data entry errors occur, they can be redirected to quarantine tables, where inconsistent records usually are stored, identifying (at least) the record, the error occurred, and the temporal data. Posteriorly, these records can be analysed using specific procedures (or even manually) to be reintroduced in the ETL workflow, or deleted in case of not recovering.

When a record suffers more than one modification, they are handled in chronologic order according to the temporal labels provided by the source. Furthermore, other verifications can be performed to detect anomalous inconsistencies and trigger events to correct them. When all the records were processed, they will be loaded into the target dimension table.

```
(...)
sig DimensionObject extends DataObject {
    vFields: some VariationField & fields,
    state: one StateField & fields
}
sig HistoryDimensionObject extends Dimension {
   controlFields: some ControlField & fields
   variatonFields: some VariationField & fields,
   foreignKey: one SKField & fields
}
sig SCD extends Transform{
   auditToHistory:one Mapping
}{
   one coreComponent.sourceToTarget (...)
}
fact consistentSCD {all scd : SCD {
    scd.dimensionToHistory.inData in DimensionObject
   (...)
}
(...)
```

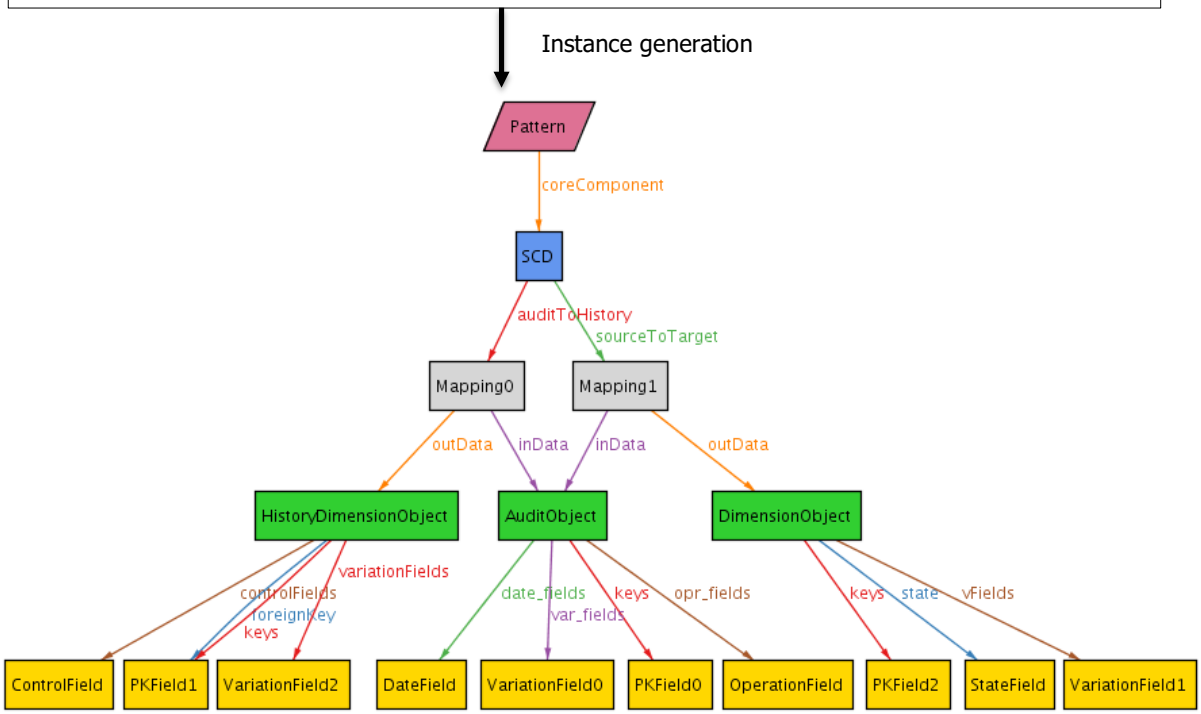


Figure 15. Describing a SCD pattern and an instance excerpt using Alloy.

The Figure 15 presents the structural and operational constraints using an Alloy specification, representing a subset of the ETL patterns meta-model for the SCD structure representation and the related objects. Additionally, it is also presented a random instance generated by the Alloy Analyser.

The "PatternCore" signature is specialised for representing SCD patterns, containing the inherited mappings with additional constraints based on specific data objects and mappings. The

"DimensionObject" signature represents the data object that keeps the current data, while the "HistoryDimensionObject" signature represents the data object that stores the historical data. Each one of these objects will have fields from several types, revelling the specific operational characteristics of each one. In addition to the inherited properties, the dimension object ("DimensionObject") is composed of a set of variation attributes ("VariationAttr") and a specific control field for identifying records state ('active' and 'inactive'), while the history dimension ("HistoryDimensionObject") is represented as a "DimensionObject" specialization for representing additional control fields ("ControlField") to store records metadata, such as the temporal data and the foreign key field ("ForeignKey"), pointing to the record's current version ("consistentSCD" fact). Additionally, we defined another mapping for the SCD signature: the "auditToHistory" relationship, which represents the mapping between the dimension fields and the history dimension fields. From each data object used, the correspondences between fields should be defined according to the constraints established for each mapping. For example, the "sourceToTarget" mapping should be defined over one audit object (as input) and a dimension object (as output), while the remaining mappings inherited from "Pattern" signature are constrained based on the data objects nature.

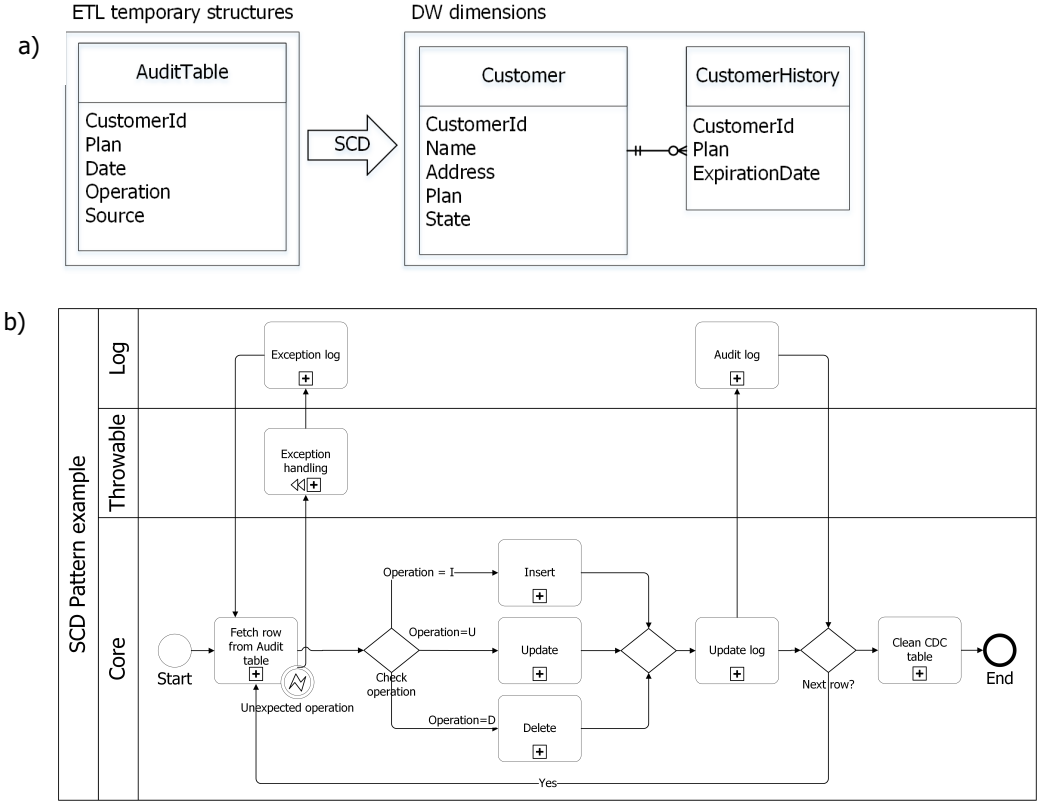


Figure 16. a) An ETL scenario representing the data structures involved with a SCD operation; b) The schematic process of a SCD instantiation represented in BPMN.

Mappings enable the preservation of variation, control, surrogate key and data fields between data objects, which are imposed by facts defined over the SCD signature. Additionally, some of the signature facts were used for enforcing the correctness of each mapping - this was necessary for mapping correctly all the related fields and avoiding invalid relationships. The pattern instance from

Figure 15 presents a "PatternCore" instance with two mappings to describe the relationship between the source "AuditObject" and the target "DimensionObject"; and the relationship between the source "AuditObject" and the target "HistoryDimensionObject" used to store historical data. To simplify the SCD instance representation, the mappings representation between the fields were omitted.

In Figure 16b it is presented a BPMN schema that represents an example of a process for feeding a "Customer" dimension that stores information about customer's data of given telecommunication company (Figure 16a). Particularly, beyond the customer personal data representation, the telecommunication plan data represents an attribute with variation that should be used for preserving historical data about the plan that each customer had at some point in time. Therefore, the history dimension table stores all historical data about changes that occurred for each customer dimension record.

The SCD pattern instance configured for handling this scenario should extract data from an "audit" object and takes a specific action based on the changes verified on the source, which can be described as the follows:

- For each new record, it is inserted a new record into the "Customer" dimension and labelled as active.
- For each updated record, it is inserted a new record with the most recent version into the "Customer" dimension and the outdated version, the record that was in the dimension table, is stored in the history table "CustomerHistory" that is associated with the dimension table, using the "Date" field from the audit table to set the "ExpirationDate" field value.
- For each deleted record, the record state is changed to 'inactive' in the dimension table "Customer".

The same methodology used to create the BPMN algorithm from Figure 14, is applied to the representation of the algorithm presented in Figure 16b. Several BPMN lanes describe different process branches related to each pattern component, grouping the activities related to each one. The "Fetch row from Audit Table" activity is represented as a BPMN "Transaction sub processes", since is composed of multiple tasks that should be applied atomically. The remaining activities are represented as BPMN "sub processes" since they can be described using multiple simpler tasks. The "Exclusive Gateways" are used to select the appropriated path that should be followed to initiate the execution of the correct strategy related to each operation value read, and to process each record extracted from the audit table. For the "Fetch row from Audit Table" sub process, a "Cancel" events are triggered for being catch by the "Exception handling" sub process, responsible to compensate the events by the application of exception handling procedures.

The strategy used in the algorithm presented in Figure 16b is based on the evaluation of the column that identifies the operation performed in source systems, triggering a specific sub process - "Insert",

"Update" and "Delete" -, respectively for each action described, considering the logic described before. Before this verification, the "Fetch row from Audit Table" sub process loads each gathered record and check for possible inconsistencies on operations over the attributes. When some inconsistencies related to the operation field values are detected when records are read from audit table (represented using the "Unexpected operation" event), the "Exception handling" sub process, which is represented in the "Throwable" pool, stores them using specific quarantine tables, returning the flow control to the "Fetch row from Audit Table" sub process. The "Log" pool represents the log procedures used either to store milestone events related to each processed record ("Audit log" sub process) or to store the exception events ("Exception log" sub process) triggered by "Exception" handling sub process represented in the "Throwable" pool that results from the identification of wrong values from audit records read. Finally, when all records are processed (an BPMN "exclusive gateway" is used to check if are more rows for processing), the audit table is cleaned.

3.3.3 The Data Quality Enhancement Pattern

Usually, operational data are stored using several data schemas built specially to serve business needs. Since these sources are used to populate the target DW, it is quite normal to deal with specific implementations based on different technologies and design perspectives. Thus, it is quite normal to occur sporadic problems on data gathering and extraction processes. Independently from the number of sources involved (single or multiple), a large diversity of problems can occur at schema or instance level every time a loading data process is executed. Considering the DW nature, uniformity and cleaning procedures should be applied properly for ensuring the correctness of data analysis processes and consequently decision-making results.

ETL processes are complex and usually work with substantial amounts of data gathered on a large diversity of heterogeneous information sources. Moreover, the complexity associated with conformation and data-cleansing tasks represents a significant issue to any ETL process. These tasks are responsible for ensuring data correctness (and homogeneity) before they are loaded into the DW, leading with business requirements that impose a lot of operational requirements that are reflected over each information source involved with, especially on gathering and conciliation tasks that should be executed. This provokes the occurrence of serious problems like attribute naming, data type conflicts, data entry errors, uncontrolled redundancy, or contradictory values. Thus, we need to define some strategies for detecting and solving these problems (Rahm and Do, 2000), as well as providing us with the necessary knowledge to create configurable components that can be instantiated to specific application scenarios. Next, we present a complete pattern description for a "Data Quality Enhancement" pattern:

Name: Data Quality Enhancement (DQE).

Classification: Transformation pattern.

Problem: Data integration complexity is highly influenced by the number of operational sources that we need to integrate into the DWS, as well as by the diversity of data schemas and inconsistencies that may exist in and between them. To avoid such undesirable cases, we need to implement strong data quality procedures for detecting and removing eventual errors or inconsistencies that operational data bring, or at least attenuate their effects.

Forces: The following aspects can compromise or difficult the application of the pattern:

- **Conflicting records**: Records from the same entity having contradictory values or domains can generate data conflicts.
- **Unexpected data**: Dictionary tables are frequently used for identifying some expected values and replace them with a specific one, according to the requirements defined in the DW schema. They can be used to replace acronyms, abbreviations or even to correct bad values already known. However, unexpected values can represent new application scenarios that do not have a correspondent value stored in dictionary tables.

Data conforming techniques involve the application of a large diversity of procedures over the records that are gathered in information sources. Some of them are: attribute decomposition and transformation, normalization of attributes contents, duplicate elimination and conflict resolution. The complexity of all these filter mechanisms and conforming data mechanisms are higher, as well as the amount of data to be processed. Additionally, in many cases, operational systems suffer sporadically from some structural changes imposed by business processes evolution. Therefore, the integration of new requirements in existing data schemas can lead to some inconsistence states. The new business requirements can force changes in the schema or the technology architecture adopted is no more adequate.

Attribute decomposition and text transformation are some of the most simpler procedures we can use on a data conforming application scenario. Regularly, the attribute decomposition task is applied over attributes like "name" or "address", providing their decomposition in smaller parts with a specific structure and a well-defined meaning. These composite attributes could be decomposed into a set of new attributes using specific-oriented rules like the string separation or tokenization based on some delimiter character. Other transformation tasks can be used, such as the capitalization of a word or a sentence, the character's transformation from upper/lower to lower/upper case characters, the renaming or addition of a new attribute, or a data schema validation. These procedures can be classified as pre-functions because they should be applied before the application of any standardization procedure.

Standardization procedures require the use of ETL metadata (e.g. mapping, equivalence or control tables) to replace source values based on a specific set of possible corrections. In these cases, we know all the incoming values that must be replaced by their correspondent target values. To do that,

we simply use a dictionary of data objects. These dictionary objects ("DictionaryObject" signature) store all possible (allowed) equivalences (surrogate values) to support data normalization or correction procedures.

```
(...)
sig DictionaryObject extends DataObject {
   sourceId: one PKField & fields,
   sourceField: one Field & fields,
   value: one Field & fields
}
abstract sig DQE extends Transform{}
abstract sig PreFunction extends DQE{(...)}
abstract sig PostFunction extends DQE{(...)}
sig DataTransform extends PreFunction{}{}
sig SchemaTransform extends PreFunction {(...)}{(...)}
sig Duplicate extends PostFunction{(...)}{(...)}
sig Standarization extends DQE{
   sourceToDictionary: one Mapping
}{
   one coreComponent.sourceToTarget (...)
}
fact ConsistentDQEStandarization {
   all dqe : DQE {
      dqe.coreComponent.sourceToTarget.inData in SourceObject
      dqe.coreComponent.sourceToTarget.outData in Tempbject
      (...)
      dqe.sourceToDictionary.inData in DataObject
      dqe.sourceToDictionary.outData in DictionaryObject
(...)
```

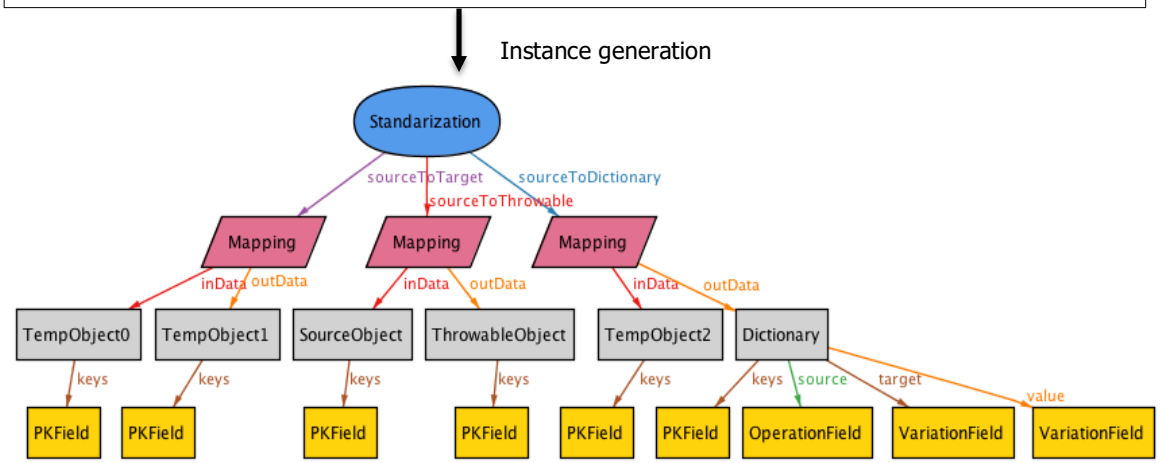


Figure 17. Describing a DQE pattern and an instance excerpt using Alloy

For example, they can be used to solve name abbreviations, replace values by correspondent synonyms or correct invalid values, according to specific bindings established between the values. Regardless the method used (single-source or multi-source), several serious problems can occur. For example, it is quite usual to proceed to the identification of conflict values that can result from the integration of data about the same identity. In such cases, the application of specific some data correction or transformation mechanisms it will be necessary in order to maintain the correctness and the data consistency across all the information sources accessed.

The duplicate elimination and conflict resolution procedures are usually used in problems that occur after the integration of data coming from multiple and disparate information sources. The first one is usually applied after the execution of all correction and standardization procedures and involves the elimination of common records based on some predefined criteria. Additionally, it may occur some conflicts with other pieces of data related to inconsistent values among similar instances. In such cases, we need to implement other procedures for solving automatically (if possible) every case of data inconsistency that may occur - for example, it is very common in business scenarios to find the same customer having different addresses in two different operational systems. Thus, to choose the most trustable values we need to establish and apply conciliation poilicies. However, if a value cannot be handled or conformed, all the records affected should be stored using quarantine repositories.

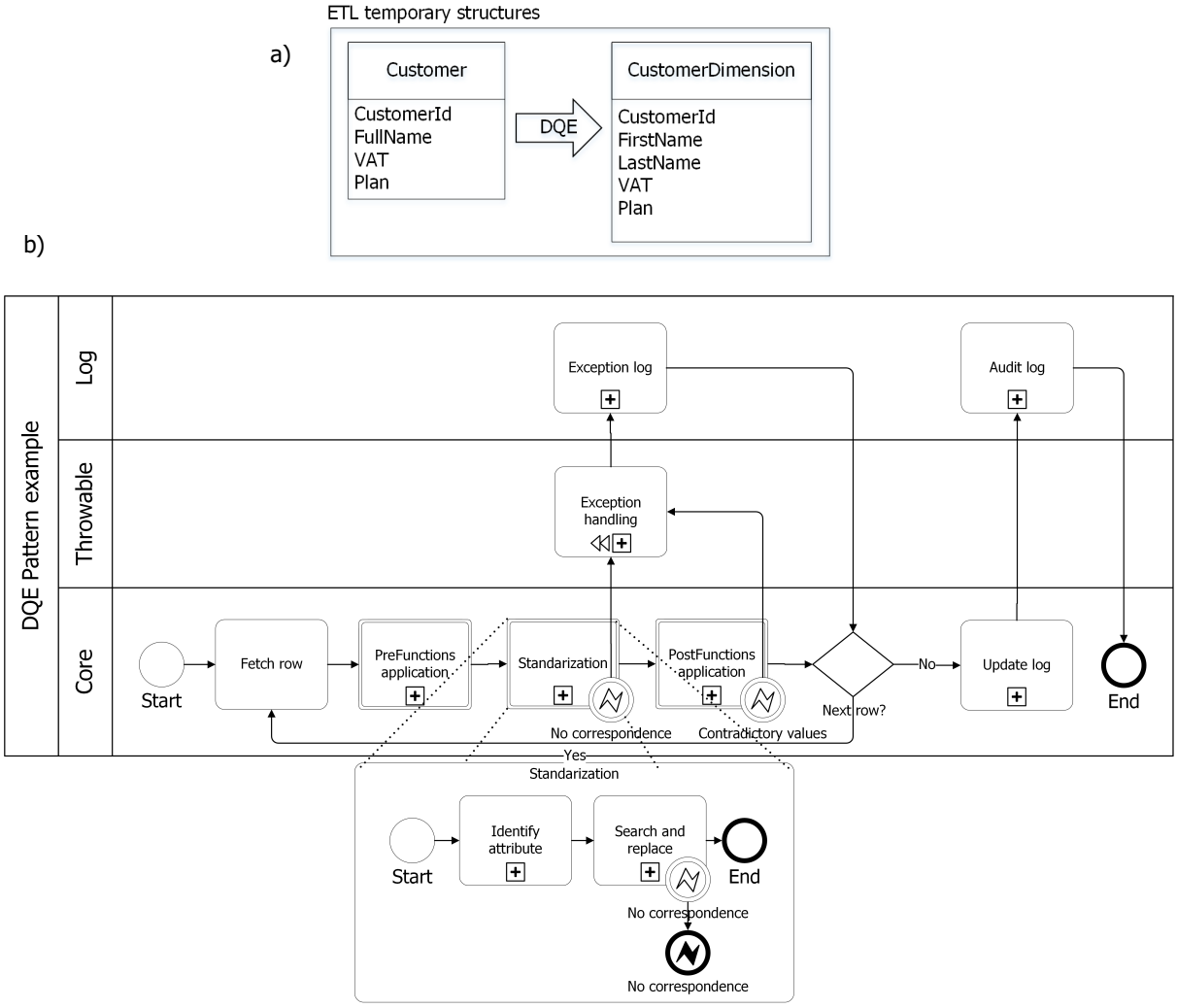


Figure 18. a) Data structures used in an DQE pattern application; b) Schematic view of a DQE instantiation process in BPMN.

The Alloy specification presented in Figure 17 shows the general concepts and constraints related to the DQE pattern structure. Each DQE procedure is represented in a hierarchical way ("PreFunction",

"Standardization" and "PostFunction" signatures) for representing the DQE signature specializations as quality enforcement procedures with particular needs.

All DQE specializations should be enriched with additional fields to support its application. For example, the "Standardization" signature defines the mapping between source data objects and the data object ("DictionaryDataObject") holding the target dictionary data. This object stores the relationship between all the expected (wrong) values ("SourceId" identifies the source and the "sourceField" the source field used) and their correspondent correction values ("value"). The remaining specification properties related to other procedures are enforced by using specific data objects for the mappings defined previously. For example, the "ConsistentDQEStandarization" fact enforces that "sourceToDictionary" mapping should be performed between one "SourceObject", as input, and one "DictionaryObject", as output. In addition to the Alloy specification excerpt, is also possible to see in Figure 17 a random instance generated by the Alloy Analyser (without the representation of mappings between attributes), showing the representation of pattern structure along with the representation of the "sourceToDictionary" mapping that describes the relationship between attributes of the source object used and the target dictionary object (the attributes representation for each source was simplified).

Let us consider an ETL sub process that is responsible for applying some conformation and correction procedures over some customers' data of a company customer's, using predefined business requirements – the data structures are presented in Figure 18a. The source table ("Customer") is composed by the attributes customer identification ("CustomerId"), customer full name ("FullName"), customer value-added tax number (VAT), and a communication plan subscribed by each customer. The target table ("CustomerDimension") holds the attributes customer name using two different fields ("FirstName" and "LastName"), the plan details that should be normalized using a specific dictionary table due the different categorization used, and the VAT field that is used to identify duplicate customers.

For migrating data between tables, we need to use three distinct procedures, namely: a decomposition procedure, for splitting customer full name into two separate attributes (first and last name), a standardization procedure, for conforming customer's telecommunication plan values and a duplicate elimination procedure for eliminating eventual duplicate customers. In Figure 18b it is presented a DQE pattern instance designed to sustain the requirements of the example we gave. For each record read ("Read record" sub process), three DQE specializations are used to describe the data quality procedures applied to each incoming record: "PreFunctions application" sub process for customer name decomposition procedure, "Standardization" sub process for normalizing plan values, and "PostFunctions" sub process to eliminate customer duplicates. Two exception scenarios were configured for handling the correspondences without match detected in the dictionary object ("No correspondence" BPMN intermediate event attached to "Standardization" sub process), and for the

identification of contradictory values between costumer's first and last name when a duplicate elimination procedure is applied ("Contradictory values" intermediate event attached to "PostFunctions application" sub process).

```
(...)
sig MappingSKPObject extends MappingObject {
   targetDimension: one DimensionObject
}
fact consistentMappingSKPObject{(...)}
sig FactObject extends OperationalObject {
    measures: some Field & fields,
    temporalFields: some DateField & fields
}
(...)
enum ConditionalOperator{LT, LTE, GT, GTE, EQ}
sig LateFactsRule{
   associationField: one Field,
   value: one Value,
   condition: some ConditionalOperator,
   mappingTable: one MappingSKPObject
}

sig SKP extends Transform{
   sourceToMapping: some Mapping,
   lateFacts: set LateFactRule
}
fact consistentSKP {
   all skp: SKP {
      skp.coreComponent.sourceToTarget.inData in FactObject
      skp.coreComponent.sourceToTarget.outData in FactObject
      (...)
      skp.sourceToMapping.inData = skp.coreComponent.sourceToTarget.inData
      skp.sourceToMapping.outData in MappingSKPObject
      (...)
   }
}
```

Instance generation

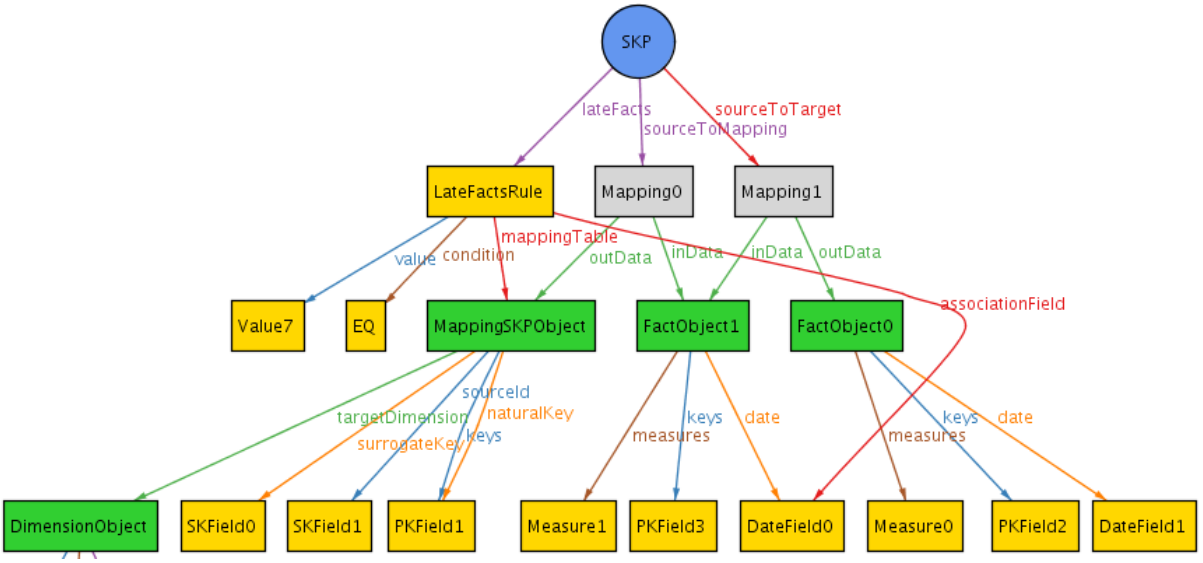


Figure 19. Describing a SKP pattern and an instance excerpt generation using Alloy.

Both events are handled by the "Exception handling" compensation sub process that stores provable inconsistent records into a predefined quarantine table. Next, the correspondent log object is updated using the "Exception log" sub process from the "Log" pool. All records detected in an inconsistent state are stored in quarantine tables for posterior analysis. After the application of the mentioned procedures on all the records, the "Update log" sub process will update the audit log object ("Audit log" sub process) to catalogue the achieved process milestone. The remaining activities were configured according to the rules described in section 3.3.1.

3.3.4 The Surrogate Key Pipelining Pattern

Due to many reasons, a fact table feeding process requires the conversion of operational keys that come from data sources to specific surrogate keys. This imposes the maintenance of a map keeping all the valid correspondences between operational and surrogate keys. Otherwise, we may lose valuable information about the origin of the facts. The Surrogate Key Pipelining (SKP) is a very common task that has a large application in many ETL systems (Kimball and Ross, 2013).

Following a multidimensional model approach, the referential integrity between the fact table foreign keys and each dimension primary key should be a guarantee. The generation of surrogate keys must be a separated process, used to generate independent keys for each record extracted from the source operational systems, maintaining the records independence between data sources and at the same time, preserving the record's source identification from they were extracted. Next, it is presented a detailed description of the SKP pattern:

Name: Surrogate Key Pipelining (SKP).

Classification: Transformation pattern.

Problem: The fact table feeding process frequently requires the conversion of operational keys that come from data sources to specific surrogate keys, imposing the maintenance of a set of mappings of all the valid correspondences between operational and surrogate keys.

Forces: The following aspects can compromise or difficult the application of the pattern:

- **Referential integrity** - After natural key replacement, the correspondent SK should be a valid foreign key for the target dimension table. If not, the records involved with may stay inconsistent.
- **Surrogate key missing** - During the replacement of operational keys, some of the records involved in the process can have new keys, which trigger non-correspondence scenarios. In cases like this, the process should be able to generate a new SK for the records or separate them from the normal ETL flow.

- **Inconsistent operational keys** – All the natural keys that are inconsistent should be identified. For example, if the same operational key refers two or more different records - Kimball and Caserta (2004) referred that such keys must be marked as 'unknown'.
- **Late arriving fact records** – All the fact events that are mapped to older versions of data should be handled by some other mechanisms to find the SK based on the time records were valid.

For each natural key, specific queries are executed over the corresponding mapping data object (a typical lookup operation) to get the corresponding surrogate key. The mapping object is similar in structure to the dictionary object, being used to maintain the correspondence between the natural keys and the corresponding surrogate key, for each record processed.

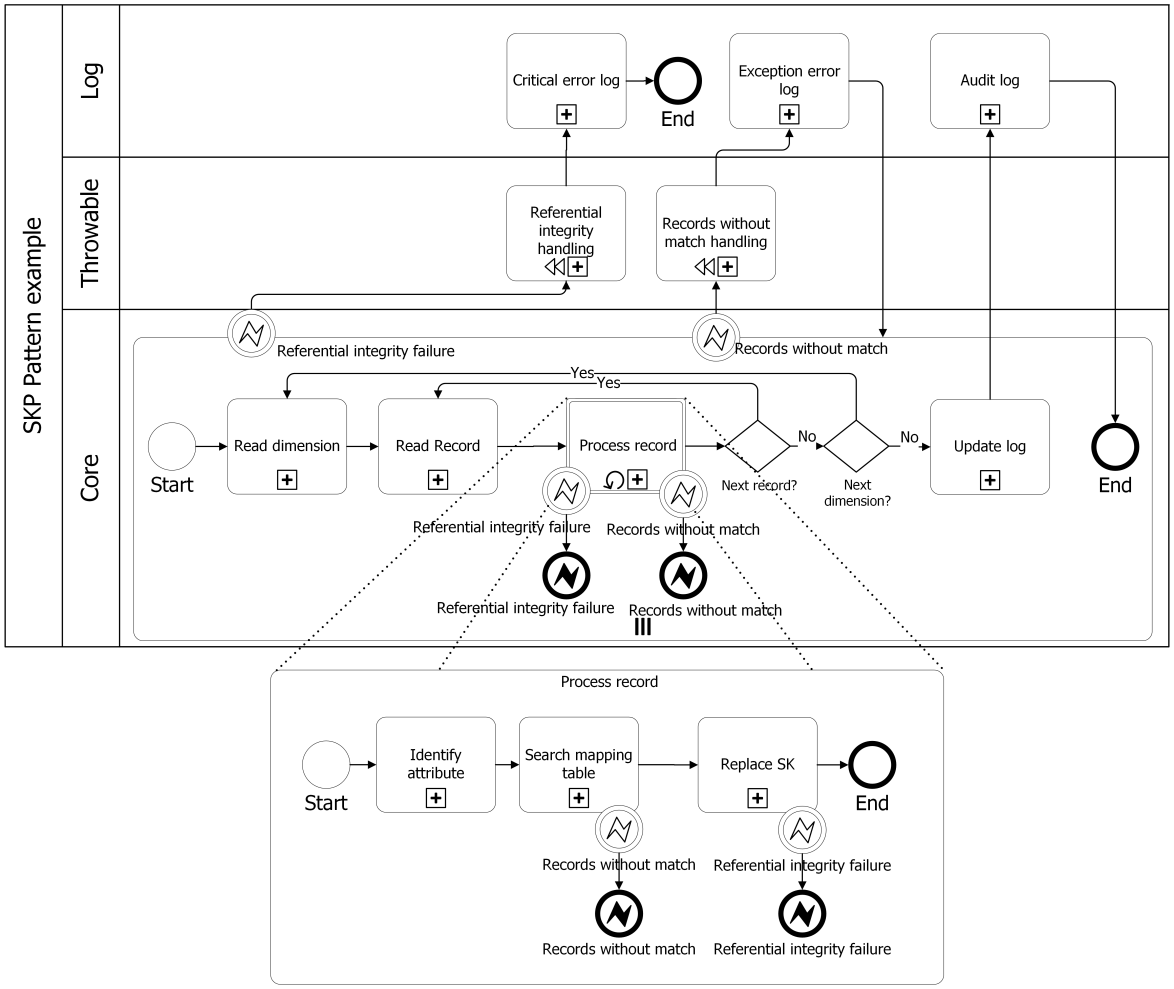


Figure 20. A SKP instantiation process represented using BPMN.

If a surrogate key does not exist for a given record, a new SK can be generated and the correspondent mapping object updated. When some inconsistent records appear, we need to put them in a quarantine zone and try to correct them automatically or store them using specific quarantine processes and objects. To improve the SKP performance, sophisticated algorithms for

ensuring parallel execution of multiple processes (if the execution environment has conditions for that) can be used with the possibility to perform each one of them as an autonomous process. This can reduce significantly the I/O cost per record, influencing positively the ETL system global performance.

The Alloy model presented in Figure 19 shows an excerpt of the main concepts related to the SKP pattern structure. The SKP pattern should represent additional mappings between the source data objects and correspondent mapping tables ("sourceToMapping") to identify the correspondent mapping table that should be used to find the respective surrogate key generated for dimension records. In the SKP pattern, the "sourceToTarget" mapping should be provided for each dimension that is involved in the SKP process. The consistency of these mappings is ensured by the "consistentSKP" Alloy fact through the preservation of structural rules. The "sourceToTarget" mapping should be defined between the source object holding the raw data ("FactObject" - a temporary fact table) as input and the target fact table ("FactObject") as output. Additionally, for each dimension used for process configuration, a correspondent mapping to the respective mapping object should be defined using the "sourceToMapping" mapping. Mapping tables specially designed ("MappingSKPObject") to identify the relationship between source data and the target fact table as well as the dimension in which data is stored.

Additionally, the "LateFactsRule" mapping represents the additional logic needed for handling late arriving facts. This scenario happens when the current dimensional record for feeding the fact records does not match any current dimensional record. This means that the record is delayed and to handle this scenario, the involved dimensions must be searched to find the correspondent dimension keys when fact event occurred. To support the logic behind this process, a specific signature ("LateFactsRule") is defined to describe the mapping that should be provided between fields along with the correspondent conditional operator ("ConditionalOperator" enumeration: e.g. 'lt' means: lesser than, 'gt' means greater than and so on) and specific temporal fields to identify the instant of time the event occurred.

The Figure 19 also presents an excerpt of an instance showing one "sourceToTarget" mapping between the source and the target fact objects, and another one, "sourceToMapping", describing the mapping between the source fact object and the target mapping object used to replace operational keys by the correspondent surrogate keys. The "LateFactsRule" is also presented, showing the field associated to the conditional rule (DateField0), the conditional operator (EQ) and the value that should be matched (Value7). The fields associations described by each mapping are not represented for the pattern instance presented.

Let us consider now a simple DW schema composed of three distinct dimensions, "Employee", "Product" and "Time", and one fact table, "Sale". To support the DW populating process, the SKP pattern instances should be configured for each dimension used. In each of them, source records are

read and using their business keys, a lookup operation is performed on the respective mapping table involved with for retrieving the correspondent surrogate key, and updating the target fact table. The BPMN process presented in Figure 20 reveals an SKP instance process designed for solving the problem we described before. All the activities described in the "Core" lane, are grouped inside an expanded BPMN sub process configured with the multi-instance marker, which means that we will use multiple process instances for handling dimension' records in a parallel way, taking into consideration the limitations of the physical environment. Parallel processes can be executed for processing each dimension table individually ("Read dimension" sub process), and for each one, the records are read ("Read Record" sub process) and processed ("Process Record"). The "Process Record" expanded sub process describes the main activities related to the identification of each record operational key using the respective mapping table, for searching the corresponding surrogate key associated. During the identification of the natural keys, an error event ("Records without match") was configured to deal with non-correspondences with the operational key that are eventually found. When the SK is found, the record is stored in the target fact table ("Replace SK" sub process). If for some reason the record stored in the target fact table causes a referential integrity violation, a specific error event ("Referential integrity failure") is triggered to stop process execution.

The "Throwable" lane describes two compensation BPMN sub processes to handle each error described before. The "Referential integrity handling" sub process stores the conflict record using a specific quarantine table, updates the log storing critical entries ("Critical error log") and terminates the process execution, while the "Records without match handling" sub process stores the new unexpected record, updates the error log ("Exception error log") and returns the control flow to the "Core" lane. When all the dimensions and their records are processed, the audit log ("Audit log" sub process) is updated to store an ETL process milestone describing the SKP process application.

3.3.5 The Surrogate Key Generator Pattern

Usually, source systems' data structures are oriented for serving data analyses requirements under a business ecosystem. Specific primary keys are simple ways to distinguish between data entities instances. In the DW context, when using more than one data source, frequently happens primary key clashes in one or more elements of data due to many disparate reasons. This provokes serious conflicts in a large variety of ETL tasks that use such business keys in preliminary stages. Furthermore, sometimes business keys are quite inefficient for searching or indexing due to inappropriate data schemas or simply because they were generated by inefficient mechanisms. For solving this, many ETL systems have implemented tasks for replacing business operational keys for meaningless surrogate keys, using an efficient way to deal with such scenarios and ensure uniqueness and appropriateness to all the DW elements.

The generation of surrogate keys is a recommended ETL procedure that helps for maintaining the independence between the DW and the data coming from information sources (Kimball and Caserta, 2004). Basically, in this SK generation process, dimension tables primary keys replace keys coming from information sources' tables, avoiding conflicts and simplifying records indexing. To ensure and guarantee an efficient SK generation process, we designed and created the SK generator pattern. The description of this pattern is the following:

Name: Surrogate Key Generator (SKG).

Classification: Transformation pattern.

Problem: The generation of DW keys without any business meaning is a very common and recommended (Kimball and Caserta, 2004) ETL procedure. A SK generation logic should consider the generation of efficient keys and at the same time must allow for tracking relationships between each operational key and its correspondent SK. The first and, posteriorly, the incremental DW load processes should also take into consideration since the logic between each one is slightly different.

Forces: The following aspects can compromise or difficult the application of the pattern:

- **Using DBMS mechanisms**: The use of some DBMS programming capabilities for supporting the generation of SK can compromise the DW performance and difficult the generation of SK (e.g. using triggers or auto increment columns).
- **Maintain SK origin meta data**: The generation of SK for DW records does not mean that operational keys should be ignored or deleted. An ETL system should maintain the track between the generated SK to the correspondent operational key and the source where it belongs for supporting subsequent load processes. Thus, we can populate a single dimension using data from different data sources, which, in some cases, involve the same data element. This means that dimension records can refer to multiple sources with different operational keys, each one representing the record source origin identification.

To avoid performance bottlenecks and inconsistency scenarios, the ETL engine should be responsible for the generation of SKs or for the control of the mechanisms in the DBMS that generate them. As discussed Kimball and Ross (2013), SKs should be simple (e.g. integer values), and easy to generate and maintain. For supporting the generation of SKs and maintain the correspondence between operational keys and their correspondent SKs, we need to define and implement some mapping tables with a similar structure as the one described previously for the specification of the SKP pattern: the business key value, the source identification and surrogate key must integrate the basic structure of the correspondent mapping table structure. Moreover, the generation process must take into consideration all the keys that were previously created for generating the new ones in DW incremental load processes.

```
(...)
sig SKG extends Transform{}
{
  sourceToMapping: one Mapping,
  sequenceField: one SequenceField
{one sourceToTarget (...)}
(...)
fact SKGConsistency {
   all skg: SKG {
      skg.coreComponent.sourceToTarget.inData in TempObject
      skg.coreComponent.sourceToTarget.outData in DimensionObject
      (...)
      skg.sourceToMapping.inData in TempObject
      skg.sourceToMapping.outData in MappingObject
      skg.sourceToMapping.inData = skg.sourceToTarget.inData
      (...)
       }
}
(...)
```

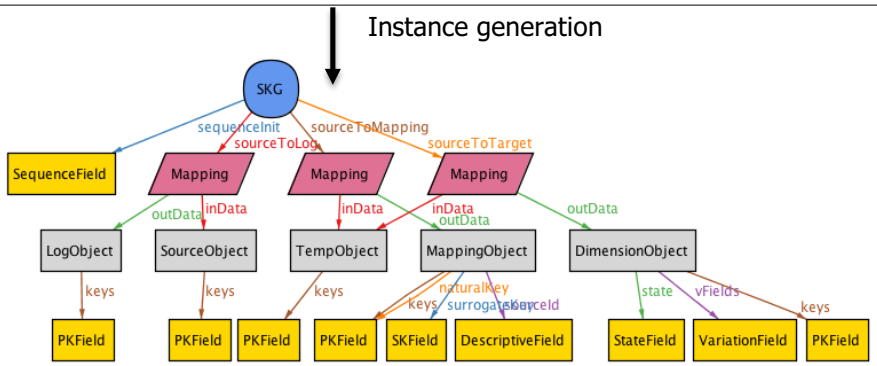


Figure 21. Alloy specification describing SKG pattern and an instance excerpt generation

In Figure 21, we can see an excerpt of the Alloy model used to support the SKG pattern and a possible instance that was generated based on Alloy specification. The SKG pattern specification uses the "sourceToMapping" mapping for describing how a source object is related to a mapping object that supports process meta data, while the "sourceToTarget" mapping should be applied between the source object and the target dimension object. The "SequenceField" signature is used to describe the field that will store the generated SK values. The pattern instance generated the by Alloy analyser shows three mappings: "SourceToTarget" describing the mapping between source object used ("tempObject" represents source data objects extracted to temporary objects stored in the DSA), "SourceToMapping" describing the relationship between source data object and the target mapping object that supports the relationship between business keys and surrogate keys, and "SourceToLog" describing the relationship between original data source object and the target log structure that holds the pattern events. Finally, the "sequenceField" represents the field used to the SK generation. The SKG instance presented in Figure 21 describes three mappings describing the relationship from the source "TempObject" to the "DimensionObject" for "sourceToTarget" mapping, to the "LogObject" for the "sourceToLog" mapping, and to the "MappingObject" for "sourceToMapping" mapping. All the SKs

are generated and stored in the "SequenceField". The fields associations described by each mapping are not visible for the pattern instance presented.

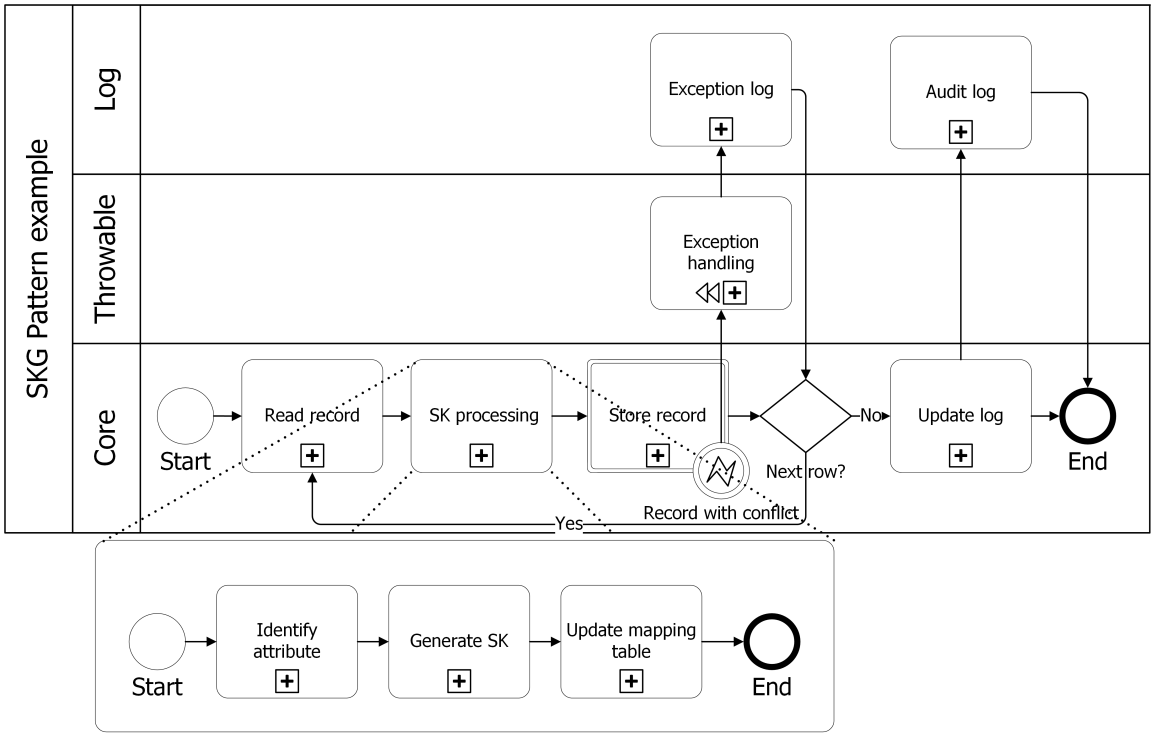


Figure 22. A BPMN schema for a SKG pattern instantiation.

As a SKG pattern application example, we consider the SK generation process for the "Product" dimension that was used for describing the SKP pattern instance presented in Figure 20. In Figure 22, we find a BPMN schema for an algorithm that can be used to generate the SK for the "Product" dimension. This BPMN schema includes all the activities needed to support the referred SKG process. For each record read from source data ("Read record" sub process), a new SK is generated based on the last SK generated using the "SK processing" sub process that is described using an expanded sub process. In this sub process, the target attribute name is identified ("Identify attribute" sub process), and the surrogate key generated ("Generate SK" sub process). For SK generation and assuming the generation of a sequence of integers, the SKG process should know the last number generated for guaranteeing the sequence number of each new SK record. The generation of new SK can be accomplished using DBMS external services or using some specific ETL engine mechanisms.

After the SK generation process is executed, the origin of each record along with its business key and the generated SK are stored in the target mapping table ("Update mapping table" sub process). In the case we have records stored in the output repository ("Store record" sub process), it is possible that may occur some conflicts between the generated SK and the stored records. These conflicts should be handled appropriately by the "Exception handling" compensation sub process, which is responsible for providing the mechanisms to solve inconsistencies or deal with roll back operations to

avoid data inconsistencies. Finally, after all, records have been processed, the audit log is updated with the metadata that sustains the pattern execution.

```
(...)
enum TypeFunction {
   MAX, MIN, AVG, SUM (...)
}
sig Function{
   field:one DescriptiveField,
   type:one TypeFunction
}
sig DataAggregator extends Transform{
   groupFields: set DescriptiveField,
   functions: some Function
}
fact DAConsistency {
   all da: DataAggregator {
      da.coreComponent.sourceToTarget.inData in TempObject
      da.coreComponent.sourceToTarget.outData in FactObject
      da.groupFields in da.coreComponent.sourceToTarget.inData.fields
      (...)
   }
}
(...)
```

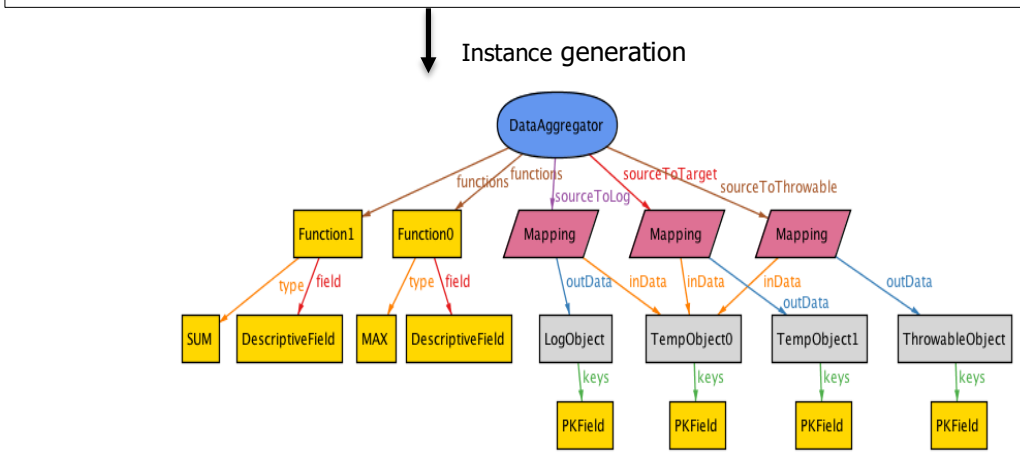


Figure 23. Alloy specification describing Data Aggregator pattern and an instance excerpt generation

3.3.6 The Data Aggregator Pattern

The process of feeding fact tables can frequently lead to the occurrence of some granularity problems between dimension and fact tables since the DW fact tables should preserve a possible atomic grain. However, sometimes the grain involves the aggregation of data involving a set of records extracted from data sources. Aggregate fact tables impose this kind of operation in order to be populated appropriately. Data aggregation procedures are used for providing data summary, serving specific reporting services or respecting analysis requirements.

Usually, a data aggregation pattern involves a set of records, which must be grouped according to some rule involving the identification of common values for one or a set of attributes following calculation method for providing a single data element. This allows for different business views, helping decision makers to better understand business to improve the decision-making processes. Next, we present a complete description of a data aggregation pattern:

Name: Data Aggregator (DA).

Classification: Transformation pattern.

Problem: Grouping together a set of records applying some specific calculations is a very typical task that is often used on data summary operations. For incremental data loads, both new and old data objects should be considered, because new records and others already stored in the DW are naturally involved with, which implies the recalculation of values previously stored.

Forces: The following aspects can compromise or difficult the application of the pattern:

- **Recalculations**: Incremental loads can lead to updating data in current calculation processes, which imposes that all records, new and old, influencing the process, should be considered for the recalculation process.
- **Incorrect aggregations**: There are cases where raw data have incorrect values, causing inconsistencies or errors in calculation processes. For example, functions, like sum and average, can only be applied to numeric types.

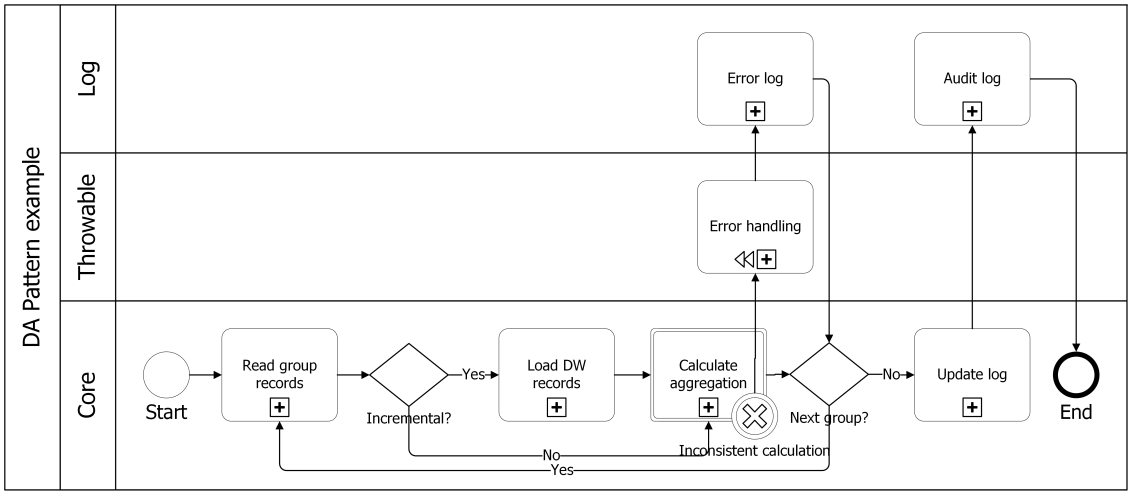


Figure 24. BPMN example algorithm for a DA instantiation.

To support incremental loads, both old and new data should be processed by the ETL system for calculating the new values for each group that need to be formed. The Figure 23 presents an excerpt of the DA pattern specification. The "sourceToTarget" mapping describes the relationship between the data objects holding the raw data and the target fact object. The group formation between similar values related to a set of attributes are defined by the "groupFields" Alloy field that uses a set of "DescriptiveField" signature to identify the target fields for group formation. The DA pattern

consistency and for this association is guarantee by the "DAConsistency" Alloy fact. For the calculations related to the group formation we can use aggregation functions such as the 'Average', 'Maximum' or 'Minimum' -, which can be identified in the Alloy specification by the "TypeFunction" enumeration. For each function used, we need to identify a specific attribute for applying the correspondent functions ("Function" signature), considering that aggregate values should always be consistent with the atomic base data.

```
(...)
enum Rule{CREATE_INSTANCE, PRESERVE, CHOOSE_BY}
(...)
sig IntegrationRules{
   association : Mapping -> Mapping,
   rule: one Rule
}
sig IntegrationRulesCompensation extends IntegrationRules {
   compensationPattern: one Transform
}{(...)}
sig IntegrationRulesChoose extends IntegrationRules {
   choosenDataObject: one DataObject
}{(...)}
sig DCI extends Transform{
   DCIrule: one IntegrationRules
}
fact DCISameOutput{all ds1, ds2:DCI.CoreComponent.sourceToTarget.outData | ds1
= ds2}
fact consistentDCI{
   all i:IntegrationRules | some dci:DCI | i in dci.DCIrule
      (...)
}
(...)
```

Instance generation

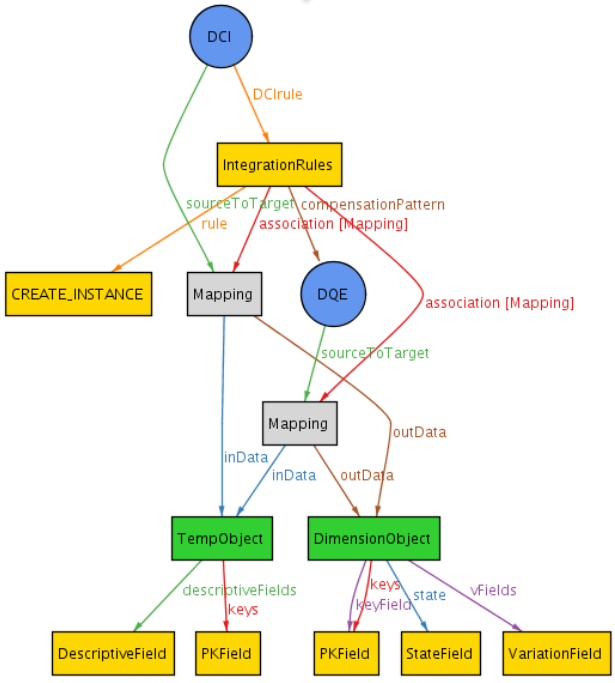


Figure 25. Describing a DCI pattern and an instance excerpt generation using Alloy.

Considering the "Sale" fact table example scenario described before for the DW schema in the section 3.3.4, and used to the SKP instance configuration, we can identify two aggregated measures, namely: 'quantity' that corresponds to the number of products sold for each sale, and its correspondent amount ('total') paid. In Figure 24 we can see a basic "Aggregator" Pattern instance that was defined for supporting the data aggregation calculations used to feed the "Sale" fact table. This process identifies the source records belonging to each group ("Read group records" sub process) based on the configuration of the group creation rule. Next, using a BPMN exclusive gateway determines if there is the need to load records ("Load DW records" sub process). In both process paths, the records are used for group calculation/recalculation ("Calculate aggregation" transactional sub process).

In case of a record causes the aggregate calculation to fail (having wrong data types for example), an error event is triggered and the process is handling by "Error handling" compensation sub process that stores the group of inconsistent records using a quarantine object, updates the error log ("Error log" sub process) and returns the control flow to the "Core" lane.

3.3.7 Data Conciliation and Integration

During the implementation of an ETL system, it is very common to handle data coming from a considerable number of operational sources that are used in distinct business branches. In cases like this, it is necessary to integrate some additional related data (data from the same entity) coming from other different data sources. Providing a Data Conciliation and Integration (DCI) pattern will allow for encapsulating all the logic about data conciliation. In general terms, integrating data coming from a set of heterogeneous sources can be categorized based on two distinct aspects: schema and semantics heterogeneity. Previously, we presented the definition for a DQE pattern, however, this pattern also covers already some aspects of data conciliation. However, the effective integration should be accomplished using some binding logic procedures, which should be implemented to specify how related fields could be linked together. A DCI pattern can be described as follows:

Name: Data Conciliation and Integration (DCI).

Classification: Transformation pattern.

Problem: Since ETL implementation involves the integration of data about the same business subject using several data sources to a specific dimension table, several conflicts can occur. These conflicts should be properly handled, preserving data consistency and ensuring well-structured data used by data reporting tools.

Forces: The application of this pattern can be compromised by the following aspects:

- **Invalid data**: At this ETL stage, data may contain some errors that, in normal circumstances, should have been solved before. A DCI pattern should be able to move that

data to specific quarantine objects or lead them with the instantiation of some related patterns for solving errors at execution time.

- **Conflicting records**: Records that come from different data sources can have contradictory semantics related to the same data object.
- **Historical data**: Handling historical data for supporting SCD dimensions can involve some additional efforts because some records may have historical versions that should be handled properly.

To support the identification of common records between each data source, we used specific data lookup processes to find related records. To do this, some binding rules must be used to identify mappings between all the involved data objects. In Figure 25 it is presented an Alloy specification about the main concepts and relationships for supporting a DCI meta model along with an example of a random instance excerpt. In the figure, the "DCI" signature represents the DCI pattern structure to support the correspondences between each data source object involved with and the correspondent target data objects. Thus, several "sourceToTarget" mappings can be provided. However, their output data objects should be the same, which is forced by the "DCISameOutput" Alloy fact.

The integration rules ("IntegrationRules" signature) should be also provided to DCI pattern for indicating the action that should be applied to each field association, including the mapping fields ("MappingFields" field), the rule ("Rule" enumeration), and the conditional operator applied ("ConditionalOperator" enumeration). Thus, in the case of using more than one source, the conflicts that can emerge can be avoided using specific rules for deciding which fields from a specific source data object are used for populating the target dimension. The "consistentDCI" fact enforces several constraints for guaranteeing the DCI pattern consistency. For example, the fields associations used for rules configuration should be the same employed in "sourceToTarget" mappings.

When DCI pattern is used, it is expected that the major data inconsistencies and errors were corrected in previous phases. However, if some problems are found in data integration phases, specific instances of other patterns can be generated and used to solve particular problems ("compensation" field). The DCI pattern instance presented in Figure 25 shows two "sourceToTarget" mappings representing the mapping rules from the "TempObject" and the "DimensionObject" (data from the same object is used to feed the dimension), the "IntegrationRules" describing the rules applied when conflicts between records is found (for example, different addresses for the same customer). The "CREATE_INSTANCE" rule is configured and associated (using the "association [Mapping]" relation) to the mapping in which the conflicts can occur. Additionally, a DQE instance is presented (using the "compensationPattern" relation) that is used to handle and correct data conflicts. To simplify DCI pattern instance representation, the relations between fields were omitted.

Following the DW "Sale" schema described before in section 3.3.4, let's consider that several data sources are used to populate the "Product" dimension without the guaranty of well-conformed data. A possible DCI pattern instance to deal with this the scenario is presented in Figure 26 using BPMN. The "Core" lane represents two cyclic structures for identifying each input data source object used ("Load source" sub process) and the records they provide ("Load record" sub process).

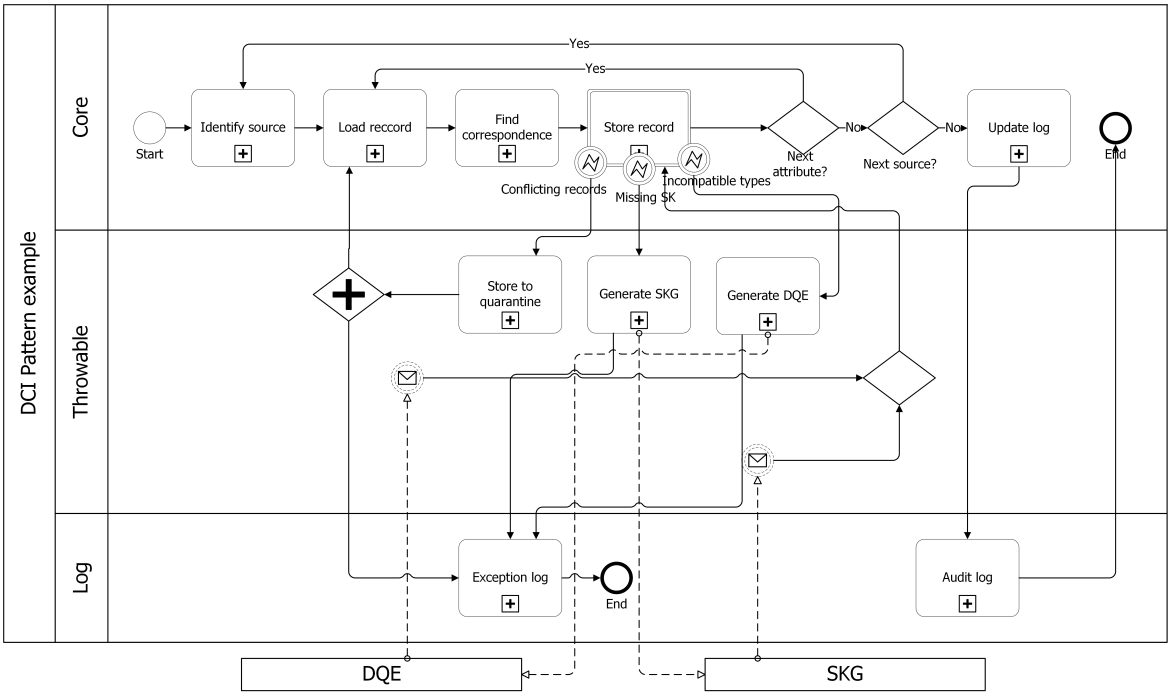


Figure 26. An example of a DCI instantiation process using BPMN.

For each record, the mappings for preserving the binding rules must be used for supporting the integration logic ("Find correspondence" sub process). When records are stored ("Store record" sub process), specific events can be triggered to identify three possible exception handled using the correspondent compensation sub process:

1) "Missing SK", referring to new records without the correspondent SK generated ("productId" field) and handled by the "Generate SKG" compensation sub process. A SKG pattern instance is generated and represented in the BPMN diagram using a collapsed BPMN pool that is responsible to generate the new SK for the record in which is missing one. The DCI pattern pool communicates with the external pool using message events that allow the communication between the participants (pools). The "SKG" intermediate catch event represents the SKG instance conclusion and return the control flow to the "Core" component to store the corrected record;

2) "Incompatible types", for data type incompatibilities verified between source and target repository on the "productId" field (e.g. numeric values with characters). This exception scenario is handled by the "Generate DQE" compensation sub process. This event handling is similar to the "Missing SK" event, using an external participant to correct the "productId" field

through the conversion of the "productId" value to the integer data type. When corrected, the record is stored using the "Store record" sub process;

3) "Contradictory records", describing contradictory data for the "productName" field. The "Store to quarantine" compensation sub process is responsible for handling this exception scenario. The contradictory record is stored in the quarantine table and then a parallel gateway is used to launch two parallel flows: one for storing the event data in the audit object and the other one to return to the "Core" pool for reading the next record.

The DCI instance process ends storing the process milestone data using the "Update log" sub process and the "Audit log" sub process presented in the "Log" pool.

3.3.8 The Intensive Data Loading Pattern

The extraction and transformation patterns described until now must be (ideally) implemented using a DSA without affecting any of the operational systems involved in the populating process. The Intensive Data Load (IDL) is a typical loading pattern that stores in the DSA all the data that go later to the target DW. "Load" patterns are included in one of the last stages of ETL system implementation. Thus, when data records arrive at this stage, they are ready for loading into the DW. This means that all other necessary patterns were previously applied, which guarantees data consistency and uniformity.

After the ETL data transformation process was accomplished, data stored in the DSA is ready to be loaded into DW's data structures. To do this, we need to manage the re-creation of all the changes into the DW. Considering the dimensional approach for modelling DW, we should preserve data referential integrity: a dimension should be populated first and only then the fact tables. Next, we present a more detailed description of the IDL pattern:

Name: Intensive Data Loading (IDL).

Classification: Load pattern.

Problem: Independently of the loading process - initial or incremental -, the loading processes should refresh DW schemes taking into consideration all the schema rules as fast as possible with the minimum computational resources.

Forces: The following aspects can compromise or difficult the application of pattern:

- **Undetected errors**: even using specific conforming and normalization procedures, data can have inconsistencies in relation to the data already stored in the DW. SK conflicts or referential integrity problems are some examples of the problems that can occur.

- **Historical dimensions**: Handling historical data involves additional logic. When data is loaded into the DW, dimension tables should be populated before its associated historical structures (if any).

```
(...)
sig IDL extends Load{}
fact consistentIDL{
   all d:TempObject | one m:Pattern.CoreComponent.sourceToTarget | d in
m.inData or d in m.outData
   (...)
   all idl: IDL {
      idl.sourceToTarget.inData in TempObject or idl.sourceToTarget.inData in
DimensionObject
      idl..sourceToTarget.outData in DimensionObject or
idl.sourceToTarget.outData in FactObject or idl.sourceToTarget.outData in
HistoryDimensionObject
   (...)
   }
}
(...)
```

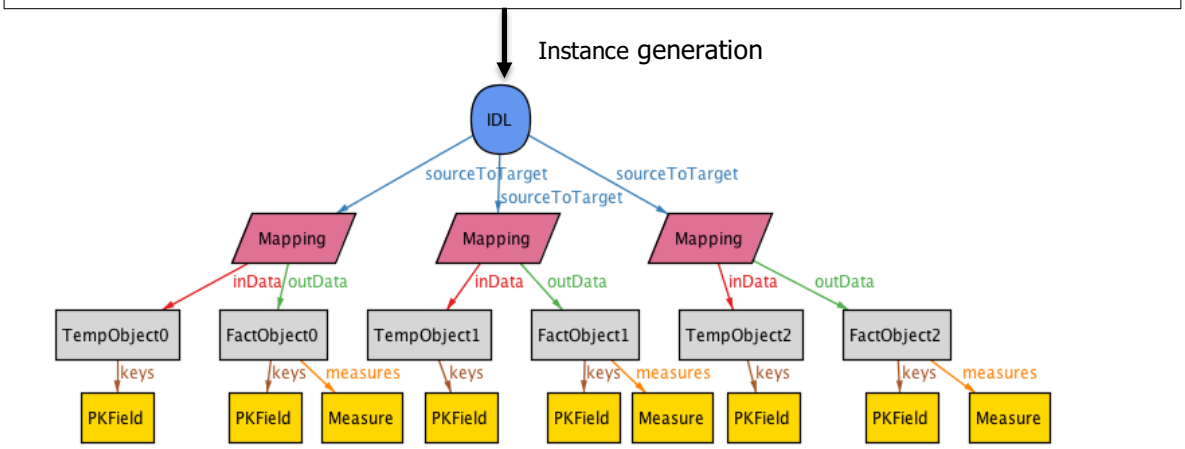


Figure 27. Alloy specification describing IDL pattern and an instance excerpt generation

If we consider an incremental approach, data records should be transferred to the DW dimensions accordingly the operations that were performed at the moment they were extracted from information sources. For dimensions with history maintenance, current data should be first loaded into the dimension table and only then insert historic records in the correspondent support structures. This is necessary for ensuring all the applied referential integrity constraints. For this, the DSA log should be accessed to find all operations made for each dimension.

Analyzing the operations performed in the DSA by previous ETL patterns, the IDL pattern creates the most appropriated loading sequence to ensure the process consistency. In Figure 27 we can see an excerpt of the Alloy specification for supporting the basic operational requirements of an IDL instantiation. To make it, we need to provide several "sourceToTarget" mappings for describing all the correspondences between DSA temporary structures for dimension, historic dimensions, and fact objects. The Alloy facts (e.g. the "consistentIDL") enforce these constraints and ensure process consistency. Moreover, a random instance generated by the Alloy Analyzer is also presented to check

instance consistency based on the provided specification. The pattern instance presented in Figure 27 shows three "sourceToTarget" mappings, used to provide the necessary relationships between three "TempObjects" stored in the DSA and the three "FactObjects" used for data load. For pattern instance visual representation, the fields representation and its relations were omitted from the Figure 27.

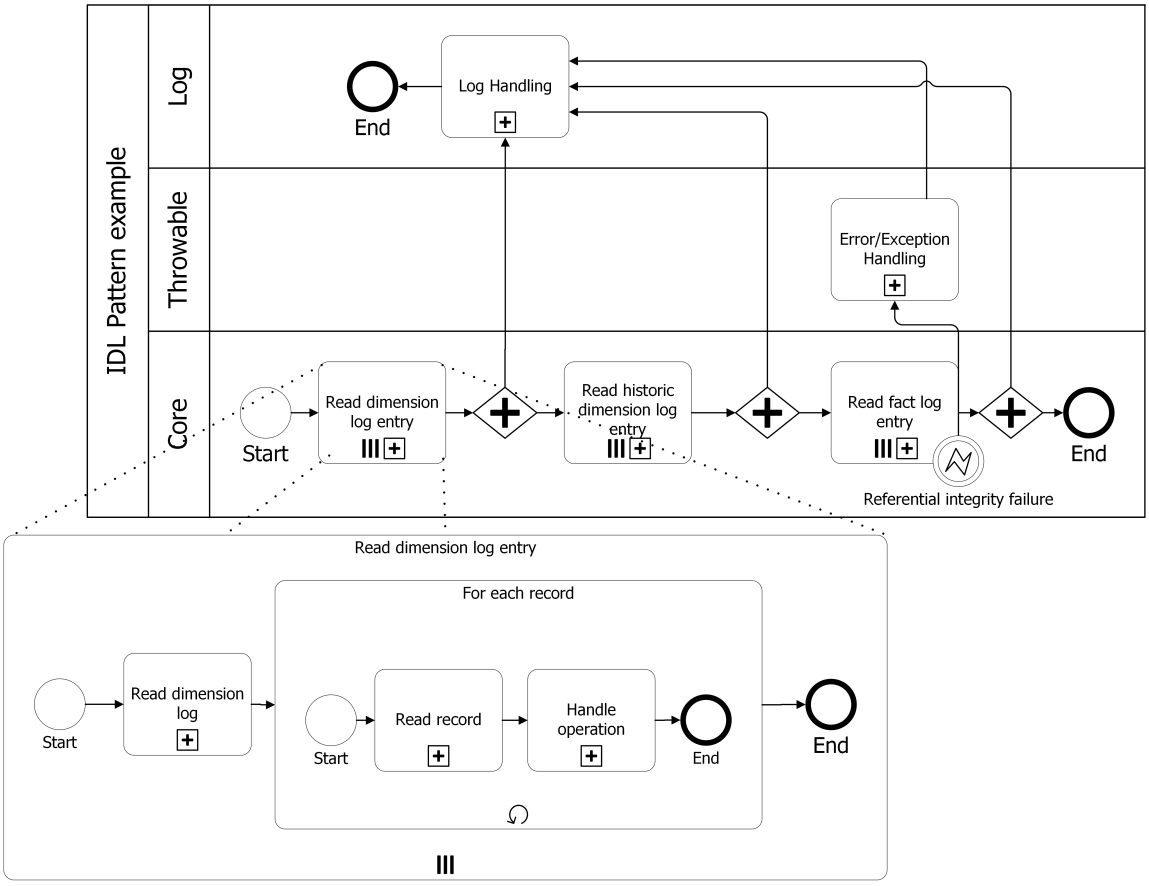


Figure 28. BPMN example algorithm for the IDL instantiation

Next, in Figure 28, we present a BPMN schema process having the main tasks for supporting DW data loading operations, involving two regular dimensions ("Product" and "Employee"), one historical table of a dimension ("ProductHistory" dimension) and a "Sale" fact table. Observing the BPMN diagram, we can see that the loading process starts with three consecutive sub processes that are used for populating each target data object type, starting by the dimension tables ("Read dimension log entry" sub process), next the historical table ("Read history dimension log entry" sub process), and finally the fact table ("Read fact log entry" sub process). For performance reasons, each sub process can decompose data in parts to run in parallel (if execution environment allows for it), and for that reason, they are represented with a BPMN parallel multi-instance configuration. For the dimension tables, the amount of data to be delivered hardly represents a bottleneck, which is typical. However, for fact records, we can face some performance problems due to the considerable amount of data that need to be delivered into the DW. In such cases, the existence of multiple process instances processing smaller group of records can reduce the time needed for fact table populating.

The "Read dimension log entry" sub process is presented with more detail, representing the dimension log processing ("Read dimension log" sub process) and their correspondent records ("Read record" expanded sub process) represented using an expanded sub process with a loop marker, indicating the iterative record handling. On the delivering dimension records from DSA to the target DW dimension table, read records ("Read record" sub process) are separated in new records and the ones that need to be updated (the separation logic is encapsulated in the "Handle operation" sub process). After dimensions populating processes, the historical tables are updated using a similar process ("Read historic dimension log" sub process). The last step is the fact table populating process ("Read fact log" sub process). Sometimes, when records are stored into the fact table, schema violations provoked by referential integrity failures related to the problems in the dimension populating process can occur. In such cases, an error event is handled by the "Exception handling" sub process that stores the inconsistent records using quarantine tables, updates the error log structure ("Log handling" sub process) and ends process execution. Between each sub process execution, BPMN parallel gateways were configured to initiate a parallel path to update process milestone into the target log file.

3.4 The Throwable Pattern

Usually, an ETL implementation and management deal with several problems that can compromise all its process activities. Even using data profiling techniques for detecting problems on data extracted from operational systems, inconsistencies provoked by bad data or by changes made to cover new business needs can occur frequently. Using specific "Throwable" instances to support each pattern configuration allows for the definition of specific strategies for planning and dealing with these recurring problems. These procedures can anticipate such kind of errors and provide suitable recovering strategies.

Any ETL process must be able to detect data inconsistent states and avoid their impact in the regular DWS functioning. We know that these inconsistencies can compromise all the ETL activities. However, stopping all processes every time an unexpected situation occurs in a specific task is quite impractical. For that reason, we need to have a more flexible approach to prevent potential process failures, and provide a way for separating inconsistent records from the regular ETL flow or, if possible, correcting them at execution time. Next, we present a more detailed description of a "Throwable" pattern:

Name: Throwable.

Problem: During an ETL normal production and considering the operational requirements of each ETL pattern, several data errors can occur compromising the operationality of the entire system. Using data profiling techniques is a good practice to know the data drawbacks before ETL starts. However, some of these errors are unpredictable or cannot be detected before process execution.

Forces: The following aspects can compromise or difficult the application of the pattern:

- **Unexpected errors**: strategies can be represented as pre-configured processes to deal with well-known errors, providing alternative ways for avoiding ETL failures and providing pre-defined solutions for correcting them on-the-fly. Eventual unexpected application scenarios should be handled in an appropriate manner without affecting the remaining activities. Additionally, records can have some dependency degree between them, which means that the process of moving records to quarantine objects should consider the affected records within a logical group.
- **Expert decisions**: some errors may have the need to be handled manually by the ETL team. Thus, these errors need to be stored using adequate quarantine structures in order to be posteriorly analysed. If corrected, these records can be reintroduced in the ETL normal flow or eventually deleted from the quarantine objects.

ETL processes deal with very particular data coming from heterogeneous systems that have very different operational and storing requirements. So, the occurrence of unexpected data errors is quite often as a result of wrong data inputs, changes in functional requirements, or even due to the implementation of wrong validation mechanisms. In Figure 29 we can see the main concepts related to a "Throwable" component using Alloy primitives, along with a random instance generated based on a specification excerpt. The "Throwable" signature represents the basis for the "Throwable" pattern using a "sourceToThrowable" mapping, describing the mapping between the source data object and the target "throwable" object. The "Throwable" signature can also be configured with a log component ("log" field), which will allow for the communication between the components. This way, when "Throwable" events are triggered, the exception and error logs can be registered properly.

The "Exception" signature represent all the data errors that do not compromise the execution of the ETL system, and, for that reason, it can be handled using specific routines or avoided, transferring incoherent records to quarantine objects ("Quarantine" signature). The quarantine objects are defined individually for each pattern, being responsible for storing descriptive metadata about the errors found along with the records that provoked them. There are two related objects that support the quarantine objects schema:

- The quarantine metadata object ("QuarantineMetaData" signature) that stores metadata about each exception.
- The quarantine data object ("QuarantineData" signature) that keeps the inconsistent records.
- The quarantine metadata structure is generally composed by a "Key" attribute, identifying each quarantine row, and a set of control attributes ("ControlField" signature), which are used to tag each row with additional metadata provided by the exception mechanisms. The temporal data ("Temporal"), the source object ("ObjectDescription") where the exceptions

are found (object and fields), the triggered exception ("Throwable"), the action that triggered the exception ("Stage"), the evaluation state ("State", by default 'invalid'), and the exception severity ("SeverityScore"), are some examples of possible control fields used to store exceptions metadata.

```
(...)
enum SeverityCode{LOW, MEDIUM, HIGH}
sig ThrowableField, StageField extends Field{}
sig ThrowableMetaData {
   temporalData:some DateField,
   objectDescription:one Field,
   event:one ThrowableField,
   stage:one StageField,
   severityCode:one SeverityCode
}
sig QuarantineData extends DataObject{metaData:one ThrowableMetaData}
abstract sig Throwable extends Pattern{
   sourceToThrowable: some Mapping,
   log:lone LogPattern
}
sig Exception extends Throwable{
   quarantineObject:one QuarantineData,
   correctionProcedure:one DQE
}
sig Error extends Throwable{metadata:one ThrowableMetaData}

(...)
```

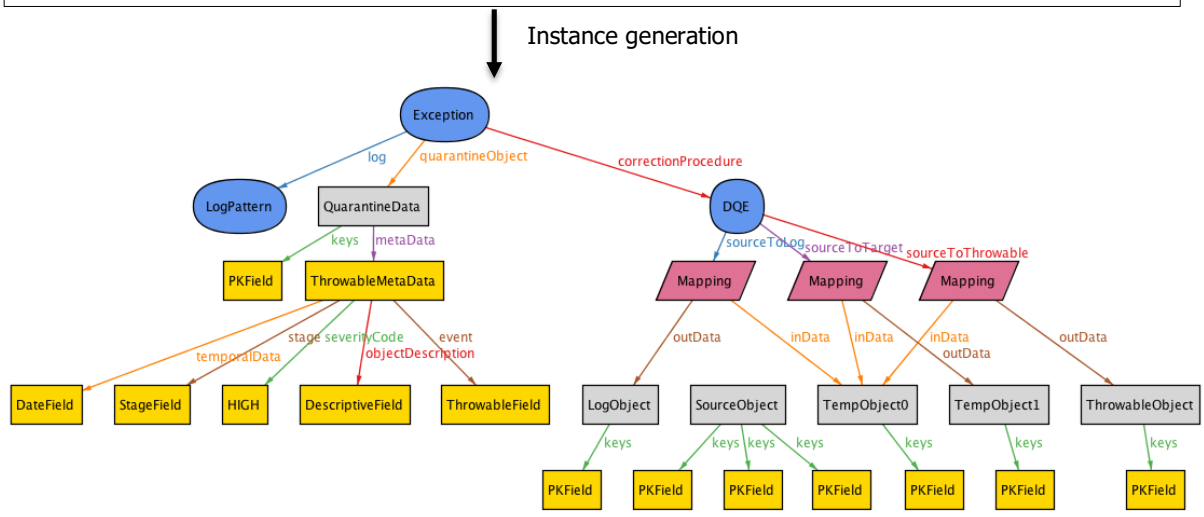


Figure 29. Alloy specification describing "Throwable" pattern and an instance excerpt generation.

We can also use some descriptive attributes to represent non-structured data and to enrich quarantine rows with information that can be used for human interpretation. The record identification ("RecordId") is used to identify source records stored in the quarantine object. After being treated by ETL administrators, the records stored in quarantine objects can be corrected using a DQE instance (pattern representing common clean/conform procedures). This pattern instance provides a way to correct records and reintroduce them in the normal flow of the ETL system.

The "Error" signature reveals serious data errors that cannot be handled automatically, and so compromising all ETL flow. When errors are detected, it is necessary to have rollback mechanisms configured appropriately for reversing all operations done so far, and return data to a previous consistent state. Otherwise, the process can be cancelled, leaving data at the current state. The "Throwable" instance presented in Figure 29 presents an "Exception" Throwable instance configured with two patterns instances: The "DQE" instance configured with three mappings to represent a correction procedure applied to the source records, and Log ("LogPattern") instance that represents the logic behind log activities. Additionally, a "QuarantineObject" is also presented, describing the "ThrowableMetaData" used to describe the storage metadata to store the unresolved records into the quarantine table.

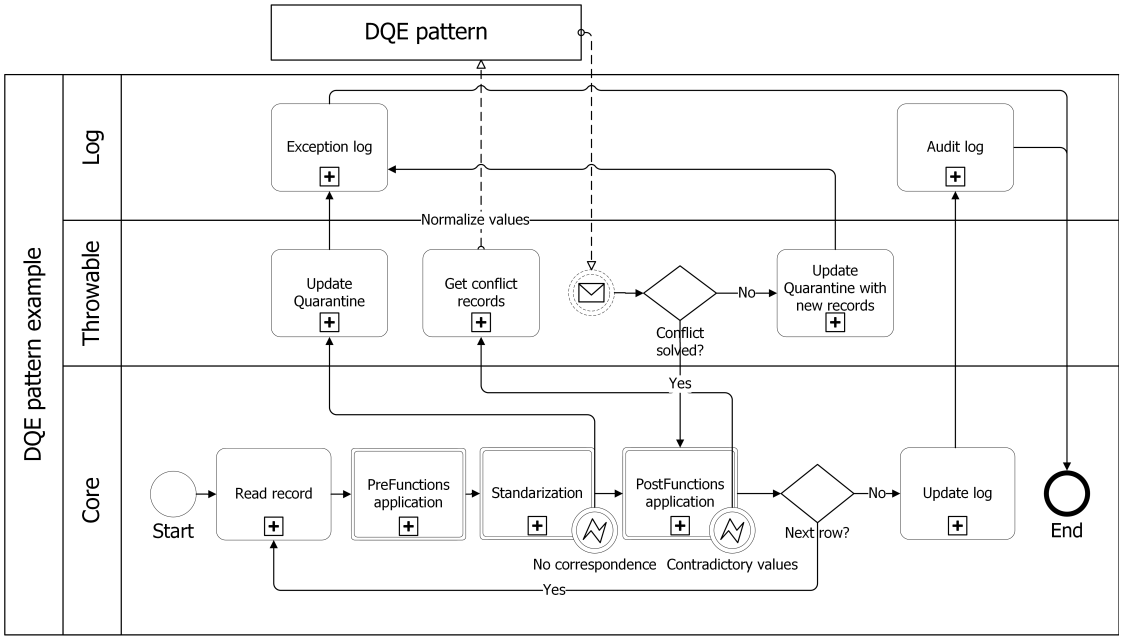


Figure 30. BPMN example algorithm for the Exception handle modules using DQE pattern.

The BPMN process presented in Figure 30 presents with more detail the "Throwable" lane configuration for the DQE pattern instantiation that was presented before in Figure 18b. In the figure, we can distinguish three different lanes, namely:

- The "Core" lane showing the DQE configuration, with two exception scenarios represented using error BPMN events: "Unexpected plan" and "Records with conflict".
- The "Throwable" lane representing with more detail the configuration related to the two exception scenarios presented.
- The "Log" lane describing the logic behind the "Exception" and "Audit" log updates carried by the sub processes that are responsible for handling "Throwable" scenarios.

As already described, the DQE instance presented in Figure 18b presents a normalization procedure used for solving the different "terminology" associated with the plan names from each record from

the data sources involved with. For each processed record, a lookup process is executed over the mapping table in order to find the correspondent plan name. When some plan rule for a given name is not stored in the dictionary table, an "No correspondence" error event is triggered. For storing this event, a special-purpose process ("Update Quarantine" sub process) is used to store inconsistent records and to update the exception log ("Exception log" sub process).

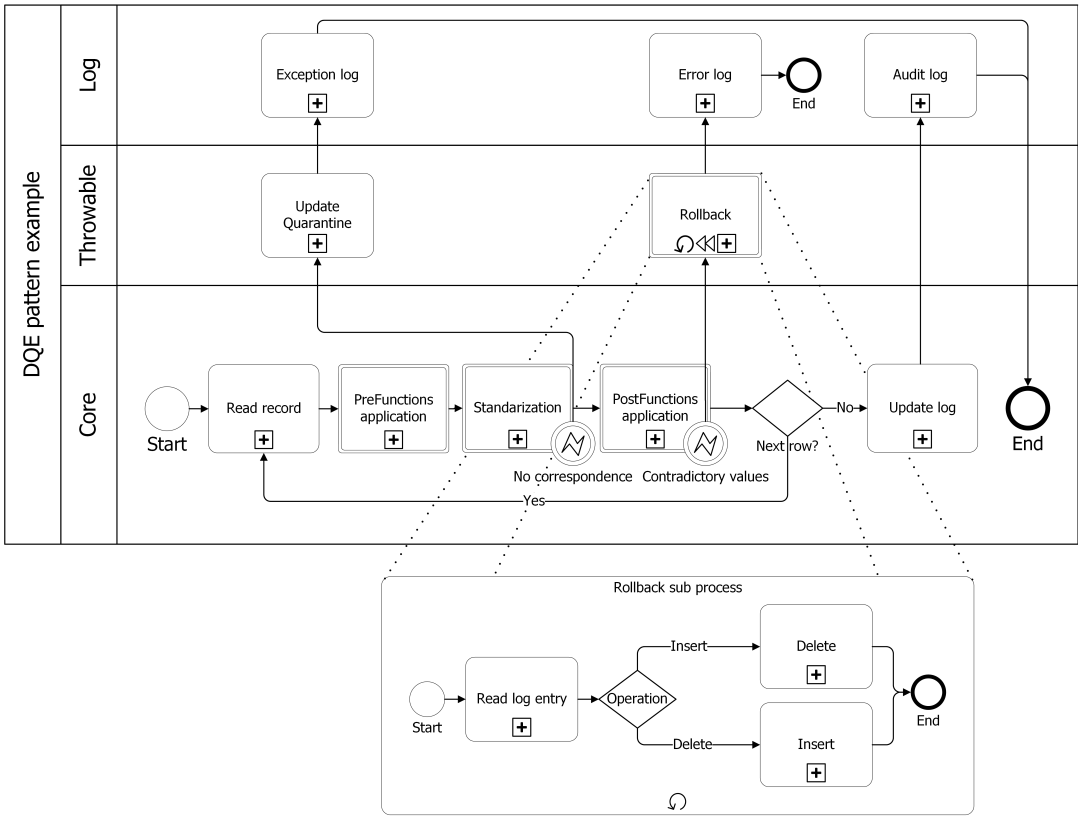


Figure 31. The Figure 30 BPMN process variation using a BPMN error event with a rollback configuration

If a situation in like two or more products share the same "VAT" number but with different plans, a "Contradictory values" event is triggered, using the "Get conflict records" BPMN sub process for handling the conflicting records. To work around this problem, it is generated a DQE pattern instance in order to apply additional transformation rules for replacing the plan names, based on a set of default values that solve conflicts between records. Like other cases, when a conflict is successfully managed, the records are reintroduced in the regular ETL flow (using a BPMN Message event). After this step, a duplication elimination procedure (from "PostFunctions application" sub process) is applied to eliminate records referring the same information. However, if the conflict persists, the conflicting records are stored in a quarantine table, triggering a new log entry to record this event ("Update Quarantine with new records" sub process).

When we use error events, we can configure rollback operations in order to restore data to a previous state by cancelling a specific transaction. This can be represented using a BPMN compensation activity. Thus, when an error occurs, the process stops and accordingly the rollback mechanisms

configured (if any), the process can return data to a consistent state. A different version for the "DQE pattern instance" pool is presented in Figure 31. In this version, it is used an error event for "Contradictory values" event with a compensation sub process representing a rollback procedure ("Rollback" sub process). The "Roll" sub process is configured as a "Transaction" sub process to denote transactional properties that are related to the rollback mechanisms used. This sub process represents a set of activities that integrate the logical unit of operations that must be performed atomically. When this activity is triggered, the rollback procedure starts and inverts all the operations performed following the records stored in the log, maintained by the "PostFunctions application" sub process. A simplified view of the tasks performed by this rollback procedure is presented using a BPMN expanded sub process. The log entries are read and the inserted records are deleted while the deleted records are reinserted. After the rollback operation, the correspondent error log object ("Error log" sub process) is updated before the process ends.

3.5 The Log Pattern

Kimball and Ross (2013) identified several ETL subsystems that can be used for managing an ETL system's environment. For example, the Workflow Monitor and the Job Scheduler as well as the backup, the recovery and the version control systems, can be used together to manage ETL processes and provide to end-users reporting applications for tracking critical aspects of ETL systems. Along with additional metadata about the operations performed, tracking ETL system activities it is an important task that can be used for identifying some data inconsistent states, errors or performance bottlenecks, and the stage they occurred, acting as support for several ETL components.

Transaction logs files are used by many software systems to collect data about the events and transactions that occurred during system daily operations. Thus, all performed actions can be stored to provide a system overview in a specific time-window. Logging is a very important component of every ETL system and can be used for maintaining consistency and reliability. For this, we need to monitor some critical tasks for ensuring quality and performance measures, identifying bottlenecks and errors that can influence the ETL process operationally. For example, we can use log mechanisms to control the performance and the ETL tasks quality measures that have the ability to check its consistency based on the execution history. Strong variations in the results can indicate the occurrence of anomalies in the source systems, which should be investigated. The log structure can also be used to create recovery mechanisms for protecting ETL procedures against critical failures, providing a way to rollback ETL data to a consistent state, such as the example presented in Figure 29. Next, we present a more detailed view of the ETL Log pattern:

Name: Log Monitoring (LM).

Problem: During the development and implementation of a regular ETL process, several problems can occur. Some of them are very hard to find, monitor and solve. A conventional ETL

process involves dozens of tasks using substantial amounts of data. All this, influence the quality of data, and the way how faster data can be provided to end users.

Forces: The following aspects can compromise or difficult the application of the pattern:

- **Identify the event nature and stage**: ETL tasks can produce several levels of events that should be identified properly. Information or Debug levels are typically used to get information about the start and end events for identifying suspicious records or for finding unexpected behaviours. Since these events can be triggered by many tasks, it is crucial to know where and when they happened.
- **Log detail level**: Logging everything can be troublesome since every log entry needs to be written to disk. This implies some additional performance costs to the ETL system. Also, the generation of these logs implies other computational costs and, in the short term, the amount of data generated could be so big that could be very hard to manage it, which makes log data quite useless.

```
(...)
abstract sig LogField extends Field{}
abstract sig EventLogField extends LogField{}
sig CompensationField, ErrorLogField, CheckPointField extends
EventLogField{}
sig PerformanceField extends LogField{}
enum LogLevel{EXCEPTION, FATAL, INFO(...)}
sig LogObject extends DataObject{
   temporalData:some DateField & fields,
   taskDescription:set PerformanceField & fields,
   logDescription:one EventLogField & fields,
   stage:lone CheckPointField & fields,
   level:lone LogLevel,
   sequence:lone Sequence & fields
}
sig Log extends Pattern{
   sourceTolog: some Mapping,
   object: one LogObject
}
(...)
```

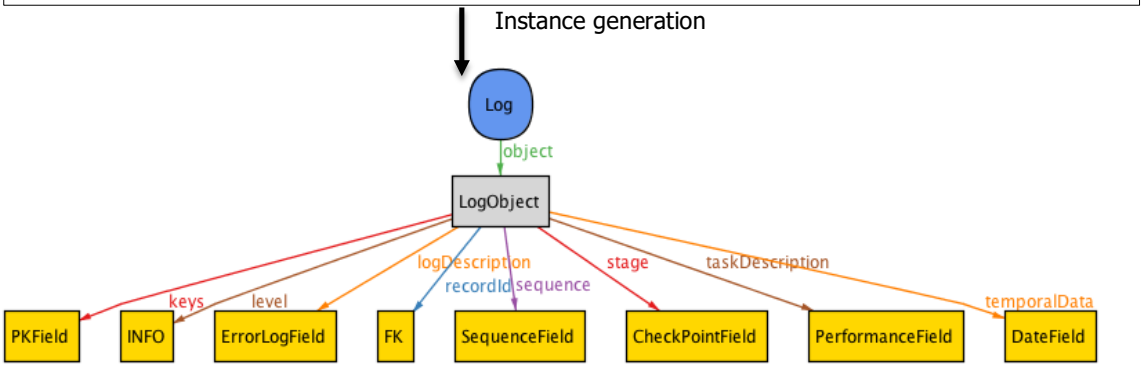


Figure 32. Alloy specification describing Log pattern.

Log file or relational log tables are just some examples of different approaches to capture information about ETL activities. The log entries can be supported by specific triggers applied to the record's

attributes or using more general conditions at process level to identify ETL milestones. We can also use external tools and built-in components to implement logging systems.

In Figure 32 we can find some of the most important concepts related to the "Log" pattern, using Alloy. A simpler instance showing a possible log object configuration is also presented in the figure. The Log pattern ("Log" signature) is composed by the "sourceToLog" mapping that relates to the field association between the source object and the target log Object ("LogObject"). Log objects use specific fields for checking quality parameters, such as the temporal ("temporalData") and performance data ("taskDescription" using specific "performanceFields") about process tasks. The number of records processed and the time needed to perform specific tasks are just two simple examples of possible performance fields that can be used.

The log level ("LogLevel" enumeration) describes the type of log used. For example, the information log can represent its entries as information logs ('INFO'), which are useful for describing informational messages that highlight progress tasks, or fatal logs ('FATAL'), which describe critical error events that make the process to abort. The log description ("logDescription") field characterizes the event that triggered the log entry, which can be, for example, an exception, error or a state event. Finally, the key attribute referencing source object in log entries can be supported using the "recordId" and the "sequence" fields. The random instance generated by the Alloy Analyser presents a simple representation of a "LogObject" configured with several fields (key, information and control fields) describing the log structure.

Table 2. Log table sample data for the DQE pattern.

Log Key	Stage	Date	Log description	Description	Sequence
671	Standardization	2016-08-17;11:24:40	Exception	Unexpected pan: *PlanXFR*	517
…	…	…	…	…	…
870	Update log	2016-08-17;11:15:20	Auditing	Success	623
…	…	…	…	…	…

Let us now consider the scenario exposed in Figure 30, which represents the "Log" component configuration ("Log" lane) for the DQE instance. The logic behind the "Exception log" and the "Audit log" sub processes is straightforward, consisting of the identification of several control attributes that should be used to store relevant information about process execution.

Thus, we can configure the metadata related to the stored entries, providing complementary data about the actions performed in the remaining lanes. In the Table 2, it is presented a simple example of two possible log entries generated by the DQE pattern already presented using the same log structure both to support the "Exception log" and "Audit log" sub processes, representing a specific set of attributes for satisfying the requirements of the "Exception log" and the "Audit log" sub processes, and supporting the following logging tasks:

- The "Log Key" column ("Key" signature), representing the unique id for each entry.
- The "Stage" column, identifying the task that triggered the event ("Stage" signature).
- The "Date" column, storing the event temporal data ("Temporal" signature).
- The "Log description" column ("LogDescription" signature), identifying the event nature ("Exception" or "Audit" signatures).
- The "Description" column ("ObjectDescription" signature), representing the pre-configured messages to describe each log entry.
- The "Sequence" column ("Sequence" signature), representing the position for each entry within the process context that can be used for ETL process debugging.

```
(...)
pred addToDimension [s,s': State, r,r': Row, scd: SCD] {
    r in scd.auditToDimension.inData.rows.s
    r' not in scd.auditToDimension.outData.rows.s
    scd.auditToDimension.inData.rows.s' =
        scd.auditToDimension.inData.rows.s - r
    scd.auditToDimension.outData.rows.s' =
        scd.auditToDimension.outData.rows.s + r'
    all f: scd.auditToDimension.association.Field |
        f.(r.values) = f.(scd.auditToDimension.association).(r'.values)
    (...)
}
assert addToDimensionCorrect {
    all s: State, s': s.next, scd: SCD, r: Row |
        (consistentSCD[s,scd] and addToDimension[s,s',r,scd]) =>
            consistentSCD[s',scd]
}
check addToDimensionCorrect for 10
pred updateDimension [s,s': State, r : Row, scd : SCD] {
   r in scd.sourceToTarge.outData.rows.s
   some rd: scd.coreComponent.sourcToTarget.inData.(rows.s), rd',rh' : Row
{
      all f: Field.(scd.coreComponentsourceToTarget.association) &
scd.souceToTarget.outData.keys | (...)
(...)
assert updateDimensionCorrect {
   all s:State, s':s.next,s'':s'.next, scd:SCD,r, r':Row |
       (consistentSCD[s',scd] &&
       addAudit[s,s',r,scd] &&
       updateDimension[s',s'',r',scd]) => consistentSCD[s'',scd]
}
check updateDimensionCorrect for 5 but exactly 1 SCD, exactly 3 State, 2
Mapping, 3 DataObject
```

Figure 33. Excerpt of the SCD pattern specification for dynamic behaviour simulation.

3.6 Patterns Behaviour Validation

Alloy is a very expressive language, very suitable for representing static properties but also expressive enough to represent the dynamic properties for a given system. These properties represent systems

executions and the related state changes. The patterns specification already described, embody the ETL patterns static definition. However, the process correctness based on the changes applied to the collected data as well the description of ETL procedures state before and after its execution was not presented.

Using Alloy, pattern dynamic behaviour can be encoded through well-established idioms, such as the local state idiom (Jackson, 2012). The "State" signature is introduced and used for representing different system states, modelling the notion of evolving in time. All the artefacts that are expected to evolve in time are then enriched with the "State" element, representing values in each instant of time. The data objects structures (schemas) and patterns structure are represented in a static way while the data contained in each data object are expected to change. Basically, the data that can be stored using the data structures presented are now represented as part of the Alloy model specification. That way, we can validate the behaviour of each pattern when records are manipulated by the pattern application context.

The system evolution is modelled using declarative predicates that establish two different states. In this perspective, while the static specification components represent pattern configuration, the dynamic components define the pattern behavior based on those configurations. Figure 33 shows an excerpt of the "addToDimension" predicate, which represents the "insert" operation modelling involving two data objects: the audit ("AuditObject" object) in which data is extracted and the target dimension ("Dimension" object) in which data from the "AuditObject" are moved. Additionally, the "updateDimension" predicate excerpt is also represented, showing an example of how the dimension updates should be performed, encoding the logic behind removing records from audit object and insert them to the target dimension object.

Being declarative, these definitions are usually comprised by pre-, post- and frame conditions:

- The pre-conditions restrict the operation to be applied only if the audit row surrogate keys to be processed do not exist in the dimension table. Otherwise, the update operation should be applied instead.
- The post-conditions states, preserving the data as defined by the associations of the SCD mappings (e.g. when a row is removed from an audit table and inserted in a target dimension).
- The frame conditions state that every other artifact, like the history table, stays unchanged.

The two states are represented by the "**s**" and "**s'**" predicate parameters. Whenever the rows field is annexed with the "s" elements it represents the data in a pre-state condition, while rows with "s'" represents the data in a post-state condition. Once these operation behavior description predicates are defined, the specification can be automatically analyzed by the Alloy Analyzer for checking whether they preserve the consistency constraints defined above within a bounded universe. These

are defined by the assert clause, testing whether the execution of the "addToDimension" operation preserves the system consistency. The "check" command is used for instructing the Alloy Analyzer to verify this assertion for a given scope. Other additional constraints, as well as more detail levels, can be added to the specification to verify more complex properties.

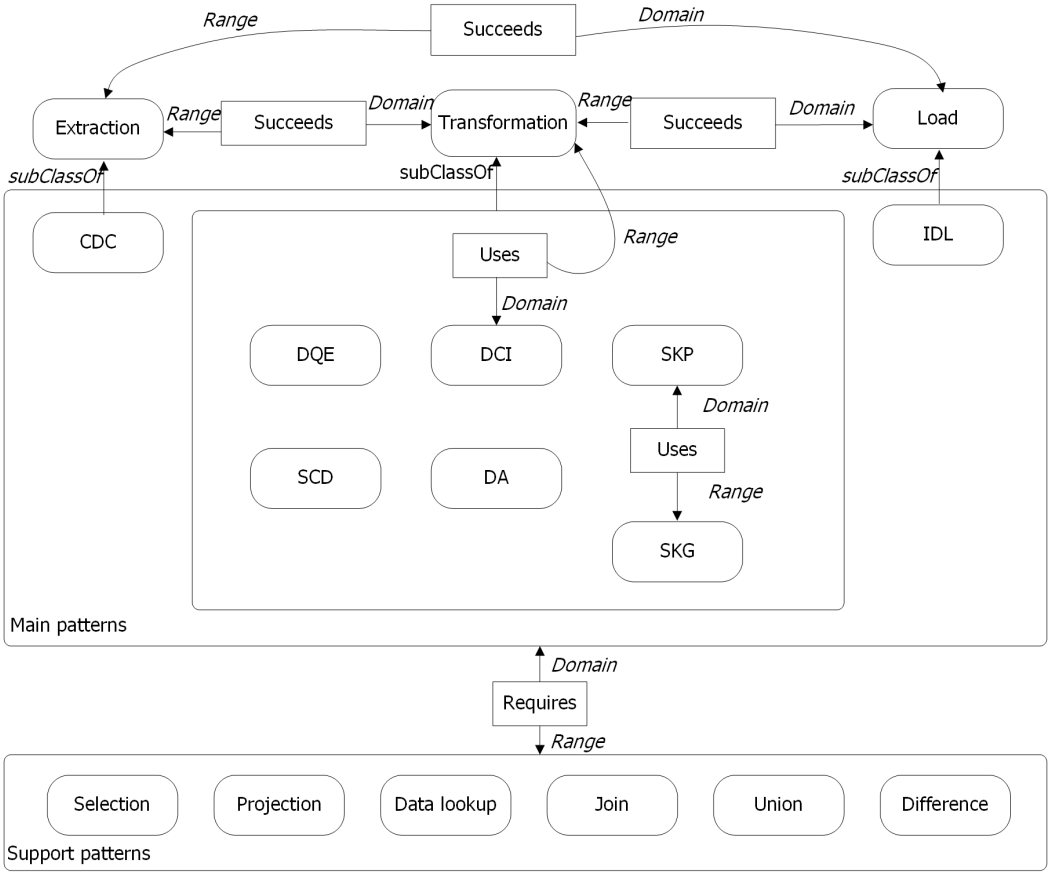


Figure 34. Pattern language example for ETL patterns representation.

Due to its declarative nature, rather than explain how states are obtained based on pattern's execution, the Alloy dynamic behaviour modelling gives the constraints that define a consistent state obtained by the relation of the old and new state. When the Alloy Analyser reveals the predicates consistency, an increased level of confidence in the procedure correctness and model consistency before proceeding to its logical representation, is provided. This way, patterns not only become easier to use than in a common granular approach but also be reused to produce better software. This can be possible because models can be checked using a powerful simulation engine before its execution.

Besides this, the Alloy model is a very good formal basis for supporting the generation of physical models without inconsistencies, which is advantageous for the ETL systems implementation. Using static definitions, we prove model consistency by the absence of the existence of counterexamples for the specified model, however this does not imply that they cannot exists (Jackson, 2012). The dynamic behaviours specification for the ETL patterns model try to fill that gap, covering the most relevant behaviours for each pattern, being helpful to understand and validate if models remain in

consistent states even when its states are affected by pattern's behaviours. It is also important to note that Alloy was used as tool for providing a certain level of confidence and validation to the proposed model that was initially described using ontology specifications. The dynamic behaviour was only briefly introduced based on ad-hoc characterizations of state changes, which is even today a research subject without there being a standardized approach to support its construction (Frias et al., 2005; Regis et al., 2017).

3.7 The Pattern Language

The use of patterns to build a complete ETL system is a way to separate theoretical knowledge from its practical design. They help to create a general picture of the future systems, revealing the most critical components and reduce the complexity involved with their subsequent implementation. However, when modelling patterns behavior, we are representing the complexity of individual components independently of how they are framed in the ETL system. To represent and express the ETL system, the relationships between patterns should be known, since on their own they are hard to use in a structure without context, mainly when specific problems require the use of multiple patterns. These relationships can be express using a pattern language (Alexander et al., 1977) that describes how each pattern should be related for creating well-formed systems, avoiding architectural contradictions and misunderstandings.

Considering all the patterns specification described before, a pattern catalog for supporting ETL system development and implementation can be provided. In addition to each pattern structure, this catalog should also provide the description of the combinatorial rules between patterns, which is important to validate their use and allows for the description of how they can be combined for creating well-formed ETL workflows. Specific rules can be provided to describe how patterns are connected to support specific ETL scenarios, reducing ambiguity and helping to splay the range of possibilities for pattern application within a same scenario. For example, a specific order between how patterns can be used can be described using precedence constraints for describing ETL semantic within workflow context. Thus, the construction rules can be formalized to guarantee the generation of well-formed pattern instances for specific domains (Henninger and Ashokkumar, 2006).

The extraction and transformation patterns have a logical dependency between them since data should be first extracted from data sources (task assured using "Extraction" patterns) and only then can be processed by "Transformation" patterns. Dependent patterns should also be described since specific patterns can imply the simultaneous use of other patterns. The DCI pattern example (Figure 26) was configured for generating DQE (section 3.3.3) and SKG (section 3.3.5) instances to solve specific exception scenarios, being just a simple example in which a pattern instance uses other pattern instance for solving a certain problem. To support the pattern language specification, we provide a set of general rules for describing how patterns are connected (Figure 34). These rules

form coherent combinations to create more abstract components, describing a whole that holds information that cannot be derived from each part of it (Salingaros, 2000).

In Figure 34 we can see the ETL patterns representation based on the ontology primitives we presented, using the same rules that we described before for ontology elements visual representation: round corner rectangles are used for identifying classes, and rectangles are used for identifying object properties that establish restrictions between concepts. For each pattern group - "Extraction", "Transformation" and "Load" -, the "Succeeds" restriction is used to describe the degree of precedence between each group of patterns. The restriction is also enforced between extraction and load patterns because ETL sub processes can be created without the need of using transformation patterns. The "Main patterns" box represents all the patterns described so far, using the "subClassOf" axiom for representing the specialization nature of each pattern group. The DCI pattern (section 3.3.7) is a special case that communicates with instances of other transformation patterns (the Figure 26 presents a DCI instance that uses DQE and SKG instances).

The hierarchical structure we can see in the graph model represents connections between elements of each group and throughout levels, which provides a set of connections between concepts allowing for the creation of coarse level constructs. The restriction "Uses" was defined to describe a non-mandatory dependency relationship between patterns. Based on the type of pattern we use, the range of values for this property can be specialized further for having a more accurate precision. The "Uses" restriction is used by DCI instances for regulating some other configurations with the remaining patterns. Additionally, we represent a new layer of patterns (Figure 34). This layer represents a more detailed pattern level, representing more specific tasks that can be encapsulated inside each pattern. In fact, support patterns can be single tasks represented by a single instruction. Of course, its application and use depends mainly on the physical environment we are using for pattern execution.

The proposed approach is extensible and enables an upward transition to more abstract levels that can be created for covering the needs of the ETL team. Thus, we can create a new business layer specifically oriented for ETL functional areas, providing a new detail level that can be specialized using the patterns presented. The "requires" property links the support and the main patterns, which denotes stronger dependencies for each pattern involved: the support patterns are required as part of a solution for other patterns. However, the support patterns representation is only possible when the target physical infrastructure is well known, since provides a logical view of the target systems based on the pattern-oriented framework we presented. All the elements for supporting the pattern language represents only a subset of the constructors that can be used to rule the implementation of well-formed ETL workflows. Support patterns and their relationships with the main patterns should be enriched, considering the needs of each project.

The general idea of a pattern language to support the ETL pattern-oriented framework represents a complex task, since the large variety of potential problems they can solve are almost impossible to catalogue. This difficulty is intrinsically related to the ETL nature: workflows of tasks specially built to solve recurring problems based on specific data structures not only to support target decision-making systems but also when dealing with data sources that frequently reveal very different ways to represent data describing similar domains. This is, in fact, a key point that affects ETL development since forever. For that reason, and based on the presented restriction, specific ontologies can be created to describe how particular problems will be solved based on the "parts" representing the entire system, properly framed within a specific domain or project. That way, we can keep the presented patterns independent from any context, preserving the pattern framework flexibility and applicability.

Chapter 4

From Conceptual to ETL Physical Implementation

4.1 An ETL Case Study

Design methodologies are used across several software areas, covering several domain requirements with the aim to improve software quality. Complex software pieces are usually supported by demanding development processes covering all the stages of software development ranging from requirements identification to implementation and maintenance phases. The design task (whether being at conceptual or logical level) assists the development team, identifying how the system should meet customer requirements and how the system should be more effective and efficient. Based on the ETL patterns metamodel presented so far, in this chapter, we reveal and discuss a three-step methodology for mapping conceptual models to logical representations, and consequently to the correspondent physical primitives that have the possibility to be executed using a commercial ETL tool.

To demonstrate the application of the methodology we purpose, we selected a real case study with the necessary characteristics for using and exploring all the ETL patterns designed. The case study is a solid urban waste collection application used for monitoring collection data from several recycling centres within a given city. Each recycling centre holds containers related to a specific type of garbage collected at different time intervals. Thus, step by step, we show how the ETL process is developed: from its initial specification, through the construction of ETL conceptual models using BPMN, its posterior meta-data enrichment with the necessary configuration parameters for describing its logical representation, and finally how both representations can be used to generate, at least, a

skeleton of the physical representation, which can be executed using the Pentaho Data Integration tool[13]. All development process steps are supported by the pattern approach we presented, showing how they can be applied in distinctive design stages and how they contribute to improving ETL development activities.

The solid urban waste collection application is recognized as an essential task in any daily city life. Day after day it assumes, even more, relevance as cities are expanding and their population increases. So, it is not surprising that the entities responsible for urban zones are more worry about the effectiveness of their waste collection processes implementation, for maximizing, within the possibilities, the usefulness of the solutions adopted.

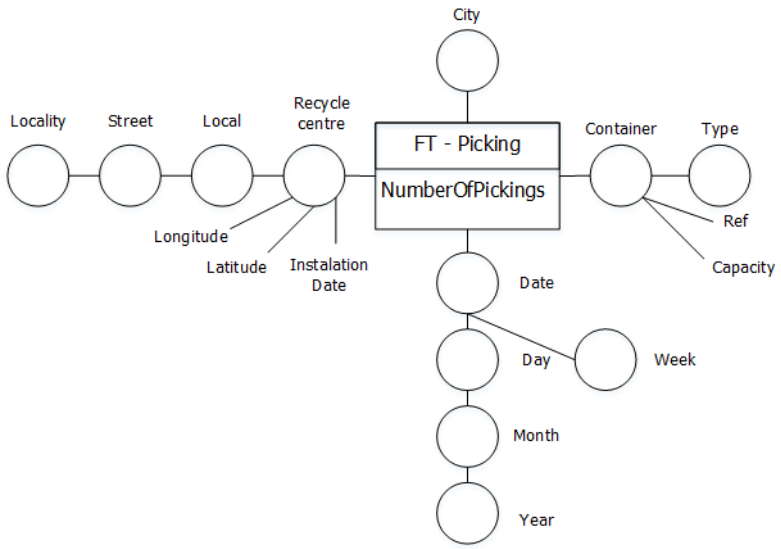


Figure 35. The DW star schema used for the solid urban waste collection case study.

To cover some of the most relevant information needs that managers usually have on waste collection management, we designed and implemented a business intelligence system especially oriented to monitor, control and predict waste management services, as well as to characterize and profile urban zones. The system works on a temporal basis, providing a general picture about waste collection processes, which were done in a specific period. The system was organized into five different scalable functional platforms, namely:

1. information sources gathering, where predefined packages of data are selected and collected for populating DW data structures;
2. data integration, which is an intermediate system's area (the DSA) that supports the execution of all data preparation processes – auditing, cleansing, transformation, etc. – accordingly to the requirements of the DW;

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

3. data warehousing, which maintains all the system's data marts. Each data mart supports a set of specific activities that are related to a decision-making area – waste collection, waste profiling, etc.;
4. data analytics, where are located the on-line analytical processing mechanisms and support structures, providing the means for exploring system's data according to the different analyses perspectives (dimensions) established previously by decision makers;
5. data visualization, which integrates a system's data visualization applications that are responsible for presenting the results of the analysis processes, using dashboards, custom reports, geo charts, gauges and other data visualization means.

In Figure 35 it is presented the DM schema that supports decision-making processes, where we can see the "Recycling centre" data and its characterization ("Recycle Centre" dimension), the garbage picking activities ("FT-Picking" fact table) for each container ("Container" dimension), the city ("City" degenerated dimension) for identifying picking activities location as well the related temporal data ("Date" dimension).

Table 3. A small excerpt of a picking data spreadsheet for a given month and city.

August					
Day 1			Day 2		
P	C	G	P	C	G
X060	O345	V713	X708	O118	V357
X732	O928	V600	X162	O099	V687
X1005	O845	V331	X106	O198	V944
…	…	…	…	…	…

System's containers are organized in three distinct categories: glass, plastic and cardboard packaging. The company uses several recycling centres spread across the city, which are usually composed of three types of containers. Following a predefined route planning and time interval, the company collects containers content for each recycling centre. The data related to the garbage pickup activity are stored using a conventional relational-based application and a set of spreadsheets (according to the approach followed by each company branch).

To reduce data inconsistency problems, specific organizational templates for spreadsheets sources were defined for each branch activity, ensuring a unique pickup structure - each spreadsheet corresponds to a specific year and stores the reference of all containers that were collected in a specific month/day. In the Table 3 we can see the representation of the spreadsheet structure that is used as data source. Basically, each spreadsheet receives data related to a set of garbage collecting

activities that are performed every day for a specific city. Each nested column represents the container code (not the recycling centre) that was collected, grouping them by type ('P' - Plastic, 'C'- Cardboard, and 'G' – Glass), which is identified by the first letter of its code. For each city, a secondary spreadsheet is also provided having a map describing the association between containers and recycling centres. Both sources (relational and spreadsheet) are used to populate the star schema presented (Figure 35). For that reason, we needed to develop specific population processes for handling their data specificities.

Table 4. Spreadsheet-to-Data Mart mappings.

Target dimension	Target field	Source object	Source field
Container	Ref	Picking spreadsheet	Each cell after header row contains the container ID where the garbage was collected along with the garbage "type" identification
	Type	Recycle centre and Picking spreadsheet	"Type" column from Picking spreadsheet under "Day" row header identifies the type of garbage collected. Additionally, the "Type" column from recycle centre spreadsheet identifies each container "type". The first characters used for container reference can also be used to identify the garbage type collected.
RecycleCentre	Local, Street and Locality	Recycle centre spreadsheet	"Local", "Street" and "Locality" fields are fields used to identify recycle centre location.
	Reference		"Ref" column
	State		"Notes" column
…	…	…	…
District	District	Picking spreadsheet	Each spreadsheet refers to a specific "district".
Date	Day	Picking spreadsheet	Each column represents a specific day included in the header row for days
	Month	Picking spreadsheet	"Month" row identifies the set of picking activities that were performed in a given month
	Year	Picking spreadsheet	Each spreadsheet refers to a specific year. Its name contains year reference (e.g. "2016_Lisbon")
FT - Picking	Total	Picking spreadsheet	Number of picking activities performed for a specific container and garbage type for a given day

Before proceeding to the ETL process design, we need to plan the source-to-target mapping to hold all logical relationship between source systems fields and target DW schema (dimensions and fact tables) fields. Possible data types incompatibilities can be identified along, including conversion logic programs and structures, business rules and SCD techniques that should be implemented for guaranteeing the system consistency.

Table 4 presents a simple example of a source-to-target data mapping that was established between the spreadsheet source and the target DM schema. As we can see, mappings are quite straightforward. However, we need to consider the variation attributes ("type'" for containers and "state" information for recycling centres) for the application of SCD procedures, as well as all the specific data transformation procedures that should be planned and implemented, namely:

- the recycle centre 'state' values normalization, that were applied without a consistent approach;
- specific spreadsheets columns identifying the pickup month as well as the pickup activities need to be matched and counted for each day based on the type of garbage collected. The data output is used for supporting the fact table populating process ("Number of Pickings").

Additionally, due to some internal specific business policies, some company branches share the collection activities data they hold about containers and recycle centres. This means that the collection activities data about each container is spread across data sources, which imposes the use of integration processes configured accordingly to specific integration rules. For demonstration purposes, the spreadsheet source was selected to illustrate all the steps needed to extract, conciliate, and integrate data into the DW schema using the ETL pattern-based approach we proposed previously in this work.

4.2 Pattern-oriented methodology for ETL development

Nowadays, software systems occupy a very relevant role in all the aspects of our society, helping people performing a lot of activities, ranging from simple to very complex and demanding critical tasks. Quality and robustness are two of the most relevant characteristics that a software piece must present, independently from the application domain. To ensure such properties, several efforts should be done right in collecting and modelling software requirements. Modelling is a delicate activity, often complex, requiring qualified personnel with proven experience and knowledge about many different techniques in the field. The models produced, whether they are conceptual or logical, are very valuable tools whose utility can be demonstrated in almost all development phases. The abstraction level that conceptual models provide is a very useful tool for supporting conversation with all the involved personnel in the project.

Conceptual models are an excellent groundwork for software validation. As such, it is not surprising the great importance that models have in the daily life of every professional in the software design and implementation. The topics we will approach next present a practical view of the pattern-oriented approach we developed, using the case study presented above. All the development stages are covered, starting with conceptual modelling using the BPMN language, its logical enrichment using a Domain Specific Language (DSL), and its final mapping to physical primitives.

4.2.1 Conceptual ETL design

The abstract view conceptual models provide simplifies the problem representation, helping designers to understand and knowing better the process, and at the same time in providing a framework that allows for the discussion of more generic details with non-technical personnel. To simplify the ETL system development, a task clustering technique was implemented to group a set of finer grain tasks into a collection of tasks, flows and control primitives. This provides a method for organizing them into software components with a clear purpose. Beyond that, we can represent them using different abstraction layers for supporting different details about its operational behaviours oriented to specific stakeholders in different project phases. Each component represents a pattern with a specific skeleton used for describing its internal behaviour and specific interfaces that set the communication boundaries between patterns for data interchange in a workflow context.

As already discussed, several contributions concerning the ETL development improvement were presented so far. Some of them approached the use of well-known modelling languages, such as the use of BPMN (El Akkaoui and Zimanyi, 2009). The BPMN language is composed of very expressive constructs, allowing for the representation of data flows in a large variety of ways and capturing both data and control flow constraints. However, such expressiveness represents a delicate problem, especially when we want to translate the ETL conceptual model for detailed representations that can be interpreted by a machine. Over the years, the research community addressed this problem, specifically in what is concerned with translating BPMN processes to the BPEL language. The detail and scope disparities between an abstract model and a physical model represent a huge distance between the two representations. This happens because they are used for different purposes. We use BPMN as the tool for representing ETL conceptual models using a new type of composite tasks, which describes semantically the patterns we presented before. Therefore, the necessary bridges for using this conceptual representation for execution primitives is accomplished by the means of an integrated approach based on the BPMN model enrichment using a specific DSL developed for supporting the requirements of each pattern.

The first step in developing an ETL system for supporting all the data migration requirements already described, starts by doing the conceptual representation using the BPMN language. This approach helps in the identification of early project requirements, its correspondent main entities, and business rules for supporting decision-making processes. As we know, BPMN provides a very simple and powerful notation for process representation (very suitable for ETL processes), coupled with his power of expressiveness, implementation constructs, and control tasks it provides. Additionally, companies use BPMN for describing and implementing internal business processes, which can reveal two advantages for ETL implementation (Wilkinson et al., 2010):

- the business users are already familiar with process construction language;

– all the existing business processes can be used and integrated as an ETL process, taking advantage of existing routines and process logic.

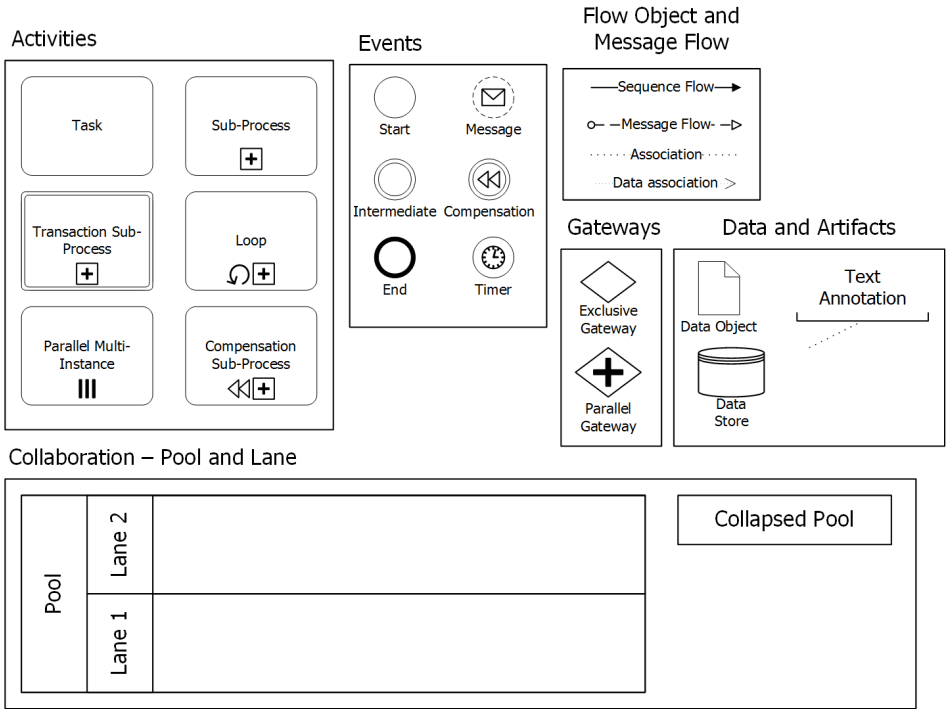


Figure 36. BPMN elements that can be used for pattern-based ETL conceptual modelling.

For supporting different abstraction levels, we can use BPMN collapsed sub processes to provide process conceptualization, since complex constructs can be decomposed into distinct levels. This is very helpful for high-level users when presenting, discussing and understanding process concepts. To show how BPMN can be customized for designing ETL processes following the proposed pattern-oriented approach, a subset of BPMN constructs were selected for integrating a conceptual palette of constructs especially oriented for the development of ETL conceptual models (Figure 36).

To describe ETL activities, atomic and composite tasks are represented using "Task" and "Sub process" elements, respectively. Both can be enriched, using specific markers (following the same combination restrictions imposed by the BPMN specification) for describing specific operational requirements. For example, the "Standard Loop" element can be used for describing a row-by-row data processing, the multi-instance marker for representing multiple activity instances that handle specific sets of data, and the "Compensation" element for embodying compensation processes for undoing/compensating actions performed by some ETL activity. Additionally, sub processes can be used for expressing a hierarchical workflow structure, encapsulating several ETL logical parts inside specific containers, and helping the representation of processes in a more readable way. The ETL patterns are represented using sub processes, more specifically using the "Transaction" sub process type, which groups a set of activities that constitute a logical unit of work that must be executed atomically.

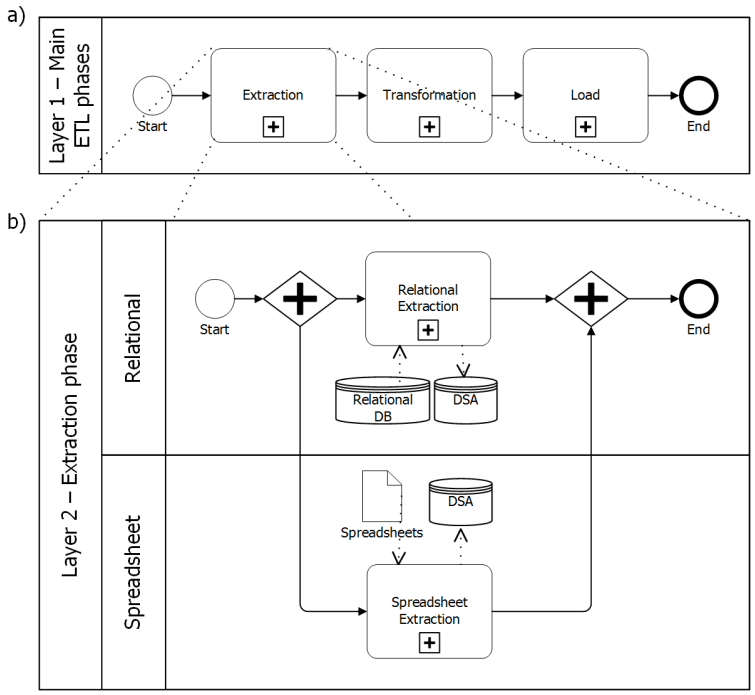


Figure 37. ETL extraction phase representation – Layer 1 (a), Layer 2 (b)

To distinguish between ordinary sub processes from ETL patterns, "Text annotations" can be used for identifying patterns and for providing a general description of more specific parameters. Additionally, the pattern initials between the hash character (e.g. '#CDC#') can also be used for identifying the pattern we are using. The "Events", "Gateways", "Flow Objects", "Message Flow", "Data" and "Artefacts" elements have the same semantic when are used in traditional BPMN processes:

- The "Events" describe something that happens and affects the behaviour of workflow activities.
- The "Gateways" are used to control process flow sequence based on some specific conditions.
- The "Flow Objects" and "Message Flow" are used for establishing relationships between the BPMN elements to describe the sequence of activities and the possible paths that can be taken, respectively.
- The "Data objects" and "Artefacts" are used for enriching BPMN processes with descriptive visual elements. "Data objects" artefacts represent data states flowing within the process, namely for describing data inputs, data outputs, and data stores. Using data elements, it is possible to represent the data received by each pattern, the data they produce as result of its execution, and the data they use to support its execution.
- The "Text annotation" artefact is used for adding extra detail to workflow objects, providing a way to express semantics associated with ETL patterns or other BPMN elements (e.g. for describing gateway conditions). The "Text annotation" description can be parameterized

using keywords for expressing specific requirements, reducing the ambiguities and allowing its future interpretation by specific tools.

- The "Pools" and "Lanes" are used for representing the process roles in the different abstract levels. These roles allow for the representation of several ETL layers using different strategies. For example, using "Lanes", it is possible to represent the data transformation process associated with a specific data source, a process entity, or covering a specific process variant. The "Collapsed pools" can also be used for hiding the process logic, being quite useful for representing exchanged messages between pools.

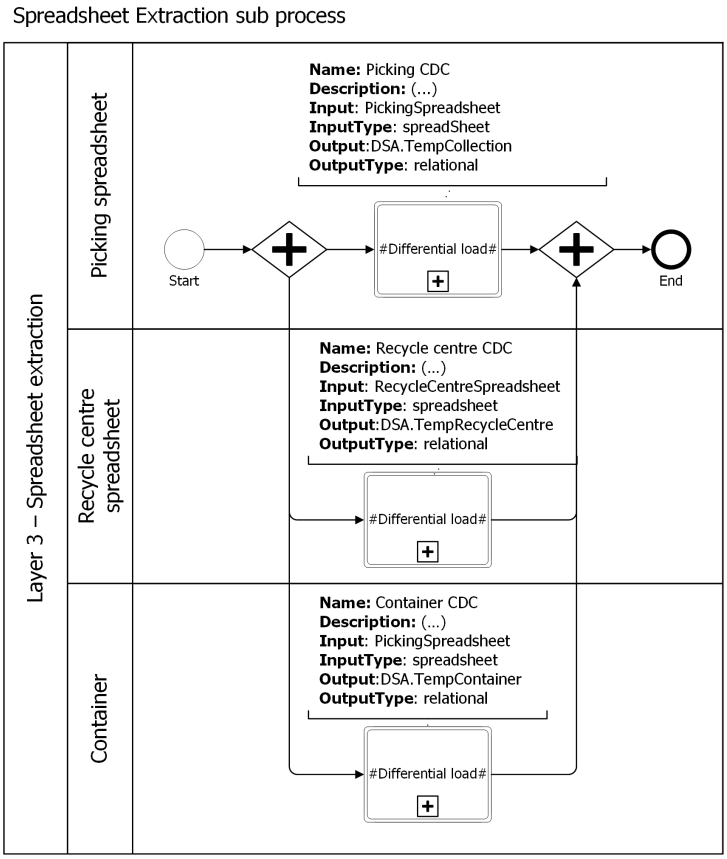


Figure 38. The ETL extraction process – Layer 3.

If we consider the source-to-target mapping from Table 4 and a possible subset of activities that are needed for covering all the requirements established to extract data from data sources, we can feed some temporary data structures in the DSA to conciliate and integrate data, and to load it into the target DW. All this is presented in several parts, which can be seen in Figures 37, 38, 39, 40 and 41. The conceptual model we designed is organized in several process layers. The most generic one - "Layer 1" (Figure 37a) - represents the most abstract level that can be derived from the three main ETL phases (Extract, Transform and Load).

The ETL extraction processes – "Layer 2" (Figure 37b) – begins with data extraction tasks applied over the spreadsheet files ("Spreadsheet Extraction" sub process) and a relational database

("Relational Extraction" sub process). To described this, we used a BPMN pool with two internal lanes, representing each one a different role related to the data source nature (relational or spreadsheet). Each BPMN sub process is composed by several finer grain tasks grouped together that can be described at more detailed layers. Two BPMN parallel gateways were used between BPMN sub processes to express the non-dependency between them, i.e. they can be executed parallelly. The first parallel gateway is used to initiate the parallel flows while the second one is used for flows coordination. Additionally, BPMN artefacts (Data Object and Data Store) were used to describe the nature of each data repository (structured data for data stores artefacts and semi-structured data for data objects artefacts) for each sub process with BPMN "Data association" artefacts describing the relationship directionality: from sub processes to Data Artefacts indicating the output data flow and from data artefacts to sub processes indicating input data flows. They only have a visual representation value, since at this level of abstraction they do not provide any kind of semantic value that can be used later. However, when used at a finer grain level, the "Data Association" Flow Object can be used for identifying the type of the source and the direction of the data flow.

Table 5. Patterns description for the Extraction process presented in the Figure 38.

Pattern	Pattern name	Description
Differential Load	Picking CDC	The pattern extracts data from the "Picking" spreadsheets that holds data about container's pickings activities for all the cities covered. Data are extracted and stored using a relational table in the DSA ("TempCollection") with a set of columns equivalent to the columns extracted from the spreadsheet.
Differential Load	Recycle centre CDC	The pattern extracts data from the "RecycleCentre" spreadsheet that holds data about all recycling centres: Local, Street and Locality, just to name a few. Data are extracted and stored using a relational table in the DSA ("TempRecycleCentre") with a set of columns equivalent to the columns extracted from the spreadsheet.
Container	Container CDC	The pattern extracts data from the "Picking" spreadsheet that holds data about all containers: the reference (code), type and the current location. Data are extracted and stored using a relational table in the DSA (Container) with a set of columns equivalent to the columns extracted from the spreadsheet.

For the spreadsheet data extraction process description, the "Layer 3" represents specific pools for process description with a more detailed level, where each lane represents the extraction processes for each processed entity (Figure 38). At this level, we defined some "Transactional sub processes" to represent ETL patterns (the "#" character identifies the pattern used), which means the process cannot be decomposed more at the conceptual level. The BPMN "transactional sub-process" embodies the notion of consistency, inferring the ACID properties applied to any defined database transaction. The spreadsheet data extraction process (Figure 38) includes three "Differential load" pattern

instances, which are used for extracting data record from the spreadsheet and comparing them with the data already stored in the DSA to identify new or changed data. In the Layer 2 (Figure 37), we used two parallel BPMN gateways for indicating that tasks belonging to each flow may be executed in parallel, which can be used for further ETL development phases. We used again two BPMN gateways for Layer 3 (Figure 38): the divergent gateway to split flows that invoke the extraction activities of each entity, and the divergent gateway to guarantee that all activities initially invoked are synchronized, i.e. ensure that they finish before proceeding to the next steps. At "Layer 3" (Figure 38), the patterns full or abbreviation name are used between hash characters (#), just because we need to identify the pattern instances and improve the process readability for the patterns identification without changing BPMN base elements.

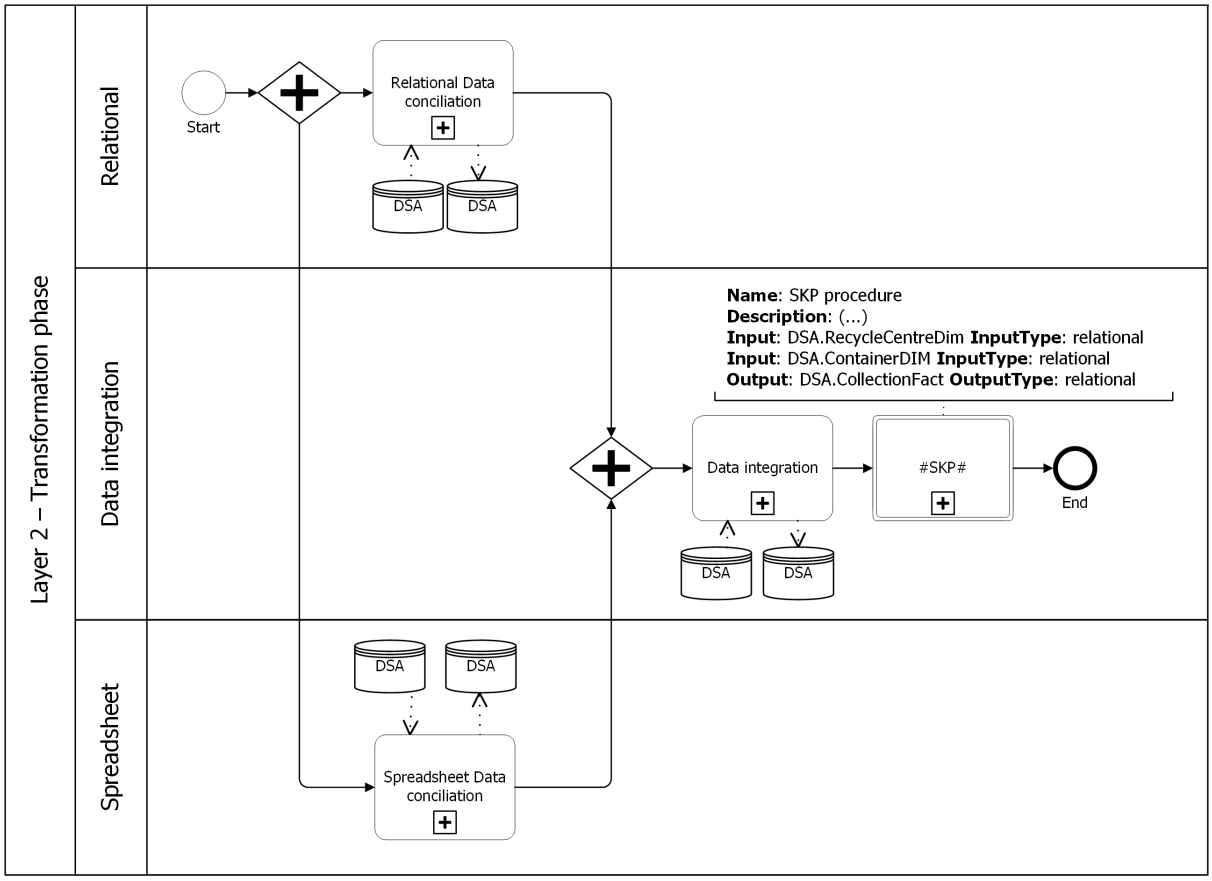


Figure 39. ETL Transformation layer – Layer 2.

Additionally, we used "Text Annotation" artefacts for enriching the conceptual model, exposing semantic details to describe better some pattern details based on a specific structure. In Figure 38, we used these artefacts to improve the meaning of pattern instances, applying some specific keywords, namely:

- "Name", for identifying the name of a pattern.

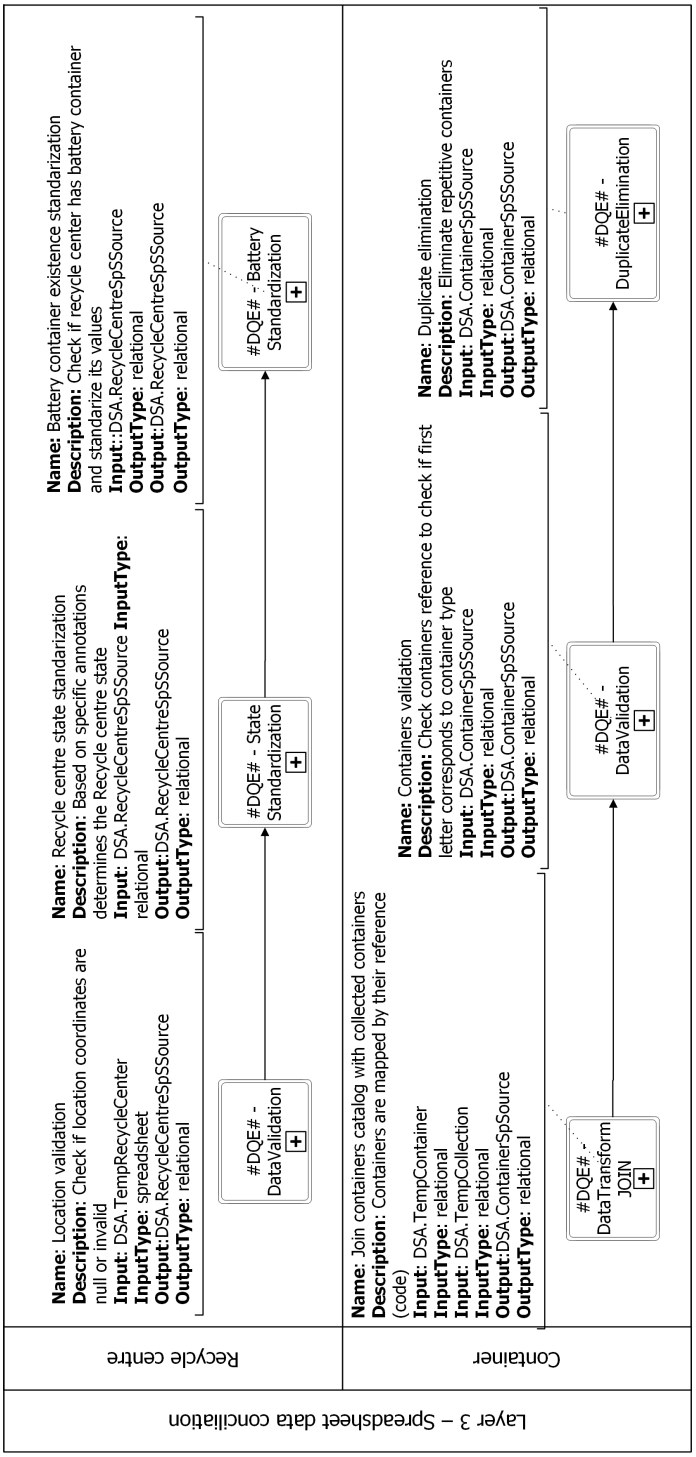


Figure 40. ETL transformation layer: Layer 3 subset (part 1) for spreadsheet data conciliation.

- "Description", for describing a useful and unstructured description of the functionality of a pattern.
- "Input", for identifying the input file name we use.
- "InputType", for presenting the technology used on data input tasks.
- "Output", for describing the output object name used to store output data.

– "OutputType", for designating the technology used to store output data. The input and output data objects identification uses the "." (dot) character to identify the path to the data, e.g. "pathToData.data".

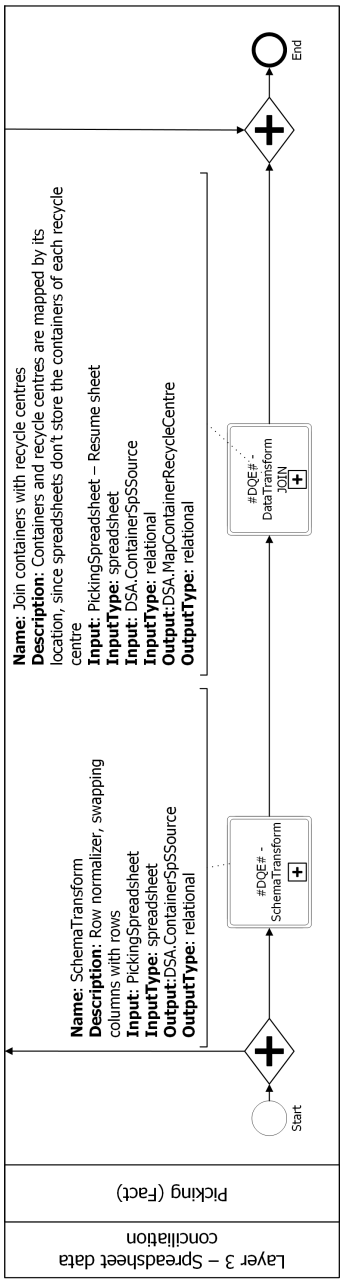


Figure 41. ETL transformation layer: Layer 3 subset (part 2) for spreadsheet data conciliation.

The Table 5 shows a more detailed description of Figure 38 extraction process and the tasks needed to extract data from each spreadsheet to the DSA area equivalent structures.

Having all the necessary data stored in the DSA, we can start all the processes related to data cleaning and normalization, in a way that do not compromise the source systems operationality. The "Transformation" sub process presented in Figure 39 encapsulates the data conciliation sub processes we designed to normalize data that come from the spreadsheet source ("Spreadsheet Data

conciliation" sub process) and from the relational source ("Relational Data conciliation" sub process). Both processes are independent (denoted by the two BPMN parallel gateways used) and are applied entirely in the DSA, which means that previous loaded data is processed and stored using the same data structure (relational table) or a new different temporary structure stored in the DSA. Such particularity is partially described using the BPMN data stores to describe the relationship between each sub process and their input and output flows.

	Day				
Month	**1**	**2**	**3**	**(..)**	**31**
	P245	G23	P34	…	G34
	F98	G32	P21	…	P25

Container	**Month**	**Day**
P245	August	1
F98	August	1
G23	August	2

Figure 42. Picking data schema transformation.

Before describing the posterior tasks related to the data integration of the data produced by the "Spreadsheet Data conciliation" and "Relational Data conciliation" sub processes, the Figure 40 presents a subset of the process created to describe the "Spreadsheet Data conciliation" sub process, representing part of the "Layer 3" for the process transformation phase. Several pattern instances are used for aligning source data structures with the target DW schema requirements with each pool representing data transformation procedures applied to the "Recycling centre" and "Container" entities for data previously extracted in the Figure 38.

To deal with the "Recycle centre" data, we used three DQE patterns instances for transforming data from each Recycle centre. They are:

- "#DQE# - DataValidation", which is used for checking the location value and their domain consistency.
- "#DQE# - StateStandardization", which is used for determining the Recycle centre state ('Active'/'Inactive') based on some data annotations.
- "#DQE# - BatteryStandardization", which is used for standardizing the values related to the existence of a battery container in each recycle centre, replacing the 'Y' value by a consistent and meaningful standard value.

For each container, the following DQE patterns instances are applied:

- "#DQE# - Data Transform JOIN" – Used to map the containers data catalogue with the containers used for the picking activities processed;

Table 6. Patterns description for the Transformation process presented in the Figure 40 and Figure 41.

Pattern name	Pool	Description
Data Validation	Recycle centre	Data previously extracted to "DSA.TempRecycleCenter" is processed to check the fields related to the current location of each recycle centre. Only the records with valid values for "District", "Zone" and "Street" remain on the "DSA.TempRecycleCenter".
State Standardization	Recycle centre	For each recycle centre, a specific column named: "Notes" is preserved in source data. This column contains unnormalized values like: "I", "A" or even using date values to indicate the date that a specific Recycle centre was deactivated. Such data is normalized simply for "Active" or "Inactive" to preserve history about picking activities on a given place.
Battery Standardization	Recycle centre	Since some Recycle centre contains a specific recipient for batteries collection, this pattern is responsible to normalized values like: "Y", "YES", "yes" for the meaningful values: "Yes" or "No".
DataTransform JOIN	Container	Containers are pre-cataloged using a specific sheet that was extracted to the "DSA.TempContainer" table. However, the process should identify the containers that were collected for the current data populating process, relating them to the existing catalogue to find their data and eventually identify new containers. For that, a typical relational join is performed using the container reference to match related records.
Duplicate Elimination	Container	Since the previous join operation naturally produces several duplicates values for containers data, this pattern instance is responsible to remove duplicate containers based on its "reference" (code).
Data Validation	Container	Container's type can be identified in two different ways: by its type or using the first letter of their reference (code). Thus, it's important to guarantee that both representations are consistent.
SchemaTransform	Picking(Fact)	This pattern instance is responsible to transpose columns to rows (Figure 42).
DataTransform JOIN	Picking(Fact)	This pattern instance executes after patterns described in the "Recycle Centre" and "Container" pools, performing a relational join between three tables produced in previous steps.

- "#DQE# - Duplicate Elimination", used to remove duplicate containers. This is necessary because the same container can be collected several times and for that reason, produces several picking entries.

- "#DQE# - DataValidation", which is used to check containers reference values, ensuring the consistency of their code with their type. As already stated, the first reference character identifies each container type and for that reason should be consistent with the pre-defined type;

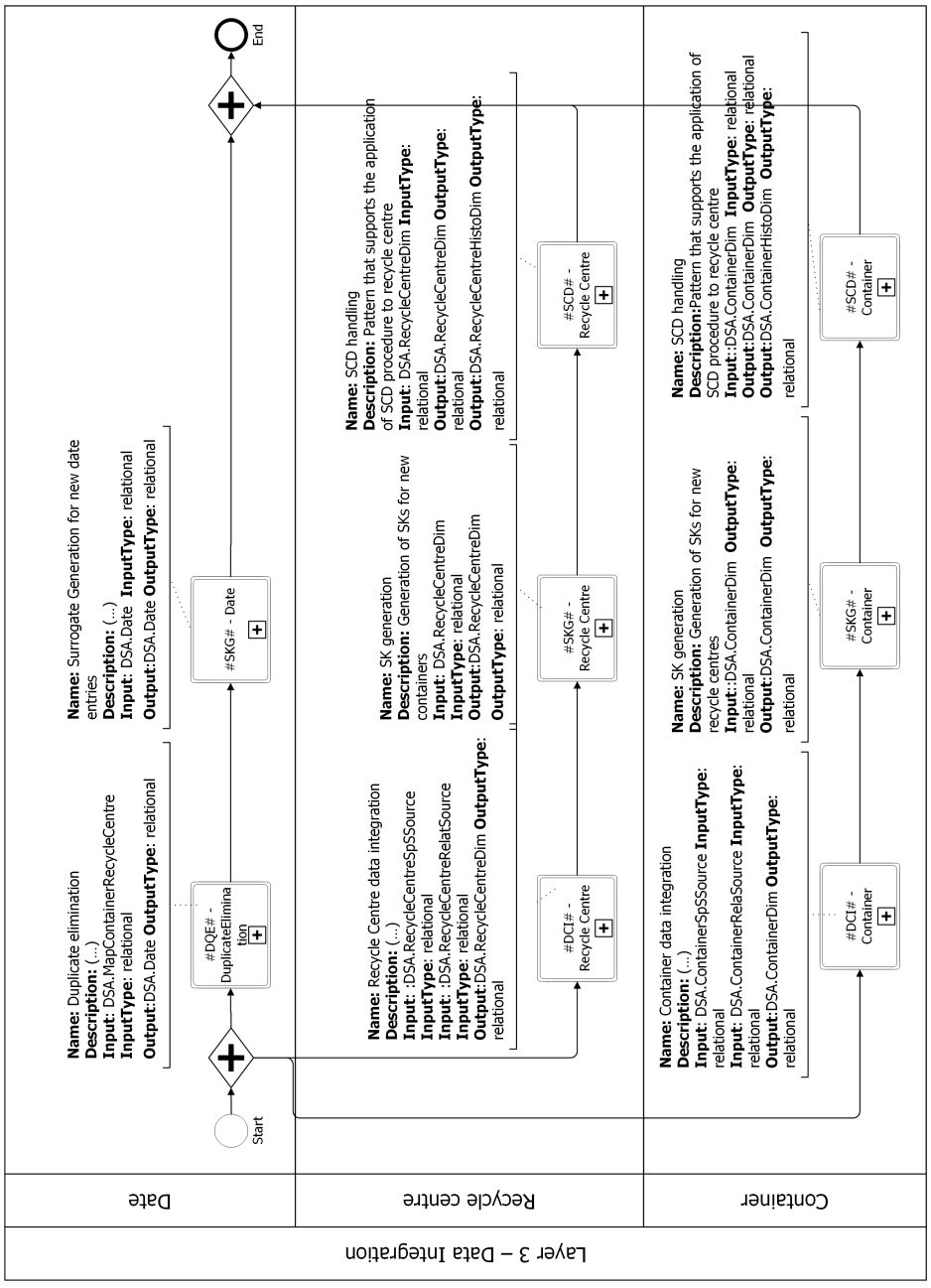


Figure 43. ETL Transformation layer: Layer 3 for spreadsheet data integration.

The remaining pool that completes the example for Figure 40 is presented in Figure 41. As its name refers, the picking spreadsheet stores data about some picking activities for each container of a given recycle centre based on the structure presented in the Table 3. This pool initiates the process related to the process "Layer 3" using a BPMN parallel gateway to distribute the flows not only for the "Picking (Fact)" pool but also for the Container and Recycle Centre pools presented in Figure 40.

For the data related to picking activities, we need to transform the original schema to get the data according to the structure of a typical relational table. Thus, a column "rotation" should be performed, transposing data contents from (some) rows to columns. Data is decomposed using the "#DQE# - SchemaTransform" pattern instance from the "Picking Fact" pool, generating distinct rows based on each garbage picking activity expressed in the picking spreadsheet (Figure 42). Next, when the pattern instances from the other pools are executed (the convergent BPMN gateway ensures that) and having the data about recycle centres and containers normalized, we can proceed to the mapping between containers, recycle centres and picking activities to identify the correct correspondences between them. For that reason, we should develop supplementary efforts for identifying the associated containers for each recycle centre. A recycle centre and a container address represent the set of attributes used to find the correspondences between recycle centres and their respective containers. Thus, the "#DQE# - DataTransform JOIN", from the "Collection (Fact)" pool, it was used to find correspondences and for identifying data related to. Any record without correspondence or having conflicts can be managed through the definition of a specific exception handler policy for managing unexpected situations and ensuring the process consistency (not included in the BPMN representations). Additional detail related to each DQE pattern configuration is presented in the Table 6.

After the conciliation phase, we proceed to the integration phase (Figure 43) for handling data about similar objects that come from relational and spreadsheet source, for storing them into a single repository. Since the picking data is exclusive in each data source type, only the "Recycle centre" and the "Container" data should be considered for the integration procedures between the two types of sources. For that reason, in Figure 43, the "Container" and "Recycle Centre" pools represent the pattern instances used to integrate data from containers and recycle centres. For both pools, similar pattern instances were used:

- "#DCI# - Data Conciliation and Integration pattern instances ("#DCI# - Recycle centre" and "#DCI# - Container") used for data integration between data related to recycle centres and containers from both sources;
- #SKG# - Surrogate Key generation pattern instances from both container and recycle centres previous integrated ("#SKG# - Container" and "#SKG# - Recycle centre") to generate new SK values for the new records;

– #SCD# - Slowly Changing Dimension pattern instances encapsulated using the "#SCD# - Recycle centre" and "#SCD# - Container") for both "Container" and "Recycle Centre" entities to deal with attributes with variation.

Table 7. Patterns description for the Integration process presented in the Figure 43.

Pattern name	Pool	Description
DuplicateElimination (DQE)	Date	Uses the data produced in the Figure 41 process to select date values and remove duplicate values.
Date (SKG)	Date	Generates new SK values for the new date records extracted and replaces business keys by surrogate keys previously generated and stored in specific mapping tables.
Recycle Centre (DCI)	Recycle Centre	Integrates data from both data sources ("DSA.RecycleCentreSpSSource" for spreadsheet recycle centres and "DSA.RecycleCentreRelatSource" for relational recycle centres) based on recycle centre location and installation date. Conflicting records are stored in quarantine tables while for consistent records, the mapping table holding the correspondences between business and surrogate keys is updated.
Recycle Centre (SKG)	Recycle Centre	Generates new SK values for the new recycle centres records identified, and replace business keys by surrogate keys previously generated and stored in specific mapping tables.
Recycle Centre (SCD)	Recycle Centre	This pattern is responsible to handle the attributes with variation: "state", applying the SCD technique with history maintenance.
Container (DCI)	Container	Integrates data from both data sources ("DSA.RecycleCentreSpSSource" for spreadsheet containers and "DSA.RecycleCentreRelatSource" for relational containers) based on container's "reference" (code). Conflicting records (for "capacity" or "type") are stored in quarantine tables while for consistent records, the mapping table holding the correspondences between business and surrogate keys is updated.
Container (SKG)	Container	Generates new SK values for the new container records identified and replaces business keys by surrogate keys previously generated and stored in specific mapping tables.
Container (SCD)	Container	This pattern is responsible to handle the attributes with variation: "type", applying the SCD technique with history maintenance.

Additionally, the "Date" pool presents a "#DQE# - DuplicateElimination" pattern instance that was used to remove duplicate date entries previously normalized by the "#DQE# - DataTransform JOIN"

pattern instance (Figure 43). The new SK values for the new "Date" entries are also generated, using a "#SKG# - Date" pattern instance. The Table 7 describes all patterns instances described in Figure 43.

The last step of transformation phase involves populating the fact table. To do this we used a "#SKP#" pattern instance, which was described in "Layer 2" (Figure 39). The "Load sub-process" from Layer 1 (Figure 37a) represents the data load processes for covering all the mechanisms need to store conciliated data into the target DW repository. To support this process, we created a specific IDL pattern instance for populating the dimensions first, and then the fact table respecting the dimension's foreign keys. This involves a lookup task to each mapping table to find the surrogate key generated to replace the operational business key. A very straightforward process when viewed from a conceptual perspective.

The use of patterns enhances the separation of workflow orchestration and data transformation patterns. The BPMN language supports quite well the workflow coordination aspects and the visual representation of the ETL patterns, leaving more freedom (and time) for process specification and runtime infrastructure. Additionally, at initial phases, we also studied other languages and notation for the ETL conceptual specification based on patterns. For example, the YAWL language (Oliveira et al., 2014; Oliveira and Belo, 2014c) and the REO specification language (Bruno Oliveira and Belo, 2013a; Oliveira and Belo, 2014a) were explored for ETL conceptual modelling with a very satisfactory success. However, the findings revealed that BPMN provides a simpler, expressive and richer language for expressing high-level requirements. For that reason, we believe that it is more suitable for the representation of abstract models. Even so, we avoided the redundancy of its constructs using the DSL specification language for configuring ETL patterns, as discussed in this work.

4.2.2 Logical ETL design

Any ETL modelling activity should reflect different control flows and data flows established between tasks. Using BPMN for specifying ETL tasks provides an expressive notation and sophisticated orchestration mechanisms that can be used for the subsequent development stages. Also, BPMN allows for control flow representation between ETL process activities, as well as permits the characterization and description of all the activities to implement. However, the real challenge is to make the correct combination of these two types of flows in one single representation model and, therefore, in a single tool (Wilkinson et al., 2010).

ETL conceptual models can be used to produce (at least) a first version of the target system. Using a domain specification language (DSL) for expressing the behaviour of each pattern can help to reduce ambiguities that are regularly associated to BPMN conceptual models. For that, we designed and implemented a specific DSL especially oriented for configuring patterns, which can be used to

produce logical models with the ability for describing in a more detail way, the behaviour of each pattern that was broadly described in the correspondent BPMN conceptual model.

The DSL covers all the general requirements of each pattern category, providing a powerful way to configure its behaviour. However, besides pattern internal behaviour, we must also consider the workflow environment in which each pattern is used since affects its future execution. For that reason, while the BPMN conceptual representation describes execution flow properties and constraints that are used for generating the physical models, the DSL was designed and developed for supporting the patterns configuration according to the BPMN specification. The pattern configuration embodies the DSL core, while the gateways and events configuration (used for describing control flow logic) can be extracted directly from the BPMN specification - the BPMN 2.0 specification supports functional properties applied to each one of these standard constructs.

Table 8. DSL constructs overview.

Language elements	**Keywords**	**Ontology property**	**Example**
Pattern selection	*PatternHierarchyPath.pattern_name*	ETL patterns hierarchy	`Transformation.DQE`
Atomic statements	`Statement_name = expression`	Class data property	`Type = relational`

Composite statements	`[`
	`  statement_name_1 = expression_1`
	`  statement_name_n = expression_n`
	`]`
		"minCardinality" value constraint	`[`
			`  data=PickingSp`
			`  type=relational`
			`]`

Main blocks	`Block_name_1{}` `Block_name_n{}`	Object properties with Pattern class	`source{}`

Multiple main blocks	`Block_name{`
	`  [block_content],[ (…)]`
	`}`
		"maxCardinality" value constraint	`source{`
			`  [`
			`  data=PickingSp`
			`  type=relational`
			`  ]`
			`},[]`

Options	`OPTIONS{`
	`  Atomic_Statement_name_1`
	`  Atomic_Statement_name_n`
	`}`
		Pattern data/object properties ("functionalProperty" or with "maxCardinality" = 1)	`OPTIONS{`
			`  Function=SUM`
			`  Field=price`
			`  (…)`
			`}`

All the DSL constructs were developed for supporting the requirements of each pattern. Following a predefined template, we can describe and formalize patterns, as well as to instruct the generator/parser engine about how patterns must be generated. The pattern structure description was already studied for general systems (Kazman et al., 2013), and also for ETL processes (Köppen et al., 2011). However, in these works patterns are not described using a technical language that enables the generation of physical models. Instead, in the pattern-oriented approach we describe,

patterns are enriched with primitives for supporting their physical instantiation that represents at least a first version of a physical model for implementing in some ETL tool.

The use of detailed tasks such as joins, selections or projections at a finer detail level affects process interpretation. In these cases, their representation is not very useful unless there is a tool for supporting their implementation. Some commercial tools are very suitable for cases like these, since they provide powerful and well-known frameworks that are used by many IT professionals around the world. Therefore, the proposed DSL - PL4ETL, can be applied on each pattern described by the ontology we presented, covering its operational stages and providing a solid framework enabling for its conversion to equivalent semantics that can be acquired and used by a commercial tool. Based on the concepts and properties presented, we designed and implemented an instance pattern generator program that produces automatically the language construction rules (syntax) and the ontology data model, for a given ETL model. To support the language syntax specification, we created a set of static type statements and keywords for describing each language component. The pattern configuration starts with the "USE" keyword followed by the pattern name. The "USE" keyword identifies the pattern path that should be used based on the pattern taxonomy tree. Top levels classes (the "Pattern" class defines the higher level) should be first defined and the reserved character: "." (dot) is used to traverse each hierarchy level from the middle levels to bottom levels.

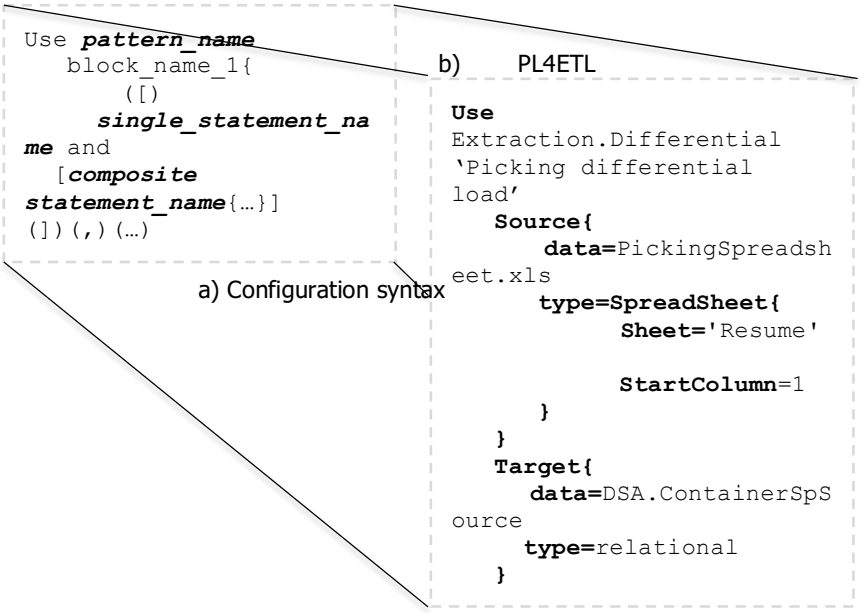


Figure 44. Examples of a PL4ETL basic pattern configuration syntax (a) and a language instantiation (b).

Next, and based on each pattern class object properties, we define a set of blocks delimited by "{}" (braces). Inside each block, it is possible to represent simple or composite assignments. In pre-defined blocks (generated based on the pattern class structure), simple assignments are formed based on the associated data properties, while in composite statements they are generated based on object properties. Each block may contain more than one occurrence according to the cardinality of

the object properties associated with the pattern class. The "OPTIONS" block is used to map the associated properties with the most specialized pattern class that was used. It can be composed of single or composite statements - Table 8 resumes the main syntax statements that can be used for supporting the DSL presented.

Based on the ontology and the syntax rules presented, the configuration language can be automatically generated for each pattern, which enhances the language flexibility. This approach guarantees that if the ontology changes, then the correspondent grammar rules will be consistent with the ontology definition.

```
(…)
  Throwable{
   [
     Type=Exception
     Event=SourceFieldNotFound
     Target{
       Data=ContainerQuarantine
      Type=relational
       Fields{
         Compensation=Eventdescription
       }
       FieldMapping{
         Target.Eventdescription="(…)"
         (…)
       }
     }
   ]
   [
     Type=Error
     Event=ContentsNotReadable
     Action=Stop
   ]
  }
  Log{
   [
     Type=Critical
     SourceEvent=SourceFieldNotFound
     Target{
       data=(…)
     }
   ]
  (…)
}
```

Figure 45. PL4ETL configuration for "Throwable" and "Log" components.

Additionally, conceptual model text annotations can be used for auto-complete parts of the PL4ETL specification automatically, namely for input and output blocks cases. In Figure 44 we can see an example of the syntax rules applied to the language constructs (Figure 44a) and a correspondent example of its instantiation (Figure 44b) we used for specifying the containers differential data load ("#Differential load#" pattern instance – Figure 38). The "Source" and "Target" blocks represent the input and output metadata, and the "data" and "type" properties the name and type of each data object used, respectively. These blocks can be derived from the annotations used in the conceptual

model specification. The "FieldMapping" block describes the field association between the source and target data objects. These three blocks correspond, respectively, to the "hasDataRepositoryInput", "hasDataRepositoryOutput" and "SourceToTarget" ontology object properties. To extract data for the "input" block, we described a spreadsheet file based on a specific sheet included in the "Picking" spreadsheet.

The "Sheet" property is used for identifying the specific sheet (the "Resume" sheet is used) and the "StartColumn" property for indicating the column the process uses for data extraction (since two properties are described, a composite statement is used). Basically, the "Output" block is composed of single statements that describe the name ("data") and the type ("type") of the target repository. Other design details, such as the database name or the server name were omitted, just because they can be configured in further steps. To describe the fields mapping we used the prefixes "Source" and "Target" for identifying the source of each field, which is especially relevant when using more than one data object to avoid ambiguous fields. In the case of multiple source or target repositories, the index identification (based on the block order) can be used for distinguishing them - for example, "Source[0].reference = Target[1].id" identifies the first "Source" block and the second "Target" block for the comparison expression. After the fields identification, the keyword "OPTIONS" is used for specifying configuration parameters (derived from properties applied to each pattern class described in the ontology we described in chapter 3) associated with "Differential" pattern instance, namely:

- "HeaderRow", which indicates if the header row describing each field is included as a data entry.
- "NoEmptyRows", which indicates that empty rows should be excluded from the result.

The "Throwable" and "Log" components are also supported by the specification using the "Throwable" and "Log" blocks. In Figure 45 we can see an excerpt of a possible configuration for these two blocks, describing an "Exception" and "Error" events for the "Throwable" component, and a critical log configuration for storing metadata related to the "Throwable" events. The "Throwable" component uses two simple statements for describing the event nature ("Type" property), the processed event ("Event" property), and a composite statement for defining the target repository ("Target" block) for storing non-conformed records. The "Data" and "Type" properties have the same purpose as referred before. The "Fields" property identifies the nature of each field that is used in the pattern configuration. In the present case, only the "Compensation" field is identified for storing exception metadata. The "FieldMapping" property is used for identifying the relationship between the source attributes and the target quarantine table attributes, which are derived from the "SourceToThrowable" ontology object property. Additionally, it was defined an error event ("Type=Error") to stop process execution ("Action" property) every time the source data cannot be read ("Event" property).

In Figure 45 it is also presented an configuration excerpt for log component that is used to store log entries related to critical ("Type" property) events that can compromise ETL execution, the source event ("SourceEvent" property) that identifies the event from the "Throwable" component, and the target log object configuration used to store data ("Target" composite block).

Using the Protégé platform[14], we can generate and manipulate an OWL XML serialization file using the Protégé-OWL API (Horridge, 2012), allowing the manipulation of the DSL syntax rules using the Xtext framework[15]. This feature allows for the ontology manipulation, providing at the same time the necessary contracts to control and implement models based on patterns configuration.

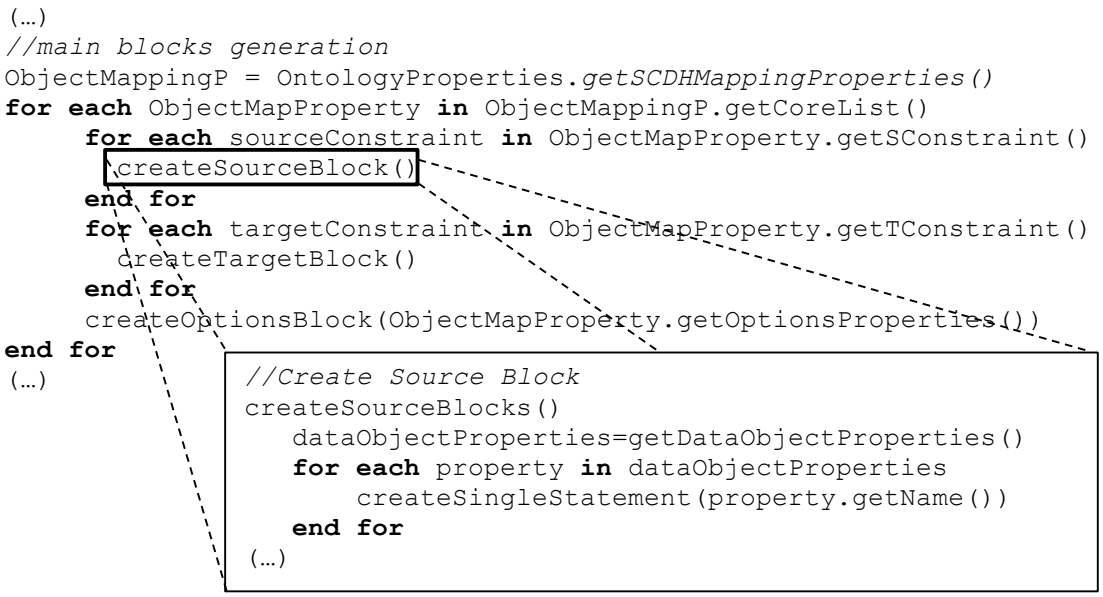


Figure 46. Ontology interpretation and DSL specification.

The syntax rules lead the physical process generation, beginning by reading all pattern objects and data properties inherited from more abstract classes. Data properties are identified and used as single statements for generating parameters such as the "data" and "type" data properties of each data source ("Data Object" entity), reachable from "sourceToTarget" object mapping property. Each pattern preserves mappings ("Mapping" entity), identified by the "hasInput" and "hasOutput" object properties that are related to the input "DataObject" (domain) and the target "DataObject" (range). The object cardinality restrictions are used to rule the number of potentials blocks that can be configured posteriorly. The number of "Source" or "Target" blocks that are available are identified based on the cardinality associated with the "hasInput" or "hasOutput" object properties, while the "Throwable" and "Log" blocks are generated based on the "sourceToThrowable" and "SourceToLog" object properties. The same procedure is followed for identifying the number of blocks that can be

[14] http://protege.stanford.edu
[15] http://www.eclipse.org/Xtext/

used. Moreover, each pattern can also enforce constraints based on their specific rules, which should be used by the generation engine to produce the language construction rules.

```
(...)
Pattern:level2=PatternLevel2'.'(('.')?blockComposition+=PatternLevel3)
*;
block:
	'Use' pattern=Pattern ':'
	input=InputBlock
	output=OutputBlock
	fields=FieldMapping
	options=PatternOptions
;
InputBlock:
	'Source' '{'
		blockComposition=BlockComposition
	'}'
;
OutputBlock:
	'Target' '{'
		('[' blockComposition+=BlockComposition ']'
			('AS' alias=ID)?)*
	'}'
;
Field: field=ID | SCDFieldType;
FieldList:fields+=Field ('.' fields+=Field)*;
FieldComposition:
	(BlockType'.')?fields+=FieldList '='
	(BlockType'.')?fields+=FieldList
;
SCDFieldType:'StateField' | 'DATE';
FieldMapping:
	'FieldMapping' '{'
		(FieldComposition+=FieldComposition)*
	'}'
;
SCDOptions:'variationFields=' field=ID (...);
PatternOptions:
	'Options' '{'
		options=SCDOptions
	'}'
;
BlockComposition:
	'data='data+=ID ('.' data+=ID)*
	'type='type=DataObjectType
;
enum DataObjectType:RELATIONAL='relational'
| SPREADSHEET='spreadsheet';
BlockType:'Source' | 'Target';
(...)
```

Figure 47. Specification of rules using Xtext for a SCD pattern instance.

The Figure 46 presents a simple excerpt for the algorithm used for interpreting the ontology and producing all the main primitives to rule the DSL pattern configuration. Additionally, to support the generation of the physical model, we developed some special-purpose templates to support the production of specific grammar rules. These templates are composed of static and dynamic parts: static rules for defining blocks that should be enriched, and dynamic blocks to be generated based on ontology elements.

The syntax rules generated for the SCD core configuration, identifying an excerpt of its composition in terms of static and dynamic blocks (identified using bold format) are presented in Figure 47. The block composition is static. This means that "input", "output", "fields" and "options" should be mandatory in any pattern configuration. Keywords such as "Use", "Source", "Target", "FieldMapping" and "Options" are used to identify these blocks, being delimited using brackets ("{}"). The assignment operators are dynamic, each one with different semantics. For example, for the SCD pattern, an equal sign ("=") is generated to accept only one syntactic declaration of "BlockComposition" (only one input source is allowed). The logic related to the language validation is based on the data properties associated with the range of "HasDataRepositoryOutput" object property ("DataObject" class): "data" and "type" features. The data property is defined with a 'String' type, enriched with syntactic rules that allow for the data objects description using the dot character to represent the path between the objects that contain target data – e.g. "server.Database.tableName".

```
Use transformation.SCD:
Source{
  data=DSA.ContainerAuditData
  type=relational
}
Target{
  [
    data=DW.ContainerDim
    type=relational
  ] AS ContainerDim
  [
    data=DW.ContainerHistoDim
    type=relational
  ] AS HistoryDimension
}
FieldMapping{
  Source.sk = ContainerDim.ref
  Source.capacity = ContainerDim.capacity
  Source.type = ContainerDim.type
  StateField = ContainerDim.state
  OperationField = Source.operation
  Source.changeDate = HistoryDimension.ExpirationDate
  Source.type = HistoryDimension.type
}
Options{
  variationFields=type
}
```

Figure 48. A configuration excerpt for a SCD pattern instance.

The range of values available for a given property is determined using enumerations. For example, the "type" property defines a property range, such as: "DataPropertyRange(:hasType DataOneOf('relational' 'spreadsheet'))", which results in DSL equivalent primitives, i.e. enum rules. The DSL "FieldsMapping" block is described using "FieldComposition", which is enriched by the SCD specific field ("Field" and "FieldList") types ("SCDFieldType"). The "FieldComposition" comprises an arbitrary number ("*") of entries that can represent direct field mapping between source and target

("BlockType") repositories. The "Source" and "Target" blocks can also be renamed (using the "AS" keyword) to simplify their use and interpretation.

In Figure 48 we can see another example for a pattern instance configuration, showing an SCD configuration that follows the logic behind the configuration described in Figure 47. The DSA 'ContainerAuditData' table is used for data input, while the 'ContainerDim' and 'ContainerHistoryDim' dimension tables are used for data output (square brackets are used for identifying each output source individually). Each one of the blocks is renamed to 'ContainerDim' and 'HistoryDimension', respectively. Next, the "FieldMapping" was used for describing relationships between the source input repository ("Source" keyword is used by default) and the alias defined for each output repository. The state field is also identified ("StateField" keyword) to properly populate the target dimension, while the operation field ("OperationField" Keyword) is used to identify the field that stores metadata about the operation performed in source systems. The "Options" block identifies the variation field for this process, using the attribute name or the name format: 'Source.AttributeName'.

4.2.3 Physical ETL design

Before proceeding to the generation of the ETL physical model, we need to have some guarantees that must be assured by earlier modelling process phases. Without them, it will be not possible to enable the physical generation of ETL models. At first place, all patterns used in the conceptual model must have their specification in BPMN, so that the program responsible for generating the physical system knows what type of physical component should be generated. Next, it must be assured that all patterns used in the BPMN schema have their behaviour described in the proposed DSL. This means that each pattern must use the DSL to set its configuration, sustaining the appropriate generation for the ETL system first version: a skeleton.

The entire transformation process relies on a converter, a program written in JAVA programming language, which takes the BPMN models, parse the correspondent DSL specification and construct the final system. Later, the produced piece of software will be imported into the ETL tool we selected: the Pentaho Data Integration (Bouman and Dongen, 2009), previously known as Kettle. This process involves the use of two distinct components: the BPMN parser, which is responsible for interpreting and validating the BPMN definition, and the grammar parser, which is responsible for validating and interpreting the configuration data associated with each pattern instance.

The BPMN conceptual specification is used to identify the patterns for supporting data transformation processes, their sequence, and, finally, the control flow constraints that allow for the identification of all the specificities for pattern execution. Even at conceptual levels, patterns can embody physical properties, such as standard loops indicating that the execution of the patterns should be performed in a row-by-row manner or using parallel gateways, indicating that patterns involved in their scope should be executed in a parallel way (if possible). In Pentaho Data Integration, the first aspect

involves the configuration of the "exec_per_row" property with a 'Y' (Yes), while the second one allows for the identification of the tasks covered by the gateways influence and the task that precedes them. For that, this task will be configured with the "parallel" property using the value 'Y' (Yes).

Events can also provide additional meta-data to the system parser. For example a start event configured with a "timer start event", indicates a periodic process execution based on a date or interval (for example every 24 hours). Thus, the correspondent Pentaho Data Integration "job" should be generated based on the pattern workflow context configuration.

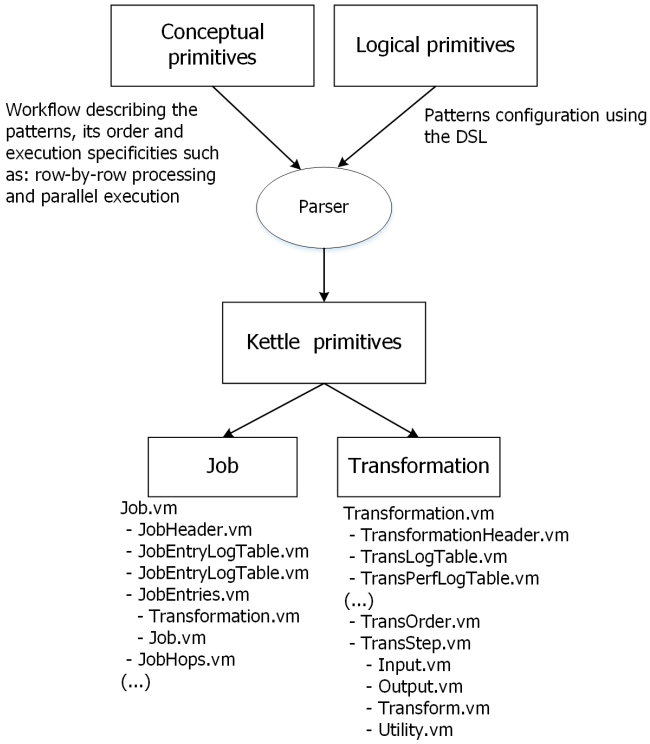


Figure 49. Physical model generation components overview.

Since the BPMN processes can be serializable using XML-based formats such as the XPDL (XML Process Definition Language)17, the configuration data can be extracted from the BPMN modelling tools used. Thus, software products can be used to interchange business process definitions without customization, providing an XML-based format representing every aspect of the BPMN process notation, including the diagram graphical description, as well pattern configuration primitives placed in BPMN "Text Annotations". For the prototype we developed, the XPDL file is used as input to the parser to provide a conceptual model description and represent the DSL code using the "Text Annotation" artefact. This provides the means for preserving details for the ETL model configuration related to process control, data transformation configuration and general ETL parameters.

[17] http://www.xpdl.org/

The conceptual primitives also provide valuable information about how tasks and patterns should be organized within the migration process. The use of BPMN sub processes allows for the representation of containers for grouping a set of related tasks. This is very useful for process organization because patterns can be represented and grouped in functional areas. The organization tree can be directly described in BPMN or be inferred based on patterns nature through the identification and workflow organization using layers describing relationships between workflow activities (patterns and ordinary BPMN activities).

a) Container Layer

```
<?xml version="1.0" encoding="UTF-8"?>

#if($numberOfTasks > 1 || !$$pattern)
      #set ($object= "Job.vm")
#else
      #set ($object = "Transformation.vm")
#end
#parse($object)
(...)
```

b) Transformation

```
  (...)
  #set ($header = "TransformationHeader.vm")
        #set ($transOrder = "TransformationOrder.vm")
        #set ($transStep = "TransformationStep.vm")

  ## File Header
  #parse("${path}$header")

  #if($pattern.patternName().equals("SCDH"))
        #set ($transStep = "SCDH/Steps.vm")
        #set ($transOrder = "SCDH/Order.vm")
        -- order
        #parse("${path}$transOrder")
        -- steps
        #parse("${path}$transStep")
  #end
```

c) SCDH transformation

```
  #set ($inputOuptut = "SCDHStepsInputOutput.vm")
  #parse($inputOutput)
  <step>
    <name>Switch &#x2f; Case - Check operation</name>
    <type>SwitchCase</type>
    <description/>
    (...)
    <cases>
      <case>
        <value>insert</value>
         <target_step>$scdh.getOuputDimension()</targe
  t_step>
      </case>
      (...)
        <value>delete</value>
        <target_step>UpdateDimension</target_step>
      </case>
    </cases>
    (...)
  </step>
```

Figure 50. Excerpt of the several template layers to support ETL patterns physical generation.

A set of transformation templates were built for encapsulating the conversion process and for transforming each pattern's internal configuration to an XML format supported by Pentaho Data Integration. This process is organized in two phases:

1) the workflow validation and the pattern configuration included in the BPMN conceptual model;
2) the generation of Pentaho Data Integration specific XML schema based on process configuration.

After the logical design phase, we used a standard transformation template for encapsulating the conversion process and providing the meanings to transform the pattern internal structure to a specific serialization format supported by the target tool. To guarantee system flexibility in relation to proprietary formats used by data migration tools, we propose a template generator language. The Apache Velocity[18] is used for describing each pattern skeleton, encapsulating standard transformation templates to generate physical instances. Thus, it is possible to maintain flexible templates and change them without affecting the logical model specification. In this process, physical details, such as connection properties, are omitted. However, the DSL we proposed could be extended for providing such details, if necessary. Due to this, we only present a semi-automatic generation of a physical model, rather than a "tool-oriented" approach. Instead, the DSL produced allows for users to choose the level of detail that they want to apply in the generation of physical models. This enhances the developing flexibility of the target system.

The templates created to support the generation physical model are built based on the tool used, according to their methodology and architecture. This means that in an ETL project, a tool expert should be engaged in this job to take advantage of desired tool particularities. The Figure 49 presents the basic instructions for generating a physical model in Pentaho Data Integration, following the primitives presented in previous modelling phases. The migration processes organize activities in "Jobs" and "Transformations". The "Transformation" construct groups data handling tasks, known as "Steps", which are used for representing the tasks responsible for extracting, transforming and loading data. "Transformation" constructs contain the most granular tasks ("steps") such as joins, unions, aggregation tasks or filtering, while "Jobs" represent higher-level constructs for controling and managing migration processes. The "Transformation" component is generated for each pattern instance, being grouped together using "Job" components. Tasks such as the transformation execution, email sending, file transferring or parallel execution of several "transformations" are only some tasks examples that only "jobs" can support. Additionally, we can also use "jobs" for grouping several "transformations" and jobs in one component, providing a way to organize ETL process in several layers (like the BPMN layers presented before).

[18] http://velocity.apache.org/

Specific process requirements can also generate additional constructs due to some specificities that Pentaho Data Integration has about process execution. For example, tasks running in parallel should be encapsulated and isolated in "job" components for ensuring process coordination. The 'On-the-fly' property is another example where the resulted pattern instance rows are passed as input to other task in-memory. When this is enabled, a more complex template will be generated to reorganize the tasks of the different Pentaho Data integration "transformations". The connection established between "Transformations" should be ensured using specific "steps", such as: "Copy rows to result" step to store records in memory and "Get rows from result" step to load rows previously stored in memory. Both "Transformation" and "Job" XML representations are described in several Apache Velocity templates, each one representing different configuration components (Figure 50), namely:

- "JobHeader.vm" and "TransformationHeader.vm", which represent the specific configuration parameters for each construct;
- "JobEntries.vm" and "TransSteps.vm", which encapsulate the internal tasks generated for supporting process specification for "jobs" and "transformations", respectively;
- "TransOrder.vm", which describes the constraints related to the execution order applied to transformation tasks.

```
<step>
    <name>$pattern.getName()</name>
    <type>
      #if($scdh.getInputType().equals("RELATIONAL"))
            TableInput
      #elseif($scdh.getInputType().equals("SPREADSHEET"))
            ExcelInput
      #end
    (...)
    </type>
</step>
```

Figure 51. Excerpt of an Apache Velocity template for a SCD transformation.

The "steps" template ("TransSteps.vm") is decomposed in other templates, encapsulating the conversion logic for each "step". Thus, the system parser only needs to know the pallet of available templates and use them based on the conversion logic. The structure of each document is statically represented using a template, while the metadata extracted from the logical model is included dynamically using the Apache Velocity primitives. For example, the Figure 50 shows an excerpt from the template logic used for converting the SCD pattern configuration for the correspondent "steps" configuration in a Pentaho Data Integration package representation.

The template for container generation (Figure 50a) is used for checking the need for the generation of "Jobs" or "Transformations" based on the requirements specified. BPMN sub processes are generated as Kettle jobs while each pattern generation process produces a transformation to support the SCD instance generation (Figure 50b). The transformation template controls the pattern that will

be generated, parsing the respective template. Based on the configuration provided, the pattern transformation template (Figure 50c) generates the instructions that will be used to materialize the model physically. At this level, all the instructions input, output and transformation tasks will be generated. Some instructions are composed of static parts, for example, the filter for identifying insert, update and delete operations; and dynamic parts, for example, the step name, fields and task configuration. For "step" generation, templates like the excerpt presented in the Figure 51 are created, in this case for checking the type of the source used for producing the correspondent output.

After physical model generation according to Pentaho Data Integration architecture, we can enrich the generated files and add complementary metadata that usually is related to the repository configuration, such as an "IP" address and an address port.

Layer 4 XML specification

```
<?xml version="1.0">
<transformation>
  <info>
<name>SCDContainer</name>
    <description/>
    (...)
  <order>
    <hop>
<from>AuditTableInput</from
>
(...)
  <step>
    <name>Delete</name>
<type>InsertUpdate</type>
(...)
<step>
    <name>Write to log
4</name>
    <type>WriteToLog</type>
(...)
```

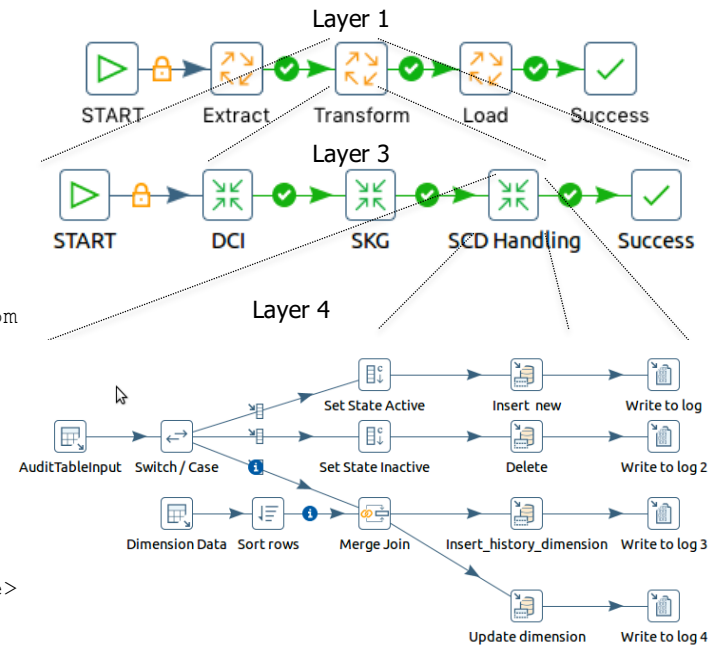


Figure 52. Physical model for a SCD instance for populating the Container dimension

In Figure 52 we can see an excerpt of the generated process using the Pentaho Data Integration modelling notation, in which we can identify three distinct layers, namely:

- the Main layer, which describes the three ETL main stages, each one encapsulating the respective patterns;
- the Layer 3, which defines a small excerpt of the transformations needed to support the data conciliation process for the containers;
- the Layer 4, which shows the application of an SCD pattern instance (Figure 48).

When a SCD job is zoomed, a "Switch/Case" step detects the action that should be performed for each record, taking in consideration the operation value stored in the audit table. The operations are

performed using the audit table as input and the temporary history and dimension tables stored in the DSA as output. The current dimension data is also used ("Dimension Data step") for supporting the update operation. A join ("Merge Join") step is configured to find related records for storing old records in the historical dimension, and current records in the dimension table. Additionally, and based on the operation performed in each data record, specific steps are also included to control log storage (for example, the "Write to log" step). All these models are supported by specific XML schemas dynamically configured by the generator engine according to the conceptual and logical primitives. Although only an excerpt of the SCD pattern templates was presented, other specific templates were also developed for each pattern presented in ETL patterns metamodel. It is also important to mention that the physical implementation presented here is only a conceivable way to deal with patterns mapping between conceptual and physical primitives. Thus, the translation process can be adjusted according to the rules, organizational practices and choices followed by each organization.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

During the last few years many research efforts have been done to improve the design of ETL systems. These efforts were quite oriented for helping the identification of common problems and the correspondent strategies for solving them appropriately. Many are the research works carried out we can refer to support this. From the aspects approached by Vassiliadis and Simitsis (2008) and Vassiliadis et al. (2000, 2005), a design technique based on a specific ETL language and notation covering from conceptual representations to its physical implementation revealed very interesting aspects related to ETL development, in particular for data repositories description along with their interrelationships based on a set of mapping rules and attribute value transformation. Independently of the technique used in later phases, this is, in fact, a very useful practice that simplifies and clarifies several aspects related to the ETL requirements. Additionally, the mapping from conceptual to logical primitives and consequently to physical implementation using the ARKTOS tool (Vassiliadis et al., 2000) revealed important considerations that contributed to the understanding of the main problems related to the relationship between different representations over the same process. Using general purpose language to reduce learning curve by adapting UML language with specific ETL components, Trujillo and Luján-Mora (2003) tried to minimize the development efforts related to ETL development, revelling how generic languages can be adapted for specific domains. Beyond that, the research approached by Akkaoui et al. (2009, 2011, 2012, 2013) contributed to the pattern-oriented approach we developed in what concerns to the use of BPMN not only for conceptual modelling but also when integrated with a model-driven approach, while Köppen et al. (2011) presented the first approach (as far as we know) covering the use of patterns for ETL development.

Despite these efforts, ETL systems still raising a lot of challenge issues in many application areas. Their nature (and goals) make them very time-consuming and error-prone pieces of software, involving many experts from different knowledge domains in their design and implementation.

Using patterns make possible to encapsulate all these kind of strategies, providing a powerful groundwork for process validation and for the identification of the most important parts of the system we want to design and build. Like in any software development process, patterns represent a well-defined template of how solving a specific problem. They provide a set of best well-proven practices that help to reduce the complexity of ETL process development. Moreover, patterns facilitate ETL design and maintenance since they represent self-contained components that are not affected by surrounding components. This means that if we change a pattern it will not affect the behaviour of other patterns, which enhances process reuse and avoids repetitive implementations of the same process every time an ETL package is developed.

We disposed a set of templates for allowing the generation of specific pattern instances representing special-purpose solutions for problems that traditionally require the configuration of several granular tasks under a very specific and technical framework. To formalize the pattern composition, we conceived an ontology (Oliveira and Belo, 2016) – Chapter 3, with the ability to describe and categorize all the ETL patterns we propose, as well as express all the construction rules, enforcing specific constraints and providing a formal and interchangeable way to share pattern knowledge between different software applications. This ontology also allows for describing the main operational components of a pattern, covering their properties and usage restrictions in workflow context. Application cases involving slowly changing dimensions policies, data quality enchantment procedures or surrogate key pipelining strategies are just some examples of typical procedures, identified and categorized based on the experience we acquired over the years. Moreover, the tasks complexity and operators involved in a complex ETL workflow consumes considerable amount of time and computational resources that must be configured and maintained. With a pattern-oriented approach, the knowledge and best practices revealed by several works (El Akkaoui et al., 2011; El Akkaoui and Zimanyi, 2009; Köppen et al., 2011; Trujillo and Luján-Mora, 2003; Vassiliadis et al., 2005a) can be put into practice using a set of software patterns that can be applied to the entire ETL development lifecycle - from conceptual phase to its physical implementation primitives. Patterns can be used in software models, providing new levels of abstraction simplifying the initial phases of software development, especially the ones related to requirements elicitation and generation of the correspondent's conceptual models.

DW conceptual modelling is a very important and demanding activity. It contributes to clarify the identification of all actors and tasks involved with it. Inside conceptual modelling, ETL processes take the greater slice of the development time and computational resources. Thus, the development of an ETL conceptual model is clearly a great advantage in a DWS project. However, despite this, the efforts spent on ETL system modelling are often not rewarded. They are frequently discarded in favour of a more detailed logical model. Based on a BPMN approach and on a set of well-defined ETL patterns (characteristics and behaviour description), we designed and implemented a specific ETL

development process (Chapter 4), which we believe enhances the importance of building ETL conceptual models for establishing a first executable version of the system – an ETL skeleton.

The use of BPMN in ETL modelling was first explored by Akkaoui and Zimanyi (2009), which proposed also its mapping and execution through the BPEL language. Later, they reinforced their original ideas proposing to model ETL generic processes, independently from their potential support platforms (Akkaoui et al., 2011). Their approach revealed a considerable potential, particularly in what was concerned with the mapping of conceptual models in a set of execution primitives. The findings of Akkaoui et al. (2011) revealed that conceptual model specification using patterns is quite appropriate since patterns can represent with accuracy the most common ETL processes. BPMN fits well the representation of pattern usage using "sub process" activities, coordinated with specific BPMN elements that are very powerful and well known by many companies and it professionals of today. BPMN was developed for supporting process execution with constructs (essentially in its 2.0 version), allowing for the representation of workflows at the physical level with the ability to support its execution. However, one of their main flaws is related, precisely, with their expressiveness that results in several ways for representing the same scenario, resulting in ambiguous processes that are difficult to map to a set of execution primitives. In fact, this was analysed and discussed by several authors (Dijkman et al., 2008; Ou-Yang and Lin, 2008). The construction freedom that BPMN provides generates also some problems related to business processes reliability, which can result from an inconsistent identification of business requirements, mainly due to the methods or model validations that are manual and error-prone.

Usually, commercial tools follow some strategies for dealing with the BPMN processes execution, providing only a subset of the BPMN constructs in some cases, specially configured to deal with different architecture requirements. For example, the BizAgi tool supports both modelling (using only a subset of BPMN constructs) and process execution based on a service-oriented architecture, representing the internal process data using structured data models based on rules for ER model's definition. This means that these tools do not support the execution of business processes in a standard way, and so they are not fitted to support specific ETL requirements.

ETL processes execution parameters also differ from traditional business processes execution platform. The conceptual approach we presented does not intend to describe how patterns will be executed, but rather establish the patterns that are used and how they are coordinated by the mechanisms provided by a BPMN specification. Additionally, pattern-based conceptual models can be exported to a model or representation that can be interpreted by a computer (several tools support BPMN processes using XPDL, for example), which is quite useful for sharing processes and pattern configurations across tools. So, the work we did at the conceptual level can be used effectively in posterior ETL development phases.

Another aspect we worked was how to avoid the BPMN model's ambiguities. In that sense, we designed and developed a specific DSL - PL4ETL (Oliveira and Belo, 2017a) – Chapter 4, which is specially oriented for describing the ETL patterns behaviour when integrated into an ETL conceptual model. The PL4ETL construction rules are derived from the ontology we already presented. They provide a flexible way for supporting pattern configuration based on the knowledge provided by the ontology. The results we got were quite satisfactory and proved the viability of the approach we followed. Using a real application case (Chapter 4), it was possible to demonstrate the application of properties used for process description at conceptual specification as a bridge for a possible physical implementation. The proposed DSL provides a way for formalizing the configuration of each ETL pattern, which allows for its posterior mapping to execution primitives independently of the architecture used. Also, it describes more naturally each part of an ETL process without having the need to program each component individually. Posteriorly, a generation engine may use the configurations, the template rules, and the coordination constraints to generate computer code that can be executed later in an ETL tool.

Posteriorly, to guarantee the consistency of the knowledge represented in the ontology, we developed a formal process specification using Alloy (Oliveira et al., 2016; Oliveira and Belo, 2017b) – Chapter 3, which provided us an automatic way for analysis and searching of false assertions through the generation of inconsistent scenarios. The complexity of the knowledge involved and the application of each pattern to specific contexts was supported using a declarative language for problem structural modelling and validation, which allows for validating previous specifications, and checking their structures and behaviour specifications using a validation engine for searching counterexamples. Using Alloy, we can develop transformation templates for translating ETL pattern configurations to a corresponding format that can be interpreted by an ETL implementation tool. All this supports pattern reusability across several systems, and so contributes to improving system robustness.

Existent data migration tools provide strong constructs covering several common transformations used in data migration processes. Recognizing the value and the potential of their frameworks, a specific parser was developed, using the pattern configuration, the template rules, and the coordination constraints to generate specific code that can be executed later in the selected ETL tool. However, they are very specific tools. They require very specialized staff having to know tool architecture specificities. The developed parser translates both the BPMN conceptual model and associated DSL configuration to an ETL skeleton built accordingly to the requirements of the target ETL tool (Pentaho Data Integration).

In brief, we can say that BPMN provides a very useful and practical way for coordinating tasks in a workflow environment, where each task represents a transformation component specially configured by the DSL. Each pattern behaviour can be generalized, which simplifies processes development and

facilitates communication between non-technical users, without compromising its mapping to a set of execution primitives. Also, the developed parser provides an automatic mapping between the conceptual and logical specifications to physical instances, which have the possibility to be executed directly. The rules and workflow logic is supported by transformation templates following a model-to-text technique for generating ETL systems code without changing the base metamodel. Due to its independence and flexibility, we believe that an ETL pattern development approach can significantly increase ETL systems quality, reduce their functional and operational errors, and decrease planning and implementation costs.

5.2 Future Work

Although the objectives planned for this work have been accomplished, several research lines can be followed for maturing and improving the pattern-oriented approach we proposed and discussed in this thesis. In order to extend and enrich the ETL patterns ontology from a practical perspective, a set of tests should be conducted to study the feasibility of the approach using different real-world scenarios in different application areas. The enrichment of the ETL patterns family is planned to support the development of the migration processes based on the needs of very specific areas. This research line is quite interesting since it will provide a more widely vision over the several areas in which the migration processes are commonly used and applied. Additionally, it is also very important to study the latest trends related to the ETL application, testing the adequacy of the presented framework to attend specific methodologies, such as the use of real-time ETL systems, or even considering the new architectures that are emerging for handling data and supporting decision-making processes.

The ETL systems specification using Alloy is also an important research aspect we intend to continue following in other research initiatives. This work only presented the first attempt for specifying formally ETL patterns using Alloy based on the specification of static patterns. At short term, a complete Alloy specification for ETL patterns is planned, particularly in what is concerned with the formalization of behaviours, assertion checking, and exception and error handling scenarios. A complete validation engine is also planned for translating pattern configurations made in PL4ETL to Alloy models, in order to provide to ETL designers a way for seamlessly and automatically checking the model's consistency.

A range of software components should also be studied for supporting data migration processes implementation and providing a more standard framework to deal with the development of ETL physical templates. To do this, it will be necessary to identify and understand each ETL tool characteristics and understand how they can receive the specification delivered by the ETL physical model. Additionally, a tool to support the entire pattern-oriented approach for ETL development can also be implemented, covering all the stages, ranging from ETL conceptual modelling, using BPMN

constructs, to the use of DSL configuration for conceptual model's enrichment and the generation of the correspondent physical model. All development stages can be supported by a validation engine ruled by Alloy specifications, for checking model consistency and the template primitives for controlling the generation of physical instances.

At long-term, a new research area covering semantic ETL approaches and computer-agent based systems may also be established for improving and complementing several aspects of the proposed approach, especially in terms of pattern validation and semantics. Semantic data (Simitsis et al., 2010) can be used for discovering and identifying source objects, and their metadata, limitations, and inconsistencies. This will help the development of conceptual models and can be used for process validation. Particularly, the use of software agents for ETL systems was already explored by Belo (2000) and Gomes et al. (2016). These works revealed some initial guidelines for implementing ETL processes based in a cooperative community of opportunistic agents. Thus, a self-organizing general purpose ETL system can be designed and implemented, which means that patterns definition, their scheduling, and execution, can be established in a dynamical manner. Being software agents, ETL patterns can be used autonomously, being activated according to given set of well-defined operational constraints and based on a specific working order. Something that we can call as "ETL as a service".

Bibliography

Alexander, C., Ishikawa, S., Silverstein, M., 1977. A Pattern Language: Towns, Buildings, Construction, Structure. Oxford University Press. https://doi.org/10.2307/1574526

Anastasakis, K., Bordbar, B., Georg, G., Ray, I., 2010. On challenges of model transformation from UML to Alloy. Softw. Syst. Model. 9, 69–86. https://doi.org/10.1007/s10270-008-0110-3

Appleton, B., 2000. Patterns and Software: Essential Concepts and Terminology. Object Mag. Online 3, 1.

Ariyachandra, T., Watson, H.J., 2006. Which Data Warehouse Architecture Is Most Successful? Bus. Intell. J. 11, 4–6. https://doi.org/10.1145/1400181.1400213

Belo, O., 2000. Gathering the Right Information at the Right Time, in: Filipe, J. (Ed.), Enterprise Information Systems. Springer Netherlands, Dordrecht, pp. 139–145. https://doi.org/10.1007/978-94-015-9518-6_14

Belo, O., Cuzzocrea, A., Oliveira, B., 2014. Modeling and Supporting ETL Processes via a Pattern-Oriented, Task-Reusable Framework, in: IEEE 26th International Conference on Tools with Artificial Intelligence. IEEE, Limassol, Cyprus, pp. 960–966. https://doi.org/10.1109/ICTAI.2014.145

Belo, O., Gomes, C., Oliveira, B., Marques, R., Santos, V., 2015a. Automatic Generation of ETL Physical Systems from BPMN Conceptual Models, in: Bellatreche, L., Manolopoulos, Y. (Eds.), Proceedings of the 5th International Conference on Model and Data Engineering. Springer-Verlag New York, Inc., Rhodes, Greece, pp. 239–247. https://doi.org/10.1007/978-3-319-23781-7_19

Belo, O., Oliveira, B., Lopes, C., Marques, R., Santos, V., 2015b. Using a Domain-Specific Language to Enrich ETL Schemas, in: Morzy, T., Valduriez, P., Bellatreche, L. (Eds.), New Trends in Databases and Information Systems. Springer International Publishing, Poitiers, France, pp. 28–35. https://doi.org/10.1007/978-3-319-23201-0_4

Berson, A., Smith, S.J., 1997. Data warehousing, data mining, and OLAP, Mcgraw-Hill Series on Data Warehousing and Data Management. Computing Mcgraw-Hill.

Bouman, R., Dongen, J. Van, 2009. Pentaho® Solutions: Business Intelligence and Data Warehousing with Pentaho and MySQL®, Solutions. John Wiley & Sons, Inc.

Breslin, M., 2004. Data Warehousing Battle of the Giants: Comparing the Basics of the Kimball and Inmon Models. Bus. Intell. J. 9, 6–20.

Büttner, F., Egea, M., Cabot, J., Gogolla, M., 2012. Verification of ATL transformations using transformation models and model finders, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 198–213. https://doi.org/10.1007/978-3-642-34281-3_16

Cunha, A., Garis, A., Riesco, D., 2013. Translating between Alloy specifications and UML class diagrams annotated with OCL. Softw. Syst. Model. 14, 5–25. https://doi.org/10.1007/s10270-013-0353-5

Dayal, U., Wilkinson, K., Simitsis, A., Castellanos, M., 2009. Business processes meet operational business intelligence. IEEE Data Eng. Bull 32, 35–41.

Dietrich, J., Elgar, C., 2007. Towards a web of patterns. Web Semant. Sci. Serv. Agents World Wide Web 5, 108–116. https://doi.org/10.1016/j.websem.2006.11.007

Dijkman, R.M., Dumas, M., Ouyang, C., 2008. Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. 50, 1281–1294. https://doi.org/https://doi.org/10.1016/j.infsof.2008.02.006

El Akkaoui, Z., Mazón, J.-N.N., Vaisman, A., Zimányi, E., Akkaoui, Z. El, Mazón, J.-N.N., Vaisman, A., Zimányi, E., 2012. BPMN-Based Conceptual Modeling of ETL Processes. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 7448, 1–14. https://doi.org/10.1007/978-3-642-32584-7_1

El Akkaoui, Z., Zimanyi, E., 2009. Defining ETL worfklows using BPMN and BPEL, in: Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP. ACM, Hong Kong, China, pp. 41–48. https://doi.org/10.1145/1651291.1651299

El Akkaoui, Z., Zimànyi, E., Mazón, J.-N., Trujillo, J., 2011. A Model-driven Framework for ETL Process Development, in: Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, DOLAP '11. ACM, New York, NY, USA, pp. 45–52. https://doi.org/10.1145/2064676.2064685

Frias, M.F., Aguirre, N.M., Pombo, C.G.L., Baum, G.A., Maibaum, T.S.E., 2005. Reasoning about static and dynamic properties in alloy: A purely relational approach. ACM Trans. Softw. Eng. Methodol. 14, 478–526. https://doi.org/10.1145/1101815.1101819

Gabriel, R.P., 1998. Patterns of Software: Tales from the Software Community. Oxford University Press.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1994. Design Patterns: Elements of Reusable Object-oriented Software, Addison-We. ed. Pearson Education.

Gomes, N., Oliveira, B., Belo, O., 2016. Modeling Agents Working on ETL Processes, in: Demazeau, YvesIto, T., Bajo, J., Escalona, M.J. (Eds.), Advances in Practical Applications of Scalable Multi-Agent Systems. Springer International Publishing, Sevilla, Spain, pp. 265–268. https://doi.org/10.1007/978-3-319-39324-7

Henninger, S., Ashokkumar, P., 2006. An ontology-based metamodel for software patterns, in: And, K.Z., And, G.S., Visaggio, G. (Eds.), 18th International Conference on Software Engineering and Knowledge Engineering, SEKE 2006. San Francisco, USA, pp. 327–330.

Horridge, M., 2012. protégé-owl api [WWW Document]. Res. Stanford Cent. Biomed. Informatics. URL http://protege.stanford.edu/plugins/owl/api/ (accessed 3.20.17).

Inmon, W.H., 2005. Building the data warehouse, Fourth. ed, Wiley technology publishing. Wiley.

Jackson, D., 2012. Software Abstractions: Logic, Language, and Analysis. MIT Press.

Jiang, L., Cai, H., Xu, B., 2010. A domain ontology approach in the ETL process of data warehousing, in: IEEE International Conference on E-Business Engineering, ICEBE 2010. pp. 30–35. https://doi.org/10.1109/ICEBE.2010.36

Kazman, R., Schmid, K., Nielsen, C.B., Klein, J., 2013. Understanding Patterns for System of Systems Integration, in: 8th International Conference on System of Systems Engineering (SoSE). Maui, HI, USA, pp. 141–146. https://doi.org/10.1109/SYSoSE.2013.6575257

Kimball, R., Caserta, J., 2004. The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley & Sons, Inc.

Kimball, R., Ross, M., 2013. The Data Warehouse Toolkit, The Definitive Guide to Dimensional Modeling, Third. ed, Vasa. John Wiley & Sons. https://doi.org/10.1145/945721.945741

Kimball, R., Ross, M., 2002. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, Second. ed. Wiley.

Kleppe, A.G., Warmer, J.B., Bast, W., 2003. MDA Explained: The Model Driven Architecture : Practice and Promise. Addison-Wesley Longman Publishing Co., Inc.

Köppen, V., Brüggemann, B., Berendt, B., 2011. Designing Data Integration: The ETL Pattern Approach. Eur. J. Informatics Prof. 13, 49–55.

Kuhlmann, M., Gogolla, M., 2012. From UML and OCL to relational logic and back, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 415–431. https://doi.org/10.1007/978-3-642-33666-9_27

Losavio, F., 2002. Standard quality model to design software architecture. J. Object Technol. 1, 165–178. https://doi.org/10.5381/jot.2002.1.1.c1

Losavio, F., Chirinos, L., Pérez, M.A., 2001. Quality Models to Design Software Architectures, in: Technology of Object-Oriented Languages and Systems. TOOLS 38. IEEE Computer Society, Zurich, pp. 123–135. https://doi.org/10.1109/TOOLS.2001.911761

Luján-Mora, S., Trujillo, J., 2002. Extending UML for Multidimensional Modeling, in: Jézéquel, J.-M., and Hussmann, H., and Cook, S. (Eds.), Proc. of the 5th International Conference on the Unified Modeling Language (UML 2002). Springer Berlin Heidelberg, resden, Germany, pp. 290–304. https://doi.org/10.1007/3-540-45800-X_23

Luján-Mora, S., Vassiliadis, P., Trujillo, J., Lujan-Mora, S., Vassiliadis, P., Trujillo, J., 2004. Data mapping diagrams for data warehouse design with UML, in: Conceptual Modeling - ER 2004, 23rd International Conference on Conceptual Modeling. Springer, Shanghai, pp. 191–204. https://doi.org/10.1007/978-3-540-30464-7_16

Macedo, N., Cunha, A., 2014. Least-change bidirectional model transformation with QVT-R and ATL. Softw. Syst. Model. 15, 783--810. https://doi.org/10.1007/s10270-014-0437-x

McGuinness, D.L., van Harmelen, F., 2004. OWL Web Ontology Language Overview, W3C recommendation 10.2004-03. https://doi.org/10.1145/1295289.1295290

McGuinness, D.L., Wright, J.R., 1998. Conceptual modelling for configuration: A description logic-based approach. Artif. Intell. Eng. Des. Anal. Manuf. 12, 333–344. https://doi.org/10.1017/S089006049812406X

Mellor, S.J., Technology, P., Clark, A.N., London, C., 2003. Model-Driven Development. IEEE Softw. 20, 14–18. https://doi.org/10.1145/1753846.1754166

Michael V., M., Zhiping, W., 2006. A framework for data warehouse refresh policies. Decis. Support Syst. 42, 121–143. https://doi.org/10.1016/j.dss.2004.11.002

Muehlen, M. zur, Ho, D.T., 2008. Service Process Innovation: A Case Study of BPMN in Practice, in: Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008). IEEE, Waikoloa, USA, pp. 372–372. https://doi.org/10.1109/HICSS.2008.388

Muñoz, L., Mazón, J.-N., Trujillo, J., 2009. Automatic Generation of ETL Processes from Conceptual Models, in: Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP, DOLAP '09. ACM, New York, pp. 33–40. https://doi.org/10.1145/1651291.1651298

Noy, N., McGuinness, D., 2001. Ontology development 101: A guide to creating your first ontology. Development 32, 1–25. https://doi.org/10.1016/j.artmed.2004.01.014

Oliveira, B., and Santos, V., and Belo, O., 2013a. Pattern-based ETL conceptual modelling, in: Cuzzocrea A., M.S. (Ed.), Model and Data Engineering: Third International Conference. Springer Berlin Heidelberg, Amantea, Italy, pp. 237–248. https://doi.org/10.1007/978-3-642-41366-7_20

Oliveira, B., Belo, O., 2017a. On the specification of extract, transform, and load patterns behavior: A domain-specific language approach. Expert Syst. Spec. issue Knowl. Discov. Bus. Intell. e12168 34. https://doi.org/10.1111/exsy.12168

Oliveira, B., Belo, O., 2017b. Validating ETL Patterns Feasibility using Alloy, in: Bernardino, J., Quix, C., Filipe, J. (Eds.), Proceedings of the 6th International Conference on Data Science, Technology and Applications. SCITEPRESS - Science and Technology Publications, Madrid, pp. 200–207. https://doi.org/10.5220/0006428002000207

Oliveira, B., Belo, O., 2016. An Ontology for Describing ETL Patterns Behavior, in: Francalanci, C., Helfert, M. (Eds.), DATA 2016 - Proceedings of 5th International Conference on Data Management Technologies and Applications, Lisbon, Portugal, 24-26 July, 2016. SciTePress, pp. 102–109. https://doi.org/10.5220/0005974001020109

Oliveira, B., Belo, O., 2015a. A Domain-Specific Language for ETL Patterns Specification in Data Warehousing Systems, in: Pereira, F., and Machado, P., and Costa, E., and Cardoso, A. (Eds.), 17th Portuguese Conference on Artificial Intelligence (EPIA'2015). Springer International Publishing, Coimbra,Portugal, pp. 597–602. https://doi.org/10.1007/978-3-319-23485-4_60

Oliveira, B., Belo, O., 2015b. Task Clustering on ETL Systems - A Pattern-Oriented Approach, in: Helfert, MarkusHolzinger, A., Belo, O., Francalanci, C. (Eds.), 4th International Conference on Data Management Technologies and Applications (DATA'2015). Springer International Publishing, Colmar, France. https://doi.org/10.1007/978-3-319-30162-4

Oliveira, B., Belo, O., 2014a. On the Conceptualization of ETL Patterns A Reo Approach, in: Almeida, A.M., Bernardino, J., Gomes, E.F. (Eds.), 18th International Database Engineering & Applications

Symposium (IDEAS). ACM, Porto, Portugal, pp. 348--351. https://doi.org/10.1145/2628194.2628247

Oliveira, B., Belo, O., 2014b. ETL Patterns on YAWL - Towards to the Specification of Platform-independent Data Warehousing Populating Processes, in: Hammoudi, S., Maciaszek, L.A., Cordeiro, J. (Eds.), Proceedings of the 16th International Conference on Enterprise Information Systems. SciTePress, Lisbon, Portugal, pp. 299–307. https://doi.org/10.5220/0004947302990307

Oliveira, B., Belo, O., 2014c. Modelling ETL workflows using YAWL, in: Hammoudi, Slimane and Maciaszek, Leszek and Cordeiro, J. (Ed.), 16th International Conference on Enterprise Information Systems (ICEIS). SCITEPRESS - Science and Technology Publications, Lda, Lisbon, Portugal, pp. 299–307. https://doi.org/10.5220/0004947302990307

Oliveira, B., Belo, O., 2013a. Using Reo on ETL Conceptual Modelling - A First Approach, in: Cuzzocrea, I.S. and L.B. and A. (Ed.), Proceedings of the Sixteenth International Workshop on Data Warehousing and Olap, Dolap 2013. ACM, San Francisco, California, USA, pp. 55–60. https://doi.org/10.1145/2513190.2513202

Oliveira, B., Belo, O., 2013b. ETL Standard Processes Modelling - A Novel BPMN Approach, in: Hammoudi, S and Maciaszek, L and Cordeiro, J and Dietz, J. (Ed.), Proceedings of the 15th International Conference on Enterprise Information Systems. SciTePress - Science and and Technology Publications, ESEO, Angers, France, pp. 120–127. https://doi.org/10.5220/0004418301200127

Oliveira, B., Belo, O., 2013. Approaching ETL conceptual modelling and validation using BPMN and BPEL, in: And, M.H., And, C.F., Filipe, J. (Eds.), DATA 2013 - Proceedings of the 2nd International Conference on Data Technologies and Applications. SciTePress, Reykjavík,Iceland, pp. 191–198.

Oliveira, B., Belo, O., 2012. BPMN Patterns for ETL Conceptual Modelling and Validation, in: Chen, L., Felfernig, A., Liu, J., Raś, Z.W. (Eds.), Foundations of Intelligent Systems: 20th International Symposium, ISMIS 2012. Lecture Notes in Computer Science, Vol 7661. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 445–454. https://doi.org/10.1007/978-3-642-34624-8_50

Oliveira, B., Belo, O., Cuzzocrea, A., 2014. A pattern-oriented approach for supporting ETL conceptual modelling and its YAWL-based implementation, in: And, M.H., And, A.H., And, O.B., Francalanci, C. (Eds.), Proceedings of 3rd International Conference on Data Management Technologies and Applications. SciTePress, Vienna, Austria, pp. 408–415.

Oliveira, B., Belo, O., Macedo, N., 2016. Towards a Formal Validation of ETL Patterns Behaviour, in:

Bellatreche, L., Pastor, Ó., Almendros Jiménez, J.M., Aït-Ameur, Y. (Eds.), Model and Data Engineering: 6th International Conference, MEDI 2016, Almería, Spain, September 21-23, 2016, Proceedings. Springer International Publishing, Cham, pp. 156–165. https://doi.org/10.1007/978-3-319-45547-1_13

Oliveira, B., Santos, V., Belo, O., 2013b. Processing XML with Java - A Performance Benchmark. Int. J. New Comput. Archit. their Appl. 3, 72–85.

Oliveira, B., Santos, V., Gomes, C., Marques, R., Belo, O., 2015. Conceptual-physical bridging - From BPMN models to physical implementations on kettle, in: Zugal, F.D. and S. (Ed.), Proceedings of the BPM Demo Session 2015 Co-Located with the 13th International Conference on Business Process Management (BPM 2015). CEUR-WS.org, Innsbruck, Austria, pp. 55–59.

Ou-Yang, C., Lin, Y.D., 2008. BPMN-based business process model feasibility analysis: a petri net approach. Int. J. Prod. Res. 46, 3763–3781. https://doi.org/10.1080/00207540701199677

Ouyang, C., Dumas, M., Hofstede, Ter, H.M., A., van der Aalst, W.M.., 2007. Pattern-based translation of BPMN process models to BPEL web services. Int. J. Web Serv. Res. 5, 42–62.

Rahm, E., Do, H., 2000. Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. 23, 3–13. https://doi.org/10.1145/1317331.1317341

Regis, G., Cornejo, C., Gutiérrez Brida, S., Politano, M., Raverta, F., Ponzio, P., Aguirre, N., Galeotti, J.P., Frias, M., 2017. DynAlloy Analyzer: A Tool for the Specification and Analysis of Alloy Models with Dynamic Behaviour, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017. ACM, New York, NY, USA, pp. 969–973. https://doi.org/10.1145/3106237.3122826

Salingaros, N.A., 2000. The structure of pattern languages. arq Archit. Res. Q. 4, 149–161. https://doi.org/10.1017/S1359135500002591

Sametinger, J., 1997. Software Engineering with Reusable Components. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-03345-6

Santos, V., Belo, O., 2011. No need to type slowly changing dimensions, in: Miguel Baptista Nunes, P.I., Powell, P. (Eds.), Proceedings of the IADIS International Conference Information Systems. Curran Associates, Inc., Avila, Spain.

Scacchi, W., 2002. Encyclopedia of Software Engineering, Encyclopedia of Software Engineering. John Wiley & Sons, Inc., Hoboken, NJ, USA. https://doi.org/10.1002/0471028959

Simitsis, A., Skoutas, D., Castellanos, M., 2010. Representation of conceptual ETL designs in natural

language using Semantic Web technology. Data Knowl. Eng. 69, 96–115. https://doi.org/10.1016/j.datak.2009.08.009

Simitsis, A., Vassiliadis, P., 2008. A method for the mapping of conceptual designs to logical blueprints for ETL processes. Decis. Support Syst. 45, 22–40. https://doi.org/10.1016/j.dss.2006.12.002

Simitsis, A., Vassiliadis, P., 2003. A Methodology for the Conceptual Modeling of ETL Processes, in: Eder, J., Missikoff, M. (Eds.), CAiSE'03: Proceedings of the 15th International Conference on Advanced Information Systems Engineering. Springer-Verlag, Klagenfurt, Austria, pp. 305–316.

Skoutas, D., Simitsis, A., 2007. Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. Int. J. Semant. Web Inf. Syst. 3, 1–24. https://doi.org/10.4018/jswis.2007100101

Skoutas, D., Simitsis, A., 2006. Designing ETL processes using semantic web technologies, in: Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP - DOLAP '06. ACM Press, New York, New York, USA, p. 67. https://doi.org/10.1145/1183512.1183526

Skoutas, D., Simitsis, A., Sellis, T., 2009. Ontology-Driven Conceptual Design of ETL Processes Using Graph Transformations, in: Journal on Data Semantics XIII. pp. 120–146. https://doi.org/10.1007/978-3-642-03098-7_5

Sommerville, I., 2004. Software engineering: Seventh Edition, 7th ed, Engineering. Pearson Addison Wesley.

Team, M.P.& P., 2009. Microsoft Application Architecture Guide (Patterns & Practices). Microsoft Press.

Trujillo, J., Luján-Mora, S., 2003. A UML Based Approach for Modeling ETL Processes in Data Warehouses, in: Song, I.-Y., and Liddle, S.W., and Ling, T.-W., and Scheuermann, P. (Eds.), International Conference on Conceptual Modeling. Springer Berlin Heidelberg, Chicago, USA, pp. 307–320. https://doi.org/10.1007/978-3-540-39648-2_25

Van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P., 2003. Workflow Patterns. Distrib. Parallel Databases 14, 5–51. https://doi.org/10.1023/A:1022883727209

Vassiliadis, P., Simitsis, A., 2009. New Trends in Data Warehousing and Data Analysis, Springer journal Annals of Information Systems, Annals of Information Systems. Springer US, Boston, MA. https://doi.org/10.1007/978-0-387-87431-9

Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., Skiadopoulos, S., 2005a. A generic and

customizable framework for the design of ETL scenarios. Inf. Syst. 30, 492–525. https://doi.org/10.1016/j.is.2004.11.002

Vassiliadis, P., Simitsis, A., Skiadopoulos, S., 2002a. Conceptual modeling for ETL processes, in: Dimitri Theodoratos, Il - Yeol Song (Eds.), Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP. ACM Press, New York, New York, USA, pp. 14–21. https://doi.org/10.1145/583890.583893

Vassiliadis, P., Simitsis, A., Skiadopoulos, S., 2002b. Conceptual modeling for ETL processes, in: Theodoratos, D., Song, I.-Y. (Eds.), Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP - DOLAP '02. ACM Press, New York, New York, USA, pp. 14–21. https://doi.org/10.1145/583890.583893

Vassiliadis, P., Simitsis, A., Skiadopoulos, S., 2002c. Modeling ETL activities as graphs, in: In Proc. 4 Th Intl. Workshop on Design and Management of Data Warehouses (DMDW). pp. 52–61.

Vassiliadis, P., Simitsis, A., Terrovitis, M., Skiadopoulos, S., 2005b. Blueprints and Measures for ETL Workflows, in: Delcambre, L., and Kop, C., and Mayr, H.C., and Mylopoulos, J., and Pastor, O. (Eds.), Proceedings of Conceptual Modeling -- ER 2005: 24th International Conference on Conceptual Modeling. Springer Berlin Heidelberg, Klagenfurt, Austria, pp. 385–400. https://doi.org/10.1007/11568322_25

Vassiliadis, P., Vagena, Z., Skiadopoulos, S., Karayannidis, N., Sellis, T., 2000. ARKTOS: A tool for data cleaning and transformation in data warehouse environments. IEEE Data Eng. Bull. 23, 42–47.

Weske, M., van der Aalst, W.M.P., Verbeek, H.M.W., 2004. Advances in business process management. Data Knowl. Eng. 50, 1–8. https://doi.org/10.1016/j.datak.2004.01.001

Wilkinson, K., Simitsis, A., Castellanos, M., Dayal, U., 2010. Leveraging Business Process Models for ETL Design, in: Parsons, J., and Saeki, M., and Shoval, P., and Woo, C., and Wand, Y. (Eds.), Conceptual Modeling -- ER 2010: 29th International Conference on Conceptual Modeling. Springer Berlin Heidelberg, Vancouver, Canada, pp. 15–30. https://doi.org/10.1007/978-3-642-16373-9_2

Zhang, Z., Wang, S., 2008. A Framework Model Study for Ontology-Driven ETL Processes, in: 2008 4th International Conference on Wireless Communications, Networking and Mobile Computing. IEEE, Dalian, China, pp. 1–4. https://doi.org/10.1109/WiCom.2008.2651