ETL Development using Patterns: A Service-Oriented Approach

Bruno Oliveira[1][a], Óscar Oliveira[1][b], Vasco Santos[1][c] and Orlando Belo[2][d]
[1]*CIICESI, School of Management and Technology, Porto Polytechnic, Felgueiras, Portugal*
[2] *ALGORITMI R&D Centre, University of Minho, Campus de Gualtar, Braga, Portugal*

Keywords: Data Warehousing Systems, ETL Patterns, ETL Component-Reuse, Services, ETL as a Service.

Abstract: Extract-Transform-Load (ETL) workflows are commonly developed using frameworks and tools that provide a set of useful pre-configured components to develop complete ETL packages. The pattern concept for ETL development is being studied as a way to simplify and improve the ETL development lifecycle. Patterns are independent composite tasks that can be changed without affecting the ETL structure. The pattern implementation reveals several challenges when used with existing ETL tools, mainly due to the monolith architectural style usually followed. The use of small and loosely-coupled components provided by the microservices architectural style can improve the way ETL patterns are used. In this paper, we present an analysis for the use of microservices for ETL application development using patterns.

1 INTRODUCTION

An Extract-Transform-Load (ETL) workflow, used as a Data Warehouse (DW) populating system, is a set of tasks that align, through complex transformation processes, operational data sources according to the target decision requirements expressed by the DW schema. Like in any workflow, tasks must be orchestrated to preserve the transformation constraints while the transformation processes are applied. Additionally, since we are dealing with a substantial amount of data, the transformation processes assume a critical role because they must guarantee both data consistency and process performance. If some of these assumptions fail, the DW adequacy is compromised.

The ETL heterogeneous nature imposes challenging problems ranging from simple access to information sources to complex strategies for conciliating data and ensure their quality. Since they are developed considering specific data requirements, each ETL system is unique, even considering the same problem domain. For that reason, the ETL development represents a critical challenge for DW implementation (Kimball and Caserta, 2004) requiring teams with specific knowledge and expertise.

Most of the ETL tools that normally support the ETL development do not provide techniques that allow reusable tasks to be applied. Also, some of the well-known design patterns for ETL development are still poorly supported (not with-standing that some tools already include standard procedures like Slowly Changing Dimension (SCD) handling (Kimball and Caserta, 2004) and the available libraries that support typical tasks like data conversion from common formats, resources accessibility like databases or files do not provide great flexibility).

In (Oliveira and Belo, 2012) a pattern-oriented approach was proposed that can be applied for ETL development, covering the conceptual, logical and physical stages. Each one of these phases represents a new detail degree covering more detailed aspects to enrich ETL skeletons, i.e., physical templates, that can be used for the generation of target executable packages to be executed using existing commercial tools.

The microservices approach for software development is emerging in several software development areas. The term is relatively new; however, this architectural style shares several

[a] https://orcid.org/0000-0001-9138-9143
[b] https://orcid.org/0000-0003-3807-7292
[c] https://orcid.org/0000-0002-3344-0753
[d] https://orcid.org/0000-0003-2157-8891

Oliveira, B., Oliveira, Ó., Santos, V. and Belo, O.
ETL Development using Patterns: A Service-Oriented Approach.
DOI: 10.5220/0007727502160222
In *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 216-222
ISBN: 978-989-758-372-8

aspects with traditional approaches used for software development. The basic idea resides in the use of small autonomous software components (services) framed within a service-oriented architecture that work together in a cohesive ecosystem.

We propose, in this paper, a new approach for ETL development considering that the microservice architectural style present several benefits and frame quite well with the pattern-oriented approach.

This paper is organized as follow. Section 2 presents the existing development approaches for ETL development, identifying the current downsides and open questions, and the approach to minimize them, namely the Pattern-oriented approach. In section 3 the main concepts related to service-oriented approach are presented and framed to the conceptual and logical design principles from the Pattern oriented approach to support ETL execution. In the last section conclusions are given and some research guidelines for future work are discussed.

2 ETL DEVELOPMENT

Despite several contributions over the years, there is still a lack of a complete ETL development methodology allowing that the effort given in the conceptual and logical phases to be effectively used in the physical implementation. The principal flaws of the existent approaches are essentially related to the use of fine-grained tasks in the ETL workflow. This typically results in big workflows hard to read, implement and maintain.

Several authors addressed this problem, providing design methodologies to simplify ETL development. Moreover, the existing commercial tools do not offer convincing approaches for ETL modelling, as they have their own notation framed with specific architectural specificities.
Vassiliadis et al. (Vassiliadis et al., 2002) presented a generic, customizable and extensible framework (Vassiliadis et al., 2005) to support ETL lifecycle (Simitsis and Vassiliadis, 2008; Vassiliadis et al., 2001)

The use of model-driven approaches was addressed by several authors (El Akkaoui et al., 2011), using models as “first-class citizens” for ETL development. These works revealed several interesting aspects not only for ETL high-level representation but also on how models can be translated to executable primitives.

Muñoz (Muñoz et al., 2009) proposed a model-driven approach to enrich ETL conceptual models, corresponding ETL processes components with model-driven artefacts, allowing for the generation of ETL processes based on Query /View/Transformation[5] (QVT). Thus, these platform independent models can be used to automatically generate final code considering a specific platform. Akkaoui et al. (El Akkaoui et al., 2011) proposed a model-driven framework for a common and integrated development strategy using vendor-independent models for ETL design. The authors used a Business Process Modelling and Language (BPMN) for ETL conceptual modelling and specific transformations were used for the automatic generation of ETL code to be interpreted by a commercial tool.

At this point and even with these interesting contributions, there is still a lack of a complete methodology that embodies the strengths both for ETL conceptual modelling and ETL execution, providing the means to translate both representations. Considering the ETL physical implementation, the ETL commercial tools such as Pentaho Data Integration or Microsoft Integration Services, provide powerful constructors that effectively help ETL development. This was already addressed using ETL patterns (Oliveira and Belo, 2012). Patterns can be understood as set of components that represent the most used processes for ETL development. They represent a reusable and partial solution to a frequent problem, enhancing the domain best practices and improve final system quality. When analysing the ETL domain several tasks such as Surrogate Key Generator (SKG), SCD and Data Quality Enhancement (DQE) (Belo et al., 2014) are recurrent procedures. The use of patterns was studied at three different development phases:

- **Conceptual Design:** Patterns are represented as BPMN sub-processes, acting as black boxes that can be configured at a later stage to produce a specific result. This allows to represent ETL processes in a simplified manner, identifying the main procedures that must be applied without implementation details;
- **Logical Design:** Based on the conceptual specification describing the main tasks and their sequence using a workflow language, at this stage, the behaviour of each pattern using a Domain-Specific Language (DSL) is specified;

[5] https://www.omg.org/spec/QVT/

- **Physical Implementation:** Considering the last previous phases, the conceptual describing the tasks and the workflow, and the logical describing the behaviour of each pattern, specific templates, i.e. skeletons, are used to generate the correspondent physical model that can be executed in existing commercial tools.

This approach covers the main ETL development phases, helping ETL designers to communicate, organize and manage all development processes.

However, inevitably ETL packages grows as new transformations are required and old ones must be removed or updated, which may revel flaws, software limitations or undesired behaviours. At certain point, the workflow is so big and complex that there is extremely difficult to maintain. Since reusability and/or scalability is hard to achieve due to tight coupling and high level of replication on the existent processes. Akkaoui et al. (El Akkaoui et al., 2013) addressed this problem extending their model-driven framework using model-to-model transformations that maintains ETL models synchronized when data sources change.

The problems related to the maintenance phase are essentially related to the nature of the physical environment. The produced conceptual models, independently of the approach taken, addresses ETL construction using high-level constructs that need to be decomposed in several low-level tasks when physical models are generated, i.e., the patterns bounded context identified and applied at more abstract levels are not preserved by physical environments.

3 SERVICE ORIENTED APPROACH FOR ETL DEVELOPMENT

The monolithic development approach, typically followed in ETL development, is easy to develop, test, deploy and share. However, real world applications are affected by business evolution and increasing data complexity, which means that the application is getting bigger and software packages are in continuous development. The monolithic approach is hard to maintain and scale (Deri, 1995) as very simple changes affect the all system leading to a new deployment.

Microservices architecture is often referred as fine-grained SOA (Newman, 2015) that follows a specific design approach. This approach is being used for software development as a componentization via services (Lewis and Fowler, 2014). This approach is based on components as small software units that can be changed independently of other components. This is possible since each service consists in processes that will always be developed and deployed together, enhancing software scability, flexibility and overall quality (Newman, 2015). The system is composed of several services, working as building blocks of this architecture, cooperating and communicating using interfaces that expose the provided functionalities. Services are developed considering a domain-driven design approach (Evans, 2003) with a natural correlation between service and context boundaries (Lewis and Fowler, 2014), leading to several advantages (Newman, 2015):

- **Technology Independency:** Instead of choosing a specific technology or language for the all system, each service can be implemented using most suitable one;
- **Error Handling:** Since the system is composed by several actors if one fails, it is simple to find the component and solve the problem without compromising the other ones;
- **Scaling:** Providing a system that can easily grow(/shrink) with dynamic load request, not requiring to upscale(/downscale) all the components at the same time;
- **Project Organization:** Since services are independent, they can be added, updated or removed without stopping the system, allowing a continuous integration and continuous deployment. The development process can be undergone with small autonomous teams responsible by specific services, reducing communication problems or incompatibilities;
- **Reusability:** Services can be reused to handle requests to respond to similar processes, enhancing system flexibility and maintenance.

Languages such as Business Process Execution language[6] (BPEL) are commonly used as mediator for all services, and task sequences are orchestrated to accomplish the target goal.
Building a service is not a straightforward task since its incorrect identification can lead to several problems. The microservices architectural style focus on two concepts (Newman, 2015): loose coupling, i.e., service isolation, and high cohesion, i.e., narrowed and bounded context services definition. All the above characteristics fits well with the ETL

[6] http://docs.oasis-open.org/wsbpel/2.0/

pattern definition discussed previously, i.e., configurable components that encapsulates logic that is needed to perform tasks, identified in the logical ETL design, intrinsically related to the DW domain.

Patterns as self-contained components should not know how the other patterns work, they just need to know how to communicate with them, enforcing independence between them and providing flexibility to the final ETL workflow implementation. Patterns can be changed without affecting the consistency of other patterns, preserving the structure of the final system. A conceptual modelling process was idealized for patterns (Oliveira and Belo, 2012), allowing for a clear separation of the coordination process (coordination layer) and the transformations applied to the data (data transformation layer). The ETL patterns aim to ensure flexibility, modularity and system evolution to support new features without affection all system. These are in fact the same key system characteristics of microservices (Dragoni et al., 2017). For that reason, microservices are evaluated to support ETL pattern-oriented approach for ETL development.

3.1 Pattern as a Service

Creating and using reconfigurable components avoid rewriting some of the most repetitive tasks that are regularly used in ETL process implementation. Several tasks, such as SKG, 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.

The pattern structure contains the pattern rules defined for supporting the operational requirements and the logic behind them. Each pattern uses input and output interfaces to communicate with ETL layers and to the data layer that supports the pattern application. The construction rules are provided by each pattern to sustain well-formed workflows, while the data from the 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.

In (Oliveira and Belo, 2013, 2012), a SOA oriented approach for ETL was proposed, in which, each pattern was represented as a web-service with specific methods responsible to invoke existing stored procedures to perform specific activity over data, leaving the complexity related to data transformation in the Database Management System and orchestration logic to BPMN or BPEL. In practice patterns acted as a bridge between the data transformation and workflow orchestration layer. Although a step forward in ETL development, this approach does not ease to extend the pattern behaviour without compromising the remaining patterns that compose the system.

Using the microservices architectural style, each pattern represents a bounded context that can be composed by nested fine-grained bounded contexts representing the pattern decomposition, creating small ecosystems that allow for pattern flexibility and extensibility.

3.2 Pattern Configuration

In (Oliveira and Belo, 2017), a Domain Specific Language (DSL) was presented to support the configuration of patterns (as defined previously). The DSL covers the requirements of each pattern category and provides a powerful way to configure behaviour. To support the language syntax specification, a set of static type statements and keywords was created to describe each language component. The language syntax and construction rules are composed of two components: a JSON Schema (Pezoa et al., 2016) and an ontology (Oliveira and Belo, 2016) that describes the main entities and their relationship within the DSL. The JSON Schema is used to validate the DSL syntax, imposing the document structure and the correctness of the language. The ontology to support the pattern description represents the knowledge base for patterns composition, covering the operational stages and providing a solid framework for pattern instances generation. Considering the ontology and the syntax rules, the configuration language can be automatically generated for each pattern, which enhances the language flexibility. Since ontology definition and language constructs are mapped, this approach guarantees that if the ontology changes, the correspondent grammar rules will continue consistent (Oliveira and Belo, 2016). The patterns relationship is derived from the class hierarchy defined in the ontology.

3.3 Patterns Communication

In a traditional ETL workflow application, components invoke one another via language-level methods or function calls. The orchestration (Mazzara and Govoni, 2005) is the most common approach to deal with ETL tasks in a workflow context. A central authority is used to rule all process invocation and handle their response. Each system

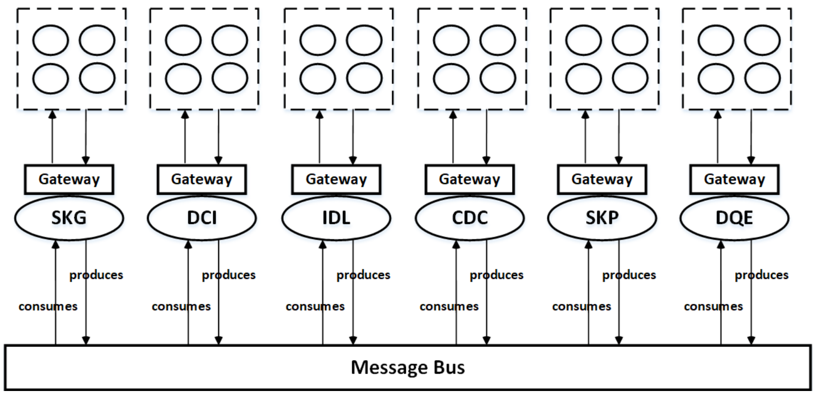

Figure 1: Architecture to support ETL patterns communication.

component receives their job, which may result in the invocation of third-party components. This approach is simple, mainly when we have several task dependencies. This approach provides a good platform to handle synchronous requests, in which the client waits for the response.

In the microservices architectural style, the choreography (Peltz, 2003) model is used for communication between each service. Services talk with each other to perform a given task or process, based on specific conditions. The interactions between multiple services are decentralized using events. As consequence, when services are invoked, the client application does not block waiting for the service response (asynchronous invocation). When comparing to orchestration, the choreography approach enhances application decoupling, improving system reliability (there isn't a single point of failure), performance (due its asynchronous nature), flexibility (services can be deployed and undeployed without affecting the remaining services) and scaling (since services can be scaled independently). However, the complexity of asynchronous calling and the breakup of process flow across several services arises several challenges for process monitoring and control.

Typically, ETL synchronous workflows are composed by long-running procedures, i.e., it returns only when the action has finished, stopping the program for the time the action takes. In the other hand, with asynchronous calls, some tasks are executed while the process flow continues to run. Figure 1 depicts a possible architecture to support ETL patterns communication considering a service-oriented approach in which several ETL patterns, namely, Surrogate Key Generator (SKG), Data Conciliation and Integration (DCI), Intensive Data Loading (IDL), Change Data Capture (CDC), Surrogate Key Pipelining (SKP) and Data Quality Enhancement (DQE). More about these patterns can be found in (Oliveira and Belo, 2016)

In this context, patterns, i.e., first tier of services, preserve their own logic provided by a set of specific finer-grain (second tier of) services within a bounded context. The pattern DSL is passed through a handler (e.g., Message Bus), to be consumed by respective service to produce the result back to the handler. All this provided by an event-based asynchronous environment.

Since services (from for first and second tier) can be replicated to handle intensive procedures, load balancers can be used to manage how the system scale.

3.4 Data Handling

The Data Staging Area (DSA) is a location (frequently a conventional database system) where data gathered from data sources remains during the patterns execution (for cleaning, surrogating, conciliating, etc.) before being loaded and integrated into the target DW (Kimball and Caserta, 2004). That way, when a transformation takes place, the operational systems normal functioning is not affected. To optimize all data transactions, the DSA is usually supported by simplified data structures. Some of this data is posteriorly deleted, while the ETL metadata can persist. For example, the mapping, dictionary, quarantine and log data is used to support ETL subsequent activities to populate the target DW.

The DSA represents a shared database to support the several ETL stages. Usually, the use of shared databases leads to highly coupled (Lewis and Fowler, 2014) systems.

Considering the SOA using patterns, for temporary data it makes sense to isolate them for each pattern service, avoiding unpredicted changes that can affect other patterns behaviour. However, several

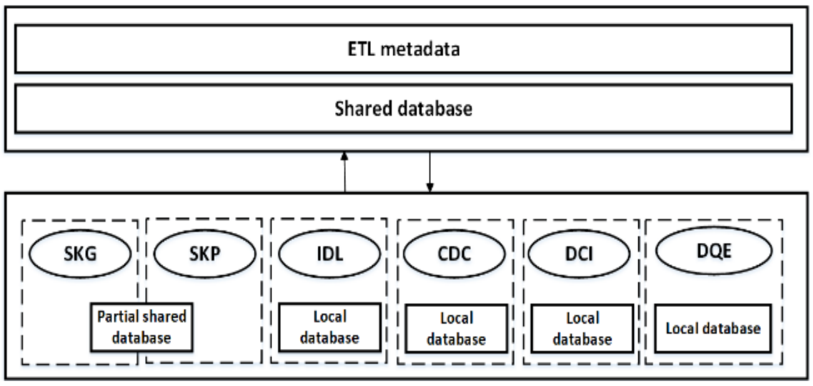

Figure 2: Data structures to preserve ETL metadata.

aspects should be considered when data is transformed by each pattern, which means the ETL designer should be able to evaluate the data storage requirements (for example to preserve process milestones for handling recovering procedures).

Thus, two approaches can be followed for temporary data, one or more databases shared between patterns (a specific Application Programming Interface (API) can be used to control the access level of each pattern), and/or a local (private) database for each pattern.

For ETL metadata, additional scenarios should be considered. The log data is primarily stored using pattern's local database, maintaining the events captured during pattern execution. Two event log types can be distinguished: local events used for pattern recovering and global event logs used to record global milestones. The pattern's local events are stored locally at pattern' context, while global events are stored in a central repository, i.e., ETL metadata (Figure 2). The Error handling metadata are stored locally in each pattern, using specific transactions identifiers to track the errors/exceptions occurred during pattern execution. Support data (for example: mapping and dictionary data) can be defined manually or can be created after some pattern execution. In both cases these "special" metadata should be shared by some patterns or by all patterns. To support the communication logic, these relationships can be, for example, maintained using a message broker with appropriated APIs to control the access to the shared data.

In software built to serve business operational needs and distributed data management, several problems can emerge related to data access and to guarantee atomicity, consistency, isolation and durability (ACID) properties. In ETL environment these problems do not affect ETL operationality since DWs are used for OnLine Analytical Processing (OLAP), i.e., they are used for complex queries to analyse data rather than process transactions.

4 CONCLUSIONS

During the last few years many research efforts have been done to improve the design of Extract, Transform and Load (ETL) systems. These efforts were quite oriented for helping the identification of common problems and the correspondent strategies for solving them appropriately. 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. In this paper, the use of composite tasks referred to as patterns were used to drive the ETL development to facilitate, improve reliability and safety to the development and maintenance actions involved in ETL systems development. In particular, the mapping from conceptual and logical design to physical primitives of patterns composition was studied under a services-oriented approach, or more specifically its finer grained approach, commonly referred to as micro-services. The architecture style fits quite well the same principles idealized for conceptual and logical design phases, as this style provide high cohesion and loose coupling concepts, enhancing scalability and system flexibility.
As future work a case study could conducted in order to validate the effectiveness of the approach presented considering a real-world scenario.

REFERENCES

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

Deri, L., 1995. Droplets : Breaking Monolithic Applications Apart. IBM Res. Rep.

Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L., 2017. Microservices: Yesterday, Today, and Tomorrow, in: Present and Ulterior Software Engineering. https://doi.org/10.1007/978-3-319-67425-4_12

El Akkaoui, Z., Zimànyi, E., Mazón, J.-N., Trujillo, J., 2013. A BPMN-Based Design and Maintenance Framework for ETL Processes. Int. J. Data Warehous. Min. 9, 46–72. https://doi.org/10.4018/jdwm. 2013070103

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, DOLAP '11. ACM Press, New York, New York, USA, p. 45. https://doi.org/10. 1145/2064676.2064685

Evans, E., 2003. Domain-Driven Design: Tackling Complexity in the Heart of Software. Folia Primatol. Int. J. Primatol. 70, 560. https://doi.org/10.1159/ 000067454

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

Lewis, J., Fowler, M., 2014. Microservices [WWW Document]. http://martinfowler.com. URL http://martinfowler.com/articles/microservices.html (accessed 2.12.19).

Mazzara, M., Govoni, S., 2005. A Case Study of Web Services Orchestration, in: Jacquet, J.-M., Picco, G. Pietro (Eds.), Coordination Models and Languages. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–16. https://doi.org/10.1007/11417019_1

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

Newman, S., 2015. Building Microservices: Designing Fine-Grained Systems, 1st ed. O'Reilly Media.

Oliveira, B., Belo, O., 2017. 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., 2016. An Ontology for Describing ETL Patterns Behavior, in: Francalanci, C., Helfert, M. (Eds.), An Ontology for Describing ETL Patterns Behavior. SciTePress, Lisboa, pp. 102–109. https://doi.org/10.5220/0005974001020109

Oliveira, B., Belo, O., 2013. Approaching ETL conceptual modelling and validation using BPMN and BPEL, DATA 2013 - Proceedings of the 2nd International Conference on Data Technologies and Applications. SciTePress, Reykjavík,Iceland. https://doi.org/10. 1007/978-3-319-62911-7

Oliveira, B., Belo, O., 2012. BPMN Patterns for ETL Conceptual Modelling and Validation. 20th Int. Symp. Methodol. Intell. Syst. Lect. Notes Artif. Intell. 7661 LNAI, 445–454. https://doi.org/10.1007/978-3-642-34624-8_50

Peltz, C., 2003. Web services orchestration and choreography. Computer (Long. Beach. Calif). https://doi.org/10.1109/MC.2003.1236471

Pezoa, F., Reutter, J.L., Suarez, F., Ugarte, M., Vrgoč, D., 2016. Foundations of JSON Schema, in: Proceedings of the 25th International Conference on World Wide Web - WWW '16. https://doi.org/10.1145/2872427.2883029

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

Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., Skiadopoulos, S., 2005. 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., 2002. 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., Vagena, Z., Skiadopoulos, S., Karayannidis, N., Sellis, T., 2001. ARKTOS: Towards the modeling, design, control and execution of ETL processes. Inf. Syst. 26, 537–561. https://doi.org/10.1016/S0306-4379(01)00039-4