ETL Patterns on YAWL
Towards to the Specification of Platform-independent Data Warehousing Populating Processes

Bruno Oliveira and Orlando Belo
ALGORITMI R&D Centre, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal

Keywords: Data Warehousing, ETL Modelling, ETL Validation and Testing, ETL Patterns, and YAWL.

Abstract: The implementation of data warehouse populating processes (ETL) is considered a complex task, not only in terms of the amount of data processed but also in the complexity of the tasks involved. The implementation and maintenance of such processes faces various design drawbacks, such as the change of business requirements, which consequently leads to adapting existing data structures and reusing existing parts of ETL system. We consider that a more abstract view of the ETL processes and its data structures is need as well as a more effective mapping to real execution primitives, providing its validation before conducting an ETL solution to its final implementation. With this work we propose the use of standard solutions, which already has proven very useful in software developing, for the implementation of standard ETL processes. In this paper we approach ETL modelling in a new perspective, using YAWL, a Workflow language, as the mean to get ETL models platform-independent.

1 INTRODUCTION

In software development reusing common software patterns to build complete software solutions is addressed by a lot of applications, programming languages and frameworks. Since early, software developers felt the need to decompose its programs in simpler ones in order to identify repetitive patterns that could be reused. All these contribute to the reduction of redundancies and foster software reuse, having a positive impact in the development time and costs of traditional software. Software patterns represent a set of more simple tasks that represent a specific set of rules that are applied in common scenarios, regardless the context that are applied. Software patterns proved to be very useful to enhance reusability, improve general quality of systems, reduce the negative impact of incomplete or incorrect software design, and minimize the impact of requirements changing. Consequently, all this contributes to reduce significantly development costs.

ETL (*Extract, Transform and Load*) processes are considered one of most difficult and time consuming tasks to be implemented in *Data Warehousing Systems* (DWS) (Kimball & Caserta 2004), being responsible for data extraction from disparate business data sources in order to transform, conform and clean data for the integration in a homogeneous data repository. The data that was integrated will be posteriorly the target for advanced data analysis tools sustaining business managers' decision-making processes. Specific decision needs are intrinsically related to specific business processes and business rules. The complexity of an ETL process is typically affected by the complexity of business processes that increases every time new business areas are integrated in a DWS. The change of business requirements difficult a lot ETL processes design and, obviously, its future maintenance. This will require adapting some parts of the process already implemented increasing system's costs. Even when we use standard solutions for DWS implementation, the specificity of business decision-making processes will lead systematically to some ETL adaptation. Moreover, it's almost impossible to find inside an organization transaction systems with the same or similar schemas. Once again, this will also lead to the development of specific ETL processes to align operational data sources with the target DWS schema (Weske et al. 2004). Furthermore, ETL modelling and implementation is often supported by proprietary tools, which dispose their own methodologies and

Oliveira B. and Belo O..
ETL Patterns on YAWL - Towards to the Specification of Platform-independent Data Warehousing Populating Processes.
DOI: 10.5220/0004947302990307
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 299-307
ISBN: 978-989-758-027-7
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

notations for ETL tasks and coordination mechanisms. All this increase the complexity of ETL implementation and maintenance, because it represents significant efforts for the ETL development, which need to understand all specificities provided by them before using them. Proprietary notations also limit communication with non-technical users. They use to be very detailed about the issues related to runtime environments. Moreover, if we need to change eventually an ETL tool, we need to spend a lot of time to rebuild data structures and tasks frequently from scratch (or almost).

In recent years, several works have presented solid and useful contributions to improve ETL processes implementation in DWS, proposing not only new notations, specially built for ETL domain, but also adapting existing general use notations, used in software development for ETL systems specification and development. Today, it's clear that ETL systems specification and development are intrinsically related to business processes and rules. Their complexity is often affected by business process, especially when new business areas are incorporated in a DWS. Changing business requirements difficult ETL design and maintenance, requiring the adaptation of some parts of the process. So far, we have been working on the identification of standard patterns that represent and characterize very common ETL tasks used for real world application scenarios - e.g. *surrogate key pipeline* (SKP), *slowly changing dimensions* (SCD) with history maintenance (SCH-HM), *change data capture* (CDC), *data quality validation* (DQV), or *intensive data loading* (IDL). We already developed and implemented some conceptual representation for such patterns, making a clear separation between coordination processes (coordination layer) and transformations applied to data (data transformation layer). Each pattern represents a proven reusable practice that can be applied in many different scenarios. Based on a set of configurable input parameters, these patterns represent specific tasks that produce an output based on their internal configuration. Other patterns involved in in the ETL system don't know how other patterns work (isolation), they just know how to communicate with them. They are autonomous. Changes on their behaviour are internal. Do not affect the consistency of the other patterns and preserve the structure of the entire ETL system.

To demonstrate the viability of this pattern-oriented approach on ETL systems development, we selected a dingle standard ETL process (IDL), using the YAWL workflow language (W M P van der Aalst & Hofstede 2003) to specify it. YAWL provides a formal but very intuitive way to represent workflows, being quite adequate to specify and validate ETL systems, at a very early stage of development, disposing powerful constructs that enable execution primitives in the definition of a workflow. So, after a brief exposure about some related work (section 2), we present a case study using YAWL (section 3), where we included some standard patterns that can be used for a complete ETL process specification. Next (section 4), we discuss a very specific pattern, IDL, as well as its respective logical mapping with YAWL and execution architecture. In section 5 we discuss the produced YAWL models, presenting the most relevant aspects of the approach we presented. Finally, we finish the paper presenting conclusions and a few guidelines for future work (section 6).

2 RELATED WORK

An ETL process is a critical component of any DWS. It requires the use of standard methodologies and models to improve project development process and maintenance. Kimball and Caserta (2004) addressed this issue providing a clear and simply methodology for guiding and supporting the entire ETL development lifecycle. After producing an initial plan, the selection of an ETL tool to support the implementation of the systems appears quite naturally. These tools helps a lot the work of developers, providing graphical notations and methodologies that allow for more accessible communication and representation of processes. Usually, these tools produce system models very detailed, considering many implementation issues - what is very useful indeed. However, often these models follow proprietary notations, and so restrict their migration to other platforms, as well as impose new learning processes to the ETL team. Taking this into consideration, many authors on the field have proposed some new ETL model specifications to minimize the impact of such "lacks". For instance, Vassiliadis et al. (2003) provided a specific notation for a complete ETL specification approaching the entire ETL development cycle, starting with the definition of more general models using specific conceptual constructs (Simitsis and Vassiliadis, 2003), mappings for logical workflow primitives (Simitsis and Vassiliadis, 2008) and finally the correspondent physical representation, supported by a tool ("Arktos") (Vassiliadis et al., 2000). On the order hand, Trujillo and Luján-Mora (2003) and Lujan-Mora et al. (2004) proposed an UML (Unified Model Language) extension for ETL modelling, showing the use of UML packages that allow the definition of ETL scenarios. More recently El-Sappagh et al. (2011) proposed an *entity-mapping diagram (*EMD) framework, consisting in a new notation and a new set of constructs for ETL conceptual modelling. In the field of general use workflow languages, Akkaoui and Zimanyi (2009) showed how to use the *Business Process Modelling Notation* (BPMN) notation to develop an ETL conceptual model and implement it using *Business Process Execution Language* (BPEL). They avoided the major drawback that usually exists between modelling conceptual specifications and their implementation. Latter, Akkaoui et al. (2011) also explored the integration of existing organizational processes with ETL processes and the definition of a meta model that formalize the workflow coordination and data transformation components. We recognize that conceptual specifications and its validation are essential, if we want to produce an effective executable instance of a model, providing its conceptualization to nontechnical users and its validation by technical users.

Previously, we had already explored the use of ETL patterns for ETL conceptual modelling (Oliveira and Belo, 2012) (Oliveira and Belo, 2013a) (Oliveira and Belo 2013b) (Oliveira and Belo, 2013c), using BPMN. We also explored how to map such abstract models to a more detailed (and executable) model using both BPMN 2.0 orchestration elements and BPEL. Latter, in other research thread, we also explored ETL pattern representation using the REO coordination language (Oliveira and Belo, 2013d), a more formal workflow language that provides an unambiguous speciation of the patterns. Although, the contributions and works that had been presented do not constitute yet a complete proposal, in order to cover all stages of ETL development.

3 ETL MODELLING USING YAWL

Nowadays, programming languages (particularly object oriented languages) and frameworks provide pre-existing components allowing for developers to build complex systems using existing patterns representing a set of tasks already included in proven solutions for many common design problems (Buschmann et al., 2007). Patterns must be context independent with the ability to approach and combine heterogeneous architectures and communicate with other patterns easily, facilitating and improve the quality of software development. Software patterns can also represent several abstraction levels, which make them very useful artefacts for the development of complex software systems.

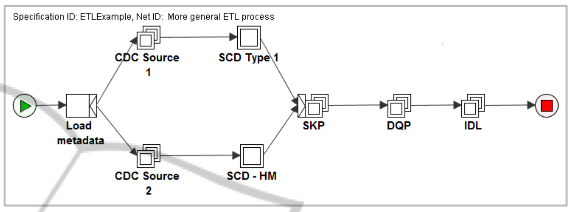

Figure 1: An Abstract ETL Representation.

We designed this work as a system especially conceived to model DWS populating processes. We consider that there is no need to “reinvent the wheel”, every time we initiate the development of an ETL system. So, our goal was to explore the existence of a design tool that provide us a set of high level patterns being capable to receive standard ETL tasks like the ones mentioned before. These tasks will free us to specify other fine grain tasks that are typically error prone and have similar characteristics, even when they are inserted in different business environment. The use of patterns simplifies communication. More specific tasks are hidden, allowing producing ETL conceptual models simpler and understandable - we should say “if we need to use a car in a trip, we just pick a car, we don’t need to develop it to use it”. Now, we are simply working in a higher level of specification, using DWS terminology (for tasks, data and control flows, and high-level specification models - DWS patterns), and well-known languages and models such BPMN or YAWL.

The use of workflow languages for the specification of ETL systems was subject of research in last few years. As we referred before, Akkaoui and Zimanyi (2009) shown that the use of *Service Oriented Architecture* (SOA) with BPMN and BPEL can be successfully applied to more specific workflow processes, like ETL. In fact, workflow languages are quite interesting, because they provide in most cases understandable notations, which contributes to improve communication at ETL conceptualization stage, separation of concerns between operations orchestration and data involved, and the necessary meaning to instantiate more

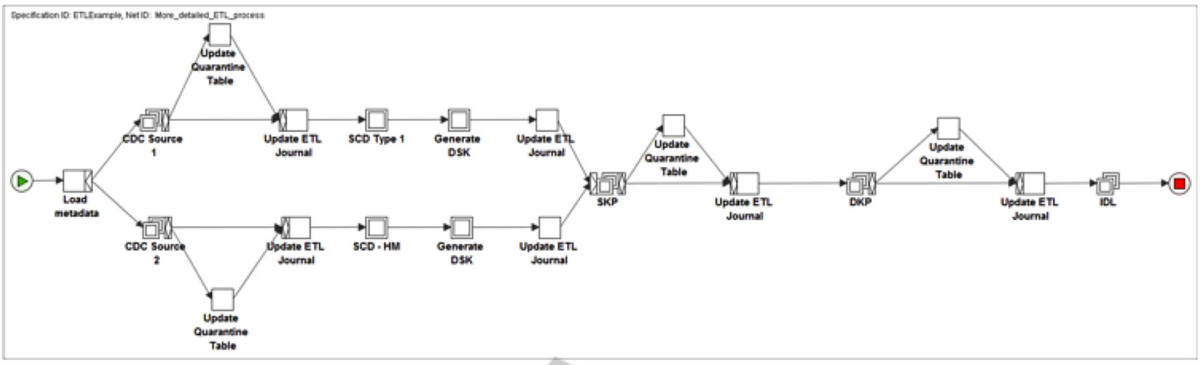

Figure 2: A more detailed view of the ETL process represented previously in figure 1.

conceptual models to execution primitives. YAWL provides all the referred characteristics, using formalisms inherited by Petri Nets concepts and workflow patterns. YAWL also extended additional features, such multiple instance support and cancellation patterns, contributing to a more detailed modelling of complex workflows. These formalisms make the language more concrete, which reduce ambiguities, making it much more “implementable”. Besides, YAWL supports also exception handle, dynamic workflows, declarative workflows and a powerful and simple notation to represent all of its constructs.

To demonstrate the use of YAWL on an ETL specification, let us consider a process for an ordinary populating process of a traditional and simple sales multidimensional schema (figure 1). In this specification, we used a set of ETL patterns that simplify the entire representation of the ETL process. Each pattern used (CDC, SCD-HM, SCD Type 1, SKP, DQP and IDL) represents itself a set of pre-defined tasks that are included in separated nets or processes. Using a atomic task configuring all the initial parameters, the process starts by loading metadata that will support the execution infrastructure, e.g. connection strings to access sources’ transaction logs files, target connection strings for holding data in the *Data Staging Area* (DSA), mapping tables, quarantine tables, and data quality procedures for IDL. Next, using an AND-Split task, two CDC processes are executed simultaneously to extract modified/new data. These two tasks are classified as Multiple Instance Composite Tasks, which are specified in a separate net allowing for the execution of multiple instances of a CDC composite task in a concurrent manner. Each source generates a specific process instance in order to improve process performance. For source 1 and source 2 we specified two different types of SCD procedures: SCD type 1 and SCD with history maintenance, respectively. Both patterns need to be classified as composite tasks since they include another YAWL net that owns elements for representing pattern behaviour. For joining the two flows an AND-Join was specified for the SKP pattern. After both flows are finished, the SKP process initiates for populating the fact table with dimensional surrogate keys generated by previous operations (we assumed that SCD patterns also includes a surrogate key generator pattern). The SKP pattern is defined as a multiple instance composite task, since for optimal performance records can be spitted by several concurrent tasks in order to populate the fact table. The DQP pattern represents a set of specific clean and conforming data tasks that can be applied concurrently dividing the original data set by all the instances generated. After records located in DSA are prepared to be loaded into the DWS through a set of IDL pattern instances that establishes the correspondences between temporary storage structures and data warehouse’s structures. Some patterns of figure 1 integrate other utility patterns in their own structure - figure 2 shows that presenting an extended version of the ETL process of figure 1. In figure 2, we represented the support patterns used previously in each pattern specification. For the two first CDC patterns, it was included the task ‘Update Quarantine Table’ and ‘Update ETL Journal’. The two flows coming from each task are launched using an OR-Split joined by an OR-Join type, because the process will activate most of these flows but not necessarily both will be activated.

If a CDC pattern finds bad records on source extraction, they will be handled by an exception task that put non-conforming records into a temporary data structure to be analysed latter. In both cases, the atomic task ‘Update ETL Journal’ will be responsible to record events into the ETL log journal. Next, it was defined two Generate DSK

(*Dimensional Surrogate Key*) patterns and two Update ETL Journal tasks, having also an exception case if necessary. The possibility to decompose patterns in sub patterns is quite useful. It provides a more clear vision of all tasks included. Original patterns still have their original configuration. However its inclusion in the final process will be composed by a set of support patterns or tasks, which will be integrated in the final process. For example, if a record is invalid on a SKP pattern, it's because something wrong happened in the last tasks. So, in spite of putting that record in a quarantine table we can cancel all (or part of the) processes in execution. This type of specification produces detailed models (figure 2), in which specific tasks are managed according ETL processes requirements. YAWL has mechanisms to manage that in a process flow.

The C-YAWL models provide a way to specify configurable tasks to establish variants in some parts of a process, adapting it for different scenarios. Thus, it's possible to identify specific parts of a process that are shared by all variants and parts for other particular variants. Configurable tasks as its input and output flows can be blocked or hided from the remaining tasks. If a flow is blocked, is no longer possible to execute the task via that port. If this happens, a new individualized YAWL model is automatically generated without the blocked tasks, being the behaviour of a hide tasks removed. This is particularly useful for ETL processes, because patterns can have configurable parts, giving more flexibility in its specification, not only to meet specific requirements but also to add or remove tasks from an ETL pattern. Additionally, YAWL provides cancellation services, which are useful for the identification of a set of tasks, conditions and flow relations (join tasks) before the completion of the tasks. For example, let us consider an excerpt of a SCD-HM pattern (figure 3). In this example, target records are processed in groups, which means that the process is not dealing with row-by-row operations, but with sets. Four sets of records are spitted by each flow according to the specified operation: 'Insert', 'Update', or 'Delete', and in case of a 'non-operation', they are labelled as 'Unknown'. When an 'Unknown' operation is finished, the task will cancel automatically the other marked operations. A pattern-oriented approach for ETL modelling is particularly useful in real world applications. Using patterns we only need to represent their behaviour. Encapsulating a considerable set of elementary operations in a single pattern we simplify ETL design and modelling. With the use of well-proven patterns, we can also reduce implementation errors. Moreover, mappings for the representation of each pattern in execution primitives are also clear and unambiguous, contributing for a successful prior validation of an ETL model.

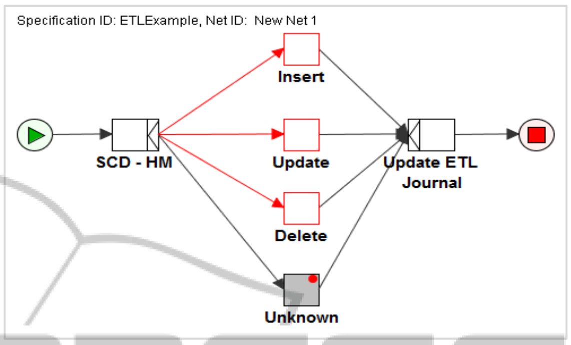

Figure 3: Simple example from a SCD with history maintenance pattern.

4 AN INTENSIVE DATA LOADING PATTERN

An IDL task is very common in ETL. It serves to load data into the data warehouse. For our scenario (figure 1 and figure 2) all data is cleaned and transformed inside the DSA. At the end of this process, the IDL pattern process just materializes conformed rows into the target data structure. When records arrive to this stage, they are already full prepared to be loaded into the data warehouse. Then, the IDL pattern will start to populate table dimensions and then the fact tables. The IDL pattern (figure 4) represents data at the finer grain level that the grain used in previous examples (Figure 1 and Figure 2). In Figure 4 we are demonstrating the pattern in terms of "record-by-record" processing, which will be executed ideally with multiple instances, one for each dimension. The process starts by reading the metadata of a specific dimension, including the connection string for the source and target dimension, and an entry log with all operation made for the dimension's data. For the first repetitive block, all dimensions' entry logs for a particular instance are read one by one and for each one is applied a conditional expression in order to identify if this particular log entry refers to an insertion or a deletion on a specific dimension table. For the delete operation, the target record is just eliminated, and for the insertion operation the specific record must be loaded and inserted in target

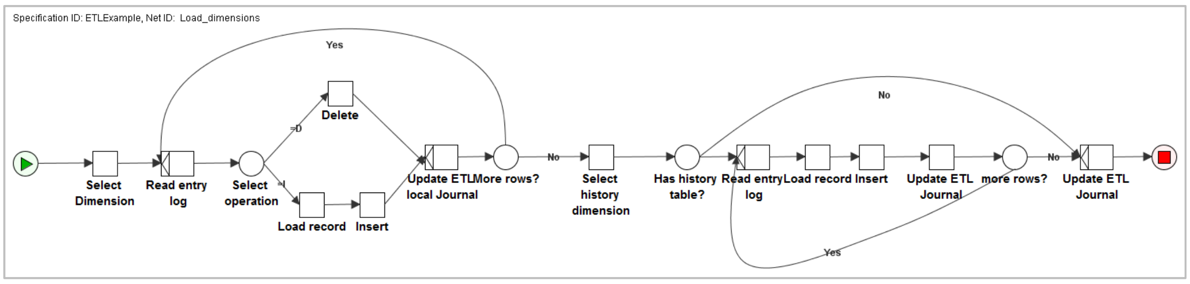

Figure 4: An IDL pattern for delivering dimension tables.

DW repository. After processing all records for a dimension, it's time to reflect changes made in records and inserted previously.

For best performance, we did not consider the update operation. If a record needs to be updated, then first is deleted and next inserted again in the dimension table. If we pretend to maintain historic data for each change made, an historic table must be maintained and updated to track all occurrences made in the correspondent dimension table. For each repetitive block, this pattern maintains locally an ETL log file. Thus, the ETL system can recover from the last consistent state in case of a system fault. After populate all dimension tables, the IDL pattern populates fact tables. In the DSA, fact records are already with the same structure that is used in the DWS, and the SKP pattern already have all transactional keys processed with the correspondence surrogate key. Figure 5 presents an excerpt of the IDL pattern, focusing on the delivering of the target fact table.

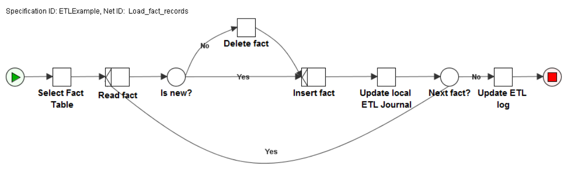

Figure 5: An IDL pattern excerpt for delivering fact tables.

In this pattern records are separated in two groups: new fact records and "updated" fact records (in case of a cumulative fact table). Typically for dimension tables, the amount of data to be delivered hardly represents a bottleneck. However, for fact records, the considerable amount of data to deliver can be a problem. As Kimball and Caserta (2004) referred, many times it's better to delete records that need to be updated, and then load the new version of those records. Thus, for the updated records, the IDL pattern first removes the old record and then inserts the new fact version. For fault recovery reasons, the IDL pattern also maintains locally an ETL journal.

YAWL provides a service-oriented architecture for the execution of the specified processes, disposing a great flexibility to the YAWL specification and providing the separation of concerns between workflows coordination and data transformations. Tasks can easily be assigned to human participants, Web Services, Java source code, or external applications. Additionally, it provides a powerful data perspective supported by powerful standards: XML for data representation, and XPath (XML Path Language) and XQuery (An XML Query Language) for data extraction and manipulation. Data inside YAWL support both net level data and task level data. The net level data is stored using net variables to be accessed and modified by net tasks. Task variables are only accessed or modified by its own or by an instance of it. Additionally, YAWL provides a strong pallet data types for process variables, allowing for the definition of complex data types using XML. This lets to define strong data types in order to support metadata for the definition of ETL patterns. In fact, the nature of each task fits our purposes for ETL pattern metadata usage. YAWL tasks support input variables where data are written for task processing, and output variables where data are read as a result of task output. For validation purposes, we propose the use of SOA as a bridge between data storage and process orchestration layers. Each atomic task is associated with a web service method that is responsible to invoke the most suitable data layer procedure (for example a stored procedure) in order to perform a specific activity. The way that transformations are applied is completely independent from the process layer. A YAWL process is capable to capture the necessary metadata for process execution, providing tasks orchestration and representing and controlling metadata involved between the tasks involved. The use of XML turns the process much more flexible, and more expressive when applied to the execution primitives.

YAWL also provides a selection service that is part of the Worklet Service, which can replace a work item in YAWL ETL process specification at runtime. That is, specific YAWL process acting as sub-net can be invoked based on specific rules at runtime based on a catalogue, handling a specific task. These features make a YAWL process non-static, being possible to specify a set of rules representing some criteria to determine the most appropriated worklet to be invoked. This is particularly useful for ETL processes domain, because it allows specifying individual behaviours based on a set of pre-defined events. This feature can be used for the specification of DQP patterns, since for this pattern we need to apply a set of specific cleaning/conforming procedures based on a set of pre-defined scenarios. The flexibility provided by a feature like this will allow for the extensibility of the DQP pattern in order to include more cleaning/conforming procedures, without reconstructing the initial process specification.

5 RESULTS ANALYSIS AND EVALUATION

In the domain of workflow languages several authors have been proposing different languages to represent conceptual representations for ETL processes and its correspondent mappings for execution primitives. Akkaoui et al. (2011) already provided a BPMN-based meta-model, where they explored the bridges for a model-to-code translation, providing its execution to specific tools. However there is still a lack of a complete approach that covers the initial process specification that easily maps an initial model to an execution process. In fact, in several occasions, BPMN proves to be very useful at the conceptual level (Akkaoui & Zimanyi 2009; Akkaoui et al. 2012; Oliveira & Belo 2012; Oliveira & Belo 2013a), but building BPMN conceptual models may lead in process specification to ambiguity situations (a same process can be represented in several different and valid ways). Akkaoui and Zimanyi (2009) and Oliveira and Belo (2013a) explored also the use of BPMN for conceptual ETL representation providing a way to execute the models specified using BPEL constructs. We also already explored the use of BPMN 2.0, including orchestration elements providing the execution of BPMN models, with some limitations. In both approaches is still clear that there is a distinction between the definition of the conceptual model and its corresponding implementation, just because both models have different detail levels and different purposes. Even using only BPMN 2.0, still exist a significant bridge between both models. It is necessary to specify meanings and metadata for model execution. Using YAWL we have a concrete and logical approach comparing to the BPMN approach. At the same time, it provides a simpler model already built with all execution support structures for its execution. Decker et al. (2008) already mentioned the conversion of YAWL models in BPMN models. However, its mapping isn't straightforward.

At logical level, YAWL provides a valuable reuse-based approach quite suitable for the definition and implementation of ETL processes. YAWL is a technology-oriented language providing useful features that enforce ETL processes quality, for reuse and process implementation: cancellation sets, C-YAWL models, selection and exception services, and a powerful mean to data representation and manipulation. With YAWL it's possible to specify a graphical representation of an ETL process without using a coupled language, and to construct very flexible ETL patterns, providing a powerful communication interface and a way to adapt their behaviour at runtime. Thus, we can easily change the order of tasks' behaviour (and input data), adding more rule-based events to meet new requirements and the continuous evolution of a pattern. Using YAWL we can have the best of two worlds: a concise and concrete modelling language and the bridges and features to provide process execution ant its continuous improvement to meet more functional requirements. However, the simplicity of the YAWL's constructs can limit the communication and expressiveness at the conceptual level. The use of abstract and descriptive models is very useful at the early stages of an ETL process design. At that point, models shouldn't include any kind of implementation specification, nor any criteria associated with its execution (Oliveira and Belo 2012). Thus, BPMN can be used in order to provide a more abstract view over an executable YAWL model. Nevertheless, the necessary meanings and rule construction need to be well defined for a successful translation between the two models. ETL processes requirements change frequently. They are one of the main problems of the implementation and maintenance of an ETL process. We believe that the flexibility provided by YAWL models could help to minimize this gap.

6 CONCLUSIONS AND FUTURE WORK

In this paper we propose the use of a pattern-oriented approach for ETL modelling and implementation. Each ETL pattern represents an ETL task (or set of tasks) that is regularly used in a real world DWS – SKP, SCD, CDC, DQV, or IDL. We can look to these patterns as "black boxes" that given a proper input metadata produce a specific output, accordingly its internal specification. This approach provides an easy method to specify complex ETL processes as also some proven software engineer practices for ETL systems implementation. With our approach it's possible to reduce some planning problems, especially the ones related to business requirements changing and implementation errors. We can change the execution order of a pattern and its input data without compromising other tasks or compromise the final process implementation.

Through the use of YAWL, we demonstrated how to design a complete ETL system using a set of ETL patterns. The YAWL specification provides a simple and very powerful notation that coupled with powerful execution primitives and data support structures turns YAWL very suitable for the validation of ETL processes before proceeding to its final implementation. Using this ETL modelling approach, designers and developers only need to know how to interact with patterns regardless of its internal specification.

As future work we intend to provide an extended family of YAWL patterns allowing for building a complete ETL system from scratch. Additionally, we expect to provide specific XML schemas for the definition of patterns' metadata and explore the use of selection and exception handling services.

REFERENCES

Van der Aalst, W M P & Hofstede, A. H. M. Ter, 2003. YAWL: Yet Another Workflow Language. Information Systems, 30, pp.245–275.

Akkaoui, Z. El et al., 2011. A model-driven framework for ETL process development. In *DOLAP '11 Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*. pp. 45–52.

Akkaoui, Z. El et al., 2012. BPMN-Based Conceptual Modeling of ETL Processes. Data Warehousing and Knowledge Discovery Lecture Notes in Computer Science, 7448, pp.1–14.

Akkaoui, Z. El & Zimanyi, E., 2009. Defining ETL worfklows using BPMN and BPEL. In *DOLAP '09 Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*. pp. 41–48.

Buschmann, F., Henney, K. & Schmidt, D. C., 2007. Pattern-Oriented Software Architecture, On Patterns and Pattern Languages, Wiley. Available at: http://books.google.pt/books?id=wzplRf3uh-EC.

Decker, G. et al., 2008. Transforming BPMN Diagrams into YAWL Nets. Business Process Management Lecture Notes in Computer Science, 5240, pp.386–389.

El-Sappagh, S. H. A., Hendawi, A. M. A. & Bastawissy, A. H. El, 2011. A proposed model for data warehouse ETL processes. *Journal of King Saud University – Computer and Information Sciences*, 23(91-104).

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

Lujan-Mora, S., Vassiliadis, P. & Trujillo, J., 2004. Data Mapping Diagrams for Data Warehouse Design with UML. In *In Proc. 23rd International Conference on Conceptual Modeling* (ER 2004. Springer, pp. 191–204.

Oliveira, B. & Belo, O., 2013a. Approaching ETL Conceptual Modelling and Validation Using BPMN and BPEL. In *2nd International Conference on Data Management Technologies and Applications* (DATA).

Oliveira, B. & Belo, O., 2012. BPMN Patterns for ETL Conceptual Modelling and Validation. The *20th International Symposium on Methodologies for Intelligent Systems*: Lecture Notes in Artificial Intelligence.

Oliveira, B. & Belo, O., 2013b. ETL Standard Processes Modelling: A Novel BPMN Approach. In *15th International Conference on Enterprise Information Systems* (ICEIS).

Oliveira, B. & Belo, O., 2013c. Pattern-based ETL conceptual modelling. In *3rd International Conference on Model & Data Engineering* (MEDI 2013).

Oliveira, B. & Belo, O., 2013d. Using Reo on ETL Conceptual Modelling - A First Approach. In ACM *Sixteenth International Workshop On Data Warehousing and OLAP* (DOLAP 2013).

Simitsis, A. & Vassiliadis, P., 2008. A method for the mapping of conceptual designs to logical blueprints for ETL processes. Decis. Support Syst., 45(1), pp.22–40. Available at: http://dx.doi.org/10.1016/j.dss.2006.12.002.

Simitsis, A. & Vassiliadis, P., 2003. A Methodology for the Conceptual Modeling of ETL Processes. In The *15th Conference on Advanced Information Systems Engineering* (CAiSE '03). pp. pp. 305–316.

Trujillo & Luján-Mora, S., 2003. A UML Based Approach for Modeling ETL Processes in Data Warehouses. Conceptual Modeling - ER 2003 - Lecture Notes in Computer Science, 2813, pp.307–320.

Vassiliadis, P. et al., 2003. A framework for the design of ETL scenarios. In *Proceedings of the 15th* *international conference on Advanced information systems engineering*. Berlin, Heidelberg: Springer-Verlag, pp. 520–535. Available at: http://dl.acm.org/citation.cfm?id=1758398.1758445.

Vassiliadis, P. et al., 2000. Arktos: A Tool for Data Cleaning and Transformation in Data Warehouse Environments. IEEE Data Eng. Bull, 23, p.2000.

Weske, M., Aalst, W. M. P. van der & Verbeek, H.M.W., 2004. Advances in business process management. Data & Knowledge Engineering 50, 50(1–8).