Using a Domain-Specific Language to Enrich ETL Schemas

Orlando Belo[1(✉)], Claudia Gomes[1], Bruno Oliveira[1], Ricardo Marques[2], and Vasco Santos[1]

[1] ALGORITMI R&D Centre, Department of Informatics, School of Engineering, University of Minho, Campus de Gualtar 4710-057, Braga, Portugal
obelo@di.uminho.pt
[2] WeDo Technologies, Centro Empresarial de Braga, 4705-319, Ferreiros, Braga, Portugal

Abstract. Today it is easy to find a lot of tools to define data migration schemas among different types of information systems. Data migration processes use to be implemented on a very diverse range of applications, ranging from conventional operational systems to data warehousing platforms. The implementation of a data migration process often involves a serious planning, considering the development of conceptual migration schemas at early stages. Such schemas help architects and engineers to plan and discuss the most adequate way to migrate data between two different systems. In this paper we present and discuss a way for enriching data migration conceptual schemas in BPMN using a domain-specific language, demonstrating how to convert such enriched schemas to a first correspondent physical representation (a skeleton) in a conventional ETL implementation tool like Kettle.

Keywords: Data warehousing systems · Data migration schemas · ETL conceptual modeling · BPMN specification models · Domain-Specific language · ETL skeletons · Kettle

1 Introduction

Nowadays, companies implemented very specialized data warehouses [6], trying to cover their information needs on decision-making processes. The process of populating a data warehouse - ETL (Extract-Transform-Load) [5] – is a sophisticated data migration process, involving a large range of tasks and operators articulated often in a complicated coordination workflow. ETL projects are quite complex and difficult to accomplish. Planning and designing assume very relevant roles on these projects in order to ensure the success of building a data warehouse. ETL tools provide today mature technologies. Most of them provide support in all stages of the ETL life cycle, from conceptual to logical, and from logical to physical ETL. However, most of them still use proprietary notations or follow other specific methodologies that cannot be generalized or easily used. Recent works have evolved around the idea of conceptually modeling ETL using BPMN patterns [7], as a part of a high-level abstraction layer of a set of commonly used tasks in the implementation of an ETL system. Despite the interesting results of these works [11] [1] [2], they do not yet solved the problem of

T. Morzy et al. (Eds): ADBIS 2015, CCIS 539, pp. 28–35, 2015.
DOI: 10.1007/978-3-319-23201-0_4

using all the potential of a conceptual model. This could be easily achieved if one used a custom language – a *Domain Specific Languages* (DSL) - with the ability to describe the semantics of ETL patterns when integrated in a BPMN conceptual model. According to [12] a DSL is «a high-level software implementation language that supports concepts and abstractions that are related to a particular (application) domain». Thus, a DSL has a very expressive power for describing the specificities of a particular domain such as the case of the description of ETL patterns in BPMN [3][4]. DSL are built by defining a grammar for the language – the logic between the components and rules – and for the components. Then, the language will be used to build or describe ETL patterns.

In this paper we present and discuss a way for enriching data migration conceptual schemas in BPMN using a DSL, especially designed and implemented to add semantics to BPMN tasks and processes. Additionally, we also demonstrate how to convert such enriched schemas to a first correspondent physical representation (a skeleton), having the possibility to be executed and validated in Kettle [8], a conventional ETL implementation tool from Pentaho. The paper is structured as follows. Section 2 describes and exemplifies how the DSL we designed and its grammar were created to describe complementary operational semantics on BPMN tasks and processes. After that, we demonstrate the use of the DSL on the specification of particular instantiations of processes in a simplified version of an ETL system (section 3). Finally, in section 4, we present some conclusions and final remarks.

2 Integrating DSL Specifications in BPMN Models

In BPMN [3] a task or an activity representing a pattern is just a visual component having a label and a small set of properties, but nothing to define its structure or its behavior. There are other approaches that propose specific oriented languages, like BPEL (Business Process Execution Language) [9] to solve this problem. This language can be seen as an extension to other programming languages, but it is focused especially on web services. A language like this can be used to validate and execute BPMN models, converting each BPMN component into some XML representation that BPEL recognizes. However, this conversion is not simple because: 1) not all elements in BPMN have a direct translation in BPEL, which provokes a certain loss of accuracy in the BPEL model; 2) there is no automated way to translate BPMN to BPEL, meaning that you have to know the elements in BPEL and make the writing of the BPEL model yourself; and 3) the tools that provide support for BPEL force the user to think at a very low level detail regarding the BPEL programming, requiring specialized users to make the conversion of a BPMN model to BPEL. Thus, we can see why BPEL is not a very suitable solution for the problem we want to approach, which stresses the need of a specific language that could add more operational semantic to an ETL conceptual model in BPMN. Generically speaking, ETL tasks can be organized into three ETL pattern categories: extractors, transformers and loaders. Each category entails the execution of various tasks and their properties, relating to different states of the development cycle of an ETL system. Therefore, each category has different properties that characterize the different states of the development of an ETL system. The first category is responsible for identifying and extracting the data that were modified in information sources. To perform this task it is

necessary to indicate the data sources' paths and the data structures that will receive data. In this category we can find patterns such as sequential log files readers or *Change Data Captures* (CDC). Data sources usually are highly heterogeneous, which impose frequently the application of data format recognition mechanisms to detect the kind of information sources – e.g. a relational database, a text file, a CSV file, or a XML file, among others. The second category includes all patterns with the ability to perform transformation tasks, such as data filtering, data quality assurance, data merging, splitting or replacement operations, or the simple conversion of an operational key. Here, some of the most common patterns are Data Quality Enhancement (DQE), Surrogate Key Pipelining (SKP), or *Error Handling Hub* (HHS). Finally, the third category: the loaders. This category includes the patterns and the operations that are responsible for loading data into a specific data repository like a data warehouse. *Slowly Changing Dimension* (SCD) or *Intensive Data Loading* (IDL) patterns are two of the most relevant elements of this category.

```
-- An excerpt of the DSL's grammar.
ETL:
  'use' (elements+=Pattern)+'on sources{' (source+=Data)+ '}'
  ('and target{' target=Data '}')? ('with mapping source{' (map+=Data)+'}')?
  ('options{'opt = Option '}')?;
-- Pattern categories.
Pattern:
  Gather | Transform | Load;
-- Gathers configuration
Gather:
  'Gather'('sort by'  sort = STRING ',')? //(gatherFields+=Fields)*;
-- Information sources configuration
Data:
  'BEGIN' (Path | Source)','type=' type=STRING'END';
Option:
  'parallel operation=' p=STRING ','number of threads=' t=STRING (','
  'number of rows in batch=' r=STRING)?;
Path:
  'data=' src=STRING;
Source:
  'name=' n=STRING ',' 'server=' s=STRING ',' 'database=' db=STRING ','
  ('table=' tb+=STRING ('{'(fields+=Fields)+'}')? ',')*
  'technology=' tech=STRING','access=' a=STRING','user=' user=STRING ','
  'password=' pwd=STRING(', port=' p=STRING)?;
Fields:
  ('source' | 'target')?'fields{' (fd+=Field)+'}';
(…)
-- Loaders configurations.
Load:
  'Load from'('each record')?;
```

Fig. 1. An excerpt of the DSL's grammar

Considering this, we designed and built a specific DSL taking into account all the properties and characteristics of each pattern category. Looking at the DSL's grammar (Fig. 1), we can see how to configure a data source, or even the destiny of the data collected, using an extensive list of attributes (source's attributes). For example, in case of the source or the destiny be a relational source, we need to specify the connection name (*name*), the server identification (*server*), the name of the database (*database*), and others. Otherwise, the source or the destiny of the data can be set

through the path of a file (*data*) containing the data required and the source type. In short, with this kind of approach we will be able to build ETL conceptual models, enrich them with task behavior descriptions using the DSL, converting the enriched models to an intermediate standard format, and produce a physical skeleton to be imported by a conventional ETL implementation tool. Fig. 2 shows an application of the DSL in the configuration of a gather pattern.

```
-- Configuration of a gather pattern.
use Gather
on sources{BEGIN data=CDR_Calls.csv,type=CSV input END}
and target{BEGIN server=localhost, database=CALLSR,table= MSISDN_ACTIVITY
  {fields{id,type,caller_id,called_id,caller_number,called_number,time_start,
  time_end,}}, technology=MySQL,access=Native,user=root,
  password=Saphira23,type=relational END}
options{loop_criteria=FOR_EACH_ROW}
```

Fig. 2. An excerpt of the behavior of an ETL task using the DSL

3 Using a DSL on Process Instantiation

Using a DSL language on the specification of data-driven workflows allows for the formalization of a set of abstract and self-contained constructors, which simplify process representation across several detail levels. To achieve the necessary adequacy, both in terms of business level and physical level, any physical model should be developed following the sequence of operations showed in Fig. 3. This way, business and physical requirements associated with the execution of a given process can be easily represented, and posteriorly used by a commercial tool to enable its real execution.

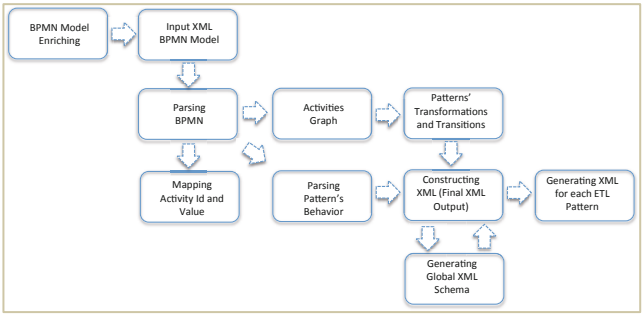

Fig. 3. Steps for producing a physical model based on a BPMN conceptual description

To start, we need to design and produce a conceptual representation in BPMN. Both semantic and physical metadata should be addressed by the DSL, providing a way to distinguish and support all the business rules involved in the process. The BPMN configuration is described using the DSL, indicating the data required by each task. The CSV Gather case (Fig. 2) shows the logical metadata involved with, the field mappings between the source CSV and the target relational schema, and the

execution support metadata describing all the connection parameters of the physical schemas as well as all the specific execution requirements. To provide a clearer picture of how BPMN can be used on early ETL development stages and how their artifacts can be mapped to execution primitives, we selected a very common data migration application case on the telecommunications domain. To support it we used a *Call Detail Records* (CDR) file from a mobile phone company having data about customers' calls. This file will act as a source data for a specific database table that identifies callers, number of calls and total call duration. The migration process (Fig. 4) aims to handle the referred data (stored in a CSV file), in which every row has a record of a call phone done by a specific customer. The record's attributes are: DATE, the date of the call; TIME, a timestamp; DURATION, the time spent with the call; MSISDN of the source customer that depends if the call direction (incoming or outgoing); OTHER_MSISDN of the target customer that depends of the call direction (incoming or outgoing); and DIRECTION of the call. The MSISDN belongs to the called customer and the OTHER_MSISDN to the caller customer, while the value 'O' represents an outgoing call, i.e. the MSISDN belongs to the caller and OTHER_MSISDN to the called customer.

The process initiates extracting data from the CSV file using a BPMN activity. The name of this activity (as many others) includes the # character to identify that the activity represents a container of tasks configured through the DSL proposed. In fact, the `Extract Source` activity configuration was already presented before in Fig. 2. The output data from each source is stored in a separated repository. The logic of this process is handled internally by each pattern.

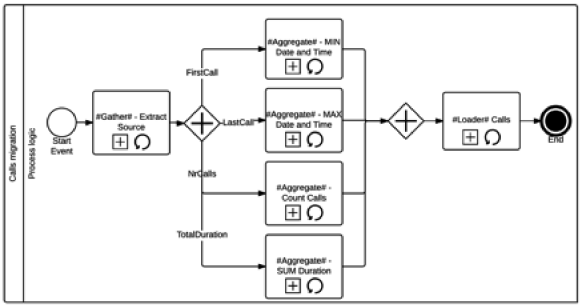

Fig. 4. The customer call registration populating process

Next, four sub-processes (`#Aggregate#`) were defined for making data aggregation. To ensure their parallel execution we used two BPMN parallel gateways: 1) a divergent gateway, indicating that each one of the fired activities will be executed in parallel; and 2) a convergent gateway to force the completion of all parallel flows. Finally, the process ends with the execution of a `#Loader#` container configured to load data into the target data structure. Additionally, a loop marker was used for each activity, meaning that each one should be executed iteratively based on some criteria. For example, the gather specification presented in Fig. 2 describes (using the `options` block) the `loop_criteria` statement with a `FOR_EACH_ROW` value, which means that the internal activity tasks should be executed for each row in the process.

For compatibility reasons, the DSL descriptions for specifying ETL patterns behavior were integrated in the BPMN model using text annotations.

The BPMN process serialization enables the representation of all BPMN elements including DSL into a *XML Process Definition Language* (XPDL) standard format [10]. The XPDL is used by some software products to interchange business process definitions between different tools, providing an XML-based format with the ability to support every aspect of the BPMN process definition notation, including the graphical descriptions of the diagram, as well as its executable properties. Thus, there is no need to extend the XPDL meta-model to ensure compatibility with other existing tools. The XPDL file is used as input of the developed parser to interpret not only the DSL code but also the logic of the workflow, providing the necessary means to preserve the details of the ETL model configuration about the process control, data transformation requirements and configuration parameters – e.g. the access to a database or a file, or the identification of a source or a target data repository (Fig. 5). Having this, it is possible to generate a physical model and provide automatically a data migration package configured accordingly.

To filter metadata enabling its transformation to a recognized specific data format by an ETL implementation tool we designed and implemented a specific parser. This parser interpreted DSL specifications and translated them to an intermediary specification format recognized by a data migration tool. To do this, we first need to convert the model, exporting it to a known serialization format. The parser generates two distinct structures: 1) a map, whose key is the identification of the activity, and a value, which is a class embodying all the details about the XML node activity; and 2) a graph, containing the information about all the existing connections in the BPMN model. BPMN tools use standard formats for process interchange, but data migration tools use their own file formats. This compromises system flexibility. To avoid this we used Acceleo (http://www.eclipse.org/acceleo/), a model-to-code transformation module. Under the MDA (*Model-Driven Architecture*) provided by Eclipse, we also selected and applied some specific Ecore models to describe each pattern skeleton and to identify process instantiation metadata. Next, a standard transformation template was built to encapsulate the conversion process and to transform the internal structure of the pattern into a XML serialization format supported by the implementation tool - Kettle. This process is executed in two steps: 1) the verification of all patterns included in the BPMN conceptual model and their conversion to a single XML representation; and 2) the generation of the general XML Kettle schema. To establish the mapping rules between a BPMN conceptual model specification and Kettle package representation it is necessary to follow a set of generic rules that are explicitly defined in what we call a transformation template. On this template: 1) the three main patterns classes - gather, transformer, and loader - are mapped directly to specific Kettle jobs or transformations; 2) the internal tasks are created inside each top-level package as well as the details related to the execution of the process. In Fig. 6 we can see four distinct examples of packages that were generated based on a BPMN model specification. The top package level (Fig. 6a) represents the three main clusters that could be explicitly represented at a conceptual level using BPMN. This layer represents a transformation that corresponds to the data extraction activity represented in the BPMN process presented before in Fig. 4. The associated model (Fig. 6b) includes a specific *CSV input* task that is responsible to load data to a *copy rows to result* task.

Rows are stored in memory since all jobs and transformations were configured to execute one row each time and the case presented does not process large volumes of data at once. The *Transformers* job (Fig. 6c) is associated with the transformation that performs, for each application scenario, the aggregation tasks. The parallel configuration described in the BPMN model was translated enabling the parallel execution of each transformation that encapsulates the logic of each aggregation task. Finally, the Loaders transformation (Fig. 6d) from top-level package represents the data load to the target relational table.

Fig. 5. A XPDL excerpt used for the serialization of a BPMN process (Fig. 4)

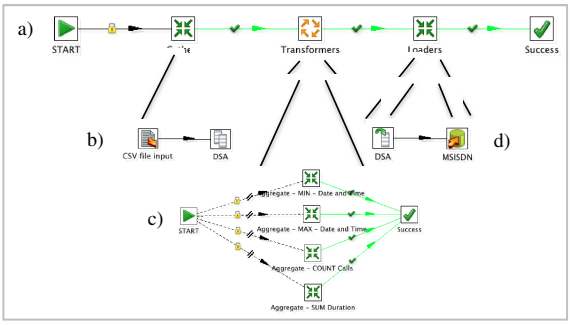

Fig. 6. The Kettle package generated from the BPMN model (Fig. 4)

4 Conclusions

Conceptual models are frequently discarded in favor of a more detailed logical model. In our opinion, ETL conceptual modeling is a very useful activity in the life cycle of a data warehousing system project. Based on BPMN and on a set of ETL patterns (characteristics and behavior description), we designed and implemented a specific ETL development process that enhances the importance of building ETL conceptual

models as a way to establish a first executable version of the system – an ETL skeleton. To enrich the expressiveness of BPMN we designed and created a DSL especially oriented to describe the behavior of an ETL pattern when integrated in an ETL conceptual model. Results are quite satisfactory, and prove the viability of the approach. It was possible to demonstrate with a real application case that it is possible to use a lot of the material applied in the conceptual specification as a catalyst for a possible physical implementation. At this stage, the ETL skeletons that can be produced still are very primary physical models. In a near future we need to improve them, in order to allow for the generation of more effective physical.

Acknowledgment. This work was developed under the project RAID B2K - RAID Enterprise Platform / NUP: FCOMP-01-0202-FEDER-038584, a project financed by the Incentive System for Research and Technological Development from the Thematic Operational Program Competitiveness Factors.

References

1. El Akkaoui, Z., Zimanyi, E.: Defining ETL workflows using BPMN and BPEL. In: DOLAP 2009 Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP, pp. 41–48 (2009)
2. El Akkaoui, Z., Zimanyi, E., Mazón, J.-N., Trujillo, J.: A BPMN-Based Design and Maintenance Framework for ETL Processes. Int. J. Data Warehous. Min. **9** (2013)
3. BPMN, OMG, in Documents Associated With Business Process Model And Notation (BPMN), Version 2.0 (2011)
4. Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices **35**(6), 26–36 (2000)
5. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Willey (2004)
6. Kimball, R., Ross, M., Thornthwaite, W., Mundy, J., Becker, B.: The Data Warehouse Lifecycle Toolkit, 2nd edn. Wiley, January 2008
7. Oliveira, B., Belo, O.: BPMN patterns for ETL conceptual modelling and validation. In: Chen, L., Felfernig, A., Liu, J., Raś, Z.W. (eds.) ISMIS 2012. LNCS, vol. 7661, pp. 445–454. Springer, Heidelberg (2012)
8. Pentaho: Pentaho Data Integration. http://www.pentaho.com/product/data-integration (accessed March 16, 2015)
9. Recker, J., Mendling, J.: On the translation between BPMN and BPEL: conceptual mismatch between process modeling languages. In: Latour, T., Petit, M. (eds.) Proceedings of the 18th International Conference on Advanced Information Systems Engineering, Luxembourg, pp. 521–32 (2006)
10. T.W.M.C. Specification: Workflow Management Coalition Workflow Standard Process Definition Interface – XML Process Definition Language (2012)
11. Trujillo, J., Luján-Mora, S.: A UML based approach for modeling ETL processes in data warehouses. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 307–320. Springer, Heidelberg (2003)
12. Visser, E.: WebDSL: a case study in domain-specific language engineering. In: Lämmel, R., Visser, J., Saraiva, J. (eds.) Generative and Transformational Techniques in Software Engineering II. LNCS, vol. 5235, pp. 291–373. Springer, Heidelberg (2008)