RAID-B2K, transforming BPMN conceptual schemas into Kettle execution primitives

Orlando Belo*, Vasco Santos, Bruno Oliveira and Cláudia Gomes

ALGORITMI R&D Centre,
Department of Informatics,
School of Engineering,
University of Minho,
Campus de Gualtar, 4710-057 Braga, Portugal
Email: obelo@di.uminho.pt
Email: vsantos@estgf.ipp.pt
Email: id4103@alunos.uminho.pt
Email: pg22818@alunos.uminho.pt
*Corresponding author

Ricardo Marques

WeDo Technologies,
Centro Empresarial de Braga,
Ferreiros, 4705-319 Braga, Portugal
Email: ricardo.marques@wedotechnologies.com

Abstract: There are many tools for designing and modelling extract-transform-load (ETL) systems, covering its entire development life cycle. However, the vast majority of them use proprietary methodologies, notations and tasks, which undermine their understanding and application. In this paper, we present a translation tool for conceptual models, with the ability to reduce the ‘gap’ that usually exists when we need to translate a conceptual model for an equivalent physical one. We will demonstrate that it is possible to automatically translate ETL conceptual models developed in business process model and notation (BPMN) into the environment of a specific ETL implementation tool (Kettle–Pentaho data integration). The BPMN models were built to produce schemes for a specific execution environment (RAID) allowing us to demonstrate the utility of the tool in the translation, validation and generation of the physical schemas which we designated as ETL skeletons – a set of execution primitives properly orchestrated.

Keywords: decision support systems; ETL systems modelling and implementation; ETL conceptual models; business process model and notation; BPMN; ETL physical models; Pentaho data integration; Kettle.

Biographical notes: Orlando Belo is an Associate Professor, with Habilitation, at the Department of Informatics of University of Minho, in Portugal. He is also a member of the ALGORITMI R&D Centre in the same university, working in business intelligence, with particular emphasis in data warehousing systems, OLAP, and data mining. His main research topics are related with data warehouse design, implementation and tuning, ETL services, and distributed multidimensional structures processing. During the last few years, he was involved with several projects in the decision support systems area designing and implementing computational platforms for specific applications like fraud detection and control in telecommunication systems, data quality evaluation, and ETL systems for industrial data warehousing systems. More recently, he was developing some research work establishing OLAP usage profiles and optimising OLAP selection methods incorporating knowledge acquired over OLAP user sessions and users’ preferences.

Vasco Santos is an Auxiliary Professor at the Polytechnic of Porto – School of Management and Technology of Felgueiras (ESTGF.IPP) in Portugal. He obtained his PhD degree in 2015 at the University of Minho under the supervision of Professor Orlando Belo. He is a member of CIICESI Research Center in the ESTGF.IPP School and is currently researching in the area of data warehouse design methodologies in particular ETL conceptual and logical design.

Bruno Oliveira is an Invited Professor at the Polytechnic of Porto – School of Management and Technology of Felgueiras (ESTGF.IPP) and a member of CIICESI Research Center in the same school. He is also an Invited Professor at the University of Minho and a member of ALGORITMI R&D Centre in the same university. He is currently developing his PhD studies in the area of data warehouse design methodologies in particular ETL conceptual and logical design.

Cláudia Gomes is currently an IT professional on the IT consulting area and has received her MSc in Computer Science from the University of Minho. In her academic background while on her Master, she was involved in the areas of business intelligence and database design, more focused on areas such as data warehousing design and implementation, ETL and reporting. Currently, she has been working professionally on rules-based engines in the area of information management.

Ricardo Marques is an IT professional with 15+ years experience in the industry, mostly on the software development area for Telcos. During this period, he has developed competencies both in technological topics – such as multi-tier application architecture and design, module-driven development and component reuse, ETL, performance and scalability – as well as business knowledge primarily related with telecommunications OSS and BSS systems. He is currently responsible for product management and architecture at the WeDo Technologies, interfacing with colleagues such as architects, product owners, project managers and QA engineers throughout the product development lifecycle.
1 Introduction

In an effort to gain competitive advantage, the development of data warehousing systems (DWS) has been one of the approaches that companies have been committed to in the last decade (Kimball et al., 2008). DWS are vital to managers in their decision-making processes. However, their development is quite complex and has a high rate of failure, mainly due to the extraordinary amount of data that DWS need to process, in a limited time window, with the goal of storing in a centralised repository a faithful representation of the companies’ business (Mannino and Walter, 2006).

Usually, the process of populating a data warehouse (DW) is known as extract-transform-load (ETL) (Kimball and Caserta, 2004). This process is responsible for gathering data from a set of information sources, clean, transform and conform the data and then load it into the DW, which makes it nothing more than a sophisticated data migration process. However, the complexity of the tasks and operators in a complicated workflow consumes great amounts of time and computational resources. Software models, in particular, the conceptual models provide a level of abstraction that is very useful in early development phases (Scacchi, 2001), providing an excellent groundwork for process validation. This modelling task requires qualified personnel with proven experience and knowledge, resulting in valuable tools whose utility can be demonstrated in almost all phases of the development of a software system. The utility of such models motivated several researchers to purpose specific frameworks to develop ETL conceptual models (Vassiliadis et al., 2002a; Simitsis and Vassiliadis, 2003; Trujillo and Luján-Mora, 2003). ETL commercial tools available use mature technologies and support 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 very specific methodologies that cannot be generalised or easily used.

In an effort to standardise the modelling phase of an ETL system, several authors have proposed new methodologies and notations (Vassiliadis et al., 2002a, 2002b), or extended existing standard modelling notations like unified modelling language (UML) (Trujillo and Luján-Mora, 2003) or business process model and notation (BPMN) (El Akkaoui and Zimányi, 2009). However, these proposals have still not been widely accepted and used by general DW practitioners and ETL commercial tools, probably due to the lack of full support of the semantics associated to an ETL process. In contrast to these works, the approach presented supports the ETL developing using composite tasks that can be used for many ETL scenarios. The majority of works presented till now supports ETL processes representation using very detailed tasks. Thus, the generated models are composed by dozens of finer grain tasks that compromises model readability. With the pattern-based approach proposed, models are generated under an abstraction layer that simplify and help to develop more consistent process through the use of well-proven techniques. In our view, one possible solution could be the adoption of a domain specific language (DSL) to characterise more accurately the specificities of an ETL system (Van Deursen et al., 2000). DSL are built by defining a grammar for the language - the logic between the components and rules – and for the components (Visser, 2008). Then you just use the language to build or describe your ETL patterns.
The tool developed and presented in this paper translates automatically a BPMN diagram complemented with our DSL to a Kettle (Pentaho, 2015) preliminary physical model. We demonstrate that by enriching a BPMN schema, it is possible to generate a first correspondent physical representation (a skeleton), having the possibility to be executed and validated in Kettle. The paper is structured as follows: Section 2 describes the case study used in this paper to demonstrate the tool developed. After that, in Section 3, we present Kettle as the execution platform of an ETL process. In Section 4, we present the developed tool that translates enriched BPMN schemas into a physical skeleton executable by Kettle. Finally, in Section 5, we present some conclusions and final remarks.

2 The case study

The case study used to demonstrate the tool we designed and developed is based on a real world scenario of a telecommunications company. The company stores in a CSV file detailed information about phone calls made by their clients and is normally called a call detail records (CDR) file. The file has records of calls identifying the date, time and duration of the call, and also the caller and the callee. The attribute direction identifies if the call was made or received by the company’s customer (Table 1). This CDR file will be used as source data for a migration process that populates a dimension table, named ACTIVITY (Table 2).

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Duration</th>
<th>MSISDN</th>
<th>Other MSISDN</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>20120701</td>
<td>104118</td>
<td>134</td>
<td>24832136</td>
<td>27045516</td>
<td>I</td>
</tr>
<tr>
<td>20120701</td>
<td>124022</td>
<td>61</td>
<td>24832136</td>
<td>27045516</td>
<td>O</td>
</tr>
<tr>
<td>20120701</td>
<td>124721</td>
<td>122</td>
<td>27045516</td>
<td>16868253</td>
<td>I</td>
</tr>
<tr>
<td>20120701</td>
<td>233148</td>
<td>978</td>
<td>27045516</td>
<td>24832136</td>
<td>O</td>
</tr>
<tr>
<td>20120701</td>
<td>135145</td>
<td>16</td>
<td>23586800</td>
<td>35947886</td>
<td>I</td>
</tr>
<tr>
<td>20120701</td>
<td>160727</td>
<td>8</td>
<td>23586800</td>
<td>13943850</td>
<td>I</td>
</tr>
</tbody>
</table>

Table 1 A fragment of a CDR source file

<table>
<thead>
<tr>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientID</td>
</tr>
<tr>
<td>FirstCall</td>
</tr>
<tr>
<td>LastCall</td>
</tr>
<tr>
<td>NumberOfCalls</td>
</tr>
<tr>
<td>TotalDuration</td>
</tr>
</tbody>
</table>

The dimension table ACTIVITY stores aggregated data of calls made by clients. The attribute FirstCall and LastCall identifies date and time of first and last call. The attributes NumberOfCalls and TotalDuration are aggregated attributes over source data that store the number of calls made and the total duration in seconds.
3 Using Kettle to create ETL processes

An ETL process represents a data-driven workflow composed by a set of tasks, their associated control flows and business rules that together express how the system should be coordinated. The commercial tools that support ETL processes development, typically implement very specific tasks that need to be grouped together each time we want to represent the same procedure. For example, the surrogate key (SK) generation is often used for each dimension table in a DW. From the process point of view, the composition of each one is the same but with different configuration. Additionally, even in early development phases, ETL processes are composed by several tasks that increase process complexity exponentially, contributing to increase time and budget not only in development but also in process maintenance phases.

Figure 1 represents a possible subset of the populating process used to integrate data to the ACTIVITY dimension table presented before. This process represents some possible tasks associated to each common groups of tasks typically used, namely: extract (gathers), transform (transformers) and load (loaders). This example was selected to illustrate all the steps needed and the specificities of real world processes used for the extraction, conciliation, and integration of data to a target dimension.

Figure 1 A part of the process used to populate the dimension table ACTIVITY (see online version for colours)

For the implementation of the process previously presented in Figure 1, several conditions about the physical execution of the process were pre-established:

- Process should execute each record individually.
- Process should be configured for periodic execution using a specific time interval or a specific schedule.
- Specific group of tasks should be executed in parallel.
- The output data from each task should serve as input of the next task without preserving temporary results, i.e., data should be passed in memory to the next step.
The Kettle populating process presented describes the general approach taken to handle data from calls stored in a CSV file to the ACTIVITY dimension table, presenting the necessary steps to extract data according to specific rules (such as the delimiter character), to clean and transform the data in order to adjust its schema to the dimension table requirements, and finally providing its load to the dimension table. Specifically, the transformation phase involves a set of data calculations to populate the FirstCall, LastCall, NumberOfCalls and TotalDuration attributes. Thus, a set of grouping operations with specific aggregation functions should be applied over the data to find most recent/old call, calls total duration and total number of calls that each customer was involved. To support these aggregation tasks, some minor data corrections should be applied to fulfil all requirements.

Kettle provides two types of constructs that are used to represent process structure. The finer grain tasks used to apply transformations over the data are recognised as step. Steps can be grouped inside transformations – a more general construct. In turn, transformations can be grouped inside jobs, which control several aspects related to the execution of transformations, other related jobs and work-flow management. These constructs provide a way to decompose process in functional areas, allowing the representation of tasks in clusters with specific execution requirements.

The top package level [Figure 1(a)] corresponds to the three main clusters that encapsulate tasks according to its purpose: gathers, transformers and loaders. The gathers transformation is composed by a specific CSV input task that is responsible to extract data from CSV file to the copy rows to result step that allows data flow across jobs in memory. The get rows from result step is used to read the records written previously in memory. This procedure is mandatory whenever data records flow across jobs or transformations. The decomposition of the transformers job is described in a second [Figure 1(b)] and third [Figure 1(c)] level of detail, composed by a data quality enforcement (DQE) job and four transformations to apply aggregation functions over the data. The aggregation tasks are executed in parallel, encapsulating the logic of each aggregation task using a transformation. Since each record is related to two customers, the DQE job decomposes each record in two records: the first one represents call data from the caller and the second one represents the same call data to the callee. Next, for each one, a specific date normalisation procedure should be performed. The raw data represents the call date and time separately and the dimension table represents the date using only one field in a date time format. Thus, a specific transformation is used for data concatenation and date conversion between two fields. The DQE job was also configured with the property: launch next entries in parallel that is responsible by the parallel execution of the outgoing tasks. Next, four aggregation tasks are executed to populate first/last call, total duration of calls and number of call of each customer. Due to Kettle synchronisation limitations (Chodnicki, 2010; Pentaho, 2015), the tasks involved in the parallel execution are grouped inside a job to force that all concurrent tasks will finish their execution before returning to the sequential work-flow execution. Finally, the loader transformation [Figure 1(d)] from top-level package is executed to load data previously handled by transformers job to the target dimension table.
4 Enriching BPMN models with ETL specifications

The amount of data that today companies need to analyse increases exponentially every day. The business evolution and the need to readjust existing business processes to satisfy new requirements are important issues that should be reflected in the operational systems. As a result, business processes are more and more complex and difficult to manage. Several approaches have been taken to simplify management and implementation of such processes. Business process management and notation (BPMN) is a work-flow language typically used by organisations to describe and implement their business processes. The BPMN notation has become a widely used notation on modelling business processes mainly because of its simplicity, expressiveness, and application scenarios. Moreover, BPMN presents a well-defined semantic structure, and provides an easy working platform. Besides conceptual representation, BPMN 2.0 also supports the execution of business processes through the use of specific constructs. This feature is very useful, since more descriptive models can be derived in order to provide, at least, an initial sketch from the final system.

The data migration processes such as ETL processes, can be considered specific business processes (El Akkaoui and Zimányi, 2009), i.e., data-oriented work-flows composed by several tasks logically related to transform and integrate raw data to a specific repository. Besides ETL logic complexity, the amount of data produced and its distribution across several spread operational branches, brings real challenges that consume a significant time and money of any decision support system. Thus, mainly in early development stages, the communication between all stakeholders should be transparent, ensuring the correct requirements identification as well to reduce the risks associated to the development of an inappropriate solution.

The ETL conceptual modelling using BPMN represents a specific application of BPMN elements that has been explored in past few years (El Akkaoui and Zimányi, 2009; El Akkaoui et al., 2012). Not only its expressiveness but also the simplicity of the generated models can support ETL development properly and support project development in main life cycle phases. However, to provide the automatic generation of executable models based of the conceptual representations, specific metadata should be provided. Since BPMN lacks to support specific business vocabularies and rules (Skersys et al., 2012), the BPMN meta model should be extended to support specific operational semantics according to the tasks typically used in ETL real world scenarios.

To match ETL requirements with business process concepts, El Akkaoui et al. (2012) proposed an ETL classification, covering the ETL semantic to produce physical models. We consider a very useful approach, however, the approach we follow represent ETL tasks at a higher abstraction level. Representing ETL operations at a finer grain level will result in a very complex process that will hinder initial development phases. At the conceptual level, users should produce models that represent the main ETL tasks, providing a way to understand and capture business knowledge more clearly. Thus, common composite tasks can be encapsulated as clusters with a set of predefined tasks and represented as black boxes, i.e., only the input and output interfaces are known, while
its internal composition is totally hidden from the users and remaining tasks. Thus, a pattern-oriented approach for ETL conceptual modelling with BPMN is proposed, along with the necessary bridges to translate conceptual representations to physical models with the possibility to be executed using commercial tools.

The BPMN process presented in Figure 2 represents the first step of the approach presented, describing the main tasks needed to extract, transform and load data from the CSV file to the ACTIVITY dimension according to the requirements presented before.

A set of abstract tasks is described and can be categorised in different groups of ETL patterns, according to its operational purpose:

- **Gathers** – they are used in extraction operations, representing tasks such as index access readings, change data captures, or log file data extractions. The task: 
  \[#Gather\] – SourceFile is used for data extraction from the CSV file provided.

- **Transformers** – these elements apply a set of transformations to integrate data according to ACTIVITY dimension requirements. The \[#DQE\] – Normalise Rows and the \[#Aggregate\] tasks are some examples of these class of patterns. The DQE decomposes each row in two new rows to store data about the caller and the callee as well, and the aggregate task calculates a set of aggregations with specific functions: SUM, COUNT, MAX and MIN.

- **Loaders** – that are used to load data to a target repository – e.g., bulk operations, slowly changing dimensions SCD (), or quarantine operations. The \[#Loader\] Calls task from Figure 2 is responsible for loading data into the ACTIVITY dimension.

Figure 2  A BPMN conceptual model to support data migration (see online version for colours)

With these categories we can group all ETL common tasks and decompose them in finer grain tasks to identify all operational stages. Depending on the needs, the process can be enriched with additional BPMN elements such as data objects or data input/output artefacts that can be used to improve process interpretation. In Figure 2, two BPMN artefacts were used to describe the input file type used in the gather pattern and to describe the output relational repository used by the loader pattern. Between these tasks data will be passed on-the-fly, i.e., physical repositories will not be used implicitly, and for that reason there is no need to represent physical repositories for these tasks. With
BPMN, not only pattern representation can be done but also define how the flows will be coordinated. The parallel gateways (divergent and convergent) are used to describe parallel execution of the aggregate tasks. Additionally, each task was configured using the loop marker that identifies that the task is to be repeated according to a criteria that must be further specified. For ETL purposes, the repetition criteria must specify the number of records that should be handled per iteration.

We believe that the presented process is very useful at early development phases to discuss and validate not only the patterns that will be used but also the order how they will be synchronised. Posteriorly, at the logical level, this conceptual representation can be enriched with a specific configuration language that allows the description of each component used. The integration of the BPMN models with ETL specifications could be accomplished merging the BPMN meta model with the pattern-based extension meta model developed. This extension does not affect the interpretation of the BPMN model by common commercial tools since there is a clear separation between the original BPMN meta model and the extension proposed.

With the metadata provided, the original BPMN meta model and the extension meta model can be used to generate a new model that can be later translated to execution primitives interpreted by commercial tools. The translation process will generate physical models based on a set of specific templates configured to generate Kettle models. The BPMN enrichment is provided using a DSL, developed to describe ETL rules and operational requirements considering the operational semantics of each class of patterns used to develop the full ETL package. With the DSL developed we are working at a logical level, providing specific grammar components to each pattern used. To support grammar validation and parsing we used Xtext (2015) framework under Eclipse EMF framework. Figure 3 shows a simple excerpt of both grammar rules construction and grammar instantiation. Particularly, a configuration applied to the gather pattern from Figure 2 is presented. Several blocks are used to configure each component related to the operational execution of Gather patterns:

- The *source* block is used to describe the input metadata used to access to the CSV file. The path and file name are identified, as well as the type parameter that indicates what type of source will be used. In this example, a CSV file is identified along with the delimiter character to provide a way to interpret extracted data.

- The *target* statement represents a simple attribution, since the output will be passed *on-the-fly* to the next task. If physical repositories are used, such as temporary files or relational databases, a configuration block should be used to describe the repository that will store the output data.

- The *fields* block describes which fields will be extracted and passed to the next pattern, working like a projection over a set of attributes.

- The *options* block describes specific execution requirements that will affect the generation of physical models. The *GatherExtractionType* property is defined as *FULL*, which means that extraction process will extract all data from the source without any criteria. The *execution* property is a complement to the standard loop defined for each task using BPMN visual components, providing a method to extract one record at a time.
The grammar definition is grouped according to the flow elements available, providing specific constructs to configure not only patterns but also flow elements from original BPMN meta model, namely events and gateways. For example, the start event used for the process presented in Figure 2 was configured to execute the whole process repeatedly every 10 seconds (Figure 4).

After the configuration of each task and the start event, all process configurations should be interpreted to match with the BPMN extension meta model developed. The modelling tool used (BizAgi), exports both the BPMN diagram and the DSL to a single document in a XML process definition language (XPDL) file (Shapiro, 2008). 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. This file contains information about all the existing connections in the BPMN model and in the configuration of each task (Figure 5). The DSL can be provided in the documentation panel associated to each flow element or using the annotation artefact.

**Figure 3** A grammar definition excerpt and the corresponding example for a gather configuration (see online version for colours)

The parser is the component that is responsible to interpret metadata thus enriching the generation of the physical model making it possible to be executed by a data migration tool such as Kettle. This step is composed by two components: the BPMN parser
responsible to interpret and validate the BPMN definition; and the grammar parser responsible to validate and interpret data associated to each flow element.

**Figure 4** Configure a start event with a DSL instance (see online version for colours)

Both components are supported by the meta model layer that defines the grammar parsing rules associated to a subset of the original BPMN meta model with specific extension elements representing patterns composition (Figure 6). For example, the gather pattern is responsible for extracting data in three different forms, each one representing a class specialisation: full extraction, incremental extraction representing change data capture techniques that can be used to extract only relevant data – trigger or logged based are common types used, and differential load used to compare data between two sources to identify relevant data.

**Figure 5** A XPDL excerpt used for the serialisation of a BPMN process (see online version for colours)

After the logical design phase, a standard transformation template was built to encapsulate the logic of the conversion process and provide the meanings to transform the pattern internal structure to a specific serialisation format supported by Kettle. To guarantee system flexibility and avoid the proprietary formats of data migration tools, we used a template generator language: Apache velocity to describe each component
skeleton, and build a specific and standard transformation template to encapsulate the conversion logic. Figure 7 provides an overview between all layers developed and how they are related.

**Figure 6** Ecore diagram representing an excerpt of a gather composition (see online version for colours)

Considering the specificities of Kettle the main pattern classes, gather, transformer and loader, will be grouped together in a single Kettle job. If inner transformations associated to each component have only one composite task (such as gather for the example provided), then a Kettle transformation task will be generated. However, if several tasks were used (as it happens in the transform phase), a Kettle job is generated. The loop marker that was used in all the tasks of the model represent a record-by-record processing that was configured for each transformation using *for each row* property. The parallel execution of aggregation tasks is performed using a job parallel execution, and the *launch next entries in parallel* property is applied over the construct used before the execution of parallel tasks. For each aggregation task, input data is copied to enable task parallelisation and its output stored in a temporary structure created for each flow. Since the ETL presented processes small number of records in each execution, records can be transformed in memory without causing performance bottlenecks, and otherwise other approach should be taken. We take this approach in order to guarantee process synchronisation and to avoid deadlocks when all tasks work on the same repository. The loader task is responsible to load data from the temporary structures to the target dimension.
To provide a flexible approach to translate the logical model to a target Kettle physical model, a set of Apache velocity templates were developed. Figure 8 shows an excerpt of the template used to convert the gather pattern configuration to target Kettle representation. The document structure is statically defined in the template, while data that can be extracted from the logical model presented is included dynamically using the Apache velocity primitives. Some specific execution properties can involve complex template generation due the Kettle particularities in process execution. The on-the-fly property defined before involves a difficult template configuration since a specific step and its connection to the remaining steps should be generated: copy rows to result. Similarly, the subsequent transformations should generate the get rows from result step as described before. Apache velocity supports quite well all the translation requirements presented. Thus, we can maintain flexible templates that can be changed without affecting the logical model core.

Specific physical details related to the physical execution platform used were ignored in the process configuration. For example, the connection properties were omitted, however the DSL proposed could be extended to provide these details if necessary to the execution platform. However, to provide a more general picture of the approach we followed, only a semi-automatic generation of physical models is presented rather than to provide a Kettle-oriented approach. Thus, with several enrichment levels, the DSL proposed allow users to choose the level of detail applied to the generation of physical models, which contributes to system flexibility.
Figure 8  Excerpt of a gather template used for generating a Kettle physical model (see online version for colours)

```xml
<transformation>
  <Info>
    ...
  </Info>
  <order>
    #foreach ($item in $gather.getTransitions())
      <hop>
        <from>$item.getFrom().getName()</from>
        <to>$item.getTo().getName()</to>
        <enabled>$</enabled>
      </hop>
    #end
    #if($gather.getTarget().getOutputType().equals("On-the-fly"))
      <hop>
        <from>$gather.getLastTransformation().getName()</from>
        <to>Copy rows to result</to>
        <enabled>$</enabled>
      </hop>
    #end
  </order>
  #foreach ($item in $gather.getTransitions())
    <step>
      <name>$item.getName()</name>
      <type>$item.getType()</type>
      ...
      <filename>$item.getFileName()</filename>
      ...
      <fields>
        #foreach ($field in $gather.getFields())
          <field>
            <name>$field</name>
            <type>String</type>
          ...
        </field>
        #end
      </fields>
    ...
  </step>
  #end
  #if($gather.getTarget().getOutputType().equals("On-the-fly"))
    <step>
      <name>Copy rows to result</name>
      <type>RowsToResult</type>
      ...
    </step>
  #end
</transformation>
(....)
5 Conclusions and future work

ETL modelling has always been a challenge. Developing an ETL system that gathers and processes all necessary data to represent faithfully the companies business is a complex task if not sustained by a proven methodology therefore minimising the risk of failure in such vital system like a DWS. Designing methodologies base themselves in the separation of conceptual, logical and physical models in an effort to gain independence between these layers. Nevertheless, existing commercial ETL tools tend to create and use proprietary diagrams and notations with clear disadvantage to portability, and with a clear focus on the logical/physical model. This means that if we change the ETL vendor, it typically implies almost a complete ETL reimplementation from scratch, simply due to the fact each specific tool follows it own methodology. Conceptual models are frequently discarded in favour of a more detailed logical model. We consider ETL conceptual modelling a useful activity in the life cycle of a DWS project, providing a way to simplify process representation with focus in patterns synchronisation rather the configuration of very detailed tasks. Since patterns are independent from each other, not only process consistency is guaranteed but also the patterns reuse is possible, improving time and reducing costs associated to ETL development.

Posteriorly, we enriched BPMN with a DSL grammar that characterises and describes common ETL tasks behaviour. Thus, we can formalise each pattern configuration, allowing its posterior mapping to specific meta model developed to support each pattern requirement. We also developed a tool that automatically translates the enriched BPMN conceptual model into an ETL skeleton that can be submitted to an execution platform like Kettle. The rules and workflow logic for each tool used is supported by a set transformation templates that will allow the generation of specific code without changing the base meta model. We believe that this approach will significantly increase ETL quality, reduce functional and operational errors, and decrease planning and implementation costs.

Currently, the parser presented supports several common used patterns associated to the three main classes presented, such as the generation of SKs and for story preservation (SCD), and loaders that support several algorithms to optimise data loading to specific sources (CSV, XML or relational). Specific error/exception handlers for each pattern are also under development to provide a way to identify and recover from errors that typically occur. Currently, we are working to expand the number of patterns provided as well to validate the approach provided in complex environments that involve personnel from different knowledge levels.

Acknowledgements

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


