Defining ETL Worfklows using BPMN and BPEL

Zineb El Akkaoui, Esteban Zimányi
Department of Computer and Decision Engineering (CoDE)
Université Libre de Bruxelles
50 Avenue F. D. Roosevelt
1050 Brussels, Belgium
zelakkao@ulb.ac.be, ezimanyi@ulb.ac.be

ABSTRACT

Decisional systems are crucial for enterprise improvement. They allow the consolidation of heterogeneous data from distributed enterprise data stores into strategic indicators. An essential component of this data consolidation is the Extract, Transform, and Load (ETL) process. In the research literature there has been very few work defining conceptual models for ETL processes. At the same time, there are currently many tools that manage such processes. However, each tool uses its own model, which is not necessarily able to communicate with the models of other tools. In this paper, we propose a platform-independent conceptual model of ETL processes based on the Business Process Model Notation (BPMN) standard. We also show how such a conceptual model can be implemented using Business Process Execution Language (BPEL), a standard executable language for specifying interactions with web services.

1. INTRODUCTION

A *data warehouse* is a specialized database that aims to prepare and manage decisional information for easy access and exploitation by analysts. A data warehouse organizes the information using fact and dimension tables [5]. A *fact table* represents the focus of analysis and contains *measures* that represent analysis indicators. A *dimension table* includes attributes allowing the user to explore the measures from different analysis perspectives. A data warehouse obtains its data from multiple sources in various formats and applies a set of transformations for cleansing and modifying the data in order to be stored into the data warehouse. This chain of activities is known as *Extraction, Transformation, and Load* (ETL) process.

Many tools provide dedicated platforms for ETL design. Consequently, there are many different logical ETL models specific to each tool. Unfortunately, this diversity of approaches generates a separated growth of best practices and methodologies. At the same time, these tools tend to make the designer worry about implementation issues from the begining of the ETL design process.

This paper proposes a high-level and platform-independent approach for ETL design. It consists of a conceptual model based on the Business Process Model Notation (BPMN) [7]. There are several reasons for using this notation. Firstly, BPMN provides a palette of conceptual tools to express business processes in an intuitive language, which can be later mapped into a target execution language. Secondly, BPMN is a notation used to design all enterprise processes uniformly so as every process can communicate easily with the others. In this paper we also study the translation of BPMN into Business Process Execution language (BPEL) [6], a standard executable workflow language for web services.

The paper is organized as follows. In Section 2 we describe related work in ETL design. Section 3 presents a running example that shows how BPMN may be applied to an ETL process. In Section 4 we briefly describe BPMN, and in Section 5 we describe our approach for using BPMN as a conceptual model for ETL processes. Section 6 presents the mapping of BPMN to BPEL. Finally, Section 7 contains final conclusions and points to further work.

2. RELATED WORK

Various approaches for designing, optimizing, and automating ETL processes have been proposed in the last few years. In this section we briefly review these different approaches.

Conceptual design of ETL processes belongs to the more general notion of conceptual data warehouse design. This notion was first introduced by Golfarelli et al. [3]. A detailed description of conceptual multidimensional models can be found in [17]. Some of these models are based on the ER model (e.g., [5, 11, 18]), or on UML (e.g., [2, 4]).

In spite of the abundant literature about ETL issues, there are only a few papers dealing with conceptual modelling for ETL processes. The model proposed by Simitsis et al. [20] represents ETL processes as a graph where nodes match to transformations, constraints, attributes, and data stores, and edges correspond to data flows, inter-attribute relations, compositions, and concurrent candidates. An approach of mapping conceptual design to logical design was proposed in [12]. Starting from a conceptual representation, matching rules allow to obtain a logical representation ready for implementation.

On the other hand, several tools are available for designing and executing ETL processes, such as Microsoft Integration Services, Oracle Warehouse Builder, Talend Open Studio, or Pentaho Data Integration. However, each one of these tools provide its own language for specifying ETL processes.

DOLAP'09, November 6, 2009, Hong Kong, China.

These languages differ considerably in many respects, in particular since they are based on different paradigms and have different expression power. As a consequence, ETL processes are incompatible across tools and this implies high costs when migrating from one tool to another. In addition, since these languages often involve implementation-level considerations they are difficult to understand, optimize, and maintain.

Optimizing ETL processes is an essential issue of ETL design. In fact, such processes are time and effort consuming, and recent technologies and requirements, such as real-time data warehouses, further compounded this complexity. Several solutions have been proposed for this. For example, [1] considers the logical level of ETL processes as a state-space search problem. The approach assimilates a node of an ETL graph to a state that changes to the next one through a transition, and an heuristic search algorithm then finds the optimal state among the constructed states. Another approach [19] provides an algorithm for optimizing the physical graph generated from the logical one. These approaches are applied during the construction phase of ETL processes, and therefore target *static* optimization. Yet, *dynamic* optimization is an essential aspect since it allows ETL processes to fit the evolution of system requirements. Dynamic optimization can be guaranteed by a permanent monitoring and a continuous improvement of the processes. Our approach, based on BPMN, allows this permanent evaluation during the whole lifecycle of ETL processes.

On the other hand, ETL processes are usually designed by non-technical users. Therefore, a pertinent aspect of ETL design is its automation. Based on source and target semantic descriptions comprising schema structures and business constraints, an approach consists in building an application ontology that acts as a common metamodel. This ontology is inferred by a reasoner to generate the workflow [13, 14], or by applying some graph operation rules to generate ETL activities step by step [15]. These approches require a significant effort to ensure a semi-automated construction of ETL processes. Further, the automation of the maintenance of ETL processes has not been addressed. The model proposed in this paper presents an adequate platform for a such purpose. Other approaches deal with the automated generation of multidimensional concepts either from a business domain ontology like in [10], or from a detailed analysis of the data sources [8].

3. RUNNING EXAMPLE

Populating a data warehouse is done by a complex ETL process composed of many tasks. Each task achieves a unit of work, e.g., loading a dimension or updating a fact table, and usually consists of several steps. To illustrate our contribution, we consider an excerpt of an ETL process that loads a geographical table DimGeo into a data warehouse. Fig. 1a shows two tables of the source database from which the data will be extracted. Fig. 1b shows the temporary tables used in the transformation process with sample data illustrating several issues that must be resolved. Fig. 1c shows the geographical dimension of the data warehouse. This dimension is composed of four tables defining a hierarchy: DimGeo, DimState, DimCountry, and DimArea.

The DimGeo table takes its data from the City, State, ZipCode, and Country attributes from tables Customer and Supplier. However, populating the DimGeo table is not straightforward since the geography data needs some cleansing and transformation operations to fit the data warehouse requirements. There are many issues that the ETL process has to resolve in order to consolidate the data. For concision reasons we will only consider here three types of issues.

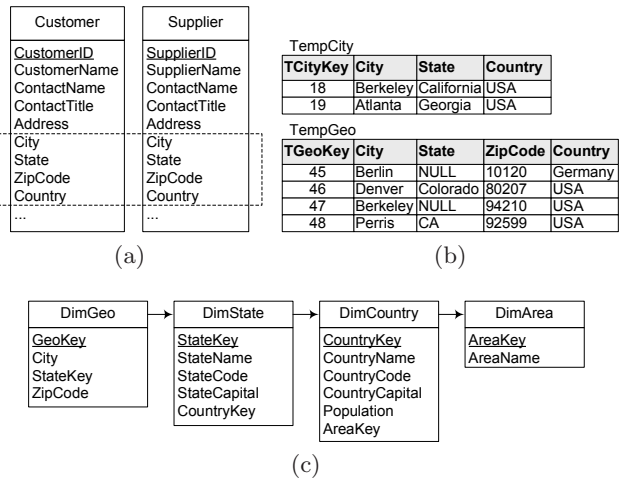

Figure 1: (a) Tables from source database; (b) Temporary tables; (c) Geography dimension in the data warehouse.

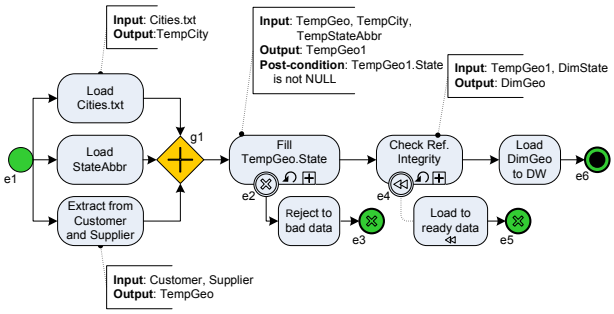

Figure 2: Example ETL workflow.

The first issue concerns *data completion*, in particular dealing with null values. As can be seen in the sample data in Fig. 1b, the attribute State may be null in the Customer and Supplier tables. Therefore, data should be filled with its corresponding value during the ETL process. For this, an external source file Cities.txt is used. The file contains three fields City, State, and Country, separated by tabs as follows:

Atlanta - Georgia - USA
Austin - Texas - USA
Berlin - Berlin - Germany
...

The second issue concerns *data consolidation*. In the source databases, the attribute State contains either a state name (e.g., California) or a state code (e.g., CA). Thus, in the latter case, the state code must be converted into the state name using an intermediate table StateAbbr with attributes StateId, StateName, and StateAbbr, which contains the link between the state name and its code.

The last issue concerns *consistency*, in particular with respect to referential integrity constraints. In our example, during the loading of the prepared data into the data warehouse, the data must comply with the referential integrity

constraint between the DimGeo and DimState tables.

The workflow depicted in Fig. 2 is used to populate the DimGeo table and takes into account the three above issues. We give next a detailed description of this workflow.

The first step consists in extracting data from the source database, the file Cities.txt, and the StateAbbr table and loading it into the temporary tables. This is done by the tasks Load Cities.txt, Load StateAbbr, and Extract from Customer and Supplier, respectively. Notice that these activities may be performed in parallel because there is no dependency between them; this allows the process execution to be optimized. After that, a synchronizing control g1 is used to ensure that all loadings have been achieved, since all the data is required for the next task.

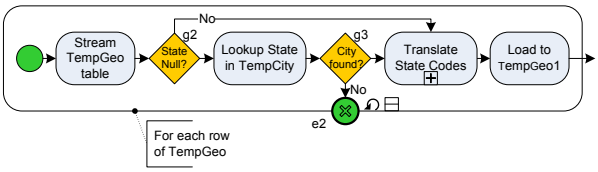

Figure 3: Fill TempGeo.State subprocess in expanded form.

The next task, Fill TempGeo.State, is shown collapsed in Fig. 2. This task is a looping container that generates a cancel event e2. Fig. 3 expands this task at a more detailed level and shows its chain of activities.

The first activity consists of streaming the table TempGeo for a row-by-row processing. Then, the null condition is checked on the TempGeo.State attribute for every row using the gateway g2. Depending on the evaluation of the condition, one of the two branches is activated: if the attribute is null, the activity Lookup State in TempCity is launched, otherwise, activity Translate State Codes is performed.

The activity Lookup State in TempCity addresses the data completion issue mentioned above. It attempts to fill in the rows with a null state by looking up the state name in the TempCity table using the city name in TempGeo.City. In the case the city is not found, an error event e2 is generated. Correct data is then ready for the outgoing task.

The next subprocess Translate State Codes handles the data consolidation issue mentioned above. It translates state code values in TempGeo.State into state name values using the table StateAbbr. This subprocess is repeated for all TempGeo rows as marked in the annotation linked to the loop container.

Finally, the subprocess Check Ref. Integrity addresses the consistency issue mentioned above. It checks the referential integrity constraint between the DimGeography and DimState tables. This subprocess filters inconsistent rows and sends them to the activity Load to Ready Data for subsequent processing and then the whole process is cancelled, whereas correct rows are carried to the task Load DimGeo to DW.

As shown in Fig. 2, the workflow is initiated by the starting event e1, which is activated either manually or automatically at the end of the previous process. The workflow is finished by either launching the end event e6 in a normal execution, or by a cancel, error, or compensation event (e.g., compensation event e5) in abnormal cases.

Notice that in this example workflow we assumed many simplifying hypothesis, such as: (1) the DimGeo table is initially empty; (2) Each city listed in the file Cities.txt matches to only one state and occurs only once; (3) the corresponding source and target attributes have the same type and format, etc. However, a realistic solution for this ETL process requires to handle these and several other issues.

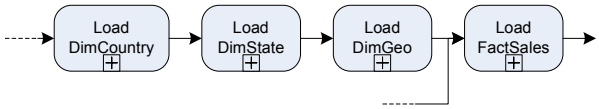

Figure 4: A high-level viewpoint.

As noted before, this workflow is a small excerpt from the ETL process populating the data warehouse. Fig. 4 shows a fragment of the whole process in a high-level viewpoint, where the workflow in Fig. 2 is represented by the activity Load DimGeo. As shown in the figure, this activity should be preceded by other tasks that load the DimCountry table, the DimState table, etc. Also, the Load DimGeo activity should precede the loading of the FactSales table.

In this section, we have illustrated the use of BPMN for describing ETL processes by an example. After a brief presentation of BPMN in Section 4, we give a more general description of this approach in Section 5.

4. MAIN FEATURES OF BPMN

A *business process* is a set of ordered tasks describing how work is performed within an organization. Business process are commonly represented using XML-based execution languages such as BPEL [6].

Many tools provide support of these execution languages. They also manage the life cycle of business processes, i.e., their design, implementation, execution, monitoring, and analysis of the improvement factors. In addition, these tools generally provide graphical notations for describing the logic of business workflows. However, these graphical notations are too complex and detailed to be used by business analysts and managers. Further, each tool provides its own graphical notation and palette of constructs, which complicates communication and understanding of business processes among different organizations.

The Business Process Model Notation (BPMN) [7] was proposed as an answer to these problems. BPMN is a standard notation for modelling business processes. It offers to users a common and understandable framework in order to describe business processes independently of the used tools. BPMN aims at providing a notation that is understood by all categories of business users: business analysts that create an initial draft of the processes, technical developers responsible of implementing the technology that perform those processes, and business people who manage and monitor those processes. Further, there are currently many research works that study the mapping of BPMN to execution languages, in particular BPEL.

BPMN allows enterprise processes to be represented in order to analyze and improve them. Many tools (e.g., TIBCO, intaglio, Sparx Systems) support BPMN and give users the possibility to represent business processes and to run simulations of them. Despite the fact that BPMN is becoming a de facto standard for modelling business processes, there are several ongoing initiatives to improve it. For example, the OMG group is working on the next version 2.0 of BPMN, in particular concerning issues like standardizing the BPMN

metamodel and enabling the exchange of business process designs and their diagrams among tools to preserve semantic integrity.

5. APPLYING BPMN TO ETL PROCESSES

An ETL process can be considered as a particular type of business process. As is the case with traditional business processes, there is no standard model for defining ETL processes, each of the existing tools provide their own model. Further, these models are often too detailed, since they take into consideration many implementation issues.

In this section we show how BPMN can be customized for designing ETL processes. We describe the constructs composing the proposed ETL palette. These constructs are grouped in four categories: flow objects, artifacts, connecting objects, and swimlanes. We divided flow objects into *ETL tasks*, which is the fundamental construct of workflows, and *control objects*, which highlight the control procedures provided by BPMN.

5.1 Flow objects

5.1.1 ETL Tasks

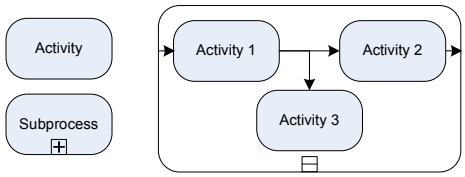

Figure 5: (a) ETL tasks; (b) Expanded subprocess.

An *ETL task* represents a simple or a composite unit of work of the integration workflow. It is the central object describing what is performed in the workflow. ETL tasks are modelled in BPMN as an activity or a subprocess, see Fig. 5a. An *activity* describes a unitary task that is not further subdivided, whereas a *subprocess* represents compound activities and expresses the hierarchical nature of workflows. In the running example, see Fig. 2, Load Cities.txt is a simple activity, while Fill TempGeo.State is a subprocess. A subprocess is also a context of exception; it allows the scope of the process where an exception is applied. For example, the above subprocess represents the context of exception for the compensation event e2. As shown in Figs. 5a and 5b, there are two ways to represent a subprocess: collapsed or expanded. Note that data annotation may be associated to both activities and subprocesses in order to describe their semantics, as will be explained later.

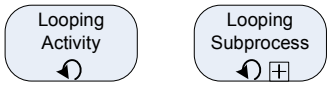

Figure 6: Activity and subprocess loop.

A loop, see Fig. 6, is an execution control feature that leads to the reexecution of an ETL task a number of times depending on a Boolean condition. There are two types of conditions depending on whether they are checked before or after activity. A loop is ended if its condition is evaluated to false. The loop condition may be written in an annotation as depicted in the expanded Fill TempGeo.State subprocess in Fig 3. The loop feature is useful in the ETL context, especially for pipelining tasks that are performed in a row-by-row manner.

ETL tasks are subdivided into three categories: row operations, rowset operations, and control operations. We briefly present next the possible types of tasks inside each group.

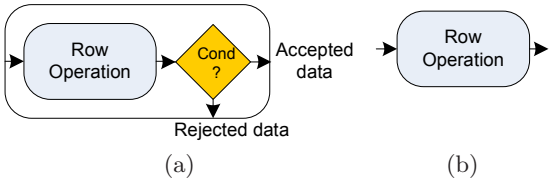

Figure 7: Row operation: (a) with and (b) without filter.

Row operations are transformations which are applied to the source or target data on a row-by-row basis. There are two types of row operations:

1. Row operations with data-based control. They apply a conditional filter and the rejected data may be either further manipulated or kept aside. These operations are depicted in BPMN as an activity linked to a data-based gateway (Fig. 7a). Such operations are used for tasks such as checking primary keys and foreign keys, unique values, not null values, domain mismatch, and lookup.

2. The second type of row operations does not apply a filter. Examples of such operations are selection, projection, type conversion, and surrogate key assignment. They are represented as a simple task, see Fig. 7b. Note that row operations are performed inside a loop that fetches data on a row-by-row basis and transfer them to the operation.

Figure 8: Rowset operations.

In contrast, *rowset operations* deal with a set of rows. For example, aggregation, sort, pivot, join, union, and difference are all rowset operations. This type of transformation is represented by a simple, splitting, or joining task, see Fig. 8.

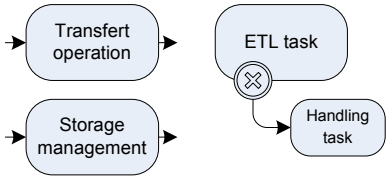

Figure 9: Control Operations.

Finally, *control operations* highlight part of the control aspect in ETL scenarios. This type of operations includes

transferring data through the network, managing files and data storage, and error control operations, as depicted in Fig. 9. Such operations are normal ETL tasks that use exception, error, cancel, and compensating events, see Fig. 11b.

5.1.2 Control Objects

Control objects manage the workflow sequence, or orchestration, independently of the data transiting through the process. BPMN includes a set of control elements, like gateways and events.

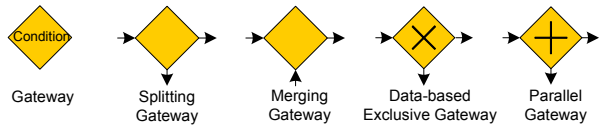

Figure 10: Different types of gateways.

Gateways, see Fig. 10, are used to control the activity sequence in an ETL process based on conditions. They may either embody the condition or be linked to a *gateway condition* artifact that includes the condition (more details are given in Section 5.2). BPMN defines several types of gateways, like exclusive, inclusive, event based, or data based events; further, they can be splitting or merging. The most used types in an ETL context are data-based exclusive gateways and parallel gateways. Examples of such types are g1 in Fig. 2 and g2 in Fig. 3, respectively. An *exclusive gateway* models a decision: depending on a data condition the gateway activates one or more of its outgoing branches. A *parallel gateway* expresses synchronisation between ingoing flows as a condition for activating the outgoing flows.

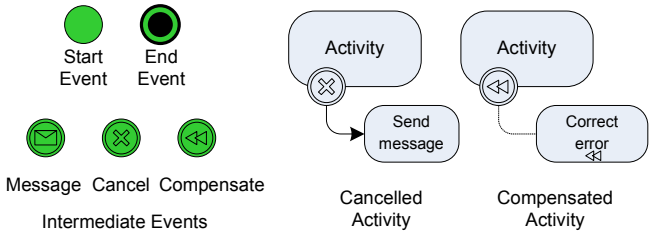

Figure 11: (a) Examples of start, end, and intermediate events; (b) Error and compensation handling.

Events, see Fig. 11a, represent something that happens that affects the sequence and timing of the workflow activities. These events may be internal or external to the task into consideration. They are categorized into start, intermediate, or end events. Events may be placed along the workflow like an activity, as is the case of the event e1 in our example, or on the boundary of the activity shape or the subprocess, as for e3. There are typical usages of events in an ETL context. For instance, a start event of type Time may be used to represent the execution schedule of the ETL process, while a normal start event may be used if the process is simply triggered by the end of its predecessor process. Further, other common events include error, message, cancel, and compensate events. In this paper we focus on error and compensation handling, and refer the reader to the BPMN specification [7] for more details.

The *error handling* is a particular intermediate event. It listens to the process errors and notify them either by an explicit action like sending message, e.g., the cancelled activity sends a message as shown in Fig. 11b, or by an implicit action that will be defined in the next steps of the process development. Besides detecting errors, *compensation handling* can also be employed to recover errors by launching specific compensation activities, which are linked to the compensation event with the association connecting object as shown in Fig. 11b. For example, an error event may send an e-mail alert notifying the failure of a task or a process, while a compensate event will try to correct an error by executing an additional activity before restarting the execution of the concerned part of the workflow.

5.2 Artifacts

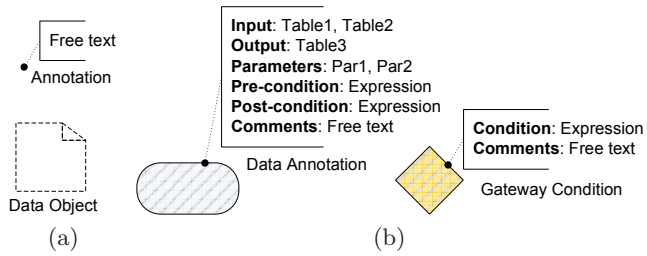

Figure 12: (a) BPMN artifacts; (b) Our anotations.

The main BPMN artifacts are annotations and data objects, see Fig. 12a. A *data object* is used traditionally to represent transiting documents between tasks (like invoices or contracts), whereas an *annotation* is used to express semantics about flow objects. In our approach, annotations are specialized into two new objects, see Fig. 12b: data annotations and gateway conditions.

Data annotations are used to make explicit the semantics of an ETL task. They include several features: *input* and *output* data of the task, *parameters*, which refer to additional data used by the task, *pre-* and *post-conditions*, and *comments*, describing any other useful semantics. Depending on whether the task is a row or rowset operation, the input and output data may be either a table or a row. As shown in our example, the traditional Table.Attribute notation is used in annotations.

Gateway condition state the conditional expression of a gateway that imposes some control on the flow, e.g., decision or synchronization. In some cases, the semantics of the gateway is expressed graphically and does not need an explicit gateway condition, as for parallel gateways. Gateway conditions include two features: *condition*, stating the conditional expression of the gateway, and *comments*, which allows to express any useful aspect of the gateway.

5.3 Connecting Objects

In BPMN there are three types of connecting objects: *sequence flows*, *message flows*, and *associations*. Fig. 13 illustrates these types of connectors.

A *sequence flow* represents the sequencing constraints between flow objects. It is the basic and essential connecting

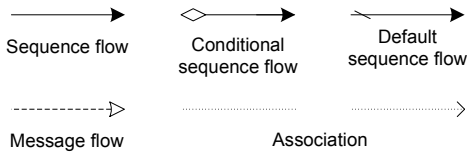

Figure 13: Connecting objects.

object in an ETL workflow. It states that if two activities are linked by a sequence flow, the target activity will start only when the source one has finished. If multiple sequence flows are emerging from a flow object, all of them will be activated after its execution. In case there is a need to control a sequence flow, it is possible to add a condition to the sequence flow by using the *conditional sequence flow* or by using a gateway. A sequence flow may be set as the *default flow* in case of many splitting flows.

A *message flow* represents the sending and receiving of messages between the pools (see below). In a BPMN diagram, separate pools represent different entities. A message flow is the only connecting object able to get through the boundary of a pool and may also connect to a flow object within that pool.

An *association* relates artifacts to flow objects. It is also used to link the compensation activity for in case of compensation handling as noted previously.

5.4 Swimlanes

A *swimlane* is a structuring object that comprises *pool* and *lanes*, see Fig. 14. Both of them allow the definition of process boundaries. As stated above, only messages are allowed between two pools, no sequence flows. Further, one workflow must be contained in only one pool. However, one pool may be subdivided into many lanes, which represent roles or services in the enterprise. Lanes within a pool do not have any special constraints and thus, sequence flows may cross a lane freely.

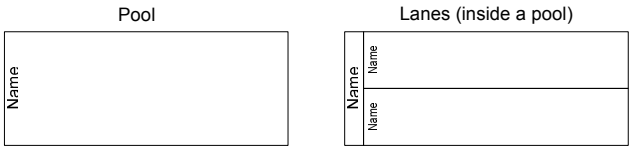

Figure 14: Swimlanes: Pool and lanes.

Swimlanes allows the organization and the hierarchization of several different and multi-level ETL processes for populating or updating a data warehouse. Swimlanes allow ETL processes to be organized according to several strategies: (1) technical architecture (e.g., localisation of tasks in servers, applications, interfaces), as is the case of the example in Fig. 15, (2) by user profile (e.g., a manager, analyst, or designer) that gives special access rights to the user, or (3) by business entities (e.g., a department or company).

We illustrate the swimlane notion in the running example, seen from its high-level viewpoint, see Fig. 4. Fig. 15 shows some ETL subprocesses: loading the geography dimension, the time dimension, and the sales fact table; it also depicts their distribution between Server 1 and Server 2. Each one of these servers is considered as a lane contained inside the pool of DW servers. The figure shows some messages between

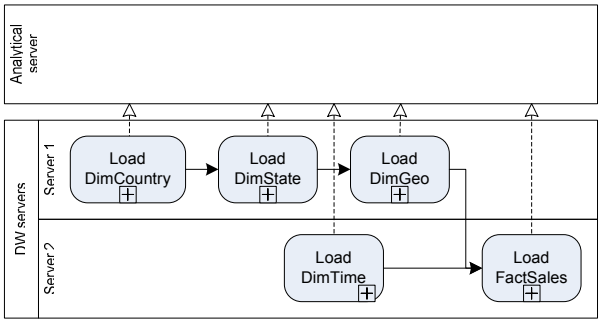

Figure 15: Swimlane example.

the DW servers and the Analytical server pools in order to highlight the interactions between those systems.

6. FROM BPMN TO BPEL

Web Services Business Process Execution Language (WS-BPEL or BPEL for short) [6] is a standard executable language for specifying interactions with web services. It defines business processes using an XML-based language. BPEL is supported by many tools, which provide sophisticated supervision and analysis of process execution.

BPMN can be used as a graphical front-end to capture BPEL process descriptions. Mappings from BPMN to BPEL have been proposed and are implemented in several tools, e.g., Oracle BPM Suite and ActiveVOS. Nevertheless, there are some differences between BPMN and BPEL. For example, these languages are used in a different stage of the life cycle of business processes: BPMN is used for designing business processes whereas BPEL is used for implementing them. Further, since BPMN is used by business analysts while BPEL is used by technical analysts and programmers, these languages use different paradigms and focus on separate issues when modelling a process. A more closed integration between BPMN and BPEL is expected in the next versions of the two languages.

In the ETL context, interactions between ETL tasks and subprocesses are managed by an ETL execution engine using dedicated languages. In our approach we have chosen BPEL for such purpose. However, in order to map a BPMN workflow into an executable language such as BPEL it is necessary that the designer provides additional information about the process. This is similar to what happens when translating conceptual models, such as the ER model, into logical ones, such as the relational model. In this activity, implementation-level information about the tasks, events, etc., must be provided. Furthermore, the designer must proceed to a decomposition of each top-level activity into a detailed subprocess. This action is repeated until atomic activities (i.e., implementable ETL operations) are reached.

Then, the mapping that consists in transforming each BPMN object into its equivalent in BPEL can be carried out. Such mapping may be automatically generated by some tools. Two approaches could be followed for this. One possibility is to describe in a very detailed way the BPMN workflows by filling in all the workflow attributes, prior to the automatic mapping. This requires an important technical effort for the designer. Alternatively, the designer could complete the skeleton of the BPEL process generated by the automatic mapping.

6.1 Introduction to BPEL

In BPEL, processes are constructed from primitive and structured activities [6]. *Primitive activities* represent basic constructs, such as invoke to call a web service, receive to wait for a particular message to arrive, reply to send a response, throw to launch faults and exceptions, etc. *Structured activities* are used for combining these primitives. The most important ones are: sequence for defining an ordered set of invoked activities, flow for parallel ones, if-else for branching, while for loops, and pick for selecting one among a number of alternative paths. Each process in BPEL can declare variables and define partner links. *Partner links* describe links to partners, where partners might be services invoked by the process, services that invoke the process, or services that do both.

6.2 Translating BPMN into BPEL

We transform next a fragment of our example ETL process depicted in Fig. 2 into BPEL. In the discussion we emphasize how BPMN objects are mapped into BPEL elements.

A first step to perform is the translation of the basic information about the whole process. The four main sections composing the BPEL process definition must be specified. These sections are the following: partnerLinks, variables, faultHandlers, and process.

Mapping basic attributes of the business process. We start by mapping the basic information about the process loading the geography dimension, such as the business process name and related name spaces. The code below depicts the header tag of a BPEL process.

```
<process name="LoadDimGeo"
 targetNamespace="http://nwetl.com/ws-bp/
   loadDimensions"
 xmlns="http://docs.oasis-open.org/wsbpel/2.0/
   process/executable"
 xmlns:lns="http://datawarehousing.org/wsdl/
   loadDimensions">
```

Mapping services to partnerLink Elements. The second step is to list the different partnerLink elements. In our approach, we consider ETL tasks in the BPMN process as being of type service; they are thus mapped to partnerlink elements. Each partnerLink is characterized by a partnerLinkType and one or two role names. This information identifies the functionality of the service or the partner providing the service.

```
<partnerLinks>
 <partnerLink name="loadTxtFile" partnerLinkType=
  "lns:loadTxtFileLT" myRole="fileLoader" />
 <partnerLink name="loadTable" partnerLinkType=
  "lns:loadTableLT" myRole="tableLoader" />
 <partnerLink name="SQLScript" partnerLinkType=
  "lns:SQLScriptLT" myRole="tableLoader" />
 ...
</partnerLinks>
```

For instance, the task Load Cities.txt will be represented in BPEL using the partner link loadTxtFile. This partner link represents all ETL operations that load text files. It is of type loadTxtFileLT and is performed by the role fileLoader. The partner link will be invoked using the BPEL variable File with value Cities.txt and will return the variable TempTable with value TempCity.

Mapping process properties to BPEL variables and messages. In BPMN, properties are used to state useful information about the process. They have a similar role as BPEL variables. During the mapping, the value of BPMN properties is transferred to BPEL variables. Variables are defined at the head of the BPEL file. For example, the following code excerpt defines the variables tempTable, which keeps the values of temporary tables, and loadFault, which keeps information about eventual loading errors.

```
<variables>
 <variable name="tempTable"
  messageType="lns:tableMessage"/>
 <variable name="loadFault"
  messageType="lns:LoadErrorMessage" />
 ...
</variables>
```

Fault handling section. This section defines the mechanisms for detecting and handling errors resulting from the invocation of services. We match the error end event in BPMN to the faultHandlers element in BPEL, which is used for interrupting a process. For example, in the code excerpt below, if an exception is detected during the sequence of services contained within the scope tag, the cannotCompleteLoading process is launched. This implements a similar behaviour as the BPMN workflow: if an error occurs during the Fill TempGeo.State subprocess, an error end event will be launched.

Main process. The process section contains the description of the normal behavior of the activities that load the geography dimension.

```
<process name="LoadDimGeo" ... >
 ...
 <bpel:sequence>
  <bpel:flow>
   <bpel:invoke partnerLink="loadTxtFile"
    inputVariable="file" outputVariable="tempTable"/>
   <bpel:invoke partnerLink="loadTable"
    inputVariable="table" outputVariable="tempTable"/>
   <bpel:invoke partnerLink="SQLScript"
   inputVariable="database" outputVariable="tempTable"/>
  </bpel:flow>
  <bpel:scope>
   <faultHandlers>
    <catch faultName="lns:cannotCompleteLoading"
      faultVariable="lookupFault">
     <reply partnerLink= "lookupData"
      portType="lns:lookupDataPT"
      variable="lookupFault" faultName=
       "cannotCompleteLoading"/>
    </catch>
   </faultHandlers>
   <bpel:while>
    <condition>
     bpel:getVariableProperty("exists","row")>0
    </condition>
    <bpel:sequence>
     <bpel:invoke partnerLink ="loadRow"
      inputVariable="tempTable"/>
     ...
    </bpel:sequence>
   <bpel:while>
  </bpel:scope>
```

```
  </bpel:sequence>
  ...
</process>
```

The above code excerpt emphasizes the general sequence of the process tasks. It shows the way to express parallelism of the first three activities using the flow construct. It also uses the catch exception inside a scope in order to launch the fault handling mechanism.

7. CONCLUSION

The Extraction, Transformation, and Load (ETL) process aims at extracting data from internal and external sources of an organization, transform this data, and load it into a data warehouse. It is well known that ETL processes are complex and costly. Therefore, it is necessary that these processes may be modelled using a conceptual language that, on the one hand, is easy to understood by end users and developers, and, on the other, can lead to their implementation into current tools. Further, current ETL tools propose specific languages for expressing such processes, which differ among tools and have different expressive power. As a consequence, there is no an agreed-upon way to specify ETL process at a conceptual level and that can be translated into executable specifications.

In this paper we have presented a conceptual language for modelling ETL processes based on the Business Process Modelling Notation (BPMN), a de-facto standard for specifying business processes. Our model provides a large and useful set of primitives that cover the design requirements of frequently used ETL processes. In addition, this set of primitives can be extended to fit the requirements of particular applications.

There are several advantages of using BPMN for specifying ETL processes. One of them is that the language is already used for specifying business processes in general and thus, end-users must not learn another language for specifying ETL processes. Further, BPMN provides a conceptual and implementation-independent specification of such processes, which hides technical details and allows end users and designers to focus on essential characteristics of these processes. Finally, we have shown that ETL process expressed in BPMN may be translated into executable specifications using Business Process Execution Language (BPEL), a standard executable language for specifying interactions with web services.

There are several ways in which this work can be pursued. One of them is to study the translation of ETL processes expressed in BPMN into the specific languages proposed of current ETL tools in order to execute such processes. Another issue is to analyze the expressive power of our model using large real-world ETL scenarios.

8. REFERENCES

[1] T. S. A. Simitsis, P. Vassiliadis. Optimizing ETL processes in data warehouse environments. In *Proc. of the 21st Int. Conf. on Data Engineering, ICDE'05*, pages 564–575, 2005. IEEE Computer Society Press.

[2] A. Abelló, J. Samos, and F. Saltor. YAM^2 (Yet Another Multidimensional Model): An extension of UML. *Information Systems*, 32(6):541–567, 2006.

[3] M. Golfarelli and S. Rizzi. A methodological framework for data warehouse design. In *Proc. of the 1st ACM Int. Workshop on Data Warehousing and OLAP, DOLAP'98*, pages 3–9, 1998. ACM Press.

[4] S. Luján-Mora, J. Trujillo, and I. Song. A UML profile for multidimensional modeling in data warehouses. *Data & Knowledge Engineering*, 59(3):725–769, 2006.

[5] E. Malinowski and E. Zimányi. *From Conventional to Spatial and Temporal Applications*. Springer, 2008.

[6] OASIS. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2007.

[7] OMG. Business Process Modeling Notation (BPMN). http://www.omg.org/docs/formal/09-01-03.pdf, 2009.

[8] C. Phipps and K. Davis. Automating data warehouse conceptual schema design and evaluation. In *Proc. of the 4th Int. Workshop on Design and Management of Data Warehouses, DMDW'02*, pages 23–32, 2002. CEUR Workshop Proceedings.

[9] M. Rafanelli, editor. *Multidimensional Databases: Problems and Solutions*. Idea Group, 2003.

[10] O. Romero and A. Abelló. Automating multidimensional design from ontologies. In Song and Pedersen [16], pages 1–8.

[11] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the E/R model for multidimensional paradigm. In *Proc. of the 17th Int. Conf. on Conceptual Modeling, ER'98*, LNCS 1507, pages 105–116, 1998. Springer.

[12] A. Simitsis. Mapping conceptual to logical models for ETL processes. In *Proc. of the 8th ACM Int. Workshop on Data Warehousing and OLAP, DOLAP'05*, pages 67–76, 2005. ACM Press.

[13] D. Skoutas and A. Simitsis. Designing ETL processes using semantic web technologies. In *Proc. of the 9th ACM Int. Workshop on Data Warehousing and OLAP, DOLAP'06*, pages 67–74, 2005. ACM Press.

[14] D. Skoutas and A. Simitsis. Ontology-based conceptual design of ETL processes for both structured and semi-structured data. *Int. Journal on Semantic Web and Information Systems*, 3(4):1–24, 2007.

[15] D. Skoutas, A. Simitsis, and T. Sellis. Ontology-driven conceptual design of ETL processes using graph transformations. In *Journal on Data Semantics XIII*, number 5530 in LNCS, pages 122–149. Springer, 2009.

[16] I. Song and T. Pedersen, editors. *Proc. of the 10th ACM Int. Workshop on Data Warehousing and OLAP, DOLAP'07*, 2007. ACM Press.

[17] R. Torlone. Conceptual multidimensional models. In Rafanelli [9], pages 69–90.

[18] N. Tryfona, F. Busborg, and J. Borch. StarER: A conceptual model for data warehouse design. In *Proc. of the 2nd ACM Int. Workshop on Data Warehousing and OLAP, DOLAP'99*, pages 3–8, 1999. ACM Press.

[19] V. Tziovara, P. Vassiliadis, and A. Simitsis. Deciding the physical implementation of ETL workflows. In Song and Pedersen [16], pages 49–56.

[20] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In *Proc. of the 5th ACM Int. Workshop on Data Warehousing and OLAP, DOLAP'02*, pages 14–21, 2002. ACM Press.