A Model-Driven Framework for ETL Process Development

Zineb El Akkaoui, Esteban Zimányi
Dept. of Computer and Decision Engineering
Université Libre de Bruxelles, Belgium
zelakkao,ezimanyi@ulb.ac.be

Jose-Norberto Mazón, Juan Trujillo
Dep. of Software and Computing Systems
University of Alicante, Spain
jnmazon,jtrujillo@dlsi.ua.es

ABSTRACT

ETL processes are the backbone component of a data warehouse, since they supply the data warehouse with the necessary integrated and reconciled data from heterogeneous and distributed data sources. However, the ETL process development, and particularly its design phase, is still perceived as a time-consuming task. This is mainly due to the fact that ETL processes are typically designed by considering a specific technology from the very beginning of the development process. Thus, it is difficult to share and reuse methodologies and best practices among projects implemented with different technologies. To the best of our knowledge, no attempt has been yet dedicated to harmonize the ETL process development by proposing a common and integrated development strategy. To overcome this drawback, in this paper, a framework for model-driven development of ETL processes is introduced. The benefit of our framework is twofold: (i) using vendor-independent models for a unified design of ETL processes, based on the expressive and well-known standard for modeling business processes, the Business Process Modeling Notation (BPMN), and (ii) automatically transforming these models into the required vendor-specific code to execute the ETL process into a concrete platform.

1. INTRODUCTION

A data warehouse is a "collection of integrated, subject-oriented databases designated to support the decision making process" [3]. By integrating different data sources, a data warehouse allows to provide a complete and accurate view on an organization's operational data, synthesized into a set of strategic indicators, or *measures*, which can be analyzed according to axes of interest, or *dimensions*. Therefore, a data warehouse enables the production of fast and efficient business reports from operational data sources.

The cornerstone component that supplies the data warehouses with all the necessary data is the ETL (Extraction/ Transformation/ Load) process. It is worth mentioning that the data sources that populate a data warehouse are typically heterogeneous and distributed, and ETL processes should therefore cope with these aspects and ensure the expected data quality required by the data warehouse. Therefore, the ETL process development is complex, prone-to-fail, and time consuming [8, 15, 18]. Actually, ETL process development constitutes the most costly part of a data warehouse project, in both time and resources, as stated by a recent Forester Research report[1]. Even more, this report argues that the complexity of integration solutions continues to grow, with higher-quality data and more-robust metadata and auditability requirements.

One of the main causes of this increased complexity is that, in practice, ETL processes are still designed by considering a specific vendor tool from the very beginning of the development process, which raises multiple drawbacks. On the one hand, these tools have a steep learning curve, due to the important effort required to assimilate the manipulated platform philosophy, e.g., the underlying languages with a wide field of functionality assortments, nonstandard ETL development scenarios, complex wizards, etc. On the other hand, the design of an ETL process using a vendor-specific tool is usually performed through a Graphical User Interface (GUI). Thus, the generated design tends to be proprietary since it can only be executed within the tool, and no available mechanisms allow to seamlessly generate its executable code for other platforms.

First attempts to improve this situation consider a conceptual modeling stage for developing ETL processes in a vendor-independent manner [7, 13, 16]. These proposals improve the development of ETL processes, since they allow documenting the ETL processes and supporting the designer tasks. However, they lack effective mechanisms for automatically generating vendor-specific code to execute the ETL process into a concrete platform.

To overcome this problem, the present work proposes a Model-Driven Development (MDD) framework for ETL processes. This framework aims at covering the overall ETL development process, including the automatic generation of vendor-specific code for several platforms. Fig. 1 shows a conceptual overview of our approach and how it is integrated with existing ETL platforms. *Vendor-independent* models of ETL processes are defined using BPMN4ETL, a platform-independent design model of ETL processes based on the Business Process Model Notation (BPMN). From this model

DOLAP'11, October 28, 2011, Glasgow, Scotland, UK.

[1]http://www.forrester.com

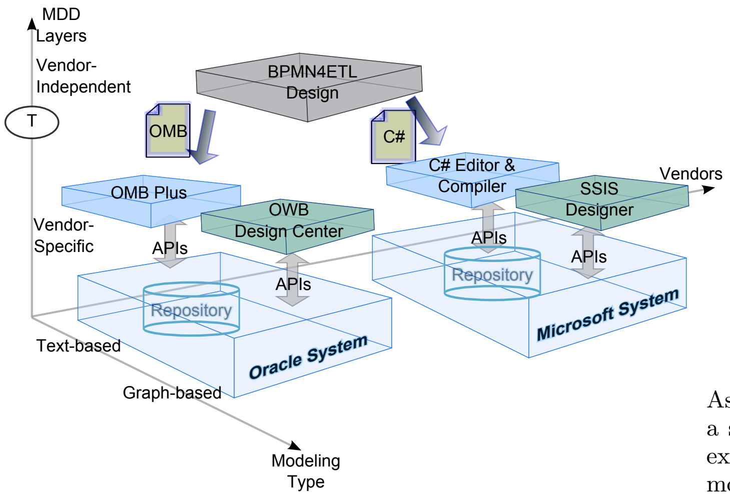


Figure 1: MDD approach for ETL process development

can be derived a *vendor-specific* implementation tailored to several technologies, e.g., Oracle Warehouse Builder (OWB) or SQL Server Integration Services (SSIS).

For creating and managing ETL processes, in addition to the traditional graphical languages, these technologies generally provide programming capabilities by a fourth-generation language (4GL). This 4GL can be either a procedural language (e.g. Oracle Metabase or OMB) or a procedural language (e.g. C#). Transformations between a vendor-independent model and this vendor-specific code are formally established by using model-to-text transformations[2], an OMG standard for transformations from models to text.

The present framework relies on our previous work as follows. First, the BPMN4ETL metamodel for designing ETL processes described in [2] has been improved to be used as the vendor-independent part of our framework. Second, the model-driven approach has been customized from a generic data warehouse approach [5] into a concrete and appropriate implementation for the ETL component. Hence, our framework is motivated by the high-expressiveness of the BPMN graphical language allowing to design any kind of ETL processes, as well as by the performance of the MDD technologies which mainly enhance the automatic implementation into any executable code. More precisely, the contributions of the present work are as follows.

Our first contribution is a vendor-independent metamodel for designing ETL processes that allows designers to focus on the ETL semantics instead of wasting efforts in implementation issues. By using our metamodel the following advantages can be highlighted:

- Reusability: the same ETL process design can be easily reused independently of the specific ETL technology. This constitutes a strong benefit since it avoids losing time in developing the same process multiple times. For example, departments of the same company can easily exchange designed ETL processes by adapting them to their requirements. Another interesting advantage, is the ability to predefine dedicated ETL subprocesses for a frequent usage through the company, regardless to the deployed technologies. For example, an ETL subprocess can be the computation patterns for common strategic indicators or cleansing patterns suitable to the company data.

- Interoperability: using a common design, ETL process models with distinct paradigms can be combined in order to prompt inter-communication or data exchange between ETL processes.

As a second contribution, our model-driven process provides a set of model-to-text transformations to be automatically executed in order to obtain code for different ETL tools from models. These transformations induce a set of benefits:

- Transformations patterns: transformations are organized into technology-independent templates. Therefore, extending our framework by adding new target ETL tools, can be easily performed by obtaining the corresponding code for each template. Since this step can be semi-automated, programming details are reused and high expertise is not more necessary.

- Validation: the user has the possibility to examine and ameliorate the generated code before its execution.

- Performance: since not all the integration issues are best solved by the same tool, the designer should be able to chose the best implementation for each issue. Each module can then be implemented in a particular tool. Moreover, modularity allows sharing the work among several engines, and the correct combination of the whole results may increase execution performance using the validation asset explained below.

- Code quality improvement: the framework should enable code enhancement by using the best practices and lesson learned when designing ETL processes.

The remainder of this paper is organized as follows. After briefly reviewing the related literature in Section 2, Section 3 thoroughly describes the proposed MDD-based framework. In Section 4, we propose a vendor-independent pattern for developing the model-to-text transformations. Then, we show how this transformation pattern can be customized and implemented for the Oracle platform in Section 5. Section 6 concludes the paper and discusses future work.

2. RELATED WORK

Existing approaches for developing ETL process highlight two main axes: (i) designing ETL processes independently of a specific vendor, and then (ii) implementing into an executable code tailored to a certain technology.

On the one hand, several authors [6, 17] argue that ETL processes can be adequately designed by a workflow language. Furthermore, other approaches [9, 10] recommend the use of ontologies for describing data sources and targets, thus providing the necessary information for solving an adequate

[2]http://www.omg.org/spec/MOFM2T/1.0/

ETL process design by inference. Other approaches as [11] apply graph operation rules in order to guide the ETL design steps, also by using ontologies. Unfortunately, building such ontologies is a tedious task since it requires a high correctness and a blow-by-blow description of the data stores. Therefore, we advocate the use of a rich workflow language that provides various enhancements for the designer work without requiring the definition of any ontology.

On the other hand, the implementation of the ETL design has been discussed from many points of view. For example, an attempt has been concerned about the optimization of the physical ETL design through a set of algorithms [14]. Also, a UML-based physical modeling of the ETL processes was introduced by [4]. This approach formalizes the data storage logical structure, and the ETL hardware and software configurations. Although both works deal with interesting adjacent issues about the implementation topic, they mainly miss to produce a code for executing ETL processes. Paradoxically, an exclusive ETL programming approach using the Python language has been claimed by [12]. Yet, this approach omits to supply an independent-vendor design which decreases the reusability and easy-of-use of the provided framework.

Finally, other works propose to take into account both ETL development axes: designing and code generation. For example, a conceptual metamodel for designing ETL processes based on BPMN, and an approach for generating its corresponding BPEL code is described in [2]. Also, some authors propose a Model-Driven Architecture approach for designing the ETL processes by means of the UML Activity Diagram, and then obtaining a representation of the ETL process through specific-vendor metamodels defined manually, [5] . However, both works lack in a wider support for more than one concrete implementation platform.

Hence, the current work constitutes an extension of these two previous proposals [2, 5]. It refines the metamodel in [2] and disseminates the code generation into any ETL programming language. Further, it improves the transformation proposed in [5] by directly obtaining code corresponding to the target platform, bypassing the laborious task of defining an intermediate representation of the target tool.

3. MODEL-DRIVEN FRAMEWORK

As already said, the present work addresses the ETL process development using a Model-Driven Development (MDD) approach. In this section, we concretely show how this approach allows to organize the various components of this framework in order to efficiently perform the design and implementation phases of the ETL process development.

3.1 MDD-Based Framework

MDD is an approach to software development where extensive models are created before source code is written. As shown in Fig. 2, the MDD approach defines four main layers (see Meta-Object Facilities (MOF)[3]): the Model Instance layer (M0), the Model layer (M1), the Meta-Model layer (M2), and the Meta-Meta-Model layer (M3).

[3]http://www.omg.org/spec/MOF/2.0/

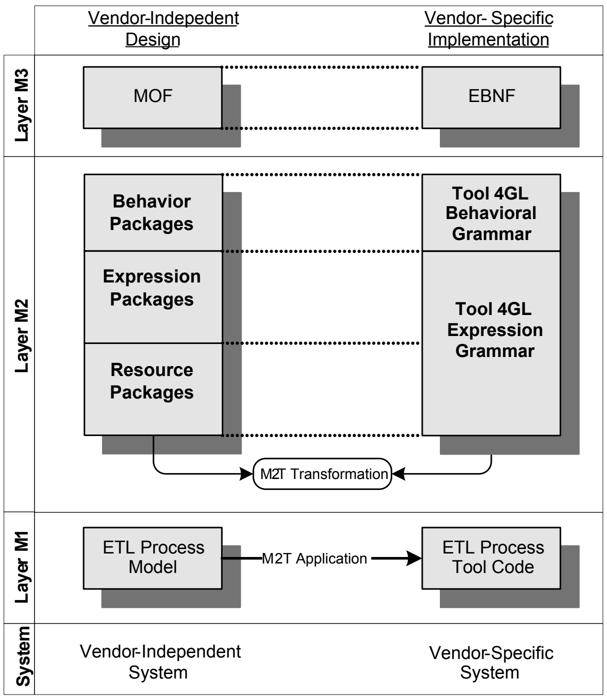


Figure 2: MDD layers for the ETL development framework

The Model Instance layer (M0) is a representation of the real-world system where the ETL process design and implementation are intended to perform. This may be represented, respectively, by a vendor-independent graphical interface and by a vendor-specific ETL engine. At the Model layer (M1), both the ETL Process Model is designed and the ETL process code is derived by applying a set of transformations, thus moving from the design to the implementation. The Meta-Model layer (M2) consists of the BPMN4ETL metamodel that defines ETL patterns at the design phase, and a 4GL grammar at the implementation phase. Finally, the Meta-Meta-Model level (M3), corresponds to the MOF meta-metamodel at the design phase, while it corresponds to the Backus Naur Form (BNF) at the implementation phase.

In the following, we focus on the two M1 and M2 layers since they describe the ETL semantics. We describe both layers by means of the small data warehouse example shown in Fig. 3. In this example, two types of data sources feed the data warehouse with the necessary data (see Fig. 3): a database and a flat file. These sources are based on the relational and the recordset paradigms, respectively, for tables and files, as defined in the Common Warehouse Metamodel[4]. The database contains the Product and Category tables, where each product belongs to a particular category. A similar information is provided by the Product_Category flat file source, with the difference that this information about products and their categories is combined within a same schema. The target data warehouse contains a simple fact table Category_Fact and its product hierarchy, as displayed in Fig. 3b. The product hierarchy is composed of both the Category_Dimension and the Product_Dimension tables.

[4]http://www.omg.org/spec/CWM/1.1/

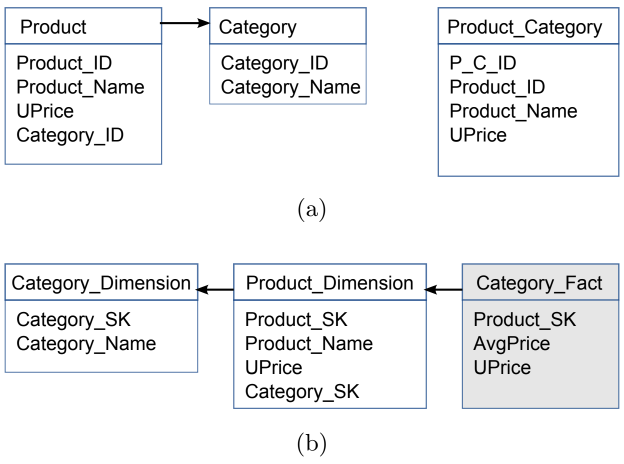


Figure 3: (a) Data source schemas; (b) Data warehouse target schema.

3.2 Vendor-Independent Design

This section describes the vendor-independent design side of the framework shown in Fig. 2. This part comprises the creation of the ETL model based on the set of constructs provided by the BPMN4ETL metamodel.

3.2.1 *BPMN4ETL Model (M1)*

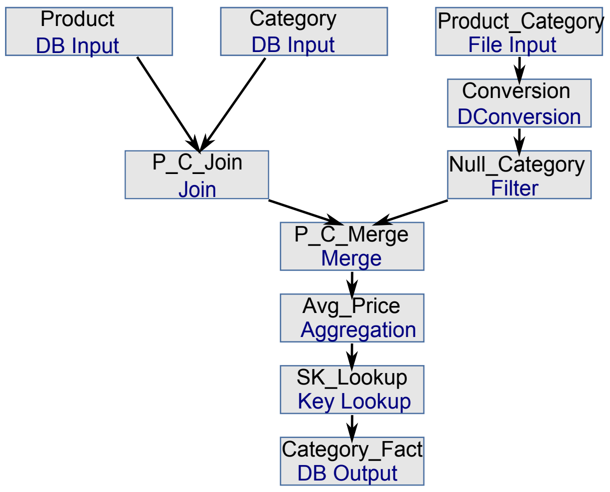


Figure 4: ETL process model for the Category_Fact load

The ETL model that provides the example data warehouse with the necessary data from the data sources is depicted in Fig. 4. This process is mainly composed of tasks of multiple types: starting from the extractions (e.g., DB Input and File Input), passing through the transformations (e.g., Filter, Join, ...) and ending by the loading (e.g., DB Output) into the data warehouse. In the example, the data from the Category table requires the creation of a new field using the variable derivation VDerivation task. Then, the product and category data can be joined together. On the other hand, extracting data from a file generates fields of a character data type. Hence, a typical behavior when dealing with files consists of converting these character types into their original data type, depicted as a DConversion task in our example. Also, the data from the file should be filtered so as to drop null Product_ID records, since there is no way to validate this constraint within a file, in contrast with a table. Then, both flows from the database and the file can be merged in order to compute the target measures for the fact table, the AvgPrice. Finally, a key lookup is included to check the referential integrity between the processed data and the old data in the Category_Fact table, before its loading.

It is worth mentioning that each task can be configured by customizing its behavior in order to fulfill the design objective. For example, the extraction and loading tasks hold many configurations, such as the loading strategies and the automatic data type mapping. Also, the transformations are composed of various computations, branching and merging tasks, and each of these categories can be customized through a set of parameters, like conditions, computations and input/output cardinalities. For simplicity, such configurations are not displayed in Fig. 4.

3.2.2 *BPMN4ETL Metamodel (M2)*

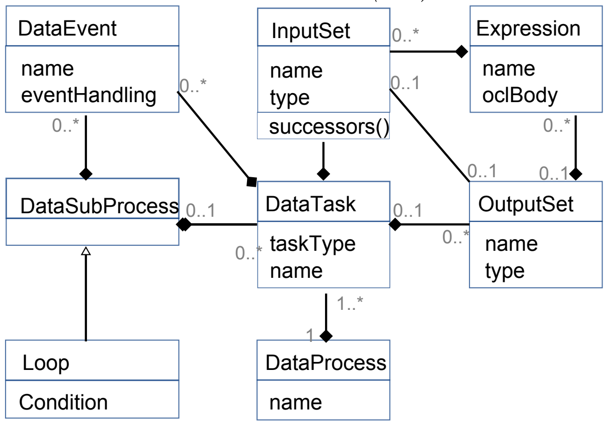


Figure 5: Excerpt of the data process package

As we have said, the example ETL process in Fig. 4 is designed according to the BPMN4ETL metamodel, shown in Fig. 5. The metamodel is mainly based on BPMN, a de facto standard notation for modeling business processes. BPMN provides a common and understandable framework for describing business processes independently of the used tools. BPMN allows users to express various pertinent features about processes, such as *events* allowing the process to interact with the external world and *containers* enabling to structure the business process and tracing its boundaries.

Our BPMN-based metamodel is structured in several packages. The root package is the ControlProcess, which captures the whole ETL process populating a data warehouse through multiple DataProcesses. Fig. 5 shows an excerpt of the data process package, a core part of ETL processes.

Typically, a DataProcess depicts the complete data flow from a set of data sources to a target data warehouse. It involves multiple DataTasks belonging to various taskTypes, such as a join, filter, merge, variableDerivation, etc. Also, the DataEvent class enables the user to custom the exception handling for each data task. Moreover, the subsequence of data tasks sharing the same objective or similar characteristics may be organized through subprocesses, represented by the DataSubProcess class. The flows incoming from and outgoing to a data task are respectively represented by the InputSet and OutputSet classes. The Expression class captures the expressions applied by a data task on its input sets in order to produce its output set.

Additional packages of our metamodel are Condition, Computation and Query, representing different types of expressions. Other packages capture fundamental aspects about data storage in ETL processes. The Resource package represents the data resources from/to which data is extracted/loaded, and the Intermediate Data Store (IDS) package, represents the stagging area for storing non-persistent data participating in the ETL process.

3.3 Vendor-Specific Implementation

In this section, we highlight the implementation components of the framework, i.e., the ETL process code and the 4GL grammar. We use Oracle as example target system.

3.3.1 Code (M1)

The Oracle MetaBase (OMB) scripting language for Oracle Warehouse Builder (OWB) can be assimilated to SQL language for the Oracle database. A snippet of the OMB code corresponding to the Category_Fact load process can be seen in Listing 1. Creating a data process (MAPPING) in OMB requires first to create a project and to define a module where to locate the mapping. Also, LOCATIONs for the resources used by the mapping should be established. Then, the mapping is defined with its data tasks (OPERATORs), and matched to each other using connections.

In our example, the merge task is created using the ADD SET_OPERATION OPERATOR command. The task properties are configured using the SET VALUES command, e.g., the type property of the merge task has the value a UNION. Then, this task is linked to the join task using the ADD CONNECTION command. Connecting operators in OMB is done by connecting each attribute of the operators inputs and outputs GROUPs, or by connecting all the input or output attributes simultaneously. For instance, the merge operator has two input connections incoming from the join and filter operators, and one output connection ongoing to the aggregation operator.

Note finally that additional code is needed in order to execute the mappings, such as the creation of the hosting project, the source connections, and the deployment code. This code can be created semi-automatically since it is common among all data processes and independent from the ETL model to implement.

Listing 1: OMB code for our Category_Fact load

```
OMBCREATE ORACLE_MODULE 'STAGING_TABLES'

OMBCREATE LOCATION 'MY_LOCATION'
SET PROPERTIES (TYPE, VERSION,
DESCRIPTION, BUSINESS_NAME)
VALUES ('ORACLE_DATABASE', '10.2',
'this is a location', 'location')

OMBCREATE MAPPING 'CATEGORY_FACT_MAP'

#DATA INPUT TASKS
OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD TABLE OPERATOR 'PRODUCT'\
  BOUND TO TABLE 'PRODUCT'

OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD TABLE OPERATOR 'CATEGORY'\
  BOUND TO TABLE 'CATEGORY'

OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD FLAT_FILE OPERATOR 'PRODUCT_CATEGORY'\
  BOUND TO FLAT_FILE '/ETLTOOL/PRODUCT_CATEGORY'

#TRANSFORMATIONS
#JOIN
OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD JOINER OPERATOR 'P_C_JOIN'
   SET PROPERTIES (JOIN_CONDITION)
    VALUES ('INGRP1.CATEGORY_ID = INGRP2.CATEGORY_ID')

#CONVERSION
OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD EXPRESSION OPERATOR 'CONVERSION'\
  ALTER ATTRIBUTE 'P_C_ID' OF GROUP
  'INGRP1' OF OPERATOR 'CONVERSION'\
   SET PROPERTIES (DATATYPE,PRECISION,SCALE)
   VALUES (INTEGER,30,0)\
    ...

#FILTER
OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD FILTER OPERATOR 'NULL_CATEGORY'\
   SET PROPERTIES (FILTER_CONDITION)\
    VALUES ('PRODUCT_CATEGORY.CODE_CATEGORY \
    IS NULL')

#MERGE
OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD SET_OPERATION OPERATOR 'MERGE'\
  SET PROPERTIES (SET_OPERATION)\
   VALUES ('UNION')

#AGGREGATION
OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
 ADD AGGREGATOR OPERATOR 'AVGPRICE'\
  SET PROPERTIES (GROUP_BY_CLAUSE, HAVING_CLAUSE)\
   VALUES ('INGRP1.CATEGORY_CODE', \
   'INGRP.CATEGORY_ID<>NULL')
 ...

#ADD CONNECTIONS BETWEEN TASKS
OMBALTER MAPPING 'CATEGORY_FACT_MAP'\
ADD CONNECTION FROM GROUP 'OUTGRP1'\
OF OPERATOR 'JOIN' TO GROUP 'INGRP1'\
OF OPERATOR 'MERGE' BY NAME
 ...
```

3.3.2 Programming Grammar (M2)

Grammars are formalisms for defining (programming) languages in the same way that metamodels represent the language of models. As shown in Fig. 2, in our framework we used the Extended Backus-Naur Form (EBNF) for defining the OMB and C# grammars. Thus, the OMB grammar defines the syntax structure for valid OMB statements. For instance, in order to generate the merge task statement shown in Listing 1, the part of grammar used during the production is shown in Listing 2.

Listing 2: OMB Grammar Snippet

```
alterMappingCommand = OMBALTER MAPPING "mappingName"
 "alterMapDetailClause"
  mappingName = "QUOTED_STRING"
  alterMapDetailClause = ADD "addOperatorClause"
  addOperatorClause = "operatorType" OPERATOR "operatorName"
  [SET "setPropertiesClause"]
  operatorType = "UNQUOTED_STRING"
  operatorName = "QUOTED_STRING"
  setPropertiesClause = PROPERTIES "propertyKeyList" VALUES
          "propertyValueList"
  propertyKeyList = "(" "propertyKey" { ","
    "propertyKey" } ")"
  propertyValueList = "(" "propertyValue" { ","
    "propertyValue" } ")"
  propertyKey = "UNQUOTED_STRING"
  propertyValue = ( "QUOTED_STRING" | "INTEGER_LITERAL" |
          "FLOATING_POINT_LITERAL" )
```

4. TRANSFORMATIONS

Transformations are used for automatically generating the ETL code from an input ETL model. In this section, we define the transformations as matching statements between the metamodel and the grammar at the M2 layer, in order to be executed at the M1 layer. At the same time, we show

how the abstract patterns for such transformations provide useful guidance when developing code generators for new ETL tools. Our approach uses OMG's model-to-text (M2T) standard for expressing the transformations.

4.1 M2T Transformations

M2T organizes transformations in a set of *templates*, responsible for generating code for the input model elements. Each template contains the code portions corresponding to a single metamodel concept. This code may be static or dynamic. Static code is replicated literally during the execution. Dynamic code corresponds to OCL expressions specified using the metamodel concepts. These expressions are computed on the input model during execution. In practice, the input model elements are sequentially iterated, and for each element, the corresponding template code is generated and appended to an output file.

Thus, the transformations between the BPMN4ETL metamodel and the OMB grammar are performed by means of templates. However, the template code depends on target ETL tool. For this reason, we show next how an abstract level for the transformation templates can be useful, since it specifies the required templates, and describes their common processing order across ETL tools.

4.2 Transformation Patterns

Since ETL code for different software tools follows a common strategy, we predefined patterns of abstract templates, such as the one shown in Fig. 6. These patterns provide to the developer a better view of the typical order in template building. Then, the code can be semi-automatically generated from the tool. Similarly to the object-oriented programming paradigm, each abstract template can be assimilated to an *abstract class*, which can be implemented by customizing its code. Finally, these patterns may be adapted to the potential dissimilarities between programming languages, as it will be discussed in Section 5.

The abstract pattern highlighted in Fig. 6 shows the creation of new code from an ETL model. Other patterns address maintaining and managing this code. This pattern algorithm depicts the following steps: It starts by creating the main control process template addNewCP and then iteratively creating its components. A main component is the data process. Thus, for each existing data process, a new empty template is created addNewDP with all connection templates corresponding to the involved resources addConnection. Note that this template is applied for each resource related to the data process depicted with the foreach loop. The data tasks templates addDTask are thereupon obtained according to their types, and they are linked to each other matchDTasks in the same order of the input model. Similarly, other control tasks templates of the control processes are created and matched. Finally, subprocess, loop, and event features are added to the process components.

5. APPLYING TRANSFORMATIONS

This section describes the Eclipse-based implementation of our framework, specifically, the code generation. It emphasizes an excerpt of the transformation code that should be applied to the ETL model in Fig 4 in order to get the corresponding OMB code.

Eclipse is a software development environment comprising an extensible plug-in system. It provides the Eclipse Modeling Framework (EMF), which comprises a number of model-driven development capabilities like modeling, inter-model transformations, and code generation. More precisely, EMF allows the definition of models and metamodels by means of the Ecore tools. In practice, metamodels are created using the Ecore meta-metamodel, which is the implementation of MOF in Eclipse. From these metamodels it is possible to generate the genmodel, which is a set of Java classes that represent each metamodel concept. The genmodel is useful for building dedicated tools, such as editors, to create and manage the models corresponding to our metamodel.

As shown in Fig. 7, the BPMN4ETL metamodel has been implemented by Ecore. Also, in order to guarantee the correctness of BPMN4ETL models created by users, the metamodel implementation contains two types of validation. Firstly, it exploits EMF default diagnostic methods for evaluating the syntactic conformance of the created models with their metamodel. Secondly, we have build specific validation rules in order to guarantee the semantics correctness. For example, some validation rules that we have implemented for a data task element are: no InputSet for data input tasks, no OutputSet data output tasks, and at least one connection for each data task (see Fig. 7). Therefore, based on this metamodel implementation, Eclipse enables to build an editor for creating valid ETL models.

Further, EMF provides model-to-text transformations support through Acceleo, the Eclipse tool for programming code generators that follows OMG's model-to-text specification. In Acceleo, each set of related transformations is called a module. Each module may contain one or several transformation templates.

Listing 3: Concrete templates code snippet for OMB

```
[template public addDProcess(dprocess : DataProcess)]
    #Creation of the corresponding mapping to the
    #data process
    OMBCREATE MAPPING '[dprocess.name.toString()/]'\
     [for (t : DataTask | dprocess.dataTasks)]
     [t.addTask()/]
     [/for]
[/template]

[template public addDTask(t : DataTask)]
    #### Add the task [t.name/] to the mapping ####
    OMBALTER MAPPING '[t.dataProcess.name/]'
     [if (t.taskType.toString() = 'DataInput')]
     [t.addDITask()/]
     [/if]
     [if (t.taskType.toString() = 'VariableDerivation')]
     [t.addVDTask()/]
     [/if]
     [if (t.taskType.toString() = 'Join')]
     [t.addJoTask()/]
     [/if]
     [if (t.taskType.toString() = 'DataOutput')]
     [t.addDOTask()/]
     [/if]
     [if (t.taskType.toString() = 'Merge')]
     [t.addMeTask()/]
     [/if]
     ...
[/template]
...
[template public convertCondition(cc : Condition)]
     [if cc.cCondition.oclIsUndefined()]
     [cc.sCondition.convertSCondition()/][/if]
     [if not(cc.cCondition.oclIsUndefined())]
     [cc.cCondition.convertCondition()/]
     [cc.sCondition.convertSCondition()/][/if]
[/template]
```

Listing 3 shows an excerpt of the templates for transforming a data process, some data tasks and a conditional expression into OMB code. Further, each abstract template includes

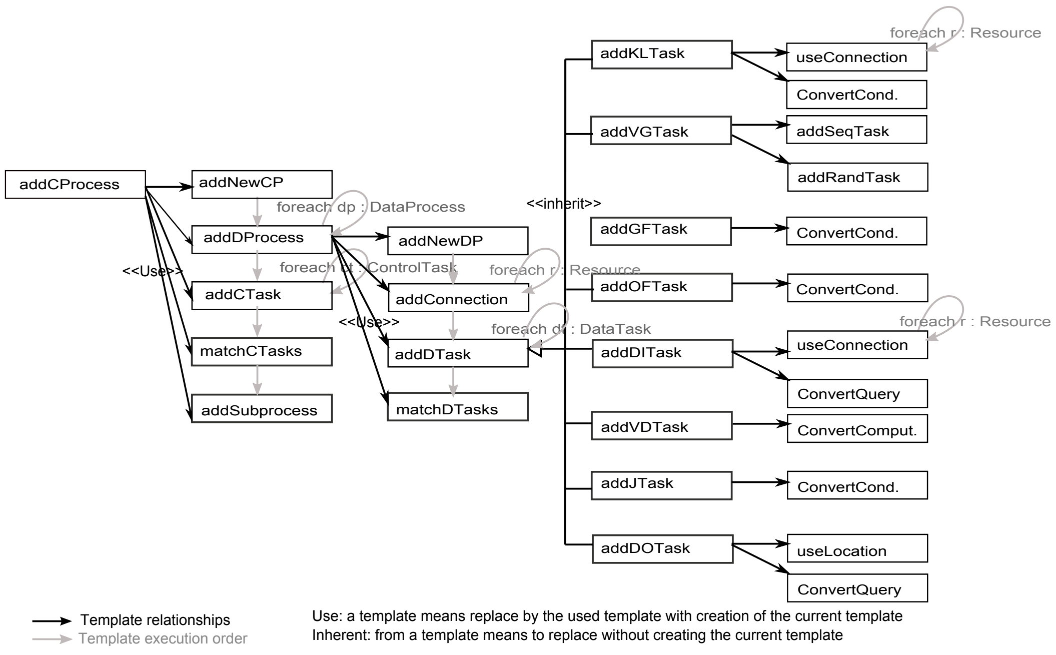


Figure 6: Abstract templates pattern for ETL process creation

the necessary code to be generated by the transformation. For example, the abstract template addDProcess depicted in Fig. 6 is implemented by the Acceleo template addDProcess, an excerpt of which is defined in Listing 3 and can be seen in the Acceleo editor in Fig. 7.

Concretely, the template addDProcess creates a new data process and calls the templates for creating all the entailed data tasks. Indeed, depending on the type of the operator taskType, a specific template is applied, e.g. addJoTask and addMeTask which aims, respectively, at adding a join and a merge tasks. The final template displays a recursive template which is responsible of transforming a combined condition into the expression language of OMB, i.e., SQL code. The combined condition is represented in our metamodel in a recursive tree structure. With regards to this structure, the transformation template calls the simple condition template convertCondition in the left part of the generated code, and recall the combined condition template for its right part.

When running the resulted OMB code generated by the transformations of our example and then opening the OWB Design Center, it is possible to verify the conformance of the resulted process with the initial ETL model.

It is worth mentioning that some custom templates, not mentioned in the abstract template pattern, may be added during the implementation. These templates differ among ETL tools, thus they need to be specified for each target tool. Further, the generated code is closely linked to how the transformations are programmed. In fact, the transformation algorithm through the abstract template pattern viewed in Section 4 is the most direct way to create the ETL code. However, a more optimized code can be generated by using specific 'macros' provided by tools. For instance, the tasks of type Lookup and DB Input in the example process in Fig. 4 can be designed differently. Both tasks can be combined and mapped to the cube operator in the OWB, instead of a direct mapping to a lookup and a table operators such as those proposed by our approach.

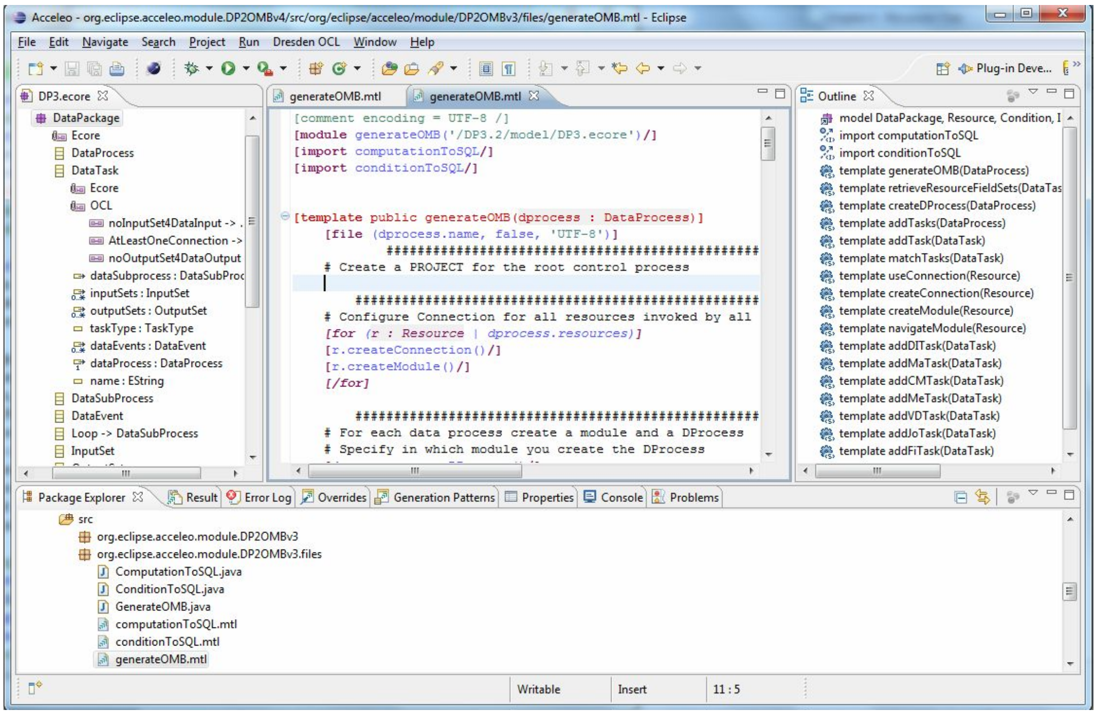
6. CONCLUSIONS AND FUTURE WORK

Even though data warehouses are used since the early 1990s, the design and modeling of ETL processes is still accomplished in a vendor-dependent way by using tools that allow to specify them according to the schemas represented in concrete platforms. Therefore, their design, implementation and even more their maintenance must be done depending on the target platform.

In this paper, we have provided, to the best of our knowledge, the first modeling approach for the specification of ETL processes in a vendor-independent way and the automatic generation of its corresponding code in commercial platforms. This is done thanks to the expressiveness of our BPMN-based metamodel and to the model-driven development capabilities, provided by Eclipse and Acceleo, in code generation.

Our approach considers Oracle and Microsoft as representative target implementation and execution tools, although only Oracle's solution has been described in this paper for space reasons. Moreover, it offers some new techniques in order to guide developers in building other code generators for new platforms.

Currently, our framework covers the design and implementation phases of the ETL process development. One future work expects to extend this framework in order to handle the whole ETL development life-cycle, i.e., by involving the analysis phase as well. This work should enhance the quality of the generated code by our approach by means of: i) en-


Acceleo - org.eclipse.acceleo.module.DP2OMBv4/src/org/eclipse/acceleo/module/DP2OMBv3/files/generateOMB.mtl - Eclipse
File Edit Navigate Search Project Run Dresden OCL Window Help
Plug-in Deve...
DP3.ecore
generateOMB.mtl
generateOMB.mtl
Outline
Package Explorer
Result
Error Log
Overrides
Generation Patterns
Properties
Console
Problems
Writable
Insert
11 : 5

```
DataPackage
  Ecore
  DataProcess
  DataTask
    Ecore
    OCL
      noInputSet4DataInput -> .
      AtLeastOneConnection ->
      noOutputSet4DataOutput
    dataSubprocess : DataSubProc
    inputSets : InputSet
    outputSets : OutputSet
    taskType : TaskType
    dataEvents : DataEvent
    dataProcess : DataProcess
    name : EString
  DataSubProcess
  DataEvent
  Loop -> DataSubProcess
  InputSet
```

```
[comment encoding = UTF-8 /]
[module generateOMB('/DP3.2/model/DP3.ecore')/]
[import computationToSQL/]
[import conditionToSQL/]


[template public generateOMB(dprocess : DataProcess)]
    [file (dprocess.name, false, 'UTF-8')]
            ##################################################
    # Create a PROJECT for the root control process

        ####################################################
    # Configure Connection for all resources invoked by all
    [for (r : Resource | dprocess.resources)]
    [r.createConnection()/]
    [r.createModule()/]
    [/for]

        ####################################################
    # For each data process create a module and a DProcess
    # Specify in which module you create the DProcess
```

```
model DataPackage, Resource, Condition, I
import computationToSQL
import conditionToSQL
template generateOMB(DataProcess)
template retrieveResourceFieldSets(DataTas
template createDProcess(DataProcess)
template addTasks(DataProcess)
template addTask(DataTask)
template matchTasks(DataTask)
template useConnection(Resource)
template createConnection(Resource)
template createModule(Resource)
template navigateModule(Resource)
template addDITask(DataTask)
template addMaTask(DataTask)
template addCMTask(DataTask)
template addMeTask(DataTask)
template addVDTask(DataTask)
template addJoTask(DataTask)
template addFiTask(DataTask)
```

```
src
  org.eclipse.acceleo.module.DP2OMBv3
  org.eclipse.acceleo.module.DP2OMBv3.files
    ComputationToSQL.java
    ConditionToSQL.java
    GenerateOMB.java
    computationToSQL.mtl
    conditionToSQL.mtl
    generateOMB.mtl
```

Figure 7: Acceleo implementation

hancing the transformation implementation using some existing knowledge-based techniques, and ii) by building the necessary metrics for proposing the 'best' implementation related to a certain ETL process.

7. REFERENCES

[1] J. Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.

[2] Z. El Akkaoui and E. Zimányi. Defining ETL worfklows using BPMN and BPEL. In *DOLAP'09*, pages 41–48. ACM Press.

[3] W. Inmon. *Building the Data Warehouse*. Wiley, 2002.

[4] S. Luján-Mora and J. Trujillo. Physical modeling of data warehouses using UML. In *DOLAP'04*, pages 48–57, 2005. ACM Press.

[5] J. Mazón and J. Trujillo. An MDA approach for the development of data warehouses. *Decision Support Systems*, 45(1):41–58, 2008.

[6] A. Simitsis. Mapping conceptual to logical models for ETL processes. In *DOLAP'05*, pages 67–76, 2005. ACM Press.

[7] A. Simitsis and P. Vassiliadis. A methodology for the conceptual modeling of ETL processes. In *CAiSE'03 Workshops*, pages 305–316, 2003. CEUR Workshop Proceedings.

[8] A. Simitsis and P. Vassiliadis. A method for the mapping of conceptual designs to logical blueprints for ETL processes. *Decision Support Systems*, 45(1):22–40, 2008.

[9] D. Skoutas and A. Simitsis. Designing ETL processes using semantic web technologies. In *DOLAP'06*, pages 67–74, 2005. ACM Press.

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

[11] 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.

[12] C. Thomsen and T. Pedersen. pygrametl: A powerful programming framework for extract-transform-load programmers. In *DOLAP'09*, pages 49–56. ACM Press.

[13] J. Trujillo and S. Luján-Mora. A UML based approach for modeling ETL processes in data warehouses. In *ER'03*, LNCS 2813, pages 307–320, 2003. Springer.

[14] V. Tziovara, P. Vassiliadis, and A. Simitsis. Deciding the physical implementation of ETL workflows. In *DOLAP'07*, pages 49–56, 2007. ACM Press.

[15] P. Vassiliadis, A. Simitsis, and E. Baikous. A taxonomy of ETL activities. In *DOLAP'09*, pages 25–32. ACM Press.

[16] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. *Information Systems*, 30(7):492–525, 2005.

[17] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In *DOLAP'02*, pages 14–21, 2002. ACM Press.

[18] L. Wyatt, B. Caufield, and D. Pol. Principles for an ETL benchmark. In *TPCTC'09*, LNCS 5895, pages 183–198, 2009. Springer.