A Framework for ETL Systems Development

Mário Sergio da Silva[1], Valéria Cesário Times[1], Michael Mireku Kwakye[2]

[1] Informatics Center - Federal University of Pernambuco, 50733-970, Recife-PE, Brazil
{mss6, vct}@cin.ufpe.br
[2] School of Electrical Engineering and Computer Science - University of Ottawa, Canada
mmire083@uottawa.ca

Abstract.

There are many commercial Extract-Transform-Load (ETL) tools, of which most of them do not offer an integrated platform for modeling processes and extending functionality. This drawback complicates the customization and integration with other applications, and consequently, many companies adopt internal development of their ETL systems. A possible solution is to create a framework to provide extensibility, reusability and identification of hotspots. Although, most academic ETL frameworks address the development of conceptual frameworks, application-oriented tools and modeling of ETL processes, they do not include a programming framework to highlight the aspects of flexibility, extensibility and reusability in the ETL systems. We present *FramETL*, which is a novel framework for developing ETL systems that offers a programmable and integrated process modeling environment, and allows the creation of customized ETL transformations. To illustrate the *FramETL* applicability in different domains, we extended our approach to facilitate the ETL processes of data warehouses modeled as star schemas, and another example to define data integration processes in a cost accounting application was also addressed. The evaluation results showed that *FramETL* is a programmable and extensible framework, offers a platform to design distinct ETL applications for modeling and specializing processes in an integrated environment. Moreover, our proposal of a programming interface which conforms to the fundamentals of ETL formulations, allowed the reuse of common transformations in the implementation of our examples of applications, while enabled the documentation of the flexibility points that facilitated the creation of customized transformations for different ETL application domains.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Language Constructs and Features—*Frameworks*; H.4 [**Information Systems Application**]: Miscellaneous—*ETL*

Keywords: data integration, datawarehouse, ETL application, software framework

1. INTRODUCTION

Extract-Transform-Load (ETL) tools are aimed at retrieving operational data from transaction processing systems, executing data transformations and populating them into decision-making databases [Kimball and Caserta 2004]. Most of the transformed data are often loaded into Data Warehouses (DW) [Inmon 2002], which are dimensional databases usually structured according to star schemas and accessed by decision support systems, such as Online Analytical Processing (OLAP) tools and Business Intelligence (BI) applications [March and Hevner 2007]. About 70% of resources to implement a DW are consumed during the ETL project [Li 2010] because the development of this kind of project is critical and time consuming since the generation of incorrect data leads to wrong decision makings [Trujillo and Mora 2003; Dayal et al. 2009]. There are currently many commercial open-source ETL suites available, namely, Pentaho Data Integration, CloverETL Software and Data Integration Talend Open Studio, amongst others. Also, vendors of DBMS have started embedding ETL functionality into their tools. However, a close examination of all these commercial systems shows that the ETL processes are modeled using predefined GUIs. Although some of them offer the

flexibility to create specialized ETL transformations, their modeling and extension environments may not be integrated and may be completely distinct. This complicates the customization for specific business and integration with other applications, and it requires the knowledge of different notations for each environment. Moreover, there is lack of standardization of the set of ETL mechanisms and of their graphical representations, as well as there is no consensus on the scripting languages available on the interfaces of these tools.

The lack of customizable ETL tools with modeling and extension environments integrated has led many companies to ignore the use of GUI-based tools and opt for the use of general-purpose programming platforms in the implementation of ETL processes [Muñoz et al. 2009; Vassiliadis et al. 2009]. Also, ETL tools with specialization interfaces based on source code have been seen as one of the most promising approaches[1], and more recently, experienced developers have argued that in many cases, textual modeling languages offer more advantages than those based on graphical modeling interfaces [Mazanec and Macek 2012].

The use of programming platforms in companies represents the internal software development for ETL and allows the modeling and specialization of ETL processes in a programmable and unified environment. This increases the level of customization of processes and the integration with existing systems in enterprises. On the other hand, the flexibility provided by a programming language lead to the handling of data in an arbitrary manner, which in turn neglects the basic ETL foundations [Kimball and Caserta 2004]. A possible solution to this drawback is creating a software framework for ETL. There has been some amount of research study about ETL process modeling approaches, ETL software architectures and ETL frameworks [Vassiliadis et al. 2002; Simitsis 2005; Mora 2005; Thomsen and Pedersen 2009; Krishna and Sreekanth 2010; Wang and Ye 2010; Akkaoui et al. 2011; Awad et al. 2011; Wojciechowski 2011]. However, most of them are conceptual frameworks for ETL because they do not address frameworks for ETL system development. Also, they ignore important issues like reuse and applicability because they do not define points of stability and flexibility in their software architecture and provide specific framework solutions for a given application domain, respectively.

In this article, we propose *FramETL*, which is a novel *software framework* for application development in the field of ETL. This framework offers a programmable and integrated environment for modeling and process execution, and enables the creation of custom ETL mechanisms. Our framework proposal is based on the concepts of flexibility, extensibility, reusability and inversion of control, as recommended in the literature on frameworks and object-orientation [Fayad and Schmidt 1997; Johnson 1997; Fayad et al. 1999; Braga and Masiero 2004]. The architecture of *FramETL* offers a programming interface containing components, such as inventory of data environments, extractors, aggregators, filters applicators, joints, unions and loaders of data, which encapsulate the fundamentals of ETL available in the literature [Kimball and Caserta 2004]. These components are reused by applications of *FramETL* for the modeling of their ETL processes, though its flexibility allows the creation of other components that encapsulate transformation rules to a specific area of ETL. A specialized component allows the construction of ETL processes that are not possible or would require the combined use of many generic components of the framework.

To evaluate our work, two real ETL-based applications were developed using the *FramETL* framework. In the first application, *FramETL* was specialized to facilitate the loading of transactional data into a DW modeled as a star schema [Inmon 2002; Kimball and Ross 2002]. Thus, an ETL process was implemented for building a data mart for sales in an industry of mineral water bottling. In the second application, *FramETL* was specialized in the business domain of cost accounting to allow the allocation of indirect costs, known as overhead proration [Banerjee 2006; Rom 2008]. For this, we implemented a process of data integration between two different DBMSs datasets related to shrimp farming in captivity. Besides the integration, data transformations were performed to distribute the

[1]Nine BI Megatrends for 2009: http://www.informationweek.com/news/software/bi/212700482

overhead among the shrimp production units, called as shrimp ponds.

This article is organized into five sections. Section 2 outlines the recent research studies on ETL frameworks for system development. Section 3 presents the proposed framework for ETL systems development, while Section 4 describes the two *FramETL* applications that were developed for evaluating our work. In Section 5, we conclude the article by summarizing the main contributions and suggesting areas of future work.

2. LITERATURE REVIEW

In [Mora 2005], researchers extended the UML conceptual language primarily for DW, and then represented the most common transformations of ETL processes modeling. In their recent study [Akkaoui et al. 2011], they presented an adaptation of the Business Process Modeling Notation (BPMN), called BPN4ETL, which unifies the modeling of ETL processes for the most known commercial ETL tools. Krishna [Krishna and Sreekanth 2010] presented a web architecture that distributes the activities of extraction and load on the server side, and the activity of processing on the client side. Awad [Awad et al. 2011] proposed a conceptual framework based on the Service Oriented Architecture (SOA) to minimize the strong coupling between components of most current ETL tools. In [Wang and Ye 2010], the authors also discussed a framework-oriented service, which generates SQL code automatically from metadata stored in a DW and other data sources. Furthermore, in [Wojciechowski 2011], a framework to handle changes in the structure of data sources is discussed, and an algorithm for semi-automatic repair of ETL processes is addressed. However, none of these studies is a programmable framework for developing systems ETL.

The area of software framework for developing ETL processes is the focus of the work described in this article and hence, a comparison analysis between our ETL software framework proposal and the only two related studies found in the literature is given next. In [Vassiliadis et al. 2005], a framework to represent logic abstractions of a ETL process is proposed, which is called Arktos and provides some ETL predefined transformations that allow customization by the use of a declarative language, called LDL notation. A metamodel with points of flexibility is presented together with a reuse architecture, but the authors do not give examples of reusability and flexibility for different domains of application. Also, the adoption of a declarative notation for describing metadata may generate complex ETL processes, promote low readability, and make the integration between their framework and other applications difficult.

The framework PygramETL [Thomsen and Pedersen 2009] is another related work, which is a programmable framework for developing ETL applications. PygramETL is aimed at optimizing the processing of large volumes of data, and was used in the implementation of physical parallelism to take advantage of the current multi-core processors [Thomsen and Pedersen 2011]. However, the flexibility and generality properties of PygramETL have not been evaluated so far, as only the performance of PygramETL processes has been evaluated. Also,its architecture does not provide the identification of areas of flexibility, which facilitates the process of specialization and instantiation. Another framework based on PygramETL, called ETLMR [Liu et al. 2011], is based on MapReduce and enables the building of DW stored in cloud computing environments. While PygramETL and ETLMR are directed to the optimization of ETL processes for building DW, *FramETL* is a more general and flexible framework for enabling the creation of data repositories and customized ETL transformations.

Table I shows the comparison criteria used in the analysis among our work and the three related studies found in the literature. These criteria are: architecture reusability (1), identification of hotspots (2), specification of design patterns (3) used in the development of the framework, evaluation results for more than a single application domain (4), evaluation results related to reusability and

extensibility (5), use of a general purpose programming language for customization and instantiation of the metadata (6), and performance evaluation results (7).

3. THE FRAMETL FRAMEWORK

The *FramETL* framework provides a software platform for the development of flexible and reusable ETL systems. This framework offers an integrated environment to model processes and extends functionalities by using a programming language, without relying on GUI. For the specification of *FramETL* that is outlined in this section, we provide the definition of analysis patterns, the identification of points of stability and flexibility, and the specification of design patterns.

3.1 The FramETL System Requirements

There is currently no consensus on the basic features of ETL systems. However, based on the concepts of ETL available so far in the literature, we classified the requirements of *FramETL* as a set of two phases, namely: *Metadata Definition* and *Operation*. In the metadata definition phase occurs the specification of the system inventory and modeling of ETL processes, while in the operation phase, the loading of metadata, the access to data sources, and the execution of ETL processes are performed. Thus, the basic requirements for an integrated ETL framework can be summarized as shown in Table II(a). These requirements were adopted as the analysis patterns for our framework project.

The *System Inventory* pattern represents the preparation of the system before starting ETL procedures. Thus, it includes information about storage environments and data structures participating in the ETL process being modeled. This storage configuration describes at least three environments: data sources, processing area and output data. The environments of data sources and output data are specified by defining their structure, attributes, data types, access credentials, and connection settings for accessing DBMS, text files, or virtual data repositories. The *System Inventory* pattern has been implemented in *FramETL*, but for the sake of space limitation, only the *Model Process* pattern and both patterns of *Operation* phase will be specified here, because these concentrate the main contribution of our proposal.

In order to model ETL processes, we need a combination of some ETL mechanisms to perform operations, like extraction, loading, and especially, data source transformations. *FramETL* implements the most common mechanisms for design ETL processes [Kimball and Caserta 2004; Mora 2005; Muñoz et al. 2009; Vassiliadis et al. 2009]. These mechanisms are analysis patterns of the second level of the *Model Processes* pattern. They are listed in Table II(b). However, this set of mechanisms is limited, such that *FramETL* allows the creation of custom mechanisms to a specific domain. Additionally, all mechanisms enable the ETL programmer intervention by handling data differently in each one of its ETL process instances. The description of the pattern *Model Processes* is given as follows.

—Context: Modeling ETL processes consists of creating a batch of jobs and a set of parameterized transformations. The context of our proposal requires the modeling and running of ETL processes programmatically, and in addition, the processes need to be reusable and flexible.

—Problem: This states how to comply the requirements on this context and which transformations are available. Moreover, it indicates how they should be combined to make the data suitable for

Table I. Comparison among programmable frameworks for developing ETL systems

	1	2	3	4	5	6	7
FramETL Proposed	Yes	Yes	Yes	Yes	Yes	Yes	No
Pygrametl Thomsen at al (2009)	No	No	No	No	No	Yes	Yes
ETLMR Liu at al (2011)	No	No	No	No	Yes	Yes	Yes
Arktos Vassiliadis at al (2005)	Yes	Yes	No	No	No	No	No

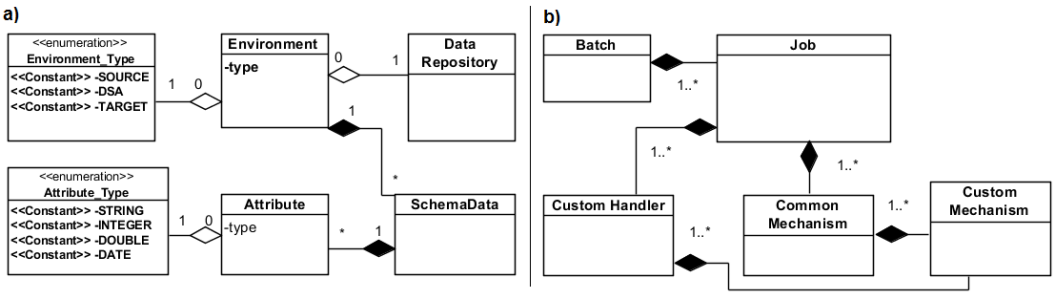

Fig. 1. Structures Diagram of the *System Inventory*(a) *Model Processes*(b) Pattern

being loaded into the decision-making database.

—Structure: It presents the possible solutions for the *Model Processes* according to the requirements of each ETL process being modeled. In *FramETL*, the transformations are represented by common mechanisms, custom mechanisms and transformation custom handlers. The transformed data are mapped into a relation source-destination attribute for each participant in the process. Figure 1b shows the structure of a simple process for reading, applying filters and loading data.

—Participants: (1) *Batch*: This is a list of jobs, which defines the scheduling and execution order of transformations. (2) *Job*: This is a source-destination mapping, which describes a collection of transformations. (3) *Common Mechanisms*: Those are the most common mechanisms found in the literature, referenced in Section 3.1, they are used to extract, transform and load when designing ETL applications (4) *Custom Mechanism*: This allows the creation of mechanisms for a specific domain. (5) *Custom Handlers*: This allows the programmer's intervention by changing the behaviour of the mechanism in each of its instances.

—Upcoming Patterns: Since the modeling processes have been adequately specified, the next task is the operation of reading the metadata.

In the *Metadata Definition* phase, the solution structure for the *Model Processes* analysis pattern varies according to the requirements of each ETL process. This pattern is the basis to identify the points of flexibility in *FramETL*. These details are discussed in Section 3.2.2. The *Operation* phase interprets the relationship between the participants of the *Metadata Definition* phase. The patterns in *Operation* phase are immutable in any ETL processes, so that they are the points of stability and represent the core of *FramETL*. They are discussed in details in Section 3.2.3.

3.2 The FramETL System Architecture

The system architecture of *FramETL* shown in Figure 2 illustrates its possibilities of usage and displays the dependency relationships among its layers, conforming the UML Model [Larman 2002]. These layers are the concepts of frameworks, the logical layer that represents the implementation of *FramETL*, and finally, the data layer.

The *Controller* is the main layer of the *FramETL* architecture, in which the analysis patterns of *Metadata Definition* and *Operation* phases are represented as class packages. The *Metadata Definition*

Table II. Analysis Patterns of FramETL

(a)

Phases	Analysis Patterns
Metadata Definition	*System Inventory*
	Model Processes
Operation	*Read Metadata*
	Perform Processes

(b)

ETL Phase	Analysis Patterns
Extraction	*Extractor*
Transformation	*Filter, Union, Aggregate, Join, Conversion Surrogate key generator, Lookup*
Load	*Loader*

package provides the interfaces for reuse and specialization required for the *System Inventory* and *Model processes* patterns. On the other hand, the *Operation* package is responsible for reading the metadata, accessing the data environments, and the execution of ETL processes. *Operation* is the core of *FramETL* and contains within it the points of stability of the framework. The *FramETL* framework architecture uses the principle of inversion of control to make calls to points of flexibility of the framework, where design patterns are used.

3.2.1 *Metadata Definition.* From the analysis patterns *System Inventory* and *Model Processes*, we designed the domain model for *FramETL* (shown in Figure 3). This shows a conceptual perspective of the project classes that provide the ETL metadata used in the implementation of ETL applications. The construction of applications based on *FramETL* consists in the creation of ETL processes from the metadata described in this section and illustrated in the domain model of Figure 3. The programming interface for the metadata instantiation is provided by two hook methods, called *DefineMetadataInventory* and *DefineMetadataProcess* (Figure 5). This separation aims at helping in the organization of the code written for instantiating the metadata. These methods are invoked during the phase *Operation* (Section 3.2.3) for the effective instantiation of metadata.

The *System Inventory* is composed of the following ETL metadata. Firstly, the entity *Repository* is responsible for ensuring the access to materialized or virtualized data repositories by storing information about physical locations, access libraries, and other security credentials that depend on the specific nature of the data repository been accessed. Secondly, the ETL *Environment* represents an area of data storage in an ETL process, which can be a data source, the processing area or a data output. Each environment is related to a data repository and a set of data structures from its repository. Thirdly, the *Data Structure* specifies DBMS tables or text files that should be available in ETL processes. Fourthly, the *Attribute* describes the attributes of each data structure and its type. The data structures and the attributes are associated with a descriptive name in the business context, which turns easily the references to these elements during the modeling process.

To model ETL processes there exists the following metadata. First, a *Batch* is a set of ETL jobs. Second, a *Job* contains a set of transformations that operate on the same environments from the same data source and data target, so that each job sets up its ETL environments for source and destination data. Third, the *Mechanisms* represent the elements of extraction, transformation and loading. The instantiation of a mechanism requires one or more input data elements, a set of parameters that

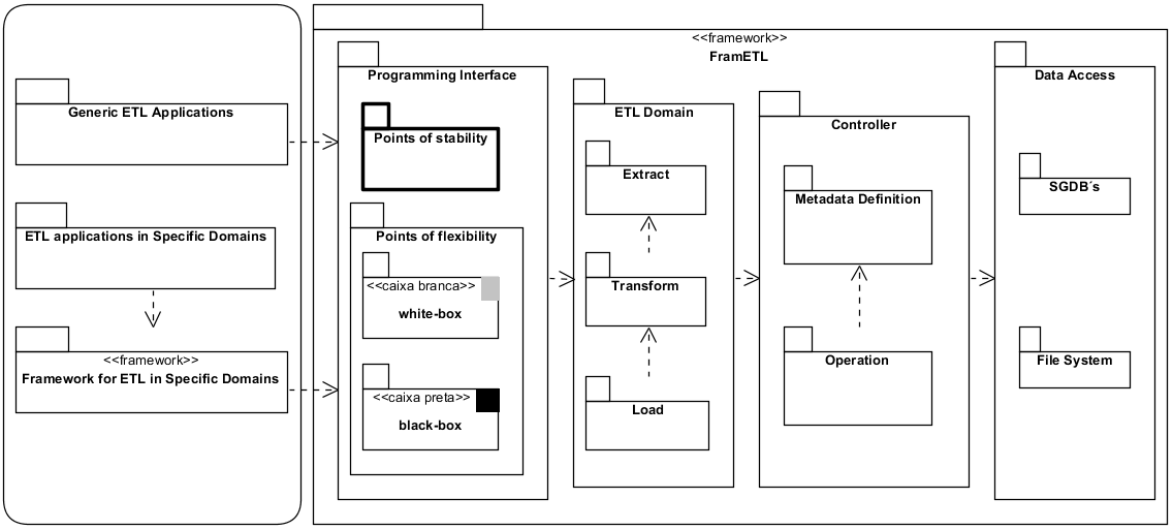

Fig. 2. The *FramETL* Architecture

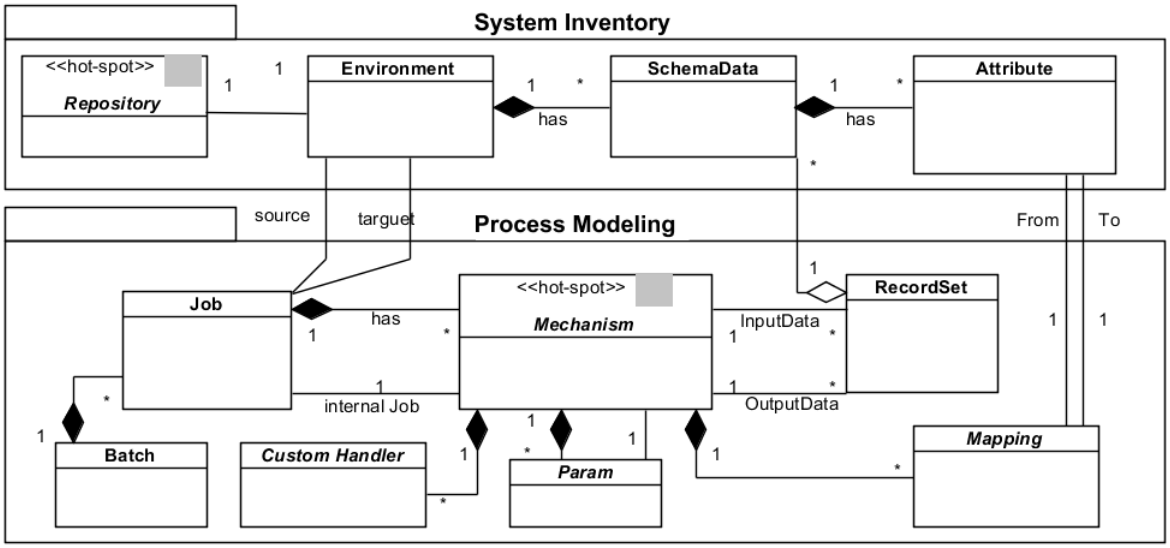

Fig. 3. Domain Model of *FramETL*

operate on the data flow and a set of mappings of attributes. If it is a *Loader* mechanism, it is needed to set up a target table. Aditionally, *Custom Handler* may be added in order to perform additional data manipulation. Fourth, the *Parameters* are variables that are expected to transform operations on the input data for which they are specific to each mechanism. Fifth, the *Mappings* are the source-destination relationships among attributes, in which an attribute of the set of input data is associated with an output attribute. Sixth, the *Recordset* are the records of input and a set of output records used by a mechanism. These records may be associated with the inventory data structures, such as a table from an OLTP system, a fact table from a DW or temporaries data in the ETL processing area. The mechanisms are related to each other through their datasets, which are processed and keep available for other mechanisms. Seven, the *Custom Handler* allows the programmer intervention during the transformation execution to provide further functionalities for the mechanism. This increases the level of flexibility by allowing that each mechanism refines its data treatment in different ways, even if they represents the same ETL operation.

3.2.2 *Identification of Flexibility Points.* The analysis patterns presented in Sections 3.1.1 and 3.1.2 show a need for providing flexibility in the configuration of data environments and in the definition of custom ETL processes. Then, we identify the points of flexibility of *FramETL* by selecting the analysis patterns *Inventory System* and *Metadata Definition* that are described as follows.

(1) *Repository*: Each *Environment* do *FramETL* is associated with an element of the entity *Repository* (Figure 3), which is responsible for ensuring the access to materialized or virtualized data repositories by providing information on physical locations, libraries of access, security clearance and instructions for reading or writing data. The physical repositories may be relational DBMS's files, XML (Extensible Markup Language) files or CSV (Comma Separated Values) files, while a virtual data repository may be a data stream that is resulted from the integration of *FramETL* with other existing ETL tools.

(2) *Custom Mechanisms*: they allow the creation of application-oriented mechanisms, which enables the creation and reuse of ETL mechanisms that had not been foreseen initially (Figure 4);

(3) *Custom Handlers*: these enable the programmer's intervention by changing the behavior of some instances of a mechanism (Figure 4). These handlers avoid the creation of a custom mechanism only to address specific details of an implementation, whose reuse would not be useful. Further-

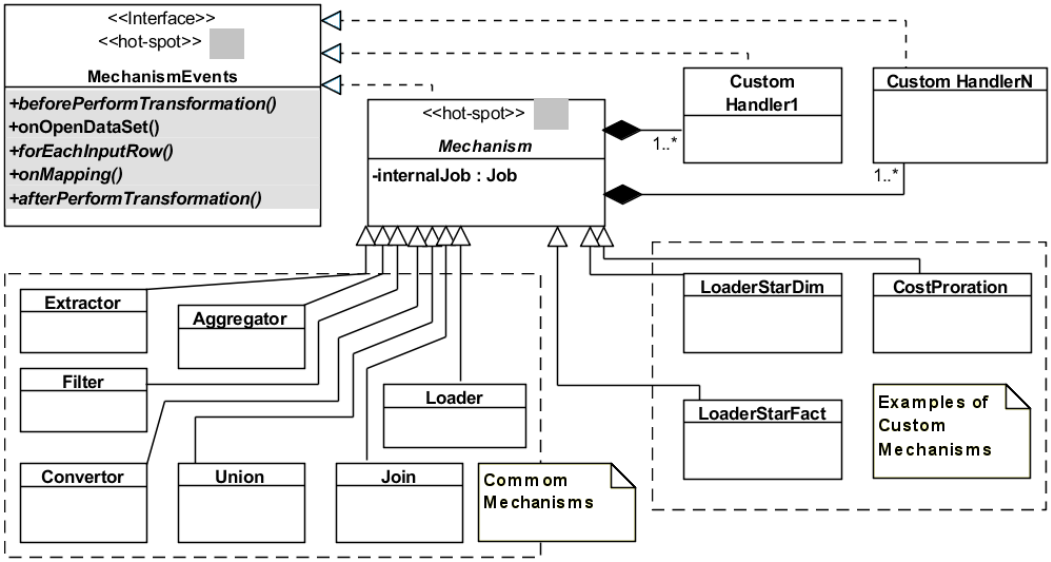

Fig. 4. Conceptual model of Points of Flexibility for *Model Process*

more, they can be reused by other mechanisms.

The points of flexibility are provided by the specialization interface of *FramETL*, which uses generics classes and interfaces. These classes provide hooks methods that must be implemented by applications based on *FramETL*. Then, these hooks methods are called by the design pattern *Template Method* during the Operation phase (Section 3.2.3).

The entity *Repository* provides a specialization interface to enable the implementation of connection procedures, the use of access libraries (drivers), the execution of authentication procedures, provision of reading and writing instructions together with other instructions that are specific to each data repository. An example of specialization of a *Repository* is the creation of a *Relational Repository* to implement the instructions of database connection, query and data manipulation for relational databases, and to encapsulate commands of the standard SQL language. This avoids the need for having SQL code written explicitly during the specialization of a *Mechanism* or the creation of *FramETL* applications. A *Repository* also provides support for semi-structured files, such as CSV or XML files (i.e. the latter from queries written in XQuery or Xpath). Another application of this point of flexibility is the specialization of a virtual *Repository* to deal with data streams resulted from the integration between *FramETL* and other existing ETL tools. Most ETL tools (e.g. Pentaho and Talend CloverETL) provide several data integration resources, such as components that run external applications, command line utilities to perform tasks outside the graphical environment, and JAVA programming interfaces for creating plugins for graphical IDE's. Thus, the first two techniques can be employed to generate the input or output *Dataset* of a *Repository*. On the other hand, an application can implement the programming interfaces provided by these tools to create a plugin based on *FramETL* to be used by the graphical environment of these tools.

Although the flexibility of the inventory system has been considered in the design and implementation of *FramETL*, our research gives a greater attention to points of flexibility for process modeling. Thus, the main hook methods of *FramETL* are in the entity *MecanismoEvents* (Figure 4), which provides programming interfaces to implement custom operations on the ETL process. The choice of these hook methods was based on the idea that it is possible to provide flexibility for the ETL Mechanisms through the notification of events during the dataflow manipulation.

Algorithm 1: *OnOpenDataSet* - implementation of *Filter* mechanism

Input: Void
Output: Void

```
implementation of hook onOpenDataSet()
        inputRepository ← ReadInputRepository(mDataInventory);
        FOR (a_param: setOfParams) DO
                filterAttr ← a_param.getAttribute()
                filterOper ← a_param.getOperation()
                filterValue ← a_param.getValue()
                inputRepository.AddWhere(filterAttr,filterOper,filterValue);
```

Algoritmo 2:*forEachInputRow(row)* - other possibility for *Filter* mechanism

Input: current record of Data Flow
Output: Boolean : True to Accept record and False to Reject record

```
implementation of hook forEachInputtRow(record)
        FOR (a_param: setOfParams) DO
                filterAttr ← a_param.getAttribute()
                filterOper ← a_param.getOperation()
                filterValue ← a_param.getValue()
                currentValue ← GetAttributeValue(record,filterAttr);
                IF CompareValues(currentValue,filterOper,filterValue) IS TRUE
                        RESULT ← TRUE
                ELSE
                        RESULT ← FALSE
```

The *Common Mechanisms* of *FramETL* (Table II(b)) are created from the entity *Mechanism* and from the implementation of its hook methods. Thus, it is possible, for example, to implement the method *OnOpenDataSet* to help in the generation of a input *Dataset* through the application of SQL instructions. To achieve this, the methods of the entity *SQL Elements* are used to group, join, filter and execute union operations. Another example is the implementation of the method *forEachInputRow* to calculate or modify initial attribute values, or even to filter input records. This method results in a acceptance or rejection of each record. To illustrate the programming specialization interface of a *Mechanism*, we used as examples two possibilities of the implementation of the *Common Mechanism Filter*. Thus, the hook methods *OnOpenDataSet* and *forEachInputRow* will be implemented as outlined in Algorithm 1 and Algorithm 2, when any other new *Mechanism* is created.

The *Custom Mechanisms* of *FramETL* (Figure 4) can be designed the same way as a *Commom Mechanism*. *Common mechanisms* can also be used as a basis for more customized mechanisms. Thus, the *Custom Mechanisms* override the implementation of the *Mechanism* methods. Moreover, it might resort to the *Internal Job* in order to instantiate other mechanisms for internal data manipulation on the input *RecordSet*. Additionally, *Custom Handlers* might self subscribe into custom or common mechanisms in order to manipulate the data transformation events too. For that, we use the *Observer* design pattern. Consequently, the handlers are notified when *FramETL* calls the *Hook Methods* during the *Operation* phase.

3.2.3 *Operation.* As mentioned in Sections 3.1 and 3.2, the *Operation* phase is the core of *FramETL*, where the points of stability of the framework are implemented. In this phase the *FramETL* reads and parses the metadata defined in *Metadata Defination* phase, and executes the ETL processes. Throughout the *Operation* procedures, the *FramETL* calls points of flexibility. To this end, the inversion of control principle is closely applied to call hook methods, which are implemented by custom mechanisms or their custom handlers. The procedures performed by the *Read Metadata* analysis pattern are basically calling two *hook methods*, which are available on the programming interface (Section 3.2.1) and must be implemented into the FramETL-based applications. The first method reads the metadata of system inventory and connects to the ETL environments defined in it, while the second method, reads the metadata of ETL processes. Thereafter, instances of entities of the domain model

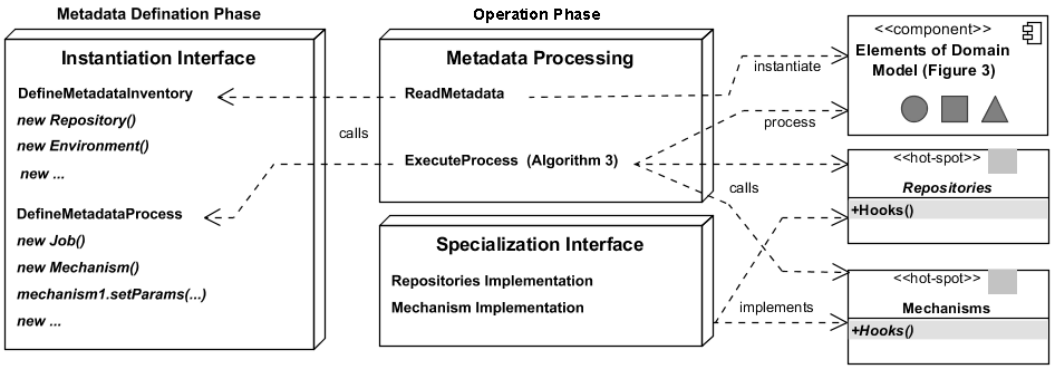

Fig. 5. Overview of *FramETL* framework

shown in Figure 3 are created, and then, they are ready to be executed.

The *Execute Processes* analysis pattern is presented as an algorithm for executing the ETL mechanisms (Algorithm 3), by reusing points of stability and calling points of flexibility. To perform this, the algorithm calls the *hook methods* of the *MechanismEvents* entity, which is part of the hotspots conceptual model shown in Figure 4. Therefore, for each *Mechanism* and *Custom Handler* instance, *FramETL* executes the algorithm, receiving as input one of these instances. Then, all the *hook methods* called by algorithm run the custom routine of the *Mechanism* instance passed as input parameter. In line 1, the *hook method BeforePerformTransformation* is called. In line 2, the set of ETL parameters is read from the *Mechanism* and stored in the variable *setOfParams*. In lines 3 and 4, it is tested if *setOfParams* is not null, then the *hook method OnOpenDataSet* is called. In line 5, the operation type is read from the *Mechanism* and stored in the variable *mechanismType*. In line 6, the output schema is read from *Mechanism* and stored in the variable *outPutSchema*. In lines 7 and 8, it is verified if *mechanismType* is equal to EXTRACT or TRANSFORM constants, then the frozen-spot method *PrepareDsaSchema* is called. In line 9, the hook method *OpenInputRecordSet* of the *Repository* entity is called, which results in a set of input data records, being stored in the variable*setOfRecords*. In line 10, for each element of *setOfRecords*, the content from lines 11 to 16 is parsed by a looping.

	Algoritimo 3:ExecuteProcess(mechanism)
	Input: a Mechanism instance
	Output: Void
1	BeforePerformTransformation(mechanism);
2	setOfParams ← ReadParams(mechanism);
3	IF (NotEmpty(setOfParams)) THEN
4	OnOpenDataSet(mechanism,setOfParams);
5	mechanismType ← ReadType(mechanism);
6	outPutSchema ← ReadOutPutSchema(mechanism);
7	IF ((mechanismType IS EXTRACT) OR (mechanismType IS TRANSFORM)) THEN
8	PrepareDsaSchema(outPutSchema);
9	setOfInputRecords ← OpenInputRecordSet(mechanism);
10	FOR (a_record: setOfInputRecords) DO
11	setOfMappings ← ReadMapping(mechanism);
12	isRowAccepted ← ForEachRow(a_record,setOfMappings);
13	IF (ISTRUE(isRowAccepted)) THEN
14	FOR (a_mapping: setOfMappings) DO
15	OnMapping(a_record,a_mapping);
16	PerformMappings(a_record,setOfMappings);
17	setOfOutPutRecords ← ReadMappingData(setOfMappings);
18	StoreOutputRecordSet(setOfOutPutRecords,outPutSchema);
19	AfterPerformTransformation(mechanism);

In this looping iteration occurs the following: (*i*) The set of attribute mappings of the *Mechanism* is read and stored in the variable *setOfMappings*. (*ii*) the *hook method forEachInputRow* is called passing a record element and the *setOfMappings* as input parameters, in order to check the record and return an acceptance or rejection state. (*iii*) If the record is accepted, the *hook method OnMapping* is called for each element *a_mapping* of *setOfMappings*. Then, in line 16, the frozen-spot method *PerformMapping* is called in order to apply the value mappings between attributes. When the iteration of input records finishes, the frozen-spot method *StoreTargetRecord* is called to store the transformed record into DSA or into the target environment. Finally, in line 19, the *hook method AfterBeforePerformTransformation* is executed.

3.3 The FramETL Specializations

To illustrate the extension and reuse functionality of *FramETL*, we developed two specialized frameworks based on *FramETL*. These frameworks, namely *FramETL4StarDW* (Section 3.3.1) and *FramETL4CostAccount* (Section 3.3.2), are both seen as contributions of this research because we found a lack of frameworks with their resources. The first was motivated by the need for a custom mechanism to load *dimensions and facts tables* into DW modeled as star schemas, as recommended by the DW literature, while the second was chosen due to a demand for data integration and prorating of indirect costs in the cost accounting. These two frameworks have special mechanisms for ETL, which are used later for the implementation of the ETL applications used to evaluate our work. Although the operations performed by these mechanisms are not complex, their modeling through a graphical ETL tool is not trivial, once this requires knowledge about a specific area in order to develop a transformation strategy. This occurs mainly in less common scenarios to people from the TI department, such as cost accounting, and therefore, the customization of a mechanism enables the separation between the implementation of the transformation strategies and the effective modeling of the ETL process. In Figures 6 and 7 we show how *FramETL* was extended and reused, and we use graphical UML elements to facilitate the representation of the customized mechanisms when designing conceptual ETL process.

3.3.1 *The FramETL4StarDW.* Transformations of aggregation, join, filter, union, creations of surrogate keys, amongst others, are commonly found in literature and on the currently available ETL tools. Although it is possible to perform ETL processes to load data into DW using these transformations, the creation of a custom mechanism hides technical details of ETL process for DW. Hence, this section presents *FramETL4StarDW*, which is a customized framework that provides mechanisms for

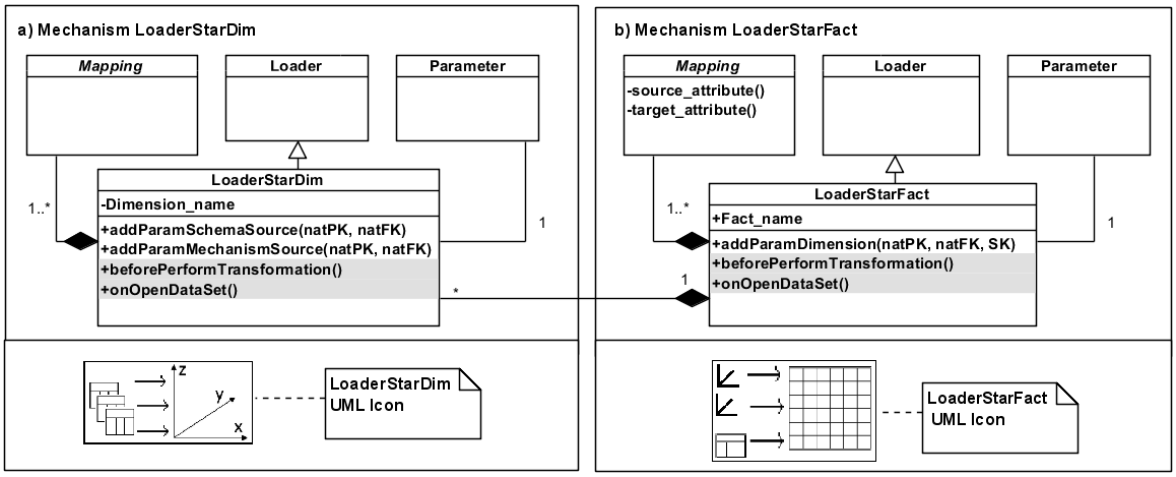

Fig. 6. Class diagram and UML representation for *LoaderStarDim*(a) and *LoaderStarFact*(b) mechanisms

loading dimensions and facts tables into DW modeled as star schemas. These mechanisms are called as *LoaderStarDim* and *LoaderStarFact*, respectively, which are specializations of the *Loader* common mechanism.

For developing these specializations, new methods were created to facilitate the input of *Parameters* that represent the source tables used in the building of a dimension, or dimensions that participate in a fact relationship. Then the hook methods *BeforePerformTransformation* and *OnOpenDataSet* were implemented, and they use the *Internal Job* of *Mechanism* to create *Common mechanisms*, and to perform operations of extraction, generation and replacement of keys according to the *Parameters* received as input data. The *LoaderStarDim* performs the denormalization of records, the creation of surrogate keys of dimensions and the loading of dimensions into target environment. Whereas, the *LoaderStarFact* loads the fact table from the relational data sources, and replaces the original foreign keys to dimensions by the previously created surrogate keys of dimensions. For the instantiation of the mechanism *LoaderStarDim*, the following data must be informed: the dimension name, the main source table, the *Parameters* that indicate the secondary source tables and their original keys, and a set of mappings for defining the members of dimension tables. For instantiating the mechanism *LoaderStarFact*, the name of the target fact table, a normalized dataset derived from the data sources, and the *Parameters* that denote the dimensions (*LoaderStarDim*) should be provided together with a set of mappings to define the measures of the fact table. Figure 6 shows the specialization diagram and a UML representation for each mechanism of *FramETL4StarDW*.

3.3.2 *The FramETL4CostAccount.* The ERP/OLTP systems are widely employed to control accounting records in the field of financial accounting. However, these systems do not provide management information, which is required by the cost accounting area. While financial accounting specialists assist companies to fulfill their legal obligations related to external agents, the cost accounting provides support for decision-making to managers. Therefore, the use of BI solutions and data integration among systems are a common practice for obtaining management reports on cost accounting. In fact, one of the main problems in this area is the allocation of overhead. The prorate technique distributes costs among departments or factory activities, which are cost consumers. Furthermore, the costs are prorated complying with specific criteria, known as cost drivers.

The *FramETL4CostAccount* is a framework applied to the field of cost accounting. This framework presents a mechanism to prorate indirect costs, which we call *CostProration*. It is extended from the *Mechanism* entity and implements the *Hook Methods BeforePerformTransformation*, *forEachInputRow* and *onMapping*. For the instantiation of the mechanism *CostProration*, the *DataSet* that represents the indirect costs and the two *Parameters* that provide the consumers and cost drivers must be informed. They are derived from *Extractors* or any other mechanism of *FramETL*. The transformation performed by *CostProration* occurs as follows. For each overhead record (OH), the value from this cost is divided in proportion to the value of each cost driver (CD). Thus, the final cost is allocated (CA) to each n-th consumer (CM) and then, the final allocated cost value is given by the

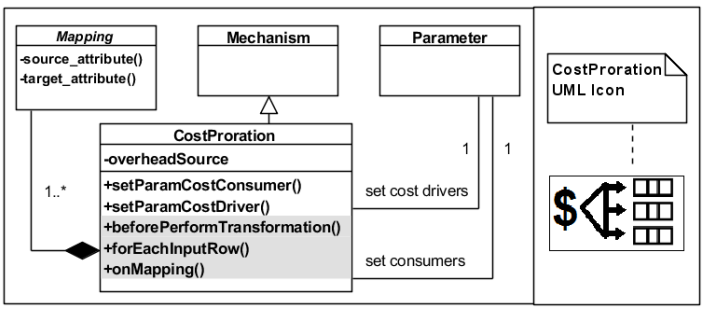

Fig. 7. Class diagram and UML representation for *CostProration* mechanism

following equation: CAn = OH * (CDn / SUM (CD)). For m records of overhead and n records of consumers, this mechanism outputs m * n records of prorated costs. Figure 7 shows the specialization diagram and a UML representation for the *CostProration* mechanism.

4. EXAMPLES OF FRAMETL-BASED APPLICATIONS

This section provides two examples of applications of *FramETL* from different domains. The first application was motivated by the need to facilitate the loading of dimensions and tables into DW structured according to star schemas, as recommended by the DW literature. The second application was chosen due to the demand for data integration, for the prorating of costs and for using BI systems in accounting costs. The development of these applications aims to illustrate both the reusability and flexibility of *FramETL* for ETL applications with different data requirements.

To evaluate the work described in this research, we implemented two *FramETL* based applications. They are based on real scenarios presented by two companies from different areas. To perform this evaluation, we used the interface instantiation of Section 3.2.1 to instantiate some *Common Mechanisms* and the *Custom Mechanisms* of the specialized frameworks detailed in Section 3.3. Thus, we did not have to rely on a predefined GUI for: (1) helping in the creation of custom DW based on star schemas as shown in Section 4.1 (i.e. *FramETL4StarDW*), or (2) assisting in the data integration tasks of prorating of costs as shown in Section 4.2 (i.e. *FramETL4CostAccount*). Additionally, we resorted to a *Custom Handler* instance to format date attribute and apply some conditional conversions for attribute values. The use of these specialized frameworks allows the ETL designer to model the processes through the reuse interface of each mechanism, without worrying about the domain-specific transformation strategies, unlike what happens when the processes are modeled using current ETL tools. To provide a comparative basis, the same ETL processes given as examples in this section were implemented through an existing ETL tool, called Pentaho Data Integration (PDI) [Bouman and van Dongen 2009].

4.1 Application I - FramETL4StarDW

In this example, we used the *FramETL4StarDW* described in Section 3.3.1, to implement an ETL process for building a Sales data mart in an industry of mineral water bottling. The data sources are relational tables (FIGURE 8a) from MySQL DBMS and a text file containing records to load the temporal dimension. The dimensional structure of the data mart is shown in Figure 8b, and this was implemented on Oracle DBMS.

The conceptual modeling of ETL process for loading the *Product* dimension is shown in Figure 9.a It presents the simplified ETL process by using *LoaderStarDim*, which encapsulate the entire procedures needed to denormalize the source tables and create the surrogate keys, as described in Section 3.3.1. Then the records for the *dim_product* dimension can be loaded into the Sales data mart environment.

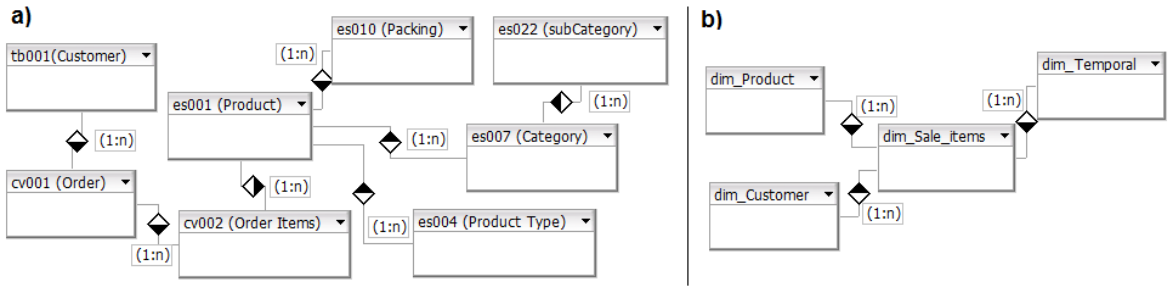

Fig. 8. ER diagram of data source tables (a) and DW Sales (b) for application 1 *FramETL4StarDW*

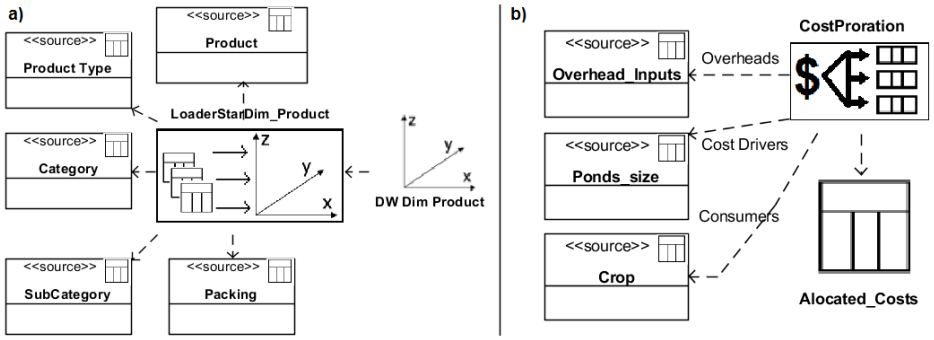

Fig. 9. UML diagrams of ETL process using *LoaderStarDim*(a) and *CostProration*(b) mechanism

Two other ETL processes like this shown in Figure 9a were also applied to the *dim_customer* and *dim_temporal*. Then, a *LoaderStarFact* mechanism was instantiated and associated with the fact source tables and with the three *LoaderStarDim* mecanismos created for each dimension. Thus, the *LoaderStarFact* performs the loading into the *fact_sale_items* fact table on the Sales data mart.

The scenario related to the generation of the dimension *Product* shown in this section was also modeled by using the ETL pentaho tool, resulted in 16 processes containing joins, sorters and key replacements and is available at *http://www.cin.ufpe.br/˜mss6/*. The transformation strategy implemented by *FramETL4StarDW* was applied with instances of *Common Mechanisms* into *Internal Job*. This led an internal transformation sequence similar to the pentaho modeling result. But, that is a difference in that the instantiation interface of *LoaderStarDim* hides technical details by passing the source tables as *Parameteres* to be used for generating the dimension *Product*.

4.2 Application II - FramETL4CostAccount

In this example, we used the *FramETL4CostAccount*, described in Section 3.3.2, to implement a process of data integration between two DBMSs: Oracle and Firebird. The first DBMS is used by a SAP ERP system, while the second DBMS is used by a vertical system for production control in a shrimp farming in captivity.

The overhead records are distributed into the production units that are the ponds used for shrimp crops, which are called in this context as cost consumers. This distribution must comply with the proportionality of the areas in hectares of each pond, so that, the size of the ponds is the cost driver for proration. To make this possible, we applied the *CostProration* customized transformation. The whole ETL process for this scenario provides *Filters* and *Joins Common Mechanisms* in order to prepare input *Recordset* for *CostProration* mechanism. Thus, the *CostProration* instance (Figure 9b) performs the computation described in Section 3.3.2 in order to populate the target table with costs allocated for each pond consumer.

The ETL process for cost proration presented in this section was also implemented by using the pentaho tool, and resulted in 13 subprocesses, including joins, merges and groupings, available at *http://www.cin.ufpe.br/˜mss6/*. From the pentaho GUI, each indirect cost record was combined with all records of cost drivers and with the sum of the attribute driver to compute the cost proration calculation. This consisted in the union of each indirect cost record with all records of cost drivers and with the sum of the attribute driver in order to perform the cost proration calculation. However, the strategy used in the *FramETLCostAccount* is simpler because do not require knowledge about cost proration techniques and because we used both the method *forEachInputRow* of each indirect cost record to iterate over all consumer data and the method *onMapping* to obtain the cost proration computation. Thus, the interface instantiation of *CostProration* hides the technical details

of the process, and allows cost proration from the *parameters Consumers* and *cost drivers*. Then, these *parameters* are applied to the set of overhead, being the input *RecordSet* of the *CostProration* mechanism. Another advantage *FramETLCostAccount* is that other instances of *CostProration* can be calculated, even if the data transformation requirements become more complex (e.g. scenario with restrictions over the accounts and the cost consumers).

5. CONCLUSION AND FUTURE WORK

The main contribution of this article is the *FramETL* framework that includes the requirements for an ETL framework for systems development in a pattern language, a system prototype architecture for a customizable ETL tool with modeling and extension environments integrated, a domain model and a novel algorithm for executing ETL customized processes by reusing points of stability and calling points of flexibility. While the identification of points of stability allows the development of generic ETL components, the formalization of points of flexibility facilitates the process of specialization and instantiation. The *FramETL* framework is useful for developing new ETL tools, creating frameworks for customized ETL processes of specific business areas, building decision support applications with embedded ETL rules, developing Enterprise Information Integration (EII) systems where the destination of the data are not necessarily data warehouses, and for constructing service-oriented ETL systems.

Additional contributions of this study are the two *FramETL*-based frameworks, one for ETL processes in data warehouses modeled as star schemas, and another for ETL processes in cost accounting. The ETL applications developed in these two contexts allowed the modeling and the execution of ETL processes conforming to requirements presented in two real business cases. The evaluation results showed that the *FramETL* is able to generate custom applications for ETL processes programmatically in different domains, without the adoption of proprietary tools or using GUI. Also, the ability to reuse common ETL elements and the flexibility to create customized mechanisms allowed the development of ETL applications, which facilitated the modeling and the executions of ETL processes, so that for well-known cases such as data warehouse, as for specific cases such as cost accounting. Furthermore, the implementation of the processes made the programming code more closer to the ETL conceptual modeling diagrams than general purpose programming language, which is not specific for ETL processes. Moreover, the cost for developing these applications should be offset by savings generated by avoiding licensing and training for use of proprietary ETL tools.

The improvement of *FramETL4CostAccount* to perform transformations specialized in different costing methodologies of cost accounting is seen as one of the next work. Moreover, the extension of *FramETL* for the extraction and generation of data in Extensible Business Report Language (XBRL) [Siqueira et al. 2009] can be investigated further, which will allow to create applications for publication of financial results. Additionally, the Business Process Modeling Notation for ETL (BPMN4ETL) [Akkaoui et al. 2011] can be applied to the proposed *FramETL* in future. Finally, the Parallel Programmable [Thomsen and Pedersen 2011] and MapReduce [Liu et al. 2011] approaches could also be applied to *FramETL* in order to help in the execution of performance tests to further evaluate our work.

REFERENCES

Akkaoui, Z. E., Muñoz, E. Z. J.-N., and Trujillo, J. A Model-Driven Framework for ETL Process Development. In *Proceedings of the international workshop on Data Warehousing and OLAP*. Glasgow, Scotland, UK, pp. 45–52, 2011.

Awad, M. M. I., Abdullah, M. S., and Ali, A. B. M. Extending ETL framework using service oriented architecture. *Procedia Computer Science* vol. 3, pp. 110–114, 2011.

Banerjee, B. OverHeads. In P. L. P. Ltd (Ed.), *Cost Accounting Theory And Practice*. Prentice-Hall, India, pp. 200–294, 2006.

Bouman, R. and van Dongen, J. Pentaho Data Integration Primer. In R. Elliott (Ed.), *Pentaho Solutions Intelligence and Data Warehousing with Pentaho and MySQL*. Wiley Publishing, Indianapolis, USA, pp. 223–258, 2009.

Braga, R. T. V. and Masiero, P. C. Finding frameworks hot spots in pattern languages. *Journal of Object Technology* 3 (1): 123–142, 2004.

Dayal, U., Castellanos, M., Simitsis, A., and Wilkinson, K. Data Integration Flow for Business Inteligence. In *Proceedings of International Conference on Extending Database Technology: Advances in Database Technology*. Saint Petersburg, Russia, pp. 1, 2009.

Fayad, M. E. and Schmidt, D. C. Object-Oriented Application Frameworks. *Communications of the ACM* 40 (10): 32–38, 1997.

Fayad, M. E., Schmidt, D. C., and Johnson, R. E. Application Frameworks. In M. Spencer (Ed.), *Building application frameworks: object-oriented foundations of framework design*. John Wiley and Sons, New York, USA, pp. 3–29, 1999.

Inmon, W. H. The Datawarehouse and Design. In R. Elliott (Ed.), *Building the Data Warehouse*. John Wiley and Sons, New York, USA, pp. 81–147, 2002.

Johnson, R. E. Frameworks= (Components+Patterns). *Communications of the ACM* 40 (10): 39–42, 1997.

Kimball, R. and Caserta, J. Metadata. In R. Elliott (Ed.), *The Data Warehouse ETL ToolKit*. Wiley Publishing, Inc., Indianapolis, USA, pp. 351–380, 2004.

Kimball, R. and Ross, M. Dimension Modeling Primer. In R. Elliott (Ed.), *The Data Warehouse ToolKit*. Wiley Publishing, Inc., Indianapolis, USA, pp. 1–27, 2002.

Krishna, R. and Sreekanth. An Object Oriented Modeling and Implementation of Web Based ETL Process. *IJCSNS International Journal of Computer Science and Network Security* 10 (2): 1, 2010.

Larman, C. Organizing the Design and Implementation Model Packages. In J. Somma (Ed.), *Applying UML and Patterns*. Prentice Hall, Upper Saddle River, USA, pp. 475–484, 2002.

Li, L. A Framework Study of ETL Processes Optimization Based on Metadata Repository. *IEEE Computer* vol. 6, pp. 125–129, 2010.

Liu, X., Thomsen, C., and Pedersen, T. B. The etlmr mapreduce-based etl framework. In *Proceedings of the 23rd international conference on Scientific and statistical database management*. Berlin, Heidelberg, pp. 586–588, 2011.

March, S. T. and Hevner, A. R. Integrated decision support systems A data warehousing perspective. *Decision Support System* 43 (3): 1031–1043, 2007.

Mazanec, M. and Macek, O. On general-purpose textual modeling languages. In *In Proceedings of DATESO'12*. Praha, Czech Republic, pp. 1–12, 2012.

Mora, S. L. *Data Warehouse Design with UML*. Ph.D. thesis, Universidad de Alicante, Alicant, Spain, 2005.

Muñoz, L., Mazon, J.-N., and Trujillo, J. Automatic Generation of ETL processes from Conceptual Models. In *Proceedings of the international workshop on Data warehousing and OLAP*. Hong Kong, China, pp. 33–44, 2009.

Rom, A. *Management accounting and integrated information systems*. Ph.D. thesis, PhD School in Economics and Business Administration - CBS / Copenhagen Business School, Copenhagen, Denmark, 2008.

Simitsis, A. Mapping Conceptual to Logical Models for ETL Processes. In *Proceedings of the international workshop on Data warehousing and OLAP*. Bremen, Germany, pp. 67–76, 2005.

Siqueira, E. R. M., da Silva, P. C., Fidalgo, R. N., and Times, V. C. A Framework for Integration of Materialized XBRL Documents. In *Brazilian Symposium on Databases*. Fortaleza, Brazil, pp. 226–240, 2009.

Thomsen, C. and Pedersen, T. B. pygrametl: A Powerful Programming Framework for Extract Transform Load Programmers. In *Proceedings of the international workshop on Data warehousing and OLAP*. Hong Kong, China, pp. 49–56, 2009.

Thomsen, C. and Pedersen, T. B. Easy and Effective Parallel Programmable ETL. In *Proceedings of the international workshop on Data Warehousing and OLAP*. Glasgow, Scotland, UK, pp. 37–44, 2011.

Trujillo, J. and Mora, S. L. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In *Conceptual Modeling - ER 2003*. Chicago, USA, pp. 307–320, 2003.

Vassiliadis, P., Mazon, A. S., and Baikousi, E. A Taxonomy of ETL Activities. In *Proceedings of the international workshop on Data warehousing and OLAP*. Hong Kong, China, pp. 25–32, 2009.

Vassiliadis, P., Mazon, A. S., and Skiadopoulos, S. Conceptual model for etl processes. In *Proceedings of the international workshop on Data Warehousing and OLAP*. McLean, Virginia, USA, pp. 14–21, 2002.

Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitisb, M., and Skiadopoulos, S. A generic and customizable frameworkfor the design of ETL scenarios. *Information Systems - Special issue: The 15th international conference on advanced information systems engineering* 30 (7): 492–525, 2005.

Wang, H. and Ye, Z. An ETL Services Framework Based on Metadata. In *International Workshop on Intelligent Systems and Applications*. Wuhan, China, pp. 1–4, 2010.

Wojciechowski, A. E-ETL: Framework For Managing Evolving ETL Processes. In *Proceedings of the Workshop for Ph.D. students in information*. Glasgow, Scotland, UK, pp. 59–66, 2011.