On the Logical Modeling of ETL Processes

Panos Vassiliadis, Alkis Simitsis, Spiros Skiadopoulos[1]

[1] National Technical University of Athens, Dept. of Electrical and Computer Eng., Computer Science Division, Iroon Polytechniou 9, 157 73, Athens, Greece
{pvassil,asimi,spiros}@dbnet.ece.ntua.gr

1. Introduction

Extraction-Transformation-Loading (ETL) tools are pieces of software responsible for the extraction of data from several sources, their cleansing, customization and insertion into a data warehouse. Research has only recently dealt with the above problem and provided few models, tools and techniques to address the issues around the ETL environment [1,2,3,5]. In this paper, we present a logical model for ETL processes. The proposed model is characterized by several templates, representing frequently used ETL activities along with their semantics and their interconnection. In the full version of the paper [4] we present more details on the aforementioned issues and complement them with results on the characterization of the content of the involved data stores after the execution of an ETL scenario and impact-analysis results in the presence of changes.

2. Logical Model

Our logical model abstracts from the technicalities of monitoring, scheduling and logging while it concentrates (a) on the flow of data from the sources towards the data warehouse and (b) on the composition of the activities and the derived semantics.

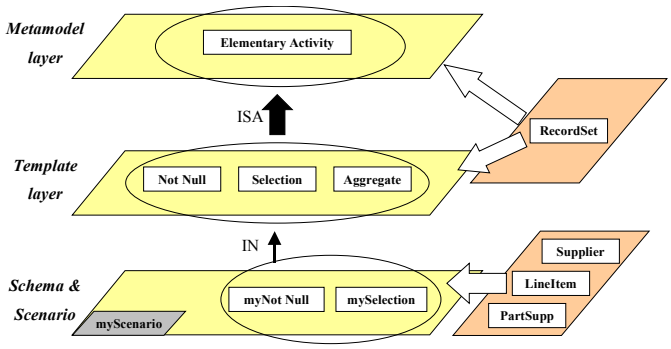

Fig. 1 The metamodel for the logical entities of the ETL environment

Activities are the backbone of the structure of any information system. In our framework, activities are logical abstractions representing parts, or full modules of code. We distinguish between three layers of instantiation/specialization for the logical model of ETL activities, as depicted in Fig.1: *Metamodel*, *Template*, and *Custom Layers*.

Metamodel Layer. The most generic type of an activity forms the upper level of instantiation (*Metamodel layer* in Fig. 1), covering the most general case of an atomic execution entity within the ETL context.

In Fig. 2 one can see the high level abstraction of the operation of an elementary binary activity. Data coming from the two inputs are propagated to the output, while the rejections are dumped to the `Rejected Rows` file. The operation is tuned through a set of parameters. Activities are depicted with triangles. The annotated edges characterize the data consumer/provider semantics. The `+` symbol denotes that data are appended to the `Output`; the `!` symbol denotes that the rejected rows are propagated towards the `Rejected Rows` file, whose contents they overwrite; the → symbol denotes that data are simply read from `Input 1`; and the `–` symbol denotes that the data from `Input 2` that match the criteria of the activity are deleted from `Input 2`, once they have been read by the activity.

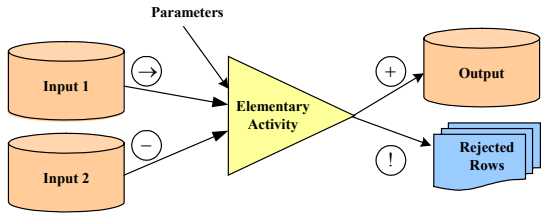

Fig. 2 The basic idea behind the mechanism of Elementary Activities

An *Elementary Activity* is formally described by the following elements:

- *Name*: a unique identifier for the activity.
- *Input Schemata*: a finite set of one or more input schemata that receive data from the data providers of the activity.
- *Output Schema*: the schema that describes the placeholder for the rows that pass the check performed from the activity.
- *Rejections Schema*: a schema that describes the placeholder for certain rows that do not pass the check performed by the activity.
- *Parameter List*: a set of pairs (name and schema) which act as regulators for the functionality of the activity.
- *Output Operational Semantics*: an SQL statement describing the content passed to the output of the operation, with respect to its input.
- *Rejection Operational Semantics*: an SQL statement describing the rejected records.
- *Data Provider/Consumer Semantics*: a modality of each schema which defines (a) whether the data are simply read (symbol → in Fig. 2) or read and subsequently

deleted from the data provider (symbol – in Fig. 2 and (b) whether the data are appended to the data consumer (symbol + in Fig. 2) or the contents of the data consumer are overwritten from the output of the operation (symbol ! in Fig. 2).

Template Layer. Providing a single metaclass for all the possible activities of an ETL environment is not really enough for the designer of the overall process. A richer "language" should be available, in order to describe the structure of the process and facilitate its construction. To this end, we provide a set of *Template Activities* (*Template Layer* in Fig. 1), which are specializations of the generic metamodel class. We have already pursued this goal in a previous effort [5]; in this paper, we extend this set of template objects and deal with the semantics of their combination (see [4] for more details).

Filters
- SELECTION ($\varphi(A_{n+1},\ldots,A_{n+k})$)
- UNIQUE VALUE (R.A)
- NOT NULL (R.A)
- DOMAIN MISMATCH (R.A, x_{low}, x_{high})
- PRIMARY KEY VIOLATION ($R.A_1,\ldots,R.A_K$)
- FOREIGN KEY VIOLATION ($[R.A_1,\ldots,R.A_K],[S.A_1,\ldots,S.A_K]$)

Unary Transformations
- PUSH
- AGGREGATION ($[A_1,\ldots,A_k],[\gamma_1(A_1),\ldots,\gamma_m(A_m)]$)
- PROJECTION($[A_1,\ldots,A_k]$)
- FUNCTION APPLICATION($f_1(A_1),\ldots,f_k(A_k)$)
- SURROGATE KEY ASSIGNMENT (R.PRODKEY,S.SKEY,x)
- TUPLE NORMALIZATION ($R.A_{n+1},\ldots,R.A_{n+k},A_{CODE},A_{VALUE}$,h)
- TUPLE DENORMALIZATION ($A_{CODE},A_{VALUE},R.A_{n+1},\ldots,R.A_{n+k}$,h')

Binary Transformations
- UNION(R,S)
- JOIN(R,S,$[(A_1,B_1),\ldots,(A_k,B_k)]$)
- DIFF(R,S,$[(A_1,B_1),\ldots,(A_k,B_k)]$)

Fig. 3 Template Activities grouped by category

As one can see in Fig. 3, we group the template activities in three major groups. The first group, named *Filters*, provides checks for the respect of a certain condition by the rows of a certain condition. The semantics of these filters are the obvious (starting from a generic selection condition and proceeding to the check of *value uniqueness*, *primary or foreign key violation*, etc.). The second group of template activities is called *Unary Transformations* and except for the most generic *Push* activity (which simply propagates data from the provider to the consumer) consists of the classical *aggregation* and *function application* operations along with three data warehouse specific transformations (*surrogate key assignment*, *normalization* and *denormalization*). The third group consists of classical *Binary Operations*, such as *union*, *join* and *difference* with the last being a special case of the classic relational operator.

Custom Layer. The instantiation and the reuse of template activities are also allowed in our framework. The activities that stem from the composition of other activities are called *composite activities*, whereas the instantiations of template

activities will be grouped under the general name of *custom activities* (*Custom Layer* in Fig. 1). Custom activities are applied over the specific recordsets of an ETL environment. Moreover, they also involve the construction of ad-hoc, user tailored elementary activities by the designer.

3. Exploitation of the Model

In order to perform operations like zooming in/out, consistency checking and what-if analysis, each scenario is modeled as a graph, which we call the *Architecture Graph* (see [4] from more details). Activities, data stores and attributes are modeled as nodes and their relationships as edges.

Zooming in and out the architecture graph. We can zoom in and out the architecture graph, in order to eliminate the information overflow, which can be caused by the vast number of involved attributes in a scenario. In a different kind of zooming, we can follow the major flow of data from sources to the targets.

Consistency Checks. We can perform several consistency checks on the architecture graph, like the detection of activities having input attributes without data providers, the detection of non-source tables having attributes without data providers, or the detection of useless source attributes.

What-if analysis. We can identify possible impacts in the case of a change, by modeling changes as additions, removals or renaming of edges and nodes. Still, out of the several alternatives, only the *drop node* operation seems to have interesting side effects. Moreover, even if atomic operations are of no significance by themselves, they are important when they are considered in the case of composite operation transitions.

References

[1] H. Galhardas, D. Florescu, D. Shasha and E. Simon. Ajax: An Extensible Data Cleaning Tool. In Proc. ACM SIGMOD Intl. Conf. On the Management of Data, pp. 590, Dallas, Texas, (2000).

[2] V. Raman, J. Hellerstein. Potters Wheel: An Interactive Framework for Data Cleaning and Transformation. Technical Report University of California at Berkeley, Computer Science Division, 2000. Available at http://www.cs.berkeley.edu/~rshankar/papers/pwheel.pdf

[3] S. Sarawagi. Special Issue on Data Cleaning (ed.). IEEE Computer Society, Bulletin of the Technical Committee on Data Engineering, **23**(4), (2000).

[4] P. Vassiliadis, A. Simitsis, S. Skiadopoulos. On the Conceptual and Logical Modeling of ETL Processes. Available at http://www.dbnet.ece.ntua.gr/~pvassil/publications/VaSS_TR.pdf

[5] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, T. Sellis. Arktos: Towards the modeling, design, control and execution of ETL processes. Information Systems, **26**(8), pp. 537-561, December 2001, Elsevier Science Ltd.