ARKTOS: A Tool For Data Cleaning and Transformation in Data Warehouse Environments

Panos Vassiliadis Zografoula Vagena Spiros Skiadopoulos Nikos Karayannidis
Timos Sellis
Knowledge and Database Systems Laboratory
Dept. of Electrical and Computer Engineering
National Technical University of Athens
{pvassil, zvagena, spiros, nikos, timos}@dbnet.ece.ntua.gr

Abstract

Extraction-Transformation-Loading (ETL) and Data Cleaning tools are pieces of software responsible for the extraction of data from several sources, their cleaning, customization and insertion into a data warehouse. To deal with the complexity and efficiency of the transformation and cleaning tasks we have developed a tool, namely ARKTOS*, capable of modeling and executing practical scenarios, by providing explicit primitives for the capturing of common tasks.* ARKTOS *provides three ways to describe such a scenario, including a graphical point-and-click front end and two declarative languages: XADL (an XML variant), which is more verbose and easy to read and SADL (an SQL-like language) which has a quite compact syntax and is, thus, easier for authoring.*

1 Introduction

A data warehouse is a heterogeneous environment where data must be integrated both at the schema and at the instance level [CGL+98]. Practice has shown that neither the accumulation nor the storage process of the information seem to be completely credible. Errors in databases have been reported to be up to 10% range and even higher in a variety of applications [WRK95]. [WSF95] report that more than $2 billion of U.S. federal loan money had been lost because of poor data quality at a single agency; manufacturing companies spent over 25% of their sales on wasteful practices. The number came up to 40% for service companies. Clearly, as a decision support information system, a data warehouse must provide high level quality of data and service. In various vertical markets (e.g., the public sector) data quality is not an option but a strict requirement for the proper operation of the data warehouse. Thus, data quality problems seem to introduce even more complexity and computational burden to the loading of the data warehouse.

To deal with the complexity of the data warehouse loading process, specialized tools are already available in the market, under the general title *Extraction-Transformation-Loading* (ETL) tools [Evo00, Ard00, Dat00]. ETL as well as *Data Cleaning* tools are pieces of software responsible for the extraction of data from several sources, their cleaning, customization and insertion into a data warehouse.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

A study for Merrill Lynch [ST98] reports some very interesting facts about the situation in the area of ETL and Data Cleaning tools, by the end of 1998. These tools cover a labor-intensive and complex part of the data warehouse processes, which is estimated to cost at least one third of effort and expenses in the budget of the data warehouse. [Dem97] mentions that this number can rise up to 80% of the development time in a data warehouse project. Still, due to the complexity and the long learning curve of these tools [ST98], many organizations prefer to turn to in-house development to perform ETL and data cleaning tasks. The major problems with data cleaning tools are complexity and price. Moreover, due to the nature of the IT departments, which are constrained in time and budget, tools for off-line tasks like data cleaning are pushed aside from the list of products to purchase.

Several commercial tools exist in the market [Evo00, Ard00, Dat00]; a detailed list can be found in [LGI00]. Research efforts include the AJAX prototype developed at INRIA [GFSS00, GFSS99]. A discussion of the research literature on data quality is found in [WSF95]. Finally, a description of our personal engagement in practical aspects of ETL and cleaning in data warehouses can be found in [Vas00]. In the sequel, we will not discriminate between the tasks of ETL and Data Cleaning and adopt the name ETL for both kinds of activities.

In order to pursue research in the field of data transformation and cleaning, we have developed a tool, namely ARKTOS, to achieve the following goals:

- graphical and declarative facilities for the definition of data warehouse transformation and cleaning tasks;
- measurement of the quality of data through specific quality factors;
- optimized execution of complex sequences of transformation and cleaning tasks.

In the rest of this paper we will present the basic ideas behind the implementation and functionality of ARKTOS as well as the declarative languages for the specification of transformation and cleaning scenarios.

2 Model and Functionality of ARKTOS

ARKTOS is based on a simple metamodel to achieve the desired functionality. The main process type of the model is the entity *Activity*. An activity is an atomic unit of work and a discrete step in the chain of data processing. Since activities in a data warehouse context are supposed to process data in a data flow, each activity is linked to *Input* and *Output* tables of one or more databases. An SQL statement gives the *logical*, declarative description of the work performed by each activity (without obligatorily being identical to the actual, *physical* code that performs the execution of the activity). A *Scenario* is a set of processes to be executed all together. A scenario can be considered as the outcome of a design process, where the designer tailors the set of activities that will populate the data warehouse. Each activity is accompanied by an *Error Type* and a *Policy*. Since data are expected to encounter quality problems, we assume that several activities should be dedicated to the elimination of these problems (e.g., the violation of the primary or the foreign key constraint). The error type of an activity identifies the problem the process is concerned with. The policy, on the other hand, signifies the way the offending data should be treated. Around each activity, several *Quality Factors* can be defined. The quality factors are measurements performed to characterize the quality of the underlying information. For the moment, quality factors in ARKTOS are implemented through the use of SQL queries. ARKTOS is also enriched with a set of "template" generic entities that correspond to the most popular data cleaning tasks (like primary or foreign key violations) and policies (like deleting offending rows, reporting offending rows to a file or table).

Connectivity. ARKTOS uses JDBC to perform its connections to the underlying data stores.

Transformation and cleaning primitives. ARKTOS offers a rich variety of primitive operations to support the ETL process. More specifically, the cleaning primitives include: (a) *Primary key violation*, (b) *Reference violation*, (c) *NULL value existence*, (d) *Uniqueness violation* and (e) *Domain mismatch*. Moreover, the tool offers *Propagation* and *Transformation* primitive operations. The propagation primitive simply pushes data to the next layer of storage. The transformation primitive transforms the data to the desired format, according to some pattern (which can be either built-in or user-defined). For example, a transformation primitive can be used to transform a date field from dd/mm/yy to dd/mm/yyyy format. These primitives are customized by the user (graphically or declaratively). The customization includes the specification of input and output (if necessary) data stores, contingency policy and quality factors. For example, in the current version of ARKTOS, the format transformation functions are performed programmatically using a freely available Java Package [ORO00] that simulates the functionality of Perl regular expressions.

Contingency policy. Once a primitive filter is defined in the context of a scenario, it is possible that some rows fulfill its criteria at runtime. For example, a particular row might violate the foreign key constraint for one of its attributes. For each such filter, the user is able to specify a policy for the treatment of the violating rows. For the moment, the policies supported by ARKTOS are: (a) *Ignore* (i.e., do not react to the error and let the row pass), (b) *Delete* (from the source data store), (c) *Report to a contingency file* and (d) *Report to a contingency table*. It is possible that the user is requested to supply some additional parameters (for example, the name of the file where the rows should be reported, or the format to which the values should be changed).

Trace Management. The physical properties of the execution of the ARKTOS scenarios are captured by detailed log information kept for this reason. The *status*, *initialization*, *commit* or *abort* information for each execution of an activity is traced.

Scheduling. ARKTOS uses a freely available Java package [Bra99] to schedule the execution of scenarios that have already been created and saved by the user. To perform this task, the user has to specify, in the correct order, the name(s) of the files where the appropriate scenarios reside. Each of the scenarios participating in a schedule can be executed either once, at a specific time point, or on the basis of a specified repetition frequency (e.g., every Monday, or every 23rd day of each month, at 11:30 am).

3 Declarative Languages for ETL Processes

As we have already pointed out, that the major obstacles the ETL tools have to overcome are the issues of user-friendliness and complexity. To this end, ARKTOS offers two possible ways for the definition of activities: *graphically* and *declaratively*. The graphical definition is supported from a palette with all the possible activities currently provided by ARKTOS. The user composes the scenario from these primitives, links them in a serial list and resolves any pending issues of their definition. In the rest of this section we will focus on the declarative definition of data warehouse processes in the ARKTOS environment.

There is a classical problem with declarative languages and formal specifications: the languages which are easy to read are hard to write and vice-versa. To overcome the problem we resort to two declarative definition languages:

- XADL (*XML-based Activity Definition Language*), an XML language for data warehouse processes, on the basis of a well-defined DTD;
- SADL (*Simple Activity Definition Language*), a declarative definition language motivated from the SQL paradigm.

The former language is rather verbose and complex to write; yet it is more comprehensible. The latter is more compact and resembles SQL; thus it is suitable mostly for the trained designer. We next give an informal presentation of the two languages, by using a motivating example (Figure 1) based on the former TPC-D standard (now TPC-H and TPC-R) [Tra00].

3.1 XADL (XML-based Activity Definition Language)

In Figure 2 we illustrate a subset of the XADL definition for the scenario of the motivating example. Lines 67-102 describe a simple activity. First, in Lines 68-85 the structure of the input table is given. Lines 86-92 describe

1. Push data from table LINEITEM of source database S to table LINEITEM of the DW database.
2. Perform a referential integrity violation checking for the foreign key of table LINEITEM in database DW, which is referencing table ORDER. Delete violating rows.
3. Perform a primary key violation check to the table LINEITEM. Report violating rows to a file.

Figure 1: Description of the scenario of the motivating example

the error type (i.e., the functionality) of the activity which involves foreign key constraint violations. The target column and table are specifically described. Lines 93-95 deal with the policy followed for the identified records and declare that in this case, we simply delete them. The quality factors of the activity are described in Lines 96-101. Each quality factor is characterized by the SQL query that computes its value and the report file where this value will be stored. The only quality factor in Figure 2 is the absolute number of violating rows and is characterized by the SQL query of Line 97. For any valid scenario that we load in ARKTOS, its XADL description can be automatically generated by the tool.

```
1. <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
...
67. <transformtype>
68.     <input_table table_name="lineitem" database_url="jdbc:informix-sqli:
         //kythira.dbnet.ece.ntua.gr:1500/dbs3:informixserver=ol_milos_tcp">
69.     <column> l_orderkey </column>
70.     <column> l_partkey </column>
...
85.     </input_table>
86.     <errortype>
87.         <reference_violation>
88.             <target_column_name> l_orderkey </target_column_name>
89.             <referenced_table_name> Informix.tpcd.tpcd.tpcd.order </referenced_table_name>
90.             <referenced_column_name> o_orderkey </referenced_column_name>
91.         </reference_violation>
92.     </errortype>
93.     <policy>  <delete/>  </policy>
94.     <quality_factor qf_name=No_of_reference_violations qf_report_file="H:\path\scenario3.txt">
95.         <sql_query> select l_orderkey from lineitem t1 where not exists
                        (select o_orderkey from order t2 where t1.l_orderkey = t2.o_orderkey)
            </sql_query>
96.     </quality_factor>
97. </transformtype>
...
140. </scenario>
```

Figure 2: XADL definition of the scenario of the motivating example, as exported by ARKTOS

3.2 SADL (Simple Activity Definition Language)

The SADL language is composed of four definition statements: the `CREATE SCENARIO`, `CREATE CONNECTION`, `CREATE ACTIVITY` and `CREATE QUALITY FACTOR` statements. A `CREATE CONNECTION` statement specifies the details of each database connection. A `CREATE ACTIVITY` statement specifies an activity and a `CREATE QUALITY FACTOR` statement specifies a quality factor for a particular activity. The `CREATE SCENARIO` statement ties all the elements of a scenario together. In Figure 3 we depict the syntax of these four statements.

Connections and activities are the first-class citizens within the context of a scenario. Thus, to declare a `CREATE SCENARIO` statement one has simply to provide the names of the respective connections and the activities of the scenario. The definition of a connection, through the `CREATE CONNECTION` statement is

```
CREATE SCENARIO <scenario_name>
WITH CONNECTIONS <con_1,...,con_m>
ACTIVITIES <act_1,...,act_n>

CREATE CONNECTION <connection_name>
WITH DATABASE <url> ALIAS <db_alias>
[USER <user_name> PASSWORD <password>]
DRIVER <class_name>
```

```
CREATE QUALITY FACTOR <qf_name> WITH ACTIVITY <activity_name>
REPORT TO <file_name>
SEMANTICS <SQL query>

CREATE ACTIVITY <activity_name> WITH TYPE <error_type>
POLICY <policy_type>
[OUTPUT <output_name>(<attr_1,...,attr_m>)]
SEMANTICS <SQL query>
```

Figure 3: The syntax of SADL for the `CREATE SCENARIO`, `CONNECTION`, `ACTIVITY`, `QUALITY FACTOR` statements

equally simple: the database URL and the class name of the respective driver are required. Since the database URL is quite big in size for the user to write down, an `ALIAS` clause is introduced. All table names are required to be in the form <table_name>@<database alias> to distinguish between synonym tables in different databases. The username and password are optional (in order to avoid storing them in the file). `CREATE QUALITY FACTOR` is also a simple statement: one has to specify the activity in the context of which a quality factor is defined, the report to which the value of the quality factor will be saved and the semantics of the quality factor, expressed by an SQL statement (in practice any SQL statement that the database driver and JDBC can support).

The `CREATE ACTIVITY` statement is somewhat more complex. One has to specify first the functionality of the activity in the `TYPE` clause. The <error_type> placeholder can take values from the set {`PUSH`, `UNIQUESS VIOLATION`, `NULL EXISTENCE`, `DOMAIN MISMATCH`, `PRIMARY KEY VIOLATION`, `REFERENCE VIOLATION`, `FORMAT MISMATCH`}. The `POLICY` clause determines the treatment of the rows affected by the activity. The <policy_type> belongs to the set {`IGNORE`, `DELETE`, `REPORT TO FILE`, `REPORT TO TABLE`}. The `OUTPUT` clause specifies the target table or file (if there exists one). If a table is to be populated, all the relevant attributes are specified too. The order of the attributes is important (it must be in one-to-one correspondence with the attributes of the input tables as specified in the SQL query, which will be described later). The `SEMANTICS` clause is filled with an arbitrary SQL query. Several issues should be noted for this clause:

- the order of the attributes in the `OUTPUT` clause should be in accordance with the order of the attributes in the `SELECT` clause of the SQL query;
- the input tables are described in the `FROM` clause of the SQL statement;
- the order of the activities in the `CREATE SCENARIO` statement is important, because it denotes the flow of the activities.

There are standard `CREATE ACTIVITY` statements for all the primitives (i.e., the specialized activities) offered by ARKTOS. In Figure 4 we list them along with syntactic sugar shortcuts which make the life of the designer much easier (remember that the type of the operation is given in the `TYPE` clause of the `CREATE ACTIVITY` statement). Note again that in XADL and SADL we refer only to the logical semantics of the activities and not to the way they are actually executed within the DBMS, which is hard-coded in the subclasses of the ARKTOS architecture.

Figure 5 expresses our motivating example in SADL. In Lines 1-4 we define our scenario, which consists of three activities. The order of the activities appearing in the figure is in descending execution priority. The connection characteristics for connecting to the data warehouse are declared in Lines 6-9. An example of the SADL description of an activity can be seen in Lines 11-16 for the reference violation checking activity. Finally, in Lines 18-22 we give the declaration of a quality factor, which reports to a file the number of foreign key violating rows.

Primitive Operation	SQL statement	SEMANTICS clause shortcuts
UNIQUENESS VIOLATION	SELECT * FROM <table> GROUP BY <attribute> HAVING COUNT(*) > 1	<table>.<attribute>
NULL EXISTENCE	SELECT * FROM <table> WHERE <attribute> IS NULL	<table>.<attribute>
DOMAIN MISMATCH*	SELECT * FROM <table> WHERE <attribute> NOT IN <domain specification>	<table>.<attribute> NOT IN <domain specification>
PRIMARY KEY VIOLATION	SELECT * FROM <table> GROUP BY (<attribute_1,...,attribute_n>) HAVING COUNT(*) > 1	<table>.(<attribute_1, ..., attribute_n>)
REFERENCE VIOLATION	SELECT * FROM <table> WHERE <attribute> NOT IN (SELECT <target_attribute) FROM <target_table>)	<table>.<attribute> NOT IN <target_table>.<target_attribute>
FORMAT MISMATCH**	SELECT APPLY(<reg_exp>,<attribute>) FROM <table> WHERE APPLY(<reg_exp>,<attribute>)	TARGET APPLY(<reg_exp>,<attribute>) SOURCE APPLY(<reg_exp>,<attribute>)
PUSH	Arbitrary SQL query	Arbitrary SQL query

* works for intervals of numbers and strings

** where <reg_exp> is PERL regular expression acting as a formatting function

Figure 4: The SADL specification for the basic primitives offered by ARKTOS.

```
CREATE SCENARIO Scenario3 WITH
CONNECTIONS S3,DW
ACTIVITIES Push_lnitem, Fk_lnitem, Pk_lnitem
...
CREATE CONNECTION DW WITH
DATABASE "jdbc:informix-sqli://kythira.dbnet.ece.ntua.gr:1500/
   dbdw:informixserver=ol_milos_tcp" ALIAS DBDW
DRIVER "com.informix.jdbc.IfxDriver"
...
CREATE ACTIVITY Fk_lnitem WITH
TYPE REFERENCE VIOLATION
POLICY DELETE
SEMANTICS "select l_orderkey from lineitem@DBDW t1 where not exists
   (select o_orderkey from order@DBDW t2 where t1.l_orderkey=t2.o_orderkey)"
...
CREATE QUALITY FACTOR "# of reference violations" WITH
ACTIVITY fk_lnitem
REPORT TO "H:\path\scenario3.txt"
SEMANTICS "select l_orderkey from lineitem@DBDW t1 where not exists
         (select o_orderkey from order@DBDW t2 where t1.l_orderkey = t2.o_orderkey)"
```

Figure 5: Part of scenario 3 expressed in SADL

4 Conclusions and Future Work

In this paper, we have presented ARKTOS, a tool we have developed for modeling and executing practical data management scenarios by providing explicit primitives for the capturing of common tasks (like data cleaning, scheduling and data transformations). Within ARKTOS, we provide three ways to describe such a scenario: a graphical point-and-click front end and two declarative languages. The first one, XADL (an XML variant) is oriented towards an easily understood description of a scenario. The second one, SADL is tailored to support the declarative definition of the ETL scenario in an SQL-like style.

In the future we plan to add more functionality to ARKTOS, in order to provide the users with richer transformation primitives. Several research issues remain open, such as (a) the development of an impact analyzer, based on the results of [VQVJ00], showing how changes in the definition of a table or an activity affect other tables or activities in the data warehouse; (b) the linkage to a metadata repository, and specifically ConceptBase [JGJ+95], in order to exploit its enhanced query facilities, and (c) the construction of an optimizer to attain improved efficiency during the execution of composite scenarios.

References

[Ard00] Ardent Software. DataStage Suite, 2000. See also www.ardentsoftware.com.

[Bra99] Branch Cut Software. JTask: Java Task Scheduler, 1999. Available at www.branchcut.com/jTask.

[CGL+98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Information integration: Conceptual modeling and reasoning support. In *Proceedings of CoopIS'98*, pages 280–291, 1998.

[Dat00] DataMirror Corporation. Transformation Server, 2000. See also www.datamirror.com.

[Dem97] M. Demarest. The politics of data warehousing, 1997. Available at www.hevanet.com/demarest/marc/dwpol.html.

[Evo00] Evolutionary Technologies International. ETI*EXTRACT, 2000. See also www.eti.com.

[GFSS99] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. An extensible framework for data cleaning. Technical Report RR-3742, INRIA, 1999.

[GFSS00] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An Extensible Data Cleaning Tool. In *Proceedings of ACM SIGMOD-2000*, June 2000.

[JGJ+95] M. Jarke, R. Gallersdorfer, M.A. Jeusfeld, M. Staudt, and S. Eherer. ConceptBase - a deductive objectbase for meta data management. *Intelligent Information Systems*, 4(2), 1995.

[LGI00] LGI Systems Inc. The Data Warehousing Information Center, 2000. See also www.dwinfocenter.org.

[ORO00] ORO Inc. PerlTools 1.2, 2000. Available at www.savarese.org/oro.

[ST98] C. Shilakes and J. Tylman. Enterprise Information Portals. Enterprise Software Team, November 1998. Available at www.sagemaker.com/company/downloads/eip/indepth.pdf.

[Tra00] Transaction Processing Performance Council. TPC-H and TPC-R, 2000. Available at www.tpc.org.

[Vas00] P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In *Proceedings of DMDW'2000*, June 2000.

[VQVJ00] P. Vassiliadis, C. Quix, Y. Vassiliou, and M. Jarke. A Model for Data Warehouse Operational Processes. In *Proceedings of CAiSE'00*, June 2000.

[WRK95] R.Y. Wang, M.P. Reddy, and H.B. Kon. Towards Quality Data: An attribute-based Approach. *Decision Support Systems*, 13, 1995.

[WSF95] R.Y. Wang, V.C. Storey, and C.P. Firth. A Framework for Analysis of Data Quality Research. *IEEE Transactions on Knowledge and Data Engineering*, 7(4), 1995.