PERGAMON

Information Systems 26 (2001) 537–561

www.elsevier.com/locate/infosys

ARKTOS: towards the modeling, design, control and execution of ETL processes

Panos Vassiliadis*, Zografoula Vagena, Spiros Skiadopoulos, Nikos Karayannidis, Timos Sellis

Knowledge and Database Systems Laboratory, Department of Electrical and Computer Engineering, Computer Science Division, National Technical University of Athens, Iroon Polytechniou 9, 157 73 Athens, Greece

Abstract

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. Literature and personal experience have guided us to conclude that the problems concerning the ETL tools are primarily problems of complexity, usability and price. To deal with these problems we provide a uniform metamodel for ETL processes, covering the aspects of data warehouse architecture, activity modeling, contingency treatment and quality management. The ETL tool we have developed, namely ARKTOS, is capable of modeling and executing practical ETL scenarios by providing explicit primitives for the capturing of common tasks. ARKTOS provides three ways to describe an ETL scenario: 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. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords: Data warehousing; Data quality; Extraction-Transformation-loading tools

1. Introduction

It has been argued in the literature [1] that viewing the data warehouse as a set of layered, materialized views is a very simplistic view. For example, the data warehouse refreshment process can already consist of many different subprocesses, like data cleaning, archiving, transformations and aggregations, interconnected through a complex schedule [2]. The reasons for such a structure of processes are both *technological* and *qualitative*.

The technological reasons are basically summarized by the observation that a data warehouse is a heterogeneous environment where data must be integrated both at the schema and at the instance level [3–6]. The schema integration alone is a painful procedure involving resolution for various types of conflicts, schema conforming and restructuring [4,7]. Still, even if the schema integration process results in a unified schema, the physical properties of data, along with timelines constraints, impose a complex process structure for the loading of the warehouse with data from the sources. To give an idea of this complexity, let us

*Corresponding author. Tel.: +30-1-772-1402; fax: +30-1-772-1442.

E-mail addresses: pvassil@dbnet.ece.ntua.gr (P. Vassiliadis), zvagena@dbnet.ece.ntua.gr (Z. Vagena), spiros@dbnet.ece.ntua.gr (S. Skiadopoulos), nikos@dbnet.ece.ntua.gr (N. Karayannidis), timos@dbnet.ece.ntua.gr (T. Sellis) .

mention a case study from our practical experience, where a small part of the loading process for a certain data source involved detecting relevant data from a COBOL file, converting EBCDIC to ASCII format, unpacking the packed numbers, reducing all address fields to a standard format and loading the result into a table in the data warehouse.

At the same time, practice has shown that neither the accumulation nor the storage 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 [8]. Ref. [9] reports that more than $2 billion of US 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, being a decision support information system, a data warehouse must provide high-level quality of data and services. In various vertical markets (e.g., the public sector) data quality is not an option but a constraint 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. To give a general idea of the functionality of these tools we mention their most prominent tasks, which include:

- the identification of relevant information at the source side;
- the extraction of this information;
- the customization and integration of the information coming from multiple sources into a common format;
- the cleaning of the resulting data set, on the basis of database and business rules and
- the propagation of the data to the data warehouse and/or data marts.

In the sequel, we will not discriminate between the tasks of ETL and data cleaning and adopt the name ETL for both these kinds of activities.

A study for Merrill Lynch [10] 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 are estimated to cost at least one third of effort and expenses in the budget of the data warehouse. Ref. [11] 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, many organizations prefer to turn to in-house development to perform ETL and data cleaning tasks. Ref. [10] sounds optimistic though, in the sense that it is expected that ETL and data cleaning tools will mature enough to outweigh their disadvantages with concrete solutions for data of high quality.

Ref. [10] discriminates between the tasks of ETL and cleaning, although it acknowledges that data cleaning is just another instance of data transformation. The involved tools, possibly due to their nature, do not always combine these functionalities. As already mentioned, most companies spend more than one third of their data warehouse budget to ETL software which is usually built in-house. The reasons for not choosing the solution of a commercial tool are the following, according to [10]:

- many of the earlier ETL tools were complex, difficult to use and expensive;
- companies have already been involved in this kind of development for their legacy systems for years and;
- many in-house developers enjoy working on this intriguing problem, which is clearly hard to solve.

Ref. [10] estimates that around the year 2000 the ETL tools would be mature enough to revert the tendency of in-house development and to reach a compound annual growth rate (CAGR) of 20% until 2002 (Fig. 1).

As for data cleaning tools, [10] is underlying their "paramount" importance to organizations that are moving towards a service-oriented profile. This is because accurate, timely and consistent information on customers can potentially produce vast cut-offs in budget expenses. The major problems with data cleaning tools are again complexity and price. Another problem is that 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 Fig. (1).

Our personal experience in the data warehouse field adds as a witness to the problems indicated by [10]. We will mention a couple of projects to show the particular problems we encountered during the construction of data warehouses. The first project involved loading data from all the health centers (i.e., hospitals, provincial medical centers and other special kinds of centers) of Greece into an enterprise data warehouse. The loading of data was performed annually and the querying was supposed to be performed mostly by pre-canned reports. The users were enabled to perform standard on-line analytical processing (OLAP) operations like filtering, roll-up, drill-down and drill-through of the data. The data warehouse was rather small and its construction took around 12 months. Still, the development of the software for the loading of the warehouse was delayed due (a) to communication problems with the administration team of the source system, (b) data quality problems and (c) evolution reasons. It was quite common a phenomenon that data were incomplete and unreliable. Moreover, sudden changes in the requirements of the users would result in the evolution of the warehouse, causing extra delays. In a second occasion, we had to build a data warehouse with pension data. The data were to be updated monthly and used by pre-canned reports. The size of data involved a few million rows per month. The source data resided again in a COBOL-based legacy system. The project lasted 9 months and could be characterized more as the construction of a data mart rather than the construction of a full data warehouse. The problems we encountered were of two kinds: (a) problems of political nature (which we will not discuss in this context, but refer the reader to [11,12]) and (b) the construction of the extraction and cleaning software. We discovered that the extraction of data from the legacy systems is a highly complex, error-prone and tiring procedure, of which we have already given an example some paragraphs ahead. The tool offered by Oracle for these purposes (SQL*Loader) was good enough only for the loading of the warehouse from ASCII files; the rest of the cleaning and transformation tasks had to be hard-coded by us.

As it has already been demonstrated, both the relevant literature and our personal experience suggest that the problems concerning the ETL tools are not only performance issues (as one would normally expect), but also issues of complexity, usability and price. *To this end, the contribution of this paper is towards the presentation* (a) *of a uniform model covering all the aspects of an ETL environment and* (b) *of a platform capable to support practical ETL scenarios with particular focus on issues of complexity, usability and maintainability*. Various aspects of our approach are contributing to the achievement of these goals. First of all, we provide a uniform metamodel for ETL processes, covering the aspects of data warehouse architecture, activity modeling, contingency treatment and quality management. Building on previous research results [1,13,14], we pay particular attention to the modeling and management of quality within the ETL process. The ETL tool we have developed, namely ARKTOS, is based on the described metamodel to show the feasibility of the proposal. ARKTOS is capable of modeling and executing practical ETL scenarios by providing explicit primitives for the capturing of common tasks (like data cleaning, scheduling and data transformations). To enhance the usability of the tool, we

	1998	1999	2000	2001	2002	CAGR
ETL	101.1	125.0	150.0	180.0	210.0	20.1%
Data Cleaning	48.0	55.0	64.5	76.0	90.0	17.0%

Fig. 1. Estimated sales in $Millions [10].

provide three ways to describe an ETL scenario: 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 constructing ETL scenarios. As far as the maintainability and extensibility of the tool is concerned, our system is characterized by a modular, easily customizable architecture based on the modularity of the metamodel. We anticipate that the modularity of the software architecture will enable us to easily experiment on alternative propagation, transformation and cleaning algorithms in the future, in order to achieve better response times from Arktos.

The structure of this paper is as follows. In Section 2, we present background research work which has influenced the definition of proposed metamodel for ETL processes, along with the metamodel itself. In Section 3, the architecture and functionality of Arktos are presented. In Section 4, we describe declarative languages for the definition of ETL processes. We discuss related work in Section 5 and conclude our results along with the presentation of future work in Section 6.

2. Background and metamodel for ETL processes

2.1. Background

The model for ETL activities lies on top of the basic ideas of the European basic research project "Foundations on Data Warehouse Quality" (DWQ) [15,16]. The DWQ approach assumes the existence of a metadata repository, where the definitions of data warehouse objects, processes and quality factors are stored. The set of basic assumptions, which have been used to describe the schema of the metadata repository, form the DWQ *metadata framework* (Fig. 2). These assumptions include:

1. A clear distinction between different *layers of instantiation*. Thus, for all the models describing different aspects of the data warehouse, there exists a generic *metamodel layer*, which deals abstractly with entities applicable to any data warehouse; also a *metadata layer* dealing with the schemata of a specific data warehouse under examination; and finally an *instance layer* representing the real world (as an instance of the previous layers). Out of the three layers of instantiation, at least the two first should be

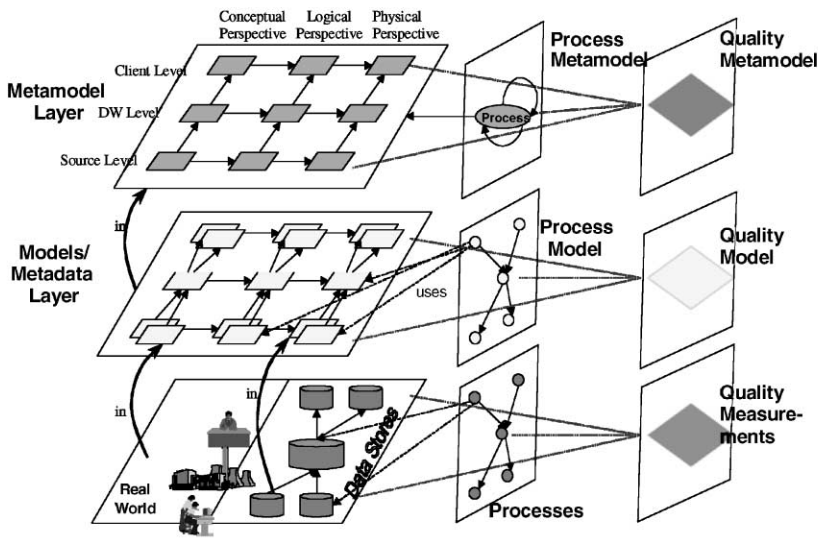


Fig. 2. Framework for data warehouse architecture [1].

clearly captured by the metadata repository. The instance layer, though, is optional as far as the metadata repository is concerned, since it deals with the actual information (and not its "meta" description).

2. A clear distinction between *perspectives*. Following a classical categorization scheme in the field of information systems, the metadata framework relies on the separation of (a) the *conceptual perspective* that captures the world in a vocabulary close to the one of the final user; (b) the *physical perspective* that covers the data warehouse environment in terms of real-world, computer-oriented components or events and finally, (c) the *logical perspective* which acts as an intermediate between these two extremes in order to provide a concrete vocabulary of the data warehouse environment, being independent, though, from strict implementation details.

In [1], the authors have presented a metadata modeling approach which enables the capturing of the *static* parts of the architecture of a data warehouse, along with information over different quality dimensions of these components. Apart from the respect of the metadata framework, a classical data warehouse separation between sources, central warehouse and clients is also introduced. In [14], this architecture model was complemented with the metamodeling for the *dynamic* part of the data warehouse environment: the *processes*. The paper extends traditional workflow modeling with a conceptual perspective to processes, quality management, and exploitation of a metadata repository.

The DWQ approach has focused on the management of data warehouse quality, based on the goal-Question-metric approach (GQM) described in [17]. The metamodel for the relationship between architecture objects and quality factors is presented in [18]. Each object in the data warehouse architecture is linked to a set of quality goals and a set of quality factors. A *quality goal* is a high-level user requirement, defined on data warehouse objects, and documented by a purpose and the stakeholder interested in it. For example, "improve the usability of the produced data cleaning software with respect to the data warehouse administrator" is a quality goal defined on the object "data cleaning software" with the purpose of "improvement" and a particular class of stakeholders involved. *Quality dimensions* (e.g., "availability") constitute the users' vocabulary to formulate quality goals. At the same time, quality dimensions act as classifiers of quality factors into different categories. A *quality factor* represents a quantitative assessment of a particular aspect of a data warehouse object, i.e., it relates quality aspects both to actual measurements and expected ranges for these quality values.

The bridge between the abstract, subjective quality goals and the specific, objective quality factors is determined through a set of *quality queries* (or questions), to which quality factor values are provided as possible answers. Quality questions are the outcome of the methodological approach described in [13]. The methodology exploits "template" quality factors and dimensions defined at the metadata level and instantiates them for the specific data warehouse architecture under examination. As a result of the goal evaluation process, a set of improvements (e.g., design decisions) can be proposed, in order to achieve the expected quality.

2.2. The metamodel of ARKTOS

In Fig. 3, we can see the basic types involved in the ARKTOS metamodel. On the upper side of the picture, one can see the *metamodel* layer, where all the generic entities are found. The entities belonging to the process model are colored white, the entities of the architecture model are colored gray and the entities of the quality model are colored black. The entity *activity* is the main process type. An activity is an atomic unit of work and a discrete step in the chain of data processing in an ETL environment. Since an activity in an ETL context is 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 actual, *physical* code that performs the execution of the activity). A *scenario*

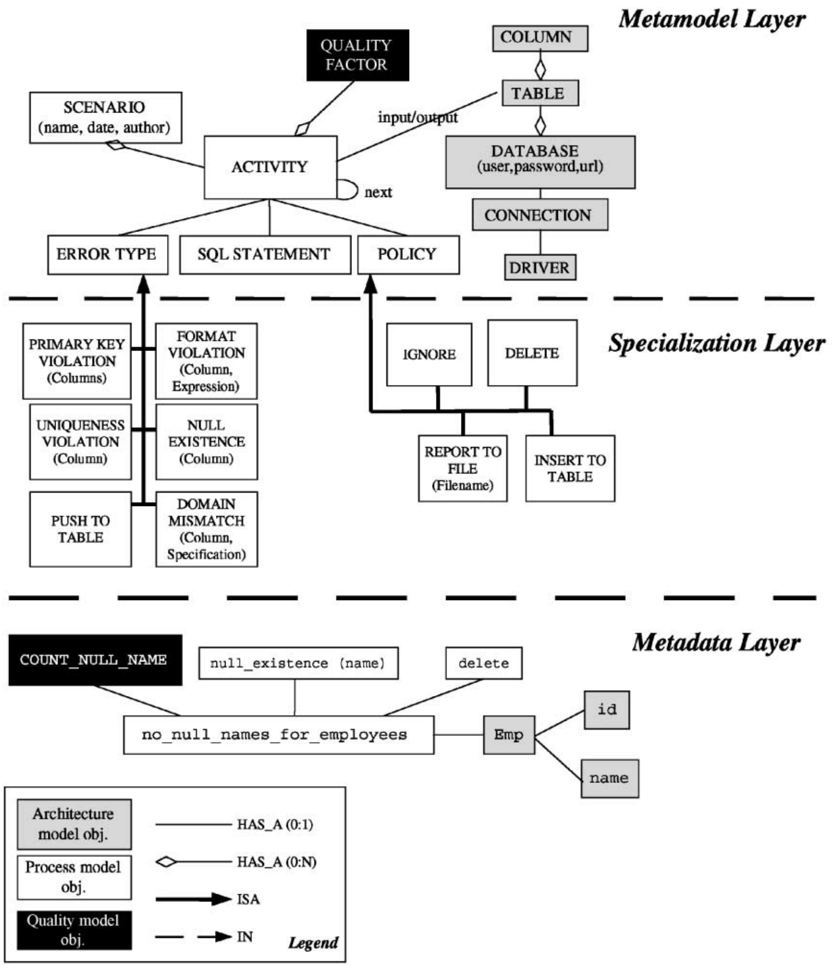


Fig. 3. The metamodel of Arktos.

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 ETL processes that will populate the data warehouse. Each activity is accompanied by an *error type* and a *policy*. Since data are likely to encounter quality problems, we assume that a large part of the activities of a scenario are dedicated to the elimination of these problems (e.g., the violation of constraints, like the primary or the foreign key constraint). The error type of an activity identifies which kind of problem the process is concerned with. The policy, on the other hand, signifies the way the offending data are going to be treated. Each activity is a fixed point in the chain of computations of the overall ETL process. Around each activity, several *quality factors* can be defined. The quality factors are measurements performed in order to characterize the status of the underlying information. For the moment, quality factors in Arktos are implemented through the use of SQL *queries*. On the right side of the metamodel layer, we depict the objects belonging to the architecture model: each *connection* refers to a set of *databases*, which comprise *tables* (and their *columns*). A *driver* determines the physical properties of the connection.

Each of the metamodel entities is generic enough to capture the majority of ETL tasks.

Still, ARKTOS is enriched with a set of "template" generic entities that correspond to the most popular ETL tasks. In the middle of Fig. 3, one can observe this *specialization* layer involving error types like *primary key violation*, *foreign key violation*, etc. and policies like *delete offending rows*, *output offending rows to a report*, etc. Clearly, the *specialization* layer is a subset of the *metamodel* layer.

The lower part of Fig. 3 presents the *metadata* layer. The metadata layer involves the particular scenarios applicable to a specific data warehouse (i.e., specific tables, activities and quality factors). In Fig. 3 we can see the case where we define an activity to delete from table `Emp` all the rows having a `NULL` value in the `Name` attribute. For clarity we use upper case letters for the entities of the metamodel layer and lower case letters for the entities of the metadata layer. Thus, the object `no_null_names_for_the_employees` is an instance of the `ACTIVITY` entity in the metamodel layer, whereas the entity `null_existence(Name)` is an instance of the `NULL EXISTENCE` entity. The entity `delete` is an instance of the entity `DELETE` in the metamodel layer. The quality factor `count_null_names` is an instance of the entity `QUALITY FACTOR` and is locally characterized by the SQL statement `SELECT * FROM EMP WHERE NAME IS NULL`. We can observe that all the entities of the metadata layer are instances of the metamodel layer. The instances of the metadata layer are the traces of the execution of the activity `no_null_names_for_the_employees` in the real world.

Formally, an *activity* can be defined in terms of the following elements:

- *Input table*(*s*). One or more input tables, belonging to a certain database.
- *ErrorType*. A characterization of the functionality of the activity. The implemented domain is {`PUSH, UNIQUESS VIOLATION, NULL EXISTENCE, DOMAIN MISMATCH, PRIMARY KEY VIOLATION, REFERENCE VIOLATION, FORMAT MISMATCH`} (with the obvious semantics). All these different alternatives are modeled as subclasses in the metamodel and the implementation of ARKTOS.
- *Policy*. A declaration of the treatment of detected tuples. The implemented domain is {`IGNORE, DELETE, REPORT TO FILE, REPORT TO TABLE`}. Again, all these different alternatives are modeled as subclasses in the metamodel and the implementation of ARKTOS.
- *Output table*. A single output table towards which the data are propagated. The field is optional for several kinds of activities.
- *Quality factor*(*s*). A set of quality factors related to the activity, each of which is characterized from an SQL statement.

For reasons of brevity, we omit the formal definition of the rest of the entities of the metamodel of ARKTOS (*scenario*, *connection*, etc.). Still, it is straightforward to derive it from the UML definition in Fig. 3.

3. Architecture and functionality of ARKTOS

3.1. The architecture of ARKTOS

ARKTOS comprises several cooperating modules, namely the *SADL* and *XADL* parsers, the *graphical scenario builder* and the *scheduler*. The architecture of ARKTOS in terms of software modules is depicted in Fig. 4.

The *graphical scenario builder* is the front-end module of the application, i.e., the module employed by the user to graphically sketch the logical structure of an ETL process. The scenario builder equips the designer with a palette of primitive transformations and data stores. The user graphically constructs the architecture of the data flow within the system. The primitive processes are instantiated with concrete values for the particular characteristics of the data stores to which they are related, as well as to the characteristics of each particular execution. The graphical user interface (GUI) consists of a number of visual objects which guide the user in the process of sketching the sequence of the transformation steps: rectangle shapes represent data stores, oval shapes represent transformation activities, rounded-corner rectangles represent user-defined quality factors. The priority of

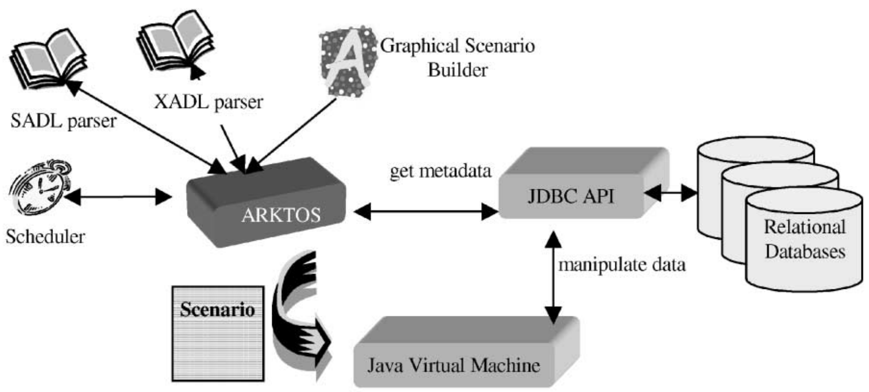


Fig. 4. The architecture of Arktos.

activities and the linkage of activities to data stores are depicted with links between them. To complete the description of the software architecture of the scenario builder we also mention a number of windows (`JFrames` in Java terminology) that provide the ability both to have a view of the already performed activities (for example, dialog boxes that show the generated SQL) and to fill in the necessary parameters.

The *XADL* and *SADL* parsers are the declarative equivalent of the graphical scenario builder. The structure of the parsers is classic (including lexical and structural analysis); thus, we do not provide any more details on these modules.

The *central Arktos module* performs three typical tasks: (a) it interprets a submitted scenario to a set of Java objects, (b) it places these objects into a sequence of executable steps and (c) it performs all the consistency checks to ensure the correctness of the design process. The interpretation of the graphical entities/declared commands within a scenario to Java objects is performed straightforwardly. It is important to note that the activities appear in linear sequence in the scenario and are similarly represented in the main memory: thus, their mapping to Java objects is performed in an one-way pass. The creation of the execution sequence is performed by placing each object in its proper place within a linear object placeholder such as vector, provided by the Java language. The consistency checking involves (a) the detection of possible cycles in the definition of a process flow, (b) the identification of isolated activities in the scenario and (c) the proper definition of all the activities within a scenario, with respect to input/output attributes, scripts and policies.

The classes used for the internal representation of activities are grouped into three packages according to their functionality. Precisely, there exists a package named `Transformtypes` that consists of the classes that model the individual transform activities, a package named `Errortypes` that consists of the classes that model the individual cleaning primitives and a package named `Policies` that consists of the classes that model the possible treatment of the detected rows from an activity. It is important to note that each instance of a class belonging to the first package is sufficient to hold all the necessary information for the successful performance of an individual transformation step (i.e., the involved source and target data stores as well as the information on the functionality of the activity are all captured by this particular instance). As a general rule, each of the aforementioned packages includes an abstract class that groups the shared attributes and methods together, acting as a base class for the rest of the classes that actually support the functionality of the tool. Thus, the metamodel of Fig. 3 is directly represented in the software architecture of Arktos. The result is enhanced modularity and extensibility without significant overhead due to the late binding of methods at run-time (remember, that in an ETL environment this overhead is only a tiny fraction of the time necessary to process millions of rows). Additionally, there exists a package named `Connections` which provides the classes to implement all the connections to the data sources, the mapping of the metadata of the underlying data stores to Java objects in the ARKTOS environment (e.g., databases, tables, data types of columns, etc.), as well as the extraction of actual data from them.

The *scheduler* is the module that is used to schedule the execution of previously created and saved scenarios. The scheduling is performed with the help of a commercial tool called jTask [19] which is used to schedule the execution of Java classes (something like a cron utility for Java classes). Precisely, jTask schedules the execution of a particular Java class of ARKTOS, which is responsible for the loading of the saved scenarios in the background and their execution in the sequel. Naturally, the whole process is completely transparent to the user.

The development of ARKTOS was performed exclusively in Java2, using several API's (SWING, Java2D, JDBC) that the JDK 1.2.2 environment provides.

3.2. Motivating example

In the sequel, we will present a motivating example upon which we will base the presentation of ARKTOS. The example is based on the former TPC-D standard (now evolved into the TPC-R and TPC-H standards [20]). We assume the existence of three source databases S_1, S_2 and S_3 as well as a central data warehouse under the schema of the TPC-D standard. The schemata of the sources and the data warehouse are depicted in Fig. 5.

Each of the sources populates a certain table of the central warehouse. Table LINEITEM, though, is populated from more than one sources. We choose to construct a different scenario for each of the

Table: DW

PART	P_PARTKEY, P_NAME, P_MFGR, P_BRAND, P_TYPE, P_SIZE, P_CONTAINER, P_RETAILPRICE, P_COMMENT
SUPPLIER	S_SUPPKEY, S_NAME, S_ADDRESS, S_NATIONKEY, S_PHONE, S_ACCTBAL, S_COMMENT
PARTSUPP	PS_PARTKEY, PS_SUPPKEY, PS_AVAILQTY, PS_SUPPLYCOST, PS_COMMENT
CUSTOMER	C_CUSTKEY, C_NAME, C_ADDRESS, C_NATIONKEY, C_PHONE, C_ACCTBAL, C_MKTSEGMENT, C_COMMENT
ORDER	O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT
LINEITEM	L_ORDERKEY, L_LINENUMBER, L_PARTKEY, L_SUPPKEY, L_QUANTITY, L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS, L_SHIPDATE, L_COMMITDATE, L_RECEIPTDATE, L_SHIPINSTRUCT, L_SHIPMODE, L_COMMENT
NATION	N_NATIONKEY, N_NAME, N_REGIONKEY, N_COMMENT
REGION	R_REGIONKEY, R_NAME, R_COMMENT
TIME	T_TIMEKEY, T_ALPHA, YYYY-MM-DD, T_YEAR, T_MONTH, T_WEEK, T_DAY

Table: S_1

PARTSUPP	(identical to the one of DW)

Table: S_2

LINEITEM	(identical to the one of DW)
ORDER	(identical to the one of DW)

Table: S_3

LINEITEM	(identical to the one of DW)

Fig. 5. The schemata of the source databases and of the central data warehouse.

sources, involving the propagation of data from the source to the warehouse as well as in some cleaning and transformation activities:

- The first scenario involves the propagation of data for table PARTSUPP (suppliers of parts) from source S_1 to the warehouse. This scenario includes some testing of NULL values for the attribute AVAILQTY of this table.
- The second scenario involves the loading of data from tables ORDER and LINEITEM of source S_2 to the central warehouse. The propagated data are checked for foreign key violation on customers. In Fig. 6, one can observe the data flow by following the vertical arrows (from the sources to the targets) and the sequence of the activities of the scenario (in oval shapes), by following the horizontal arrows.
- The third scenario loads S_3: LINEITEMS from sources S_3. Redundancy (i.e., primary key violation) as well as foreign key violations are present during this loading and are detected from the ARKTOS primitives.

One can observe that we choose to construct small-size scenarios dealing with portions of the overall loading processes and coordinate them later, through the scheduler. This reflects a broader problem that we faced in the making of our design choices both for the functionality and the architecture of ARKTOS. Actually, in the design of ARKTOS, we had to face a dilemma between two conflicting approaches. On the one hand, our practical experience indicated that the loading of the warehouse involves several small-size scenarios, possibly performed at different time instances (i.e., not fully synchronized). On the other hand, one could possibly envision complex lengthy scenarios coordinating all the possible loads. We believe that the dual nature of the programming facilities of ARKTOS that we chose, can serve both categories of ETL scenarios. Still, in the context of complex scenarios, the testing of the efficiency and the usability of ARKTOS is not yet fully explored.

3.3. *Functionality of* ARKTOS

The construction of the scenario of the ETL process is supported by several functionalities offered by ARKTOS.

Connectivity. ARKTOS is based on the JDBC protocol to perform its connections to the underlying data stores. All relational databases that

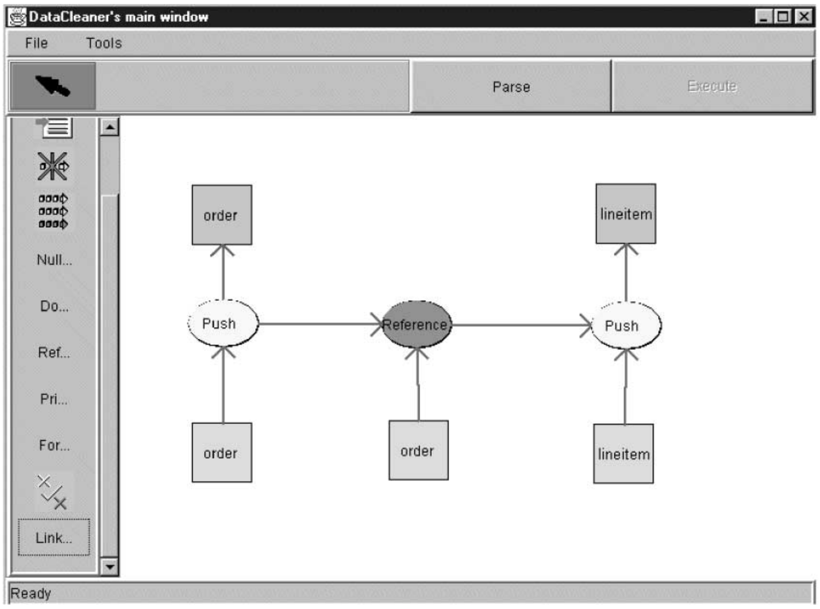


Fig. 6. Scenario for the propagation of data from sources S_2 to the data warehouse.

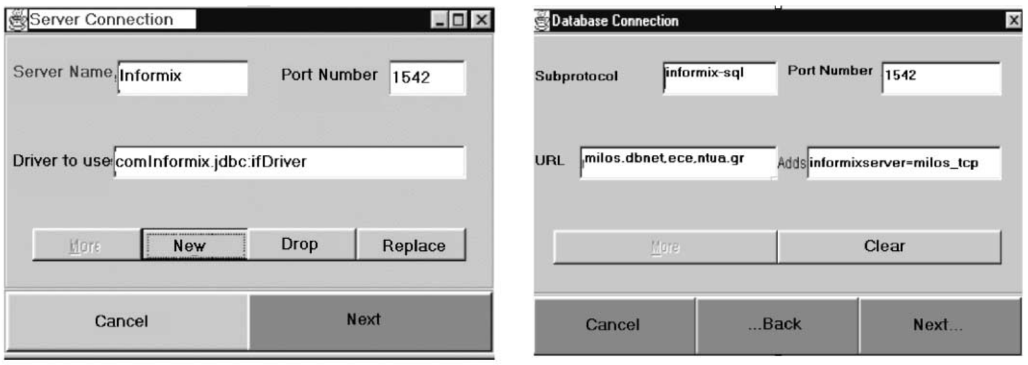
provide a JDBC driver of Types 3 and 4 can be used as input or output data stores for the ETL process. In order to connect to a data store, a number of parameters must be provided, i.e., the name of the driver to be used, the URL of the database server, the name of the target database as well as additional parameters about the driver and the employed protocols. The connection process is presented in Fig. 7.

The tool automatically retains the provided information so that the latter can be reused in subsequent scenario builds.

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 checking*, (b) *reference violation checking*, (c) *NULL value existence checking*, (d) *uniqueness violation checking*, and (e) *domain mismatch checking*. Moreover, ARKTOS offers *propagation* and *transformation* primitive operations. The propagation primitive simply pushes data to the next layer of storage and the transformation primitive transforms the data in 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 into 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. Each operation is mapped to an SQL statement; still it is not obligatory that its execution is performed the same way. For example, in the current version of ARKTOS, the format transformation functions are performed programmatically using a freely available Java Package [21] that simulates the functionality of Perl regular expressions. Below follows a description of the steps required to customize each primitive operation of ARKTOS.

To clean the rows violating a *primary key constraint*, the user has to provide the columns that should be checked as well as the columns to be retrieved. It should be pointed out that the tool takes care of the correctness of the derived SQL statement (i.e., that the columns checked should also exist in the select part of the statement). To detect the rows with a *reference violation* the user has to provide the column to be checked, as well as the database, table and column being referenced. If a primary key exists at the target table, then the tool points it out to the user by displaying it as a default selection. To check for rows having NULL values in a particular column, the user has to provide the column that should be checked and the set of columns to be retrieved (i.e., the schema of the output of the activity). Since the set of columns to be reported or propagated may be different from the set which is checked for NULL values, the user has to explicitly specify which of them he is interested in. Similar parameters should be provided in the case of checking values that present a distinct value violation. Again the tool helps the


Server Connection
Server Name: Informix
Port Number: 1542
Driver to use: comInformix.jdbc:ifDriver

Database Connection
Subprotocol: informix-sql
Port Number: 1542
URL: milos.dbnet.ece.ntua.gr
Adds: informixserver=milos_tcp

Fig. 7. Connecting to a relational database.

user by displaying the primary key of the target table, if one exists. In the case of *domain checking* the tool searches for rows on a particular column whose type is numeric. Specifically, the user specifies the range of permissible numeric values for the data of the examined column. For example, he has the ability to restrict the integer values of a column to just the integers greater than a given value.

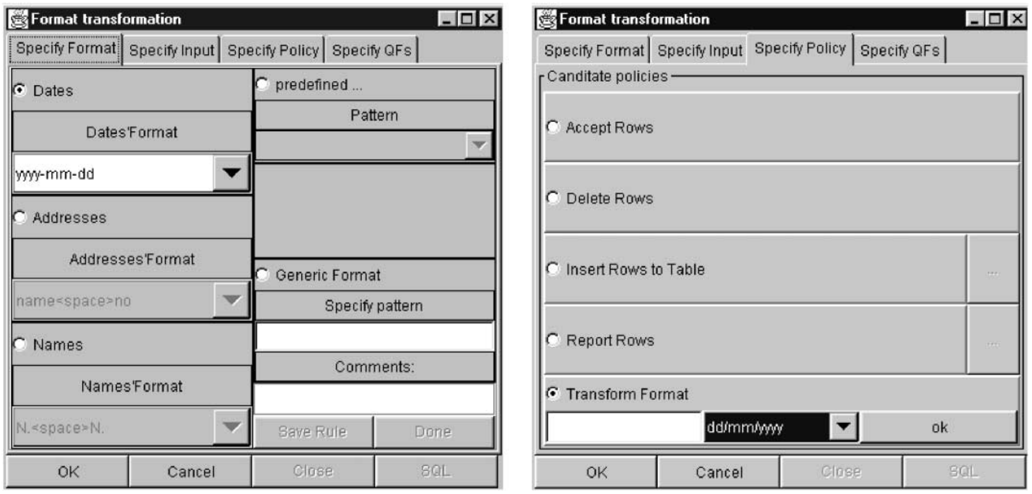
In the case of a *format transformation* a specified column is checked against a provided string pattern. ARKTOS supports a number of commonly encountered formats for dates, addresses and primary names, along with other predefined or ad hoc formats. The user has to provide the column to be checked, the columns to be retrieved, as well as the pattern against which the check is to be performed. Let us take Fig. 8 as an example of the functionality of format transformation. In this case, the user wants to detect the rows from table `Order`, having their `o_orderdate` attribute in the form `yyyy-mm-dd` and transform them in the form of `dd/mm/yyyy`. On the left side of Fig. 8, one can observe that the user has chosen the appropriate filter for the input rows. The policy for the output rows is shown on the right side of Fig. 8: the user specifies that the output is transformed into a row with its `o_orderdate` field in `dd/mm/yyyy` format. As already mentioned, ARKTOS also gives the ability to the users to define a new pattern by providing an ad hoc regular expression or to use previously defined formats.

Contingency policy. Once a primitive filter is defined in the context of a scenario, it is possible that several 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 as follows:

- *Ignore* (i.e., do not react to the error and let the row pass)
- *Delete* (from the source data store)
- *Report to a contingency file*
- *Report to a contingency table*
- *Transformation from one format into another* (*in the case of format matching*).

Within the tool, the user specifies the desired treatment for the rows identified by the error type of the activity, by selecting one of the aforementioned policies. 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—see Fig. 8b).

Trace management. As mentioned in [14,22,23], there are different *perspectives* to view a process:



[✓ Dates] Dates'Format: yyyy-mm-dd
Addresses Addresses'Format: name<space>no
Names Names'Format: N.<space>N.
predefined ...

Accept Rows
Delete Rows
Insert Rows to Table
Report Rows
[✓ Transform Format] dd/mm/yyyy

Fig. 8. Specifying the check for format transformation.

what elements it consists of (*logical* perspective), *why* these elements exist (*conceptual* perspective) and *how* these elements perform during the execution of their functionality (*physical* perspective). Although ARKTOS is not currently covering the conceptual part of this triplet, it fully covers the logical and physical perspectives. The physical properties of the execution of the data warehouse ETL processes are captured by detailed log information reporting the individual steps of the whole process. In order to provide a clear idea on the functionality of each element the *status*, *initialization*, *commit* or *abort* information for each execution of an activity is traced, as shown in Fig. 9.

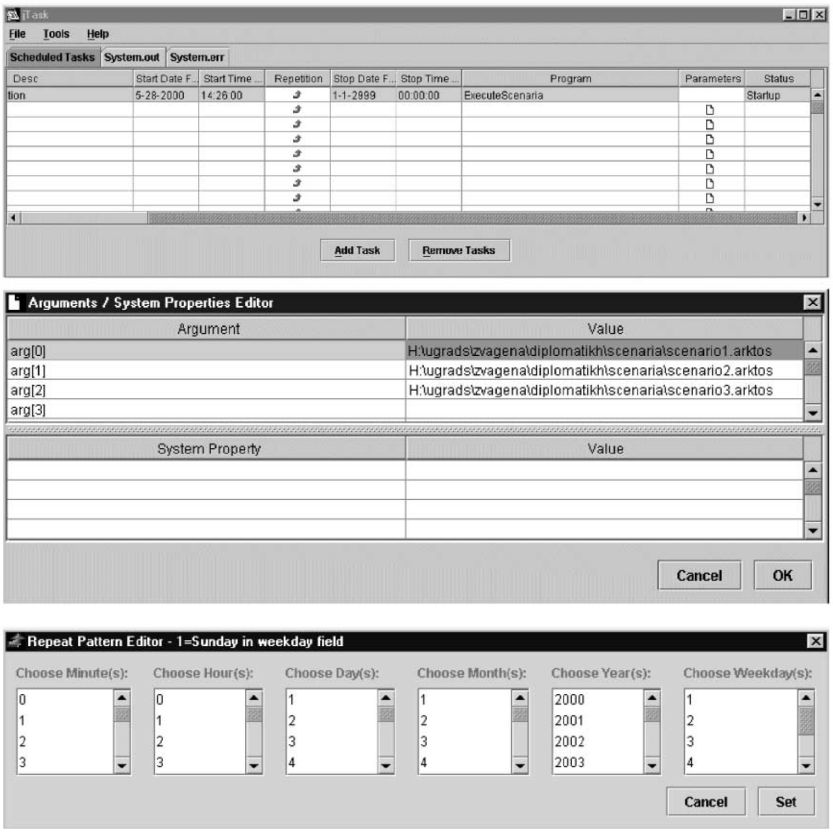
Scheduling. ARKTOS uses a freely available Java package [19] to schedule the execution of scenarios that have already been created and saved by the user. To perform this task, the user has to specify, in the correct order, the name(s) of the files where the appropriate scenarios reside. Each of the scenarios 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 a.m.). In Fig. 10 we see (a) the main window, where the user specifies a composite scenario, comprising of the three scenarios of our motivating example, to be executed in serial order; (b) the way this composition is performed and (c) the options for the time scheduling of the composite scenario.

4. Declarative languages for ETL processes

As we have already pointed out, 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. Since we have already elaborated on the facilities for the graphical definition of a scenario,

```
Log File:scenario1.log
-----------------------------------------------
Results:
Activity   Name='push   data   from   table   partsupp   to   table
partsupp',State='init',time='Wed May 31 13:56:07 GMT+03:00 2000'
Activity   Name='push   data   from   table   partsupp   to   table
partsupp',State='commit',time='Wed May 31 13:56:32 GMT+03:00 2000'
Activity Name='Identify null value rows',State='init',time='Wed May 31
13:56:32 GMT+03:00 2000'
Activity Name='Identify null value rows',State='commit',time='Wed May
31 13:56:32 GMT+03:00 2000'
Activity  Name='Apply  Policy  Accept',State='init',time='Wed  May  31
13:56:32 GMT+03:00 2000'
Activity  Name='Apply  Policy  Accept',State='commit',time='Wed  May  31
13:56:32 GMT+03:00 2000'
Activity  Name='Compute  %  of  nulls',State='init',time='Wed  May  31
13:56:32 GMT+03:00 2000'
Activity  Name='Compute  %  of  nulls',State='commit',time='Wed  May  31
13:56:33 GMT+03:00 2000'
Activity Name='Report QFs for table partsupp',State='init',time='Wed
May 31 13:56:33 GMT+03:00 2000'
Quality Factors for table partsupp
#null valued rows  -->  75
#percentage of null valued rows  -->  0.09375
Activity Name='Report QFs for table partsupp',State='commit',time='Wed
May 31 13:56:33 GMT+03:00 2000'
'**...........................................**
```

Fig. 9. A log for the execution of Scenario 1.



Desc	Start Date F...	Start Time ...	Repetition	Stop Date F...	Stop Time ...	Program	Parameters	Status
tion	5-28-2000	14:26:00		1-1-2999	00:00:00	ExecuteScenaria		Startup

Arguments / System Properties Editor

Argument	Value
arg[0]	H:\ugrads\zvagena\diplomatikh\scenaria\scenario1.arktos
arg[1]	H:\ugrads\zvagena\diplomatikh\scenaria\scenario2.arktos
arg[2]	H:\ugrads\zvagena\diplomatikh\scenaria\scenario3.arktos
arg[3]	

System Property	Value

Repeat Pattern Editor - 1=Sunday in weekday field

Choose Minute(s):	Choose Hour(s):	Choose Day(s):	Choose Month(s):	Choose Year(s):	Choose Weekday(s):
0	0	1	1	2000	1
1	1	2	2	2001	2
2	2	3	3	2002	3
3	3	4	4	2003	4

Fig. 10. Basic screens for the scheduler.

in the rest of this section we will focus on the facilities for the declarative definition of data warehouse scenarios within 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:

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

The former language is rather verbose and complex to write; yet it is more comprehensible since it is quite detailed in its appearance and produces programs which is easily understandable even for a non-expert. The latter is more compact and resembles SQL; thus it is mostly suitable for a trained designer. We find SADL easier to write than the XML variant.

In order to explain the internals and the functionality of the two languages we will employ a subset of our motivating scenario. In Fig. 11 we depict the activities for the loading of data from source `S`$_3$, as composed in a scenario, namely Scenario 3. The examples of the two next subsections will show the implementation of this scenario in SADL and XADL.

4.1. SADL

The SADL language is composed of four definition statements: the `CREATE SCENARIO,`

1. Push data from table LINEITEM of source database S_3 to table LINEITEM of DW database.
2. Perform a referential integrity violation checking for the foreign key of table LINEITEM in database DW, which is referencing the ORDER table. Delete violating rows.
3. Perform a primary key violation checking to the same LINEITEM table. Report violating rows to a file.

Fig. 11. Description of Scenario 3 of the motivating example.

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 Fig. 12 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 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} and the <output_name> variable defines the name of the output file or table (wherever appropriate). 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 (they 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.

```
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_name>]
[OUTPUT
     <output_name>(<attr_1,...,att_m>)]
SEMANTICS <SQL query>
```

Fig. 12. The syntax of SADL for the CREATE SCENARIO, CONNECTION, ACTIVITY, QUALITY FACTOR statements.

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 and
- 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 Fig. 13 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, which is hard-coded in the subclasses of the ARKTOS architecture.

Returning to our motivating example, let us observe Fig. 14. 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 depict the declaration of a quality factor, which counts the number of the rows which do not pass the foreign key violation check. The quality factor is traced into a log file.

4.2. XADL

For any valid scenario that we load in ARKTOS, its XADL description can be automatically generated by the tool. In Fig. 15, we illustrate a subset of the XADL definition for the scenario of Fig. 11, as it is exported by ARKTOS. In lines 3–10 the connection instructions are given for the source database S_3 (the data warehouse database is described similarly). Line 4 describes the URL of the source database (remember that ARKTOS uses the JDBC protocol which assigns a particular URL to every service of a DBMS instance). Naturally, the username and passwords are not depicted. Line 8 gives the class name for the employed JDBC driver (JDBC communicates with an instance of a DBMS through a driver, particular to this given DBMS—in our case, since the source database is an informix instance, we employ the respective driver).

Lines 67–102 describe the second activity of Scenario 3. First, in lines 68–85 the structure of the input table is given. Lines 86–92 describe the error

Primitive Operation	SQL statement	SEMANTICS clause shortcut
UNIQUENESS	SELECT * FROM <table> GROUP BY <attr> HAVING COUNT(*) > 1	<table>.<attr>
NULL EXISTENCE	SELECT * FROM <table> WHERE <attr> IS NULL	<table>.<attr>
DOMAIN MISMATCH*	SELECT * FROM <table> WHERE <attr> NOT IN [domain specification]	<table>.<attr> NOT IN <domain specification>
PRIMARY KEY VIOLATION	SELECT * FROM <table> GROUP BY (<attr$_1$,..., attr$_n$>) HAVING COUNT(*) > 1	<table>.(<attr$_1$,..., attr$_n$>)
REFERENCE VIOLATION	SELECT * FROM <table> WHERE <attr> NOT IN (SELECT <target_attr) FROM <target_table>)	<table>.<attr> NOT IN <target_table>.<target_attr>
FORMAT MISMATCH**	SELECT APPLY (<reg_expr>,<attr>) FROM <table> WHERE APPLY (<reg_expr>,<attr>)	TARGET APPLY (<reg_expr>,<attr>) SOURCE APPLY (<reg_expr>,<attr>)
PUSH	Arbitrary SQL query	Arbitrary SQL query

* works for intervals of numbers and strings

** where `<reg_expr>` is a Perl regular expression acting as a formatting function

Fig. 13. 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)"
…
```

Fig. 14. Part of Scenario 3 expressed in SADL.

type (i.e., the functionality) of the activity: we declare that all rows that violate the foreign key constraint should be deleted. 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. A quality factor returning the absolute number of violating rows is described in lines 96–98. This quality factor is characterized by the SQL query of line 97, which computes its value and the report file where this value should be stored. For lack of space, the full DTD for the XADL language is given in the Appendix A.

5. Related work

In this section, we discuss the state of art and practice for some commercial ETL tools and standards. We also refer to research prototypes and solutions given in the academic community.

5.1. Standards and commercial tools

Standards. The *Metadata Coalition* (*MDC*), is an industrial, non-profitable consortium with aim to provide a standard definition for enterprise metadata shared between databases, CASE tools and similar applications. The *Open Information Model* (*OIM*) [24] is a proposal (led by Microsoft) for the core metadata types found in the operational and data warehousing environment of enterprises. The MDC OIM uses UML both as a modeling language and as the basis for its core model. The OIM is divided into submodels, or *packages*, which extend UML in order to address different areas of information management. The *Database and Warehousing Model* is composed from the Database Schema Elements package, the Data Transformations Elements package, the OLAP Schema Elements package and the Record Oriented Legacy Databases package. The *Data Transformations Elements* package covers basic transformations for relational-to-relational translations. The package is not a data warehouse process modeling package (covering data propagation, cleaning rules, or the querying process), but covers in detail the sequence of steps, the functions and mappings employed and the execution traces of data transformations in a data warehouse environment.

Ardent Software. Ardent Software [25] provides a tool suite for extract-transform-load processes in a data warehouse environment, under the general

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<scenario name="scenario3.xml" date="Tue May 30 15:58:49 GMT+03:00
    2000">
<connection>
    <database database_url="jdbc:informix-
    sqli://kythira.dbnet.ece.ntua.gr:1500/dbs3:informixserver=ol_milos
    _tcp">
<user_name/>
<password/>
</database>
<jdbc_driver> classname = "com.informix.jdbc.IfxDriver" >
</jdbc_driver>
</connection>
...
<transformtype>
<input_table table_name="lineitem" database_url="jdbc:informix-
    sqli://kythira.dbnet.ece.ntua.gr:1500/dbs3:informixserver=ol_milos
    _tcp">
<column> l_orderkey </column>
<column> l_partkey </column>
<column> l_suppkey </column>
<column> l_linenumber </column>
...
</input_table>
<errortype>
<reference_violation>
<target_column_name>l_orderkey</target_column_name>
<referenced_table_name> Informix.tpcd.tpcd.tpcd.order </referenced_
    table_name>
<referenced_column_name>o_orderkey</referenced_column_name>
</reference_violation>
</errortype>
<policy>
<delete/>
</policy>
<quality_factor qf_name="# of reference violations"
    qf_report_file="H:\path\scenario3.txt">
<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>
</quality_factor>
</transformtype>
...
</scenario>
```

Fig. 15. XADL definition of Scenario 3 as exported by Arktos.

name *Datastage Suite*. Datastage is a client server tool for designing, controlling and engineering the ETL process. The *Datastage Server*, is the engine that accepts a series of transformation requests and takes over their execution. On the client side, there is a set of tools that enable the graphical design and deployment of the ETL applications. The *Datastage Manager* is a tool supporting the importing/exporting/sharing of the environment metadata. The datastage server is the central engine responsible for executing a sequence of transformation operations. Its capabilities include combining rows from heterogeneous data sources, division, enrichment and merging of rows, sorting and aggregation, parallelism, invocation of external tools and many other features. The Datastage Manager catalogues and organizes the components of a Datastage Project. The tool presents relational catalogs via standard RDBMS API's and exports these metadata using the technology of Microsoft repository [26].

DataMirror. DataMirror's *Transformation Server* (*TS*) [27] is an engine-based data transformation and replication tool that enables users to easily and seamlessly move and share data in real time among mixed system environments. The main features of TS include bi-directional, real time, periodic or asynchronous replication, the full replication of databases and the support of a publish/subscribe model to distribute data. All the metadata necessary to implement this replication process are stored in the native databases (sources and targets), where also the replication engine runs. TS can track changes in the source databases and extract only them.

ETI. ETI [28] provides a tool suite, named *ETI·EXTRACT FX*, which is a code generating software product that automates data transformation, reorganization and integration. The ETI·EXTRACT generates all the codes and necessary scripts to extract data from source systems, transform them as required and load them into the target systems. To perform these tasks, the software relies on ETI's Data System Libraries with information on individual data storage systems. Users specify the integration of the data using ETI's editors. The program analyzes the integration project and generates the actual code in the Generation Engine. This code is transferred to the source and target systems. For metadata acquisition and storage ETI·EXTRACT uses the *Metastore* component which is a centralized, object-oriented database that captures a full audit trail of the information ETI·EXTRACT acquires in the process of automatic data consolidation.

Microsoft. The tool offered by Microsoft to implement its proposal for the Open Information Model is presented under the name of *Data Transformation Services* [29]. Data transformation services (DTS) are the data-manipulation utility services in SQL server version 7.0 that provide import, export, and data-manipulating services between OLE DB [30], ODBC and ASCII data stores. The software modules that support DTS are packaged with MS SQL server. These packages include (a) *DTS Designer*, which is a GUI used to interactively design and execute DTS packages; (b) *DTS Export and Import Wizards*, i.e., wizards that ease the process of defining DTS packages for the import, export and transformation of data and (c) *DTS Programming Interfaces*, which include a set of OLE automation and a set of COM interfaces to create customized transformation applications for any system supporting OLE automation or COM.

In an overall assessment we could say that commercial ETL tools are responsible for the implementation of the data flow in a data warehouse environment. Most of the commercial ETL tools are of two flavors: *engine-based*, or *code-generation based*. The former assume that all data have to go through an engine for transformation and processing. Moreover, quite often the engine takes over also the extraction and loading processes, making the Extract-Transform-Load processes one big process, where the intermediate steps are transparent to the user. On the other hand, in code-generating tools all processing takes place only at the target or source systems. The tools offered by Ardent, DataMirror and Microsoft are engine based, while the tool from ETI is code-generation based. In ARKTOS we have followed the code-generation approach, which enables greater flexibility and customization to the user. Compared to the commercial tools, ARKTOS is based on a concrete meta-model specific for ETL processes. Although one could argue that ARKTOS includes a limited set of operations compared to a full-blown commercial product, it is the generality along with the clarity of the meta-model which gives it the advantage of extensibility and simplicity. Moreover, we believe that providing alternative, declarative ways to describe ETL scenarios, as we have done in ARKTOS, instead of just a point-and-click front-end, considerably enhances the usability of an ETL tool.

5.2. Research efforts

Research on the general problem of ETL. The *AJAX* system [31] is a data cleaning tool developed at INRIA France. It deals with typical data quality problems, such as *the object identity problem* [32], *errors due to mistyping* and *data inconsistencies* between matching records. This tool can be used either for a single source or for integrating multiple data sources. AJAX provides a framework wherein the logic of a data cleaning program is modeled as a directed graph of data transformations that start from some input source data. Four types of data transformations are supported: (a) *mapping transformations* that standardize data formats (e.g., date format) or simply merge or split columns in order to produce more suitable formats; (b) *matching transformations* that find pairs of records which most probably refer to same object. These pairs are called *matching pairs* and each such pair is assigned a similarity value; (c) *clustering transformations* which group together matching pairs with a high similarity value by applying given grouping criteria (e.g., by transitive closure) and (d) *merging transformations* which are applied to each individual cluster in order to eliminate duplicates or produce new records for the resulting integrated data source. AJAX also provides a declarative language for specifying data cleaning programs, which consists of SQL statements enriched with a set of specific primitives to express mapping, matching, clustering and merging transformations. Finally, an interactive environment is supplied to the user in order to resolve errors and inconsistencies that cannot be automatically handled and support a stepwise refinement design of data cleaning programs. The theoretical foundations of this tool can be found in [33], where apart from the presentation of a general framework for the data cleaning process, specific optimization techniques tailored for data cleaning applications are discussed.

Ref. [34] presents the *Potter's wheel* system, which is targeted to provide interactive data cleaning to its users. The system offers the possibility of performing several algebraic operations over an underlying data set, including *format* (application of a function), *drop*, *copy*, *add* a column, *merge* delimited columns, *split* a column on the basis of a regular expression or a position in a string, *divide* a column on the basis of a predicate (resulting in two columns, the first involving the rows satisfying the condition of the predicate and the second involving the rest), *selection* of rows on the basis of a condition, *folding* columns (where a set of attributes of a record is split into several rows) and *unfolding*. Optimization algorithms are also provided for the CPU usage for certain classes of operators.

Research on data transformations. In [35] the authors present WOL, a Horn-clause language, to specify transformations between complex types. The transformations are specified as rules in a Horn-clause language. An interesting idea behind this approach is that a transformation of an element can be decomposed into a set of rules for its elements, thus avoiding the difficulty of employing complex definitions. In [36] the authors discuss the general setting for schema and data integration. The Garlic system is used as the paradigm for mediator/wrapper architectures, to explore the issues of (a) transformations in the wrapper side; (b) view definitions in the wrapper and (c) data integration. A prototype under construction is also sketched. The paper is also characterized by an informative revision of the relevant literature in each of these areas. In [37], the authors discuss several issues around ETL tools in the data warehousing context. The authors conclude that a language respecting logical and physical independence is needed as a definition language for the ETL process. Another basic observation is that ad hoc querying and precanned definition should ideally use the same language. All these observations sum up to the proposal of SQL99 (which supports function definition and invocation) as the proper language for this task. The authors go on to present the Cohera System, which is a federated DBMS and supports SQL99 as the mapping language between the relations of a set of federated databases.

Research on data cleaning. Data cleaning is another step in the ETL process, which unfortunately has not caught the attention of the research community. Still, [38] provides an extensive overview of the field, along with research issues and a review of some commercial tools. Ref. [39] discusses a special case of the data cleaning process, namely the detection of duplicate records and extends previous algorithms on the issue. Ref. [40] focuses on another subproblem, namely the one of breaking address fields into different elements and suggests the training of a Hidden Markov Model to solve the problem.

Workflow and process modeling. Modeling ETL scenarios can be considered as a special case of the general problem of workflow and process modeling. Although beyond the scope of this paper, we mention several research efforts in the field [3,41–47] as well as a widely accepted standard proposed by the Workflow Management Coalition (WfMC) [48]. As far as process modeling is concerned, we reference the interested reader to [49] for a recent overview of the field.

In another line of research, [2] is the first attempt to clearly separate the data warehouse refreshment process from its traditional treatment as a view maintenance or bulk loading process. The authors provide a conceptual model of the process, which is treated as a composite workflow.

6. Conclusions and future work

In this paper, we have presented a uniform model covering all the aspects of an ETL environment as well as a platform capable to support practical ETL scenarios with particular focus on issues of complexity, usability and maintainability. The proposed metamodel covers the aspects of data warehouse architecture, activity modeling, contingency treatment and quality management. The ETL tool we have developed, namely ARKTOS, is capable of modeling and executing practical ETL scenarios by providing explicit primitives for the capturing of common tasks (like data cleaning, scheduling and data transformations). Furthermore, we provide three ways to describe an ETL scenario: 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.

In the future we plan to integrate even more functionality in our ETL tool, in order to provide the users with richer transformation primitives. From the viewpoint of research, several issues remain open and we discuss them in the sequel.

The impact analyzer. An important aspect in the data warehouse lifecycle is the issue of data warehouse evolution. Different user requirements may impose the evolution of schema, functionality and data of the warehouse. Due to the complexity of the data warehouse architecture, evolution is hard, since it is possible that several objects are affected due to a simple change. In [14] we have already suggested algorithms that exploit the logical description of a process to perform impact analysis in the presence of evolution intentions. We are currently working on implementing these algorithms and incorporating these results in ARKTOS.

Linkage to a metadata repository. It is quite common practice to use a meta-database as a repository for meta-information on the components of a data warehouse. Metadata repositories offer the possibility of exploring crucial meta-information on the structure and content of the information system either through interactive polling or querying APIs. In our case, the storage of scenarios inside a metadata repository will give us the flexibility of complex querying to retrieve interesting information (e.g., possibly hidden interrelationships, linkage to the conceptual perspective for data warehouse processes [14] and other). In all our background work, the architecture, quality and process models have been represented in Telos [50], a conceptual modeling language for representing knowledge about information systems, and implemented in the object-oriented deductive database system ConceptBase [51], that provides query facilities, and a language for constraints and deductive rules. ConceptBase offers Java and C APIs to external applications to query and retrieve stored objects, which we plan to exploit in order to link it to ARKTOS.

The optimizer. We have argued that a primary concern with ETL tools is usability and functionality. Having said that, we do not advocate though, that performance is not a crucial issue, especially as the size of data warehouses grows. For us, the optimization problem of ETL processes is posed as follows:

- Identification of a small set of algebraic operators, capable enough to support the efficient execution of the primitives we have defined at the user level. A mapping from the declarative specification of activities to these algebraic operators is also required.
- Local optimization of ETL activities. Each logical activity may be mapped to different

combinations of physical activities (in the same fashion that there are more than one way to execute a relational join). For example, checking for distinct values might be performed by sorting, hashing, self-join and possibly other techniques. To determine which is the most efficient local plan is an open research issue.

- Global (multiple) optimization of ETL activities. The execution of a whole scenario (or a set of scenarios) with the optimal way, might require to employ different plans than simply optimizing each of the activities in isolation [52]. Producing, thus the optimal plan for a scenario is not a straightforward operation.

Acknowledgements

This research has been partially funded by the European Union's Information Society Technologies Programme (IST) under project EDITH (IST-1999-20722). Many thanks are also due to the anonymous reviewers of previous versions of this paper for their fruitful comments and suggestions.

Appendix A. The DTD of XADL

```
<?xml version=''1.0'' encoding=''UTF-8''?>
<!--DTD that species the structure of the xml description of the scenario-->
<!ELEMENT scenario (connection+,transformtype+)>
<!ATTLIST scenario_name CDATA #REQUIRED>
<!ELEMENT connection (database,jdbc_driver)>
<!ATTLIST connection_name CDATA #REQUIRED>
<!ELEMENT database (user_name,password)>
<!ATTLIST database database_url CDATA #REQUIRED>
<!ELEMENT user_name EMPTY>
<!ELEMENT password EMPTY>
<!ELEMENT jdbc_driver EMPTY>
<!ATTLIST jdbc_driver class_name CDATA #REQUIRED>
<!ELEMENT transformtype (input_table, errortype,policy, output_table?, quality+factor*)>
<!ATTLIST input_table table_name CDATA #REQUIRED database_url CDATA #REQUIRED>
<!ELEMENT input_table (column+)>
<!ELEMENT column (#PCDATA)>
<!ELEMENT quality_factor (sql_query+)>
<!ATTLIST quality_factor qf_name CDATA #REQUIRED qf_report_file CDATA #REQUIRED>
<!ELEMENT sql_query (#PCDATA)>
<!ELEMENT errortype (push_to_table|uniquness_violation|null_existence|domain_mismatch|
          primary_key_violation|reference_violation|format_mismatch)>
<!ELEMENT push_to_table EMPTY>
<!ELEMENT uniqueness_violation EMPTY>
<!ATTLIST uniquness_violation target_column_name CDATA #REQUIRED>
<!ELEMENT null_existence EMPTY>
<!ATTLIST null_existence target_column_name CDATA #REQUIRED>
<!ELEMENT domain_mismatch (target_column_name, specication_string)>
<!ELEMENT target_column_name (#PCDATA)>
<!ELEMENT specication_string (#PCDATA)>
<!ELEMENT primary_key_violation (target_column_name)+>
<!ELEMENT reference_violation (target_column_name, referenced_table_name, referenced_
          column_name)>
```

```
<!ELEMENT referenced_table_name (#PCDATA)>
<!ELEMENT referenced_column_name (#PCDATA)>
<!ELEMENT format_mismatch (target_column_name, pattern_match_string)>
<!ELEMENT pattern_match_string (#PCDATA)>
<!ELEMENT policy (ignore|delete|report_to_file|insert_to_table|transform_format)>
<!ELEMENT ignore EMPTY>
<!ELEMENT delete EMPTY>
<!ELEMENT report_to_file EMPTY>
<!ATTLIST report_to_file lename CDATA #REQUIRED>
<!ELEMENT insert_to_table EMPTY>
<!ATTLIST output_table table_name CDATA #REQUIRED database_url CDATA #REQUIRED>
<!ELEMENT transform_format EMPTY>
<!ATTLIST transform_format transform_string CDATA #REQUIRED>
<!ELEMENT output_table (column+)>
```

References

[1] M. Jarke, M.A. Jeusfeld, C. Quix, P. Vassiliadis, Architecture and quality in data warehouses: an extended repository approach, Inform. Systems 24 (3) (1999) 229–253. (A previous version appeared in: Proceedings of the 10th Conference of Advanced Information Systems Engineering (CAiSE'98), Pisa, Italy, 1998.

[2] M. Bouzeghoub, F. Fabret, M. Matulovic, Modeling data warehouse refreshment process as a workflow application. Proceedings of the International Workshop on Design and Management of Data Warehouse, Heidelberg, Germany, June 1999.

[3] F. Casati, S. Ceri, Barbara Pernici, Giuseppe Pozzi, Workflow evolution, DKE 24 (3) (1998) 211–238.

[4] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, Information integration: conceptual modeling and reasoning support, Proceedings of the Sixth International Conference on Cooperative Information Systems, 1998, pp. 280–291.

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, Data integration in Data Warehousing, Int. J. Cooperative Inform. Systems 10 (13) (2001) 237–271.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati, A principled approach to data integration and reconciliation in data warehousing, Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99), available at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-19/.

[7] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis (Eds.), Fundamentals of Data Warehouses, Springer, Berlin, 2000.

[8] R.Y. Wang, M.P. Reddy, H.B. Kon, Towards quality data: an attribute-based approach, Decision Support Systems 13 (1995) 349–372.

[9] R.Y. Wang, V.C. Storey, C.P. Firth, A framework for analysis of data quality research, IEEE Trans. Knowledge Data Engi. 7(4) (1995) 623–640.

[10] C. Shilakes, J. Tylman, Enterprise information portals, Enterprise Software Team, November 1998; available at www.sagemaker.com/company/downloads/eip/indepth.pdf.

[11] M. Demarest; The politics of data warehousing, available at http://www.hevanet.com/demarest/marc/dwpol.html.

[12] P. Vassiliadis, Gulliver in the land of data warehousing: practical experiences and observations of a researcher, Proceedings of Design and Management of Data Warehouses (DMDW'2000) Second International Workshop (in conjunction with the 12th Conference on Advanced Information Systems Engineering—CAiSE'00), Stockholm, Sweden, 2000, pp. 12.1–12.16.

[13] P. Vassiliadis, M. Bouzeghoub, C. Quix, Towards quality-oriented data warehouse usage and evolution, Inform. Systems, 25 (2) 89–115 2000. (A previous version appeared in Proceedings of the 11th Conference of Advanced Information Systems Engineering (CAiSE'99), Heidelberg, Germany, 1999, pp. 164–179).

[14] P. Vassiliadis, C. Quix, Y. Vassiliou, M. Jarke, A model for data warehouse operational processes. Proceedings of the 12th Conference on Advanced Information Systems Engineering (CAiSE'00), Stockholm, Sweden, 2000.

[15] M. Jarke, Y. Vassiliou, Foundations of data warehouse quality—a review of the DWQ project, Proceedings of the Second International Conference on Information Quality (IQ-97), Cambridge, MA, 1997; available at http:/www.dbnet.ece.ntua.gr/~dwq.

[16] M. Jarke, C. Quix, D. Calvanese, M. Lenzerini, E. Franconi, S. Ligoudistianos, P. Vasiliadis, Y. Vassiliou, Concept based design of data warehouses: the DWQ demonstrators, Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data,

Vol. 29, Dallas, USA, 14–19 May 2000 (Demonstration), ACM 2000, ISBN 1-58113-218-2, p. 591.

[17] M. Oivo, V. Basili, Representing software engineering models: the TAME goal-oriented approach, IEEE Trans. Software Eng. 18 (10) (1992) 886–898.

[18] M.A. Jeusfeld, C. Quix, M. Jarke, Design and analysis of quality information for data warehouses, Proceedings of the 17th International Conference on the Entity Relationship Approach, Singapore, 1998.

[19] Branch Cut Software, Jtask: Java Task Scheduler, available at http://www.branchcut.com/jTask/.

[20] Transaction Processing Performance Council, TPC-H and TPC-R, 2000, available at www.tcp.org.

[21] ORO, Inc. PerlTools 1.2., available at http://www.savarese.org/oro/.

[22] M. Jarke, M.A. Jeusfeld, T. Rose, A software process data model for knowledge engineering in information systems, Inform. Systems 15 (1) (1990) 85–116.

[23] E. Yu, J. Mylopoulos, Understanding 'why' in software process modelling, analysis and design, Proceedings of the 16th International Conference Software Engineering, 1994.

[24] MetaData Coalition, Open information model. version 1.0, 1999, available at www.MDCinfo.com.

[25] Ardent Software. DataStage Suite, available at http://www.ardentsoftware.com/.

[26] P.A. Bernstein, Th. Bergstraesser, J. Carlson, S. Pal, P. Sanders, D. Shutt, Microsoft repository version 2 and the open information model, Inform. Systems 24 (2) (1999) 78–93.

[27] DataMirror Corporation, Transformation server, available at http://www.datamirror.com.

[28] Evolutionary Technologies International, ETI*EXTRACT, available at http://www.eti.com/.

[29] Microsoft Corp., MS Data transformation services, available at www.microsoft.com/sq.

[30] Microsoft Corp., OLEDB specification, available at www.microsoft.com/data/oledb.

[31] H. Galhardas, D. Florescu, D. Shasha, E. Simon, Ajax: an extensible data cleaning tool. Proceedings of the ACM SIGMOD International Conference on the Management of Data, Dallas, TX, 2000, 590.

[32] W. Cohen, Some practical observations on integration of Web information, WebDB'99 Workshop in conj. with ACM SIGMOD, 1999.

[33] H. Galhardas, D. Florescu, D. Shasha, E. Simon, An Extensible Framework for Data Cleaning, Technical Report, INRIA, 1999, RR-3742.

[34] 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.

[35] S.B. Davidson, A.S. Kosky, Specifying Database Transformations in WOL, Bulle. Techn. Committee Data Eng. 22 (1) (1999) 25–30.

[36] L. Haas, R. Miller, B. Niswonger, M. Tork Roth, P. Schwarz, E. Wimmers. Transforming heterogeneous data with database middleware: beyond integration, Bull. Tech. Committee Data Eng. 22 (1) (1999) 43–49.

[37] J.M. Hellerstein, M. Stonebraker, R. Caccia, Independent, open enterprise data integration. Bull. Techn. Committee Data Eng. 22 (1) (1999) 31–36.

[38] E. Rahm, H. Hai Do, Data cleaning: problems and current approaches, Bull. Tech. Committee Data Eng. 23 (4) (2000) 3–13.

[39] A. Monge, Matching algorithms within a duplicate detection system. Bull. Tech. Committee Data Eng. 23 (4) (2000) 14–20.

[40] V. Borkar, K. Deshmuk, S. Sarawagi, Automatically extracting structure form free text addresses, Bull. Techn. Committee Data Eng. 23 (4) (2000) 27–32.

[41] F. Casati, M. Fugini, I. Mirbel, An environment for designing exceptions in workflows, Inform. Systems 24 (3) (1999) 255–273.

[42] F. Casati, S. Ceri, B. Pernici, G. Pozzi, Conceptual modeling of workflows, Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Modelling (ER'95), Gold Coast, Australia, 1995, pp. 341–354.

[43] P. Dadam, M. Reichert (Eds.), Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, GI Workshop Informatik'99, 1999; available at http://www.informatik.uni-ulm.de/dbis/veranstaltungen/Workshop-Informatik99-Proceedings. pdf.

[44] R. Klamma, Readings in workflow management; annotated and linked internet bibliography, RWTH Aachen; available at http://sunsite.informatik.rwth-aachen.de/WFBib/.

[45] D. Kuo, M. Lawley, C. Liu, M. Orlowska, A general model for nested transactional workflows, Proceedings of the International Workshop on Advanced Transaction Models and Architecture, India, 1996.

[46] O. Marjanovic, M. Orlowska, On modeling and verification of temporal constraints in production workflows, Knowledge Inform. Systems 1 (2) (1999) 157–192.

[47] W. Sadiq, M. Orlowska, Applying graph reduction techniques for identifying structural conflicts in process models. Proceedings of the 11th International Conference on CAiSE'99, Heidelberg, Germany, 1999.

[48] Workflow Management Coalition, Interface 1: process definition interchange process model, Document number WfMC TC-1016-P, 1998; available at www.wfmc.org.

[49] C. Rolland, A comprehensive view of process engineering, Proceedings of the 10th International Conference on Advanced Information Systems Engineering, Pisa, Italy, pp. 1–25.

[50] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos—a language for representing knowledge about

information systems, ACM Trans. Inform. Systems 8 (4) (1990) 325–362.

[51] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, S. Eherer, ConceptBase—a deductive objectbase for meta data management, J. Intelli. Inform. Systems (Special Issue on Advances in Deductive Object-Oriented Databases) 4 (1995) 167–192.

[52] T. Sellis, Multiple-query optimization, TODS 13 (1) (1988) 23–52.