A Taxonomy of ETL Activities

Panos Vassiliadis
University of Ioannina
Ioannina, Greece

pvassil@cs.uoi.gr

Alkis Simitsis
HP Labs
Palo Alto, CA, USA

alkis@hp.com

Eftychia Baikousi
University of Ioannina
Ioannina, Greece

ebaikou@cs.uoi.gr

ABSTRACT

Extract-Transform-Load (ETL) activities are software modules responsible for populating a data warehouse with operational data, which have undergone a series of transformations on their way to the warehouse. The whole process is very complex and of significant importance for the design and maintenance of the data warehouse. A plethora of commercial ETL tools are already available in the market. However, each one of them follows a different approach for the modeling of ETL activities; i.e., of the building blocks of an ETL workflow. As a result, so far there is no standard or unified approach for describing such activities. In this paper, we are working towards the identification of generic properties that characterize ETL activities. In doing so, we follow a black-box approach and provide a taxonomy that characterizes ETL activities in terms of the relationship of their input to their output and provide a normal form that is based on interpreted semantics for the black box activities. Finally, we show how the proposed taxonomy can be used in the construction of larger modules, i.e., ETL archetype patterns, which can be used for the composition and optimization of ETL workflows.

Categories and Subject Descriptors

H.2.7 [**Database Administration**]: Data warehouse and repository.

General Terms

Management, Design, Experimentation.

Keywords

Data Warehouses, ETL, Taxonomy, Optimization.

1. INTRODUCTION

The back-stage of data warehouse comprises many software modules responsible for its population with fresh data, extracted from the appropriate sources, transformed, and cleansed to comply with the target schemata. Such software constructs are commonly known as Extract-Transform-Load (ETL) activities and as they cooperate all together they compose ETL workflows responsible for populating and maintaining data warehouses. ETL workflows are quite complex by nature, mostly due to the plethora and the large volume of different activities contained in such processes.

Typical activities are schema transformations (e.g., pivot, normalize), cleansing activities (e.g., duplicate detection, check for integrity constraints violations), filters (e.g., based on some regular expression), sorters, groupers, flow operations (e.g., router, merge), function application (e.g., built-in function, script written in a declarative programming language, call to an external library –hence, functions having 'black-box' semantics) and so on.

Nowadays, a large number of ETL tools are available in the market [e.g., 5, 7, 10, 11]. However, in general, they follow different design and modeling techniques, and use different internal language. Until recently, ETL was faced as a software technicality in the data warehouse architecture, and, so far, the research community has not dealt with and agreed upon the basic characteristics of ETL workflows and activities. Without a formal way to represent ETL activities, and at the same time, by using ad hoc design techniques, it is not possible to improve the quality and efficiency of ETL workflows in a systematic manner or to perform other crucial operations like what-if and impact analysis.

In this paper, we work toward the determination of a principled, reference way to model ETL workflows by investigating their main characteristics. To this end, we provide insights for the modeling of ETL flows and its exploitation in three levels: (a) the characterization of activities with respect to the relationship of their input-output schemata and tuples, (b) the characterization of activities via a powerful "normal form" representation that can be used to describe both activities and workflows, and (c) the possibilities for efficient operation opened by a set of recurring archetype patterns for parts of ETL flows.

Our first contribution is a *taxonomy* of ETL activities based on the relationship of activity input to its output in terms of both its schemata and the way it processes incoming tuples. The motivation for the taxonomy is the possibility of exploiting the taxonomical characteristics for the logical and physical optimization of a workflow, the parallelization of activities, and any other tuning that may improve their efficiency and resilience to failures. The foundations of the taxonomy lie in the possibility of local processing of input tuples: if every tuple that arrives in an activity can be locally processed, then there is flexibility in parallelizing the activity, changing its physical implementation (e.g., to exploit tuple ordering), and so on. On the other hand, if aggregations or routing of tuples to multiple destinations are performed, the possibilities are more constrained. The proposed taxonomy classifies activities on these grounds and, as a proof of concept, we relate its categories with activities provided by popular ETL tools.

The second contribution of this paper deals with the need for providing a unique *formalism* for the taxonomy classes. We believe that the most convenient way to handle ETL activities for our purposes is to express them via black-box semantics. We draw an analogy between ETL activities and the *structure of the matter*,

from the domain of physics, as an intuitive means for identifying ETL components of different complexity as well as their compositions. We introduce a normal form for ETL activities and use it to express different operations such as composition, split, and swap. Moreover, we show that simple operations discussed in the related literature, such as the swapping of activities can be handled by this normal-form based "language".

Finally, we demonstrate that it is possible to express complicate workflows through the combination of typical, recurring activity compositions, which we call *patterns*. This provides the possibility to equip the research community with a testbed structure over which design, optimization and resiliency methods can be tested, and the industrial community with methods to build ETL tools that exploit any such occurrence of internal structure in deployed workflows. At the end, we present results in terms of pattern-based scheduling and optimization of ETL flows.

Outline. The rest of the paper is as follows. Section 2 discusses the rationale for the foundations of the proposed taxonomy and Section 3 describes the taxonomy intuitively. Section 4 introduces a normal form for ETL activities. Section 5 presents ETL archetype patterns. Section 6 presents the related work. Finally, Section 7 summarizes our approach.

2. A RATIONALE FOR THE TAXONOMY

An ETL workflow can be seen as a directed graph. The nodes of this graph are activities and recordsets. The edges of the graph are provider relationships that combine activities and recordsets. Following the common practice, we envisage ETL activities to be combined in a workflow. Therefore, we do not assume that the output of a certain activity will be necessarily directed towards a recordset, but rather, that the recipient of this data can be either another activity or a recordset.

Figure 1 abstractly depicts the combination of an activity (*computeAmts*) with its providers and consumer. Each input schema of an activity should be mapped to a provider; i.e., an output schema of another activity or the schema of a recordset. Similarly, each output schema of an activity should be mapped to a consumer; i.e., an input schema of another activity or the schema of a recordset. In the example of Figure 1, *computeAmts* is populated by two providers *Person* and *Service* and populates a single consumer, *Payments*. Internally, the input schemata of an activity populate its output schemata by means designated by the operational semantics of the activity. In Figure 1, *computeAmts* combines employee history (*YrsService*) with salary (*Sal*) and produces an output schema containing two new attributes (*Bonus* and *Tax*) that populates its consumer (*Payments*).

2.1 ETL activities

In this section, we discuss the different types of ETL activities based on the interrelationship of their input and output. We begin with a high-level classification with respect to input (e.g., unary, binary, n-ary) or output (e.g., routers, filters) schemata and within each such category, we discuss the mappings between input and output tuples.

Unary activities. These activities take the data from the input schema, perform a transformation or cleaning operation to them, and direct the processed data to the output. Unary activities have exactly one input and one output schemata.
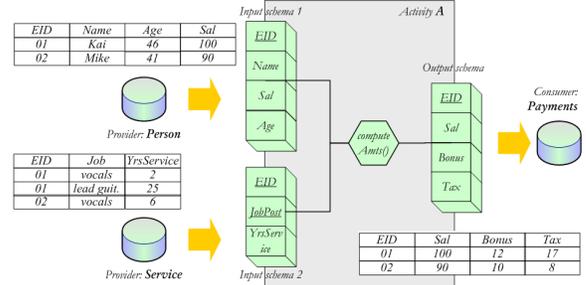

Figure 1. Graphical representation of an exemplary activity A

Within a unary activity, the combinations that can occur among its input and output tuples are as follows:

$$
\left. \begin{array}{c} 0 \\ 1 \\ N \end{array} \right\} : \left\{ \begin{array}{c} 0 \\ 1 \\ M \end{array} \right. , \neq \{0,0\}
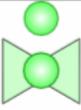$$

Based on that, the most interesting values for the cardinality of mapping in unary activities are the following:

- *1:1*, an input tuple is mapped to exactly one output tuple.

- *1:M*, an input tuple is mapped to more than one output tuples.

- *N:1*, more than one input tuples are combined to produce exactly one output tuple. Observe that this relationship introduces a set of classes among input tuples: all tuples belonging to the same class correspond to the same output tuple. If each input tuple corresponds to at most one class, then these are equivalence classes.

- *0:M*, some functions or constant values are employed to produce one or more output tuples.

- *N:M*, the relationship among a certain group of input tuples and a certain group of output tuples cannot be simplified to one of the above categories.

N-ary activities. N-ary activities combine information from multiple inputs and populate one output schema. Different tools provide different implementations regarding the input schemata. An n-ary activity (e.g., a multi-way join) may have *n* inputs or can be implemented as a series of binary activities. Although our analysis covers both, for the sake of presentation we discuss the case of binary activities, which involves two popular configurations:

- *Primary flow*. These are binary activities where one of their inputs is a part of a primary flow that probes a second input to test whether their values qualify for further propagation. An example primary flow – see Figure 8(a) – may contain a series of the binary *surrogate_key* operators, which replace the production keys of the incoming data (this data would be the first input) with surrogate keys found in lookup tables (these tables would constitute the second input). The primary flow may employ the vocabulary for unary activities (in fact, most of these binary activities can be classified under the 1:1 category.) Here, we do not focus on how the input values are combined, but rather, how the tuples are related to each other. In the case of primary flow, for each outgoing tuple there is exactly one input tuple in the primary flow that corresponds to it (but not vice versa). At the same time, typically (but not obligatorily), there is at most one corresponding tuple in any non-primary input schema, for any tuple of the primary flow.

Table 1. Built-in transformations provided by commercial ETL tools

Atom class	Particle class	Transformation Category	SQL SSIS	DataStage	Oracle Warehouse Builder
	1:1	Row-level: Function that can be applied locally to a single row	– Character Map – Copy Column – Data Conversion – Derived Column – Script Component – OLE DB Command – Other filters (not null, selections, etc.)	– Transformer (A generic representative of a broad range of functions: date and time, logical, mathematical, null handling, number, raw, string, utility, type conversion/casting, routing.) – Remove duplicates – Modify (drop/keeps columns or change their types)	– Deduplicator (distinct) – Filter – Sequence – Constant – Table function (it is applied on a set of rows for increasing the performance) – Data Cleansing Operators (Name and Address, Match-Merge) – Other SQL transformations (Character, Date, Number, XML, etc.)
	N:1	Unary Grouper: Transform a set of rows to a single row	– Aggregate – Pivot	– Aggregator – Combine/Promote records	– Aggregator – Pivot
	1:N	Unary Splitter: Split a single row to a set of rows	– Unpivot	– Make/Split subrecord – Make/Split vector	– Unpivot
	N:M	Unary Holistic: Perform a transformation to the entire data set (blocking)	– Sort – Percentage Sampling – Row Sampling	– Sort (sequential, parallel, total)	– Sorter
		Binary or N-ary: Combine many inputs into one output	Union-like: – Union All – Merge Join-like: – Merge Join (MJ) – Lookup (SKJ) – Import Column (NLJ)	Union-like: – Funnel (continuous, sort, sequence) Join-like: – Join – Merge – Lookup Diff-like: – Change capture/apply – Difference (record-by-record) – Compare (column-by-column)	Union-like: – Set (union, union all, intersect, minus) Join-like: – Joiner – Key Lookup (SKJ)
		Routers: Locally decide, for each row, which of the many outputs it should be sent to	– Conditional Split – Multicast	– Copy – Filter – Switch	– Splitter

– *Combinators*. These are binary (or n-ary in general) activities whose output instances are a combination of values from more than one input schema.

Routers & Filters. The previous two classes involve exactly one output schema. It is possible, however, that ETL activities possess more than one output schema to perform their task. Typically, such activities are used as in the following cases:

– *Routers*. These activities direct tuples to a specific path of the workflow, according to the value of one of their attributes. Routers are applicable to all cases where a tuple must undergo different kind of processing and storage, depending on a certain value it has.

– *Filters*. These activities block the further processing of unnecessary tuples and allow the propagation of tuples, which respect a specific selection criterion. The non-blocked tuples populate one or more output schema (e.g., a typical filter populates exactly one output schema, whilst a conditional filter direct the outgoing tuples to many directions). The rejected (or blocked) tuples go to a dedicated output (e.g., an error log). This category contains also activities possessing *quarantine* error schemata that are responsible for isolating records with offending values from further regular processing and directing them towards quarantine or specific processing.

Table 1 illustrates how our classification fits in existing ETL technology. Although each tool follows a different modeling technique and provides its own palette of transformation, these transformations fall seamlessly into our classification. Table 1 shows that each of the aforementioned categories in terms of mapping input to output tuples is practically corresponding to several activity types used in popular ETL tools. A similar analysis of ETL activities provided by ETL tools, but from a different perspective, has been presented in [14].

3. AN INTUITIVE PRESENTATION OF THE PROPOSED TAXONOMY

In this section, we describe the fundamental concepts of the proposed taxonomy in an intuitive way. Our discussion employs a terminology coming from the domain of physics, concerning the structure of the matter in an attempt to enhance the intuition behind the introduced concepts. We want to cover both the case of ETL tools and the case of built-in ETL code; so our taxonomy is broad enough to incorporate both approaches (still, for reasons of intuition, we will frequently resort to discussions from the tools).

Commercial ETL tools provide a palette of reusable ETL transformations that are customized per scenario. In previous research [22] we have called this kind of built-in transformations template activities. Figure 2 presents an example ETL design drawn in the canvas of a commercial tool [11] and a palette of template activities, called particles in our terminology.

ETL Particles. An ETL particle involves a single transformation or cleaning task. When the developer adds an activity in the canvas, he introduces a particle in the design. Assuming a library of template tasks, a particle is a materialization of a template for a specific schema-respecting input. Thus, we can capture the semantics of the particle via a simple predicate with commonly agreed upon semantics.

ETL Atoms. An ETL atom is a simple ETL activity that performs exactly one job and involves exactly one ETL particle. When the developer customizes the schemata of an activity and connects it to providers and consumers he defines an atom. The number of output schemata of an ETL atom can be more than one and several input attributes can be projected out, whereas new attributes can be generated at the output (e.g., in the case of a function application, like a conversion of dollars to euros, a new attribute can be added). The particle involved is called the *nucleus* of the atom.
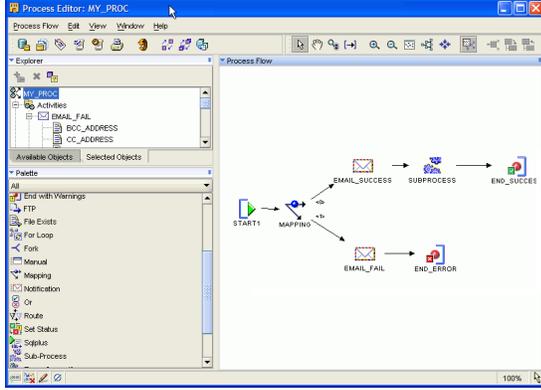
Figure 2. Example ETL workflow [11]

Figure 3 depicts the internals of a transformation atom that produces a new output field. The atom has: (a) an input schema of 6 attributes, (b) a single particle that performs a simple transformation (e.g., calculation of profit given price, quantity, cost and tax of items) and (c) an output schema that comprises a subset of the input attributes along with the newly generated attribute. Also, the attributes that have been projected out are also depicted.

Figure 4 depicts a binary atom that merges two input schemata. The atom is the most complicated atom possible as it performs all the individual subtasks an atom can do: (a) it merges 2 input schemata, (b) it performs a computation of two new attributes, (c) it routes the result to the appropriate output schema on the basis of a set of selection conditions (or if-then-else, switch programs), and (d) it projects out several attributes. Observe that the new values appear only at the *Output schema 3*.

ETL Molecule. An ETL molecule is a combination of ETL atoms that are merged in a larger construct. This is a frequent case in hand-tailored code where several functionalities are merged within the same script. In this case, instead of a single particle, there is a linear workflow of particles in between these two groups of schemata. The line of particles between the merger of the inputs and the router for the outputs will be called the *chain* of the molecule (see Figure 5 for an example of an ETL molecule). The semantics of a molecule can be defined as follows: for each output, the semantics are expressed as the conjunction of the predicates all the way to the inputs. Special care is paid to the "merge" predicate though, since it is *merge(I1(),I2(), ..., In())*.

ETL Compound. Activities are composed to form workflows: to capture this case, we model workflows as compounds where activities and recordsets (relations or structured files) are mapped to each other to form a workflow graph.

4. NORMAL FORMS FOR ETL ACTIVITIES

In this section, first we present different categories for ETL particles and atoms. Then, we provide a generic common representation of particles and atoms, which are treated as normal forms.

4.1 Foundations

Assume an infinitely countable set of attribute names Ω. A *schema S* is a finite list of attributes $S = [A_1…, A_n]$, $A_i \in \Omega$, $i = 1…n$. Each attribute A is accompanied by a domain $dom(A)$. An *atomic formula* of a selection condition is *true*, *false* or an expression of the form $x\,\theta\,y$, where θ is an operator from the set $(>, <, =, \geq, \leq,$

$\neq)$ and each of x and y can be one of the following: (a) an attribute A, (b) a value l belonging to the domain of an attribute $l \in dom(A)$. A *selection condition φ* is a formula that combines atomic formulae in disjunctive normal form.

Assume also an infinitely countable set of template activity names Λ. Each template activity $t \in \Lambda$ is accompanied by a predicate name $P_t()$ and a finite set of parameter names $D = \{D_1, …, D_m\}$. The predicate carries commonly accepted, interpreted semantics for the template. For example, assume a template activity *notNull*, with commonly accepted semantics of testing inputs for not null values over a specific attribute, expressed as the parameter D_1. A particle is practically an instantiation of the template activity over a concrete schema that maps the parameter names of the template to a specific set of attributes $P_t(X)$, $X = [X_1,…,X_n]$, $X_i \in \Omega$, $i = 1…n$. Hence, the template activity *notNull*, can be materialized as *notNull(Age)* and D_1 has been substituted by an attribute named *Age*.

A specific subset of the template activities M involves activities that merge several input schemata (e.g., *join*(), *diff*(), *sortedUnion*(), *partialDiff*(), and so on). We refer to the members of this set as *mergers*.

A router r is defined as a finite set of selection conditions (not necessarily disjoint with each other).

Definition (ETL Atom). An ETL atom is a pentad of the form (I, $m()$, $P(X)$, r, O), where:

- I is a finite set of (input) schemata,
- m is a merger,
- $P(X)$ is a materialization of a template predicate over the schema X, which we call *functionality schema* of the atom,
- r is a router,
- O is a finite set of (output) schemata.

The following well-formedness constraints must hold for an ETL atom:

- X is a subset of the union of attributes of the schemata of I.
- There is a 1:1 mapping between the selection conditions of r and the output schemata of O. Assuming $O = [O_1,…,O_n]$, and $r = [\varphi_1,…, \varphi_n]$, we will consider that condition φ_i corresponds to schema O_i, for all $i = 1…n$.

Assuming $X = [X_1,…,X_n]$, the semantics of a tuple t arriving at an output schema O_i are $merge(I) \wedge P(t.X_1,…,t.X_n) \wedge \varphi_i$.

Observe that single inputs have a *true* merger particle and single outputs have a single valued {*true*} router particle.

Example. Let us see how the particle classes of Table 1 correspond to the abovementioned formal definitions. As an example of unary atoms, unary groupers are atoms of the form: (I_1, *true*, $group(X_{groupers}, X_{grouped})$, *true*, O_1). An example of a binary atom is the traditional join of two inputs, which is expressed as: ($I(I_1,I_2)$, $join(join\text{-}fields)$, *true*, *true*, O_1). More complex atoms with one particle can also be expressed. Assume a specific case where a join atom merges items and orders, performs a conversion of Euros to Dollars values over attribute cost, and routes the results to output O_1 if the dollar cost is higher than 500 or to output O_2 in any other case. This transformation is expressed as:

(I (I_{ORDERS}, I_{ITEMS}), $join(O.I\_ID=I.IID)$, $€2\$(€Cost,\$Cost)$, {$\$Cost>500$, $\$Cost <=500$ }, $O(O_1,O_2)$).
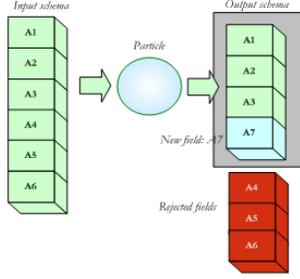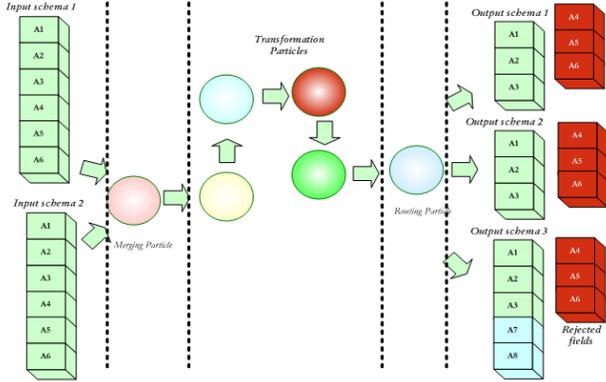
Figure 3. A unary ETL atom
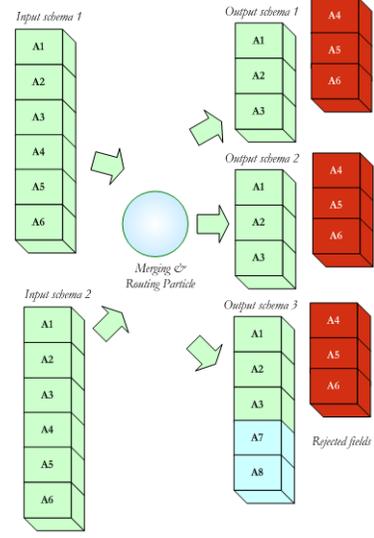

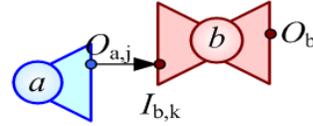**Figure 5. An ETL molecule**

Figure 4. A binary ETL atom

Figure 6. **Coupling of two ETL molecules**

Definition (ETL molecule). An ETL molecule is defined as a pentad of the form $(I, m(), P, r, O)$. The definition is similar to the definition of ETL atoms, with the following additions:

– $P = [P_1(X_1), \ldots, P_n(X_n)]$ is a list of predicates, each corresponding to a particle. In other words, the particles form a line, with the order of the predicates respecting the order of the particles in the line.

– The semantics of a tuple t arriving at an output schema O_i are $merge(I) \wedge P(t.X_{11},\ldots,t.X_{1m}) \wedge \ldots \wedge P(t.X_{n1},\ldots,t.X_{nm}) \wedge \varphi_i$, assuming the respective schemata $X_i = [X_{i1},\ldots,X_{im}]$.

Several operations are defined for ETL molecules. *Coupling* two molecules explains how the output of a molecule connects to the input of the other; this operation formalizes the workflow construction and it is needed for defining *ETL compounds* too. *Swapping* two molecules allows rearranging their execution order; this operation is useful for the algebraic optimization of ETL workflows, as for example, for executing a more selective activity before a less selective one. In addition, we can *merge* two molecules into one and *split* a molecule to two others. Next, we elaborate on these operations.

4.2 Coupling of two molecules

The coupling of two molecules is a simple act of mapping the output of one molecule to the input of the other. Therefore, we need a mapping M among the output schema O_X of a molecule a and the input schema I_Y of another molecule b.

Definition. Assume a molecule a $(I_a, m_a(), P_a, r_a, O_a)$ and one of its output schemata $O_{a,j}$ and an activity b $(I_b, m_b(), P_b, r_b, O_b)$ and one of its input schemata $I_{b,k}$ (see Figure 6). Assume also a mapping M between $O_{a,j}$ and $I_{b,k}$. Finally, assume an arbitrary output of b, say O_b. Then, the following hold:

– For each tuple arriving at $O_{a,j}$ the semantics are $sem(I_{a,j})$: $m_a(I_a) \wedge P_a \wedge \varphi_j$.

– For each tuple arriving at O_b the semantics are $sem(O_b)$: $m_b(I_{b1},\ldots, I_{bn}) \wedge P_b \wedge \varphi_{Ob}$.

– After the composition, the second semantics is $m_b(I_{b1}, \ldots, I_{bk-1}, M(I_{bk}), I_{bk+1}, \ldots, I_{bn}) \wedge P_b \wedge \varphi_{Ob} = m_b(I_{b1}, \ldots, I_{bk-1}, (m_a(I_a) \wedge P_a \wedge \varphi_j), I_{bk+1}, \ldots, I_{bn}) \wedge P_b \wedge \varphi_{Ob}$

Similarly, semantics can be defined for all inputs of molecule b.

Example. A simple atom with one input and one output can be coupled with another atom of the same family as follows: $sem(O_a) = sem(I_a) \wedge P_a$, meaning that $sem(O_b) = sem(I_b) \wedge P_b = sem(M(I_b)) \wedge P_b = sem(I_a) \wedge P_a \wedge P_b$.

Having defined couplings, we can now introduce compositions of molecules (i.e., activities) to compounds (i.e., workflows).

Definition (ETL Compound). An ETL atom is a tetrad of the form (D_f, D_s, M, C), where:

– D_f, is a finite set of input fountain data stores,

– D_s, is a finite set of intermediate or target data stores,

– M is a finite set of molecules

– C is a finite set of correspondence mappings between molecules and data stores

The following well-formedness constraints must hold for an ETL compound:

– The schemata of fountain data stores in D_f are mapped to input activity schemata and only. Every schema of the data stores of D_s has the output schema of at least one activity mapped to it. A special case of sink, i.e., target, data stores are not further mapped to other schemata.

- No molecule has unmapped schemata.
- The graph having a finite set of data stores and molecules as nodes and the mappings among them as directed edges is acyclic.

4.3 Composition and Splitting of ETL activities

Composition of molecules is an act of merging two particles into one. The inverse act, splitting, subtracts a molecule from another.

Serial composition. Assume two ETL molecules, $a_1 = (I_1, m_1(), P_1, r_1, O_1)$ and $a_2 = (I_2, m_2(), P_2, r_2, O_2)$. Then, under certain conditions, it is possible to merge these two molecules into one. At the same time it is also possible to show that there are cases where two molecules cannot be composed.

Theorem. Assume a molecule a_1 that has exactly one output O and a molecule a_2 has exactly one input I. Assume also that the attributes of O are a superset of the attributes of I. In this case, we can define a new molecule $a_3 = a_1 \circ a_2$, $a_3 = (I_3, m_3(), P_3, r_3, O_3)$ such that: $I_3 = I_1$, $m_3() = m_1()$, $P_3 = P_1 \cup P_2$, $r_3 = r_2$, $O_3 = O_2$.

Proof sketch. It is clear that a mapping can be devised among the two schemata. Then, the semantics for the output of the second molecule are the same with the ones for molecule a_3.

Still, serial composition is not always possible. On the contrary, the fact that routers are exactly before the outputs imposes a necessary constraint for composition.

Theorem. Serial composition of two molecules is not a closed operation.

Proof. Assume a molecule a_1 that has exactly 2 outputs $O_{1,1}$ $O_{1,2}$ and an atom a_2 that has exactly one input I and one output O. Assume also that we want to compose a_2 with $O_{1,1}$. This is the simplest possible non-feasible case of serial composition. If we compose a_1 and a_2, into one molecule $a_3 = a_1 \circ a_2$, then $a_3 = (I_1, m_1(), P_1 \cup P_2, r_1, \pi^-_2, \pi^+_2, O)$. This is problematic since the tuples arriving at $O_{1,2}$ will have semantics:

$$merge(I_1) \wedge P_{1,1} (X_{1,1}) \wedge P_{1,2} (X_{1,2}) \wedge P_2 (X_2) \wedge \varphi_2$$

as opposed to the appropriate

$$merge(I_1) \wedge P_{1,1} (X_{1,1}) \wedge P_{1,2} (X_{1,2}) \wedge \varphi_2 \qquad \square$$

Subtraction. Subtraction is the inverse operation of composition and produces a molecule with less particles, or schemata. Formally, assume two molecules a_1 and a_2 that have the same merger m. Then, we can define a new molecule $a_3 = a_1 - a_2$, $a_3 = (I_3, m, P_3, r_3, O_3)$ such that:

$I_3 = \{I_{1i} - I_{2i}\}$ for all the input schema of I_1

$P_3 = P_1 - P_2$,

$r_3 = [\varphi_1, \ldots, \varphi_n]$, s.t. $\neg \varphi_{1,i} \Rightarrow \varphi_{2,i}$ for all the selection conditions of the router r_1

$O_3 = \{O_{1i} - O_{2i}\}$ for all the output schemata of O_1,

and where the attributes participating in the merger and router are still present after the subtraction of the input schemata.

4.4 Management of Schemata

A subtle point not covered so far is the management of schemata and the mapping among them. The complete semantics of a molecule are given via a mapping M (which is not necessarily a function) that maps input to output attributes. Then, M: attributes(I) $\rightarrow$ attributes(O) which is onto, but not necessarily total or bijective.
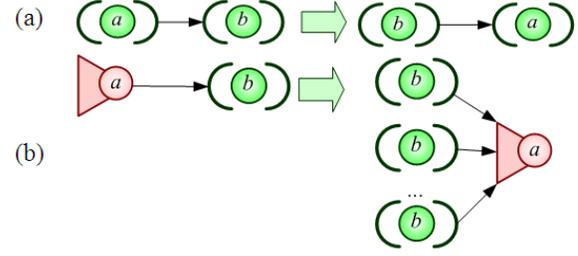

Figure 7. Activity swapping

In the case that M is not total it means that there are attributes that are not propagated from the output of an activity to the corresponding input of a subsequent activity. Another point is that due to the particles in the chain of the molecule, new attributes are generated. A straightforward extension of the normal form with attributes involves the following actions:

- Inclusion of two schemata, π^+, π^- the first with generated attributes and the second with attributes projected-out.
- Each particle is defined as $P(X,Y)$, with X being its input parameters and Y being the generated parameters.
- A constraint that guarantees that for every particle $P_a(X_a,Y_a)$ in the chain (the router included), its input parameters are a subset of the union of attributes of all the input schemata and the generated attributes of the previous particles.

Then, a molecule can be defined as $(I, m(), P(), r, \pi^-, \pi^+, O)$.

The treatment of schemata is useful, since there are two ways to populate the schema mapping function with the appropriate pairs: (a) automatically or (b) manually (as currently happens in ETL tools). The automatic way has been described in previous work [15] and it computes schemata from the target of the workflow towards its start based on the templates. The templates' parameters need to be substantiated by specific attributes involved in the schema (e.g., the template $NotNull_t(p)$, p being a template parameter can be instantiated as $NotNull(Sal)$, with Sal being a concrete input attribute). In this case, we need to assign π^+, π^- to compute the exact attributes that participate in the computed schemata.

4.5 Swapping of two activities

A straightforward application of the manual generation of schemata involves the swapping of activities. In principle, two molecules a and b can be swapped in two ways. The first (Figure 7a) involves two unary activities and the swapping can be performed if the attributes needed for a to operate are still present after the swapping. The second (Figure 7b) brings a unary activity a before all the input schema of an n-ary activity b. Again, the same constraint needs to hold. The formal proof for this result has been provided in [15] and fits gracefully in the current framework.

5. ETL ARCHETYPE PATTERNS

The previous sections discussed how we can introduce a taxonomy and a normal form as a basis for a theory for ETL activities and workflows. In this section, we go one step further and suggest that apart from a normal form of activities (as discussed in Section 4), it is meaningful to come up with normal forms for whole workflows, too with an aim of exploiting them in terms of optimization, resource allocation, and scheduling.

We refer to these frequently reused compounds as *patterns* or

butterflies [14] due to their structure, since they are composed of three parts: (a) the *left wing*, which deals with the combination, cleaning and transformation of source data on their way to the warehouse, (b) the *body of the butterfly*, which involves the main points of storage of these data in the warehouse, and, (c) the *right wing*, which involves the maintenance of data marts, reports, and so on, after the fact table has been refreshed; all are abstracted as materialized views that have to be maintained. Typical butterflies are: the *line*, *wishbone*, *tree*, *fork* workflows, the *primary flow* and so on (see [14] for more details).

Figure 8 depicts (a) a *primary flow* and (b) a combination of a *line* and *fork workflows*. A primary flow is a typical pattern for the beginning of ETL processing, when data should be assigned surrogate keys for every dimension of the warehouse. A fork scenario, on the other hand, is a typical scenario at the end of the ETL process, where data that are already cleansed and ready for loading in the fact table are also ready for multiple aggregations in order to populate reports, data marts and materialized views (all abstracted as materialized views in our examples). The line workflow combines linearly a set of unary activities.

As an alternative use of patterns, besides their use for the operational side of ETL tools, we have found that the treatment of large ETL scenarios as compositions of archetypical structures is beneficiary for the purpose of benchmarking ETL as well [14, 21].

Next, we discuss two practical problems that can be tackled using the ETL archetype patterns.

## 5.1 Archetype patterns for the physical optimization of ETL workflows

In previous research, we experimented with the physical optimization of ETL processes and the exploration of alternatives for the execution of ETL scenarios. Part of this work, concerned the exploration of intentional addition of sorters to the workflows to exploit the order of data at the physical level [21]; once the cost for sorting has been paid, costly operations like joins and aggregations may exploit the ordering of data and complete much faster.

For example, Figure 9 shows that for different ETL patterns and different settings (not shown in the figure) we get different gains when adding sorters. As a general rule, when data volumes are high (left wing of the butterfly, early stages of the ETL process) sorting is not that beneficial. Once data have been cleansed and integrated, sorting can accelerate the aggregations of the right wing (materialized view loading part) of the ETL scenario.

However, the main insight here is that in principle, once the ETL scenario has been designed as the composition of archetype patterns, an optimizer can avoid spending the time needed to explore alternative configurations and directly go to the appropriate heuristics for each of its parts.

5.2 Scheduling ETL workflows

We have also experimented with the usage of alternative scheduling policies (like round robin, minimum cost, minimum memory) for different archetype patterns [8]. Again, the presence of archetype patterns can serve as the basis for the decisions taken by a scheduler of an ETL tool. Once the designer has studied the effect of an ETL application to the ETL archetype patterns, then he/she is able to tune the whole ETL workflow accordingly.
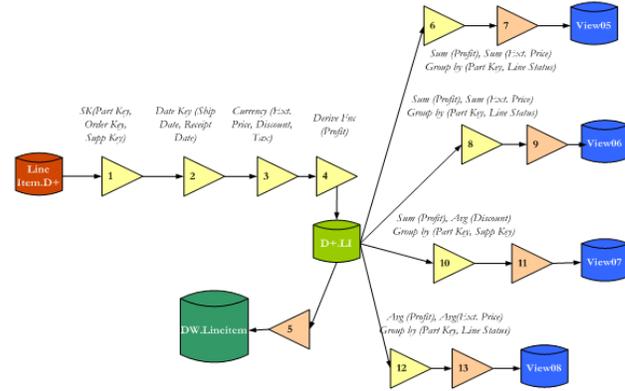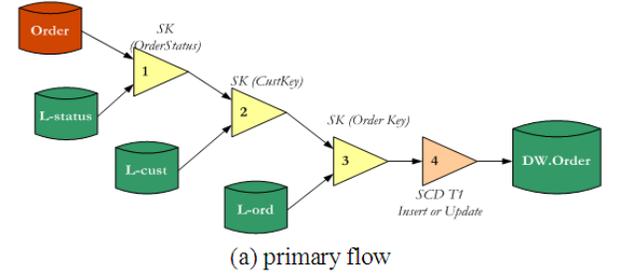
(a) primary flow

(b) combination of line and fork workflows

Figure 8. **Example ETL archetypical patterns**

Observe for example Figure 10. In the case of a primary flow, we can find a scheduling policy that minimizes the peaks in memory consumption. In the case of a fork scenario, the optimal choice is dependent upon the selectivity of the scenario; in this case, the scheduler needs to take this kind of statistical information under consideration before scheduling this part of the scenario. Having acquired such knowledge, one may tune the scheduling of an ETL workflow containing these two constructs appropriately.

5.3 Overall observations

The ETL design can be viewed as the act of compound composition via atoms and molecules. Whereas atoms can be of arbitrary nature, it is very important that compounds (i.e., workflows) are constructed in ways that can later be exploited by the optimizer or the scheduler of an ETL tool. Archetype patterns have already been suggested [14] and it is important to realize that the synthesis of ETL scenarios should be performed in a manner that allows the underlying engine to exploit them. In simple terms, *designers should try to compose compounds in ways that resemble archetype patterns; the design tools, on the other hand, could present the user with the opportunity to construct the design in terms of such patterns that are later exploitable by the execution engine*.

6. RELATED WORK

Several research approaches have dealt with ETL modeling by exploring either UML [e.g., 9, 20], semantic web [e.g., 17] or sui generis [23] modeling techniques . Other efforts have focused on the optimization of ETL processes [e.g., 6, 15, 21] and on individual operators [12] or phases, like the load [18, 19]. However, none of the existing works has dealt with the problem of categorizing ETL activities based on their characteristics.

	Number of Sorters	Cost of Sorters	Percentage of Sorter Cost
Fork	1	695.594	41%
	1	695.594	29%
	1	695.594	29%
	1	695.594	29%
	0	0	0%
Primary Flow	0	0	0%
	1	118.221	30%
	1	132.877	32%
	1	132.877	32%
	2	251.099	51%

Figure 9. Effect of adding sorters in ETL workflows [21]

Figure 10. Effect of workflow selectivity to memory peaks under different scheduling policies

Apart from research efforts, there is a variety of ETL tools available in the market [e.g., 5, 7, 10, 11]. Given that so far there is no standard categorization of ETL activities, each tool follows a different approach. Interestingly, our categorization (see Table 1) covers the functionality provided by these tools.

Work on schema mappings [e.g., 4, 13] is related to ETL processes; however, existing efforts focus on a subset of mappings that usually we encounter in typical ETL scenarios. Some works deal with the issue of input and output relationships both, in terms of the involved schemata [3] and tuples [1]. The former work presents a set of mapping operators for entities and attributes. It discusses a classification of possible mapping cases based on the cardinality of the schemata. However, our work delves into the semantics of an ETL activity and also, considers the physical operation of ETL activities. The latter work is representative of a large part of the related literature concerned with the issue of input-output relationship in terms of the involved instances — usually under the name of lineage tracing. The goal of that work is to identify the originating tuples out of which a data warehouse tuple was produced (with the obvious benefit of being able to identify which part of the ETL process must be resumed in the case of a failure, once intermediate results are rescued from the failure). We are looking at the instance mappings from a different perspective: the categorization of ETL activities based on those mappings.

7. CONCLUSIONS

As Business Intelligence deals with continuously increasing amounts of information and more complex environments, there is a stressing need for standardizing ETL processes that feed data warehouses [2, 16]. In this paper, we have introduced a generic categorization of ETL activities and have demonstrated that the ETL transformations used by popular commercial ETL tools fall into our categorization. We have presented how this simple characterization can lead to a normal form for performing simple operations for ETL activities, like composition, coupling and swapping. Apart from a normal form for the formal treatment of ETL activities and workflows, we also discuss normal forms at the macro level (i.e., design patterns) and demonstrate evidence that these reusable patterns can be used for improving the efficiency of ETL flows.

An interesting challenge for future work is to construct an ETL optimizer that will be able to automatically decompose ETL workflows into archetype patterns and then, optimize them based on the proposed principles.
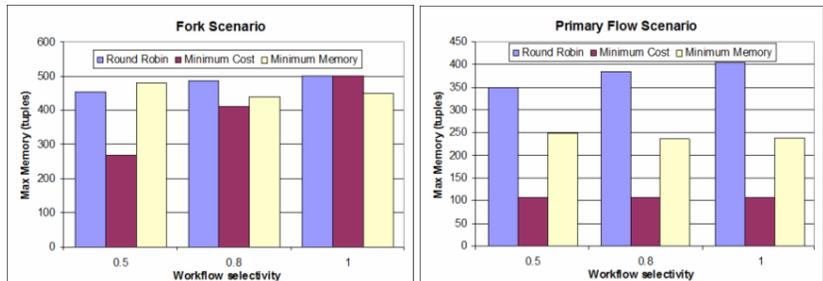
8. REFERENCES

1. Y. Cui, J. Widom. Lineage tracing for general data warehouse transformations. In VLDB J. 12(1): 41-58, 2003.
2. U. Dayal, M. Castellanos, A. Simitsis, K. Wilkinson. Data integration flows for business intelligence. In EDBT, pp. 1-11, 2009.
3. A. Dobre, F. Hakimpour, K.R. Dittrich. Operators and Classification for Data Mapping in Semantic Integration. In ER, pp. 534-547, 2003.
4. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In SIGMOD, pp. 805-810, 2005.
5. IBM, "IBM Data Warehouse Manager", in the Web, available at: http://www-306.ibm.com/software/data/integration/datastage/,2009.
6. Informatica. How to Achieve Flexible, Cost-effective, Scalability and Performance through Pushdown Processing. White paper, 2007.
7. Informatica, "PowerCenter", in the Web, available at: http://www.informatica.com/products/powercenter/, 2009.
8. A. Karagiannis, P. Vassiliadis, A. Simitsis. Macro-level Scheduling of ETL Workflows. Submitted for publication, 2009.
9. S. Luján-Mora, P. Vassiliadis, J. Trujillo. Data Mapping Diagrams for Data Warehouse Design with UML. In ER, pp.191-204, 2004.
10. Microsoft. SQL Server 2005 Integration Services (SSIS), in the Web, available at: http://technet.microsoft.com/en-us/sqlserver/bb331782.aspx, 2009
11. Oracle, "Oracle Warehouse Builder 10g", in the Web, available at http://www.oracle.com/technology/products/warehouse/, 2009.
12. N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, N.-E. Frantzell. Supporting Streaming Updates in an Active Data Warehouse. In ICDE, pp. 476-485, 2007.
13. E. Rahm, P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. In VLDB Journal, 10(4): 334-350, 2001.
14. A. Simitsis, P. Vassiliadis, U. Dayal, A. Karagiannis, V. Tziovara. Benchmarking ETL Workflows. In TPC-TC, 2009.
15. A. Simitsis, P. Vassiliadis, T. K. Sellis. Optimizing ETL Processes in Data Warehouses. In ICDE, pp. 564-575, 2005.
16. A. Simitsis, K. Wilkinson, M. Castellanos, U. Dayal. QoX-driven ETL design: reducing the cost of ETL consulting engagements. In SIGMOD Conference, pp. 953-96, 2009.
17. D. Skoutas, A. Simitsis. Designing ETL processes using semantic web technologies. In DOLAP, pp. 67-74, 2006.
18. M. Thiele, U. Fischer, W. Lehner: Partition-based workload scheduling in living data warehouse environments. In DOLAP, pp. 57-64, 2007.
19. C. Thomsen, T.B. Pedersen, W. Lehner. RiTE: Providing On-Demand Data for Right-Time Data Warehousing. In ICDE, pp. 456-465, 2008.
20. J. Trujillo, S. Luján-Mora: A UML Based Approach for Modeling ETL Processes in Data Warehouses. In ER, pp. 307-320, 2003.
21. V. Tziovara, P. Vassiliadis, A. Simitsis. Deciding the physical implementation of ETL workflows. In DOLAP, pp. 49-56, 2007.
22. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis. A Framework for the Design of ETL Scenarios. In CAiSE, pp. 520-535, 2003.
23. P. Vassiliadis, A. Simitsis, S. Skiadopoulos: Conceptual modeling for ETL processes. In DOLAP, pp. 14-21, 2002.