A Taxonomy of ETL Activities

Panos Vassiliadis
University of Ioannina
Ioannina, Greece
pvassil@cs.uoi.gr

Alkis Simitis
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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DOLAP’09, November 6, 2009, Hong Kong, China.
Copyright 2009 ACM 978-1-60558-801-8/09/11...$10.00.

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 tested 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 schema of an activity populate its output schema 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 schema.
Table 1. Built-in transformations provided by commercial ETL tools

<table>
<thead>
<tr>
<th>Atom class</th>
<th>Particle class</th>
<th>Transformation Category</th>
<th>SQL, SSIS</th>
<th>DataStage</th>
<th>Oracle Warehouse Builder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td></td>
<td>Raw-level: Function that can be applied locally to a single row</td>
<td>Character Map, Copy Column, Dim Conversion, Desired Columns, Script Component, OLE DB Command, Other filters (not null, selections, etc.)</td>
<td>Transforms (A generic representative of a broad range of functions: date and time, logical, mathematical, null handling, number, raw, str, string, type conversion/existing, counting)</td>
<td>Deduplicator (demon) Filter, Sequence, Concatenator, Table function (it is applied on a set of rows for increasing the performance) Data Cleansing Operators (Numeric and Address, Match-Merge) Other SQL transformations (Character, Date, Number, XML, etc.)</td>
</tr>
<tr>
<td>N:1</td>
<td>Unary Group sep: Transform a set of rows to a single row</td>
<td>Aggregate, Pivot</td>
<td>Aggregator, Combine/Promote records</td>
<td>Aggregator, Pivot</td>
<td></td>
</tr>
<tr>
<td>L:1</td>
<td>Unary Splitter: Split a single row to a set of rows</td>
<td>Unpivot</td>
<td>Make/Split subrecord, Make/Split vector</td>
<td>Unpivot</td>
<td></td>
</tr>
<tr>
<td>N:M</td>
<td>Unary Holistic: Perform a transformation to the entire data set (blocking)</td>
<td>Sort, Percentage Sampling, Raw Sampling</td>
<td>Sort (sequential, parallel, total)</td>
<td>Sorter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Blazar or N:ary: Combine many inputs into one output</td>
<td>Union-like, Union All, Merge, Join-like, Merge Join (M1), Lookup (SK1), Import Column (NLI)</td>
<td>Union-like (distinct, continuous, sort, sequence) Join-like, Join, Merge, Lookup, Diff-like, Change capture/apply, Difference (record by record), Compare (columns by column)</td>
<td>Union-like (set, union, union all, intersect, minus) Join-like, Inter, Key Lookup (SK1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Routers: Locally decide, for each row, which of the many outputs it should be sent to</td>
<td>Conditional Split, Multicast</td>
<td>Copy, Filter, Switch</td>
<td>Splitter</td>
<td></td>
</tr>
</tbody>
</table>

- **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.
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 \( \Omega \). A schema \( S \) is a finite list of attributes \( S = [A_1, \ldots, A_n], A_i \in \Omega, i = 1 \ldots n \). Each attribute \( A \) is accompanied by a domain \( \text{dom}(A) \). An atomic formula of a selection condition is \( \text{true} \), false or an expression of the form \( x \ \theta \ y \), where \( \theta \) is an operator from the set \( \{\neq, \leq, \geq, \lt, \gt, =\} \) 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 \( l \in \text{dom}(A) \). A selection condition \( \phi \) is a formula that combines atomic formulae in disjunctive normal form.

Assume also an infinitely countable set of template activity names \( \mathcal{A} \). Each template activity \( t : A \) is accompanied by a predicate name \( P(t) \) and a finite set of parameter names \( D = \{D_1, \ldots, D_m\} \).

The predicate carries commonly accepted, interpreted semantics for the template. For example, assume a template activity \( \text{notNull} \), 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 \( P(X), X = \{X_1, \ldots, X_n\}, X_i \in \Omega, i = 1 \ldots n \). Hence, the template activity \( \text{notNull} \), can be materialized as \( \text{notNull}(Age) \) and \( D_1 \) has been substituted by an attribute named \( Age \).

A specific subset of the template activities \( \mathcal{M} \) involves activities that merge several input schemata (e.g., \( \text{join}(\cdot), \text{diff}(\cdot), \text{sortedUnion}(\cdot), \text{partialDiff}(\cdot) \), 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 \( O = \{O_1, \ldots, O_n\} \), and \( r = [\phi_1, \ldots, \phi_n] \), we will consider that condition \( \phi_i \) corresponds to schema \( O_i \), for all \( i = 1 \ldots n \).

Assuming \( X = \{X_1, \ldots, X_n\} \), the semantics of a tuple \( t \) arriving at an output schema \( O_i \) is \( \text{merge}(I_i) \land P(t, X_i, \ldots, t, X_n) \land \phi_i \).

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: \( (I_1, \text{true}, \text{group}(X_{\text{group}}, X_{\text{group}}), \text{true}, O_1) \). An example of a binary atom is the traditional join of two inputs, which is expressed as: \( (I_1, I_2, \text{join}(\text{join-fields}), \text{true}, \text{true}, O_1) \). 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 \( O_2 \), if the dollar cost is higher than 500 or to output \( O_3 \) in any other case. This transformation is expressed as:

\[
(I_\text{IORDERS}, I_\text{ITEMS}, \text{join}(O_1, \text{ID}=\text{IID}), \text{\$28(\text{\$Cost} - \text{\$Cost})}, \\
\{\text{\$Cost}>500, \text{\$Cost}<=500\}, O_2, O_3).
\]
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:

- \(P = \{P_1(X_1), \ldots, P_s(X_s)\}\) 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 \(\text{merge}(I_i) \land P(tX_i) \land \cdots \land P(tX_m) \land \phi_i\) assuming the respective schemata \(X_i = [X_{i1}, \ldots, X_{in}]\). 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 \(O_a\) of a molecule \(a\) and the input schema \(I_b\) 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 \(O_{a,b}\) 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 \(O_{a,b}\) 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 \(\text{sem}(I_{a,j}) = m_{a}(I_{a,j}) \land P_{a} \land \phi_{a,j}\).
- For each tuple arriving at \(O_{b}\) the semantics are \(\text{sem}(O_{b}) = m_{b}(I_{b,1}, \ldots, I_{b,n}) \land P_{b} \land \phi_{b,0}\).
- After the composition, the second semantics is \(\text{sem}(O_{b}) = m_{b}(I_{b,1}, \ldots, I_{b,k-1}, M(I_{b,k}), I_{b,k+1}, \ldots, I_{b,n}) \land P_{b} \land \phi_{b,0} = m_{b}(I_{b,1}, \ldots, I_{b,k-1}, (m_{a}(I_{a,j}) \land P_{a} \land \phi_{a,j}), I_{b,k+1}, \ldots, I_{b,n}) \land P_{b} \land \phi_{b,0}\).

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: \(\text{sem}(O_{a}) = \text{sem}(I_{a}) \land P_{a} \land \phi_{a}\), meaning that \(\text{sem}(O_{b}) = \text{sem}(I_{b}) \land P_{b} = \text{sem}(M(I_{a})) \land P_{a} \land \phi_{a}\).

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 \((D, D', M, C)\), where:

- \(D\) is a finite set of input fountain data stores,
- \(D'\) 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 \(D\) are mapped to input activity schemata and only. Every schema of the data stores of \(D\) 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 \( a_3 = a_1 \cup a_2, a_3 = (I, m_3(), P_3, r_3, O_3) \) such that: \( I_1 = I, m_1() = m_1(), P_1 \cup P_2, r_1 = r_2, O_1 = O_2 \).

**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 \( a_3 = a_1 \cup a_2 \), then \( a_3 = (I_1, m_1(), P_1 \cup P_2, r_1, \pi_{2,1}, \pi_{2,2}, O) \). This is problematic since the tuples arriving at \( O_{1,2} \) will have semantics:

\[
\text{merge}(I_1) \times P_{1,1} (X_{1,1}) \times P_{1,2} (X_{1,2}) \times P_2 (X_2) \times \varphi_2
\]

as opposed to the appropriate

\[
\text{merge}(I_1) \times P_{1,1} (X_{1,1}) \times P_{1,2} (X_{1,2}) \times \varphi_2
\]

**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 \( a_3 = a_1 \setminus a_2, a_3 = (I_3, m, P_3, r_3, O_3) \) such that:

\[
I_3 = \{I_{1,i} - I_{2,i}\} \quad \text{for all the input schema of } I_1
\]

\[
P_3 = P_1 \setminus P_2,
\]

\[
r_3 = [\phi_1, \ldots, \phi_n] \text{ s.t. } \neg \phi_{i,1} \Rightarrow \phi_{2,i} \quad \text{for all the selection conditions of the router } r_1
\]

\[
O_3 = \{O_{1,i} - O_{2,i}\} \quad \text{for all the output schema of } O_i,
\]

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, \( M: \text{attributes}(I) \rightarrow \text{attributes}(O) \) which is onto, but not necessarily total or bijective.

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
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 operations.

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 beneficial 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.

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.
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