Conceptual Modeling for ETL Processes

Panos Vassiliadis, Alkis Simitsis, Spiros Skiadopoulos[1]

[1] National Technical University of Athens, Dept. of Electrical and Computer Eng.,

Computer Science Division, Iroon Polytechniou 9, 157 73, Athens, Greece

{pvassil,asimi,spiros}@dbnet.ece.ntua.gr

1. INTRODUCTION 2

2. RELATED WORK AND STATE OF THE ART 6

3. CONCEPTUAL MODEL 7

3.1 CONCEPTS AND ATTRIBUTES 10

3.2 TRANSFORMATIONS, CONSTRAINTS AND NOTES 10

3.3 PART-OF AND CANDIDATE RELATIONSHIPS 12

3.4 PROVIDER RELATIONSHIPS AND SERIAL COMPOSITION OF TRANSFORMATIONS 13

4. METHODOLOGY FOR THE USAGE OF THE CONCEPTUAL MODEL 16

5. INSTANTIATION AND SPECIALIZATION LAYERS 20

6. CONCLUSIONS 23

ACKNOWLEDGMENTS 24

REFERENCES 24

Conceptual Modeling for ETL Processes

Panos Vassiliadis, Alkis Simitsis, Spiros Skiadopoulos[1]

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

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. Research has only recently dealt with the above problem and has not established commonly accepted models, formalisms and techniques for tasks like the job definition, scheduling, monitoring and error handling in an ETL environment. In this paper, we focus on the problem of the definition of ETL activities and provide formal foundations for their representation. The proposed model is (a) customized for the tracing of inter-attribute relationships and the respective ETL activities in the early stages of a data warehouse project; (b) enriched with a 'palette' of a set of frequently used ETL activities, like the assignment of surrogate keys, the check for null values, etc; and (c) constructed in a customizable and extensible manner, so that the designer can enrich it with his own re-occurring patterns for ETL activities.

1. Introduction

Extraction-Transformation-Loading (ETL) tools is a category of specialized tools with the task of dealing with data warehouse homogeneity, cleaning and loading problems. [ShTy98] reports that ETL and Data Cleaning tools are estimated to cost at least one third of effort and expenses in the budget of the data warehouse while [Dema97] mentions that this number can rise up to 80% of the development time in a data warehouse project. [Inmo97] mentions that the ETL process costs 55% of the total costs of data warehouse runtime. 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. In fact, while data warehouse expenses are expected to come up to 14 billion dollars worldwide, projected sales for ETL and data cleaning tools are expected to rise to only (!) 300 million dollars. Thus, it is apparent that the design, development and deployment of ETL processes, which is currently performed

in an ad-hoc, in house fashion, needs modeling, design and methodological foundations. Unfortunately, as we shall show in the sequel, the research community has a lot of work to do to confront this shortcoming. In the rest of the paper, we will not discriminate between the tasks of ETL and Data Cleaning and adopt the name ETL for both these kinds of activities.

In Fig. 1.1 we abstractly describe the general framework for ETL processes. In the bottom layer we depict the data stores that are involved in the overall process. On the left side, we can observe the original data providers. Typically, data providers are relational databases and files. The data from these sources are extracted (as shown in the upper left part of Fig. 1.1) by extraction routines, which provide either complete snapshots or differentials of the data sources. Then, these data are propagated to the *Data Staging Area* (DSA) where they are transformed and cleaned before being loaded to the data warehouse. The data warehouse is depicted in the right part of the data store layer and comprises the target data stores, i.e., (a) fact tables for the storage of information and (b) dimension tables with the description of the multidimensional, roll-up hierarchies of the stored facts. Eventually, the loading of the central warehouse is performed through the loading activities depicted on the upper right part of the figure.

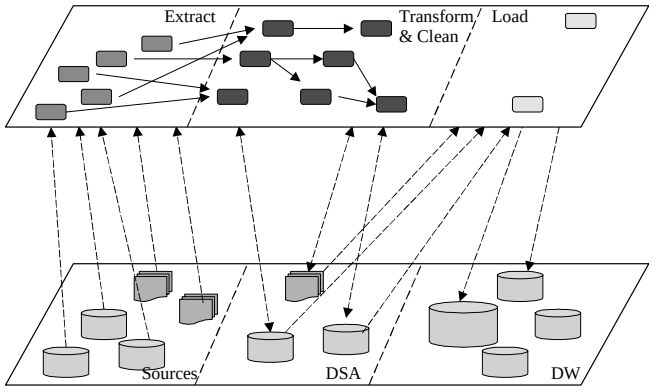


Fig. 1.1 The environment of Extract-Transform-Load processes

In [KRRT98] one can find a detailed description of the different aspects of the ETL process, including: the job definition of the process; time/event based scheduling; monitoring (i.e., on-line information about the progress/status of the process); logging (i.e., off-line information, presented at the end of the execution); error handling (for incoming data

violating consistency constraints or business rules); crash recovery and stop/start capabilities (e.g., committing a transaction every few rows) and miscellaneous other features.

In this paper, we focus on the conceptual part of the definition of the ETL process. More specifically, we are dealing with the earliest stages of the data warehouse design. During this period, the data warehouse designer is concerned with two tasks which are practically executed in parallel. The first of these tasks involves the *collection of requirements* from the part of the users. The second task, which is of equal importance for the success of the data warehousing project, involves *the analysis of the structure and content of the existing data sources* and their *intentional mapping to the common data warehouse model*. Related literature [KRRT98] and personal experience [Vass00] suggest that the design of an ETL process aims towards the production of a crucial deliverable: the mapping of the attributes of the data sources to the attributes of the data warehouse tables.

The production of this deliverable involves several interviews that result in the revision and redefinition of original assumptions and mappings; thus it is imperative that a simple conceptual model is employed in order to (a) facilitate the smooth redefinition and revision efforts and (b) serve as the means of communication with the rest of the involved parties.

We believe that the formal modeling of the starting concepts of a data warehouse design process has not been adequately dealt by the research community. To this end, in this paper we propose a conceptual model for this task, with a particular focus on (a) the interrelationships of attributes and concepts and (b) the necessary transformations that need to take place during the loading of the warehouse. The latter part is directly captured in the proposed metamodel as a first class citizen; we employ *transformations* as a generic term for the restructuring of schema and values or for the selection and even the transformation of data. Attribute interrelationships are captured through *provider* relationships that map data provider attributes at the sources to data consumers in the warehouse. Apart from these fundamental relationships, the proposed model is able to capture constraints and transformation composition, too. Due to the nature of the design process, in this paper, we present the features of the conceptual model in a set of design steps, which lead to the basic

target, i.e., the attribute interrelationships. These steps constitute the methodology for the design of the conceptual part of the overall ETL process.

The proposed model is characterized by different instantiation and specialization layers. The generic metamodel that we propose involves a small set of *generic constructs* that are powerful enough to capture all cases. We call these entities the *Metamodel* layer in our architecture. Moreover, we introduce a specialization mechanism that allows the construction of a 'palette' of *frequently used ETL activities* (e.g., transformations like the surrogate key transformation or checks for null values, primary key violations, etc.). This set of ETL-specific constructs, constitute a subset of the larger metamodel layer, which we call the *Template Layer*. The constructs in the Template layer are also meta-classes, but they are quite customized for the regular cases of ETL processes. All the entities (data stores, inter-attribute mappings, transformations, etc.) that a designer uses in his particular scenario are instances of the entities in the metamodel layer. For the common ETL transformations, the employed instances correspond to the entities of the template layer.

More specifically, our contribution lies in:

1. The proposal of a novel conceptual model which is customized for the tracing of inter-attribute relationships and the respective ETL activities in the early stages of a data warehouse project.
2. The construction of the proposed model in a customizable and extensible manner, so that the designer can enrich it with his own re-occurring patterns for ETL activities.
3. The introduction of a 'palette' of a set of frequently used ETL activities, like the assignment of surrogate keys, the check for null values, etc.

This paper is organized as follows. In Section 2, we present related work. Section 3 presents the conceptual model for ETL processes. In Section 4, we demonstrate the methodology for the usage of the conceptual model. In Section 5, we introduce the instantiation and the specialization layers for the representation of ETL processes. Finally, in Section 6 we conclude our results.

2. Related Work and State of the Art

In this section we will review related work in the fields of conceptual modeling for data warehousing and ETL in general. For lack of space, we refer the interested reader to [VVS+01] for an extended discussion of the issues that we briefly present in this section.

Conceptual models for data warehouses. The front end of the data warehouse has monopolized the research on the conceptual part of data warehouse modeling. In fact, most of the work on conceptual modeling, in the field of data warehousing, has been dedicated to the capturing of the conceptual characteristics of the star schema of the warehouse and the subsequent data marts and aggregations (see [Tsoi01] for a broader discussion). Research efforts can be grouped in four major trends, including: (a) *dimensional modeling* [Kimb97, KRRT98], (b) *(extensions of) standard E/R modeling* [SBHD98, CDL+98, CDL+99, HuLV00, MoKo00, TrBC99] (c) *UML modeling* [NgTW00, TrPG00] and (d) *sui-generis models* [GoMR98, GoRi98, Tsoi01] without a clear winner. The supporters of the dimensional modeling method argue that the model is characterized by its minimality, understandability (especially by the end-users) and its direct mapping to logical structures. The supporters of the E/R and UML methods models base their arguments on the popularity of the respective models and the available semantic foundations for the well-formedness of data warehouse conceptual schemata. The sui-generis models are empowered by their novelty and adaptation to the particularities of the OLAP setting.

Conceptual models for ETL. There are few attempts around the specific problem of this paper, although we are not aware of any other approach that concretely deals with the specifics of ETL activities in a conceptual setting. We can mention [BaFM99] as a first attempt to clearly separate the data warehouse refreshment process from its traditional treatment as a view maintenance or bulk loading process. Still, the proposed model is informal and the focus is on proving the complexity of the effort, rather than the formal modeling of the activities themselves. [CDL+98, CDL+99] introduce the notion of intermodel assertions, in order to capture the mappings between the sources and the data warehouse. However, any transformation is dereferenced for the logical model where a couple of generic operators are employed to perform this task. In terms of industrial approaches, the model that stems from [KRRT98] would be an informal documentation of the overall ETL process.

Related work on ETL logical and physical aspects. Finally, apart from the commercial ETL tools [Arde01,Data01,Micr01,ETI01,Orac01] there also exist research efforts including [GFSS00, RaHe01, Mong00, BoDS00, RaHa00, VQVJ01, VVS+01, VaSS02, LWGG00]. The management of quality in data warehouses is discussed extensively in [JJQV99, JLVV00, JeQJ98].

We would like to stress that, in this paper, we do not propose another process/workflow model; thus, we do not intend to cover the composite workflow of ETL activities for the population of the warehouse. There are two basic reasons for this approach. First, in the conceptual model for the ETL process, the focus is on documenting/formalizing the particularities of the data sources with respect to the data warehouse and not in providing a technical solution for the implementation of the process. Secondly, the ETL conceptual model is constructed in the early stages of the data warehouse project during which, the time constraints of the project require a quick documentation of the involved data stores and their relationships, rather than an in-depth description of a composite workflow (see also Section 5 for this). In this sense, our approach is complementary to the aforementioned logical models, since it involves an earlier stage of the design process. We refer the interested reader to [VaSS02] for a formal modeling of this workflow, which is beyond the scope of this paper.

3. Conceptual Model

The purpose of this section is to present the conceptual model for ETL activities. Our goal is to specify the high level, user-oriented entities which are used to capture the semantics of the ETL process. First, we will present the graphical notation and the metamodel of our proposal. Then, we will detail and formally define all the entities of the metamodel. Throughout all the section, we will clarify the introduced concepts through their application to a motivating example that will support the discussion.

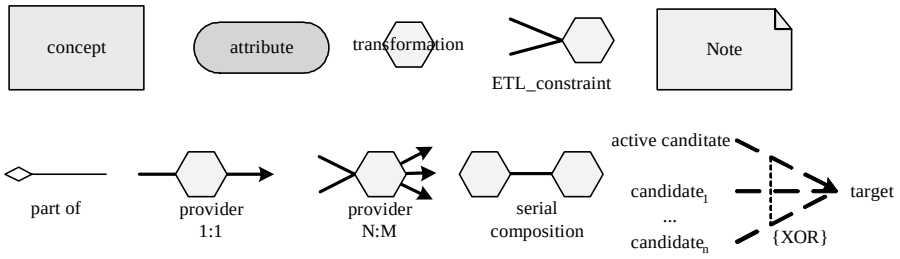


Fig. 3.1 Notation for the conceptual modeling of ETL activities

In Fig. 3.1 we graphically depict the different entities of the proposed model. We do not employ standard UML notation for concepts and attributes, for the simple reason that we need to treat attributes as first class citizens of our model. Thus, we do not embed attributes in the definition of their encompassing entity, like for example, a UML class or a relational table. We try to be orthogonal to the conceptual models which are available for the modeling of data warehouse star schemata; in fact, any of the proposals for the data warehouse front end can be combined with our approach, which is specifically tailored for the back end of the warehouse.

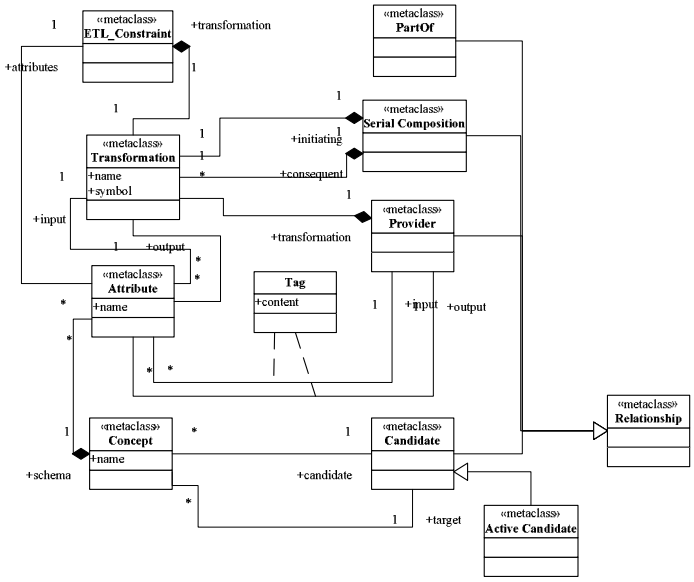


Fig. 3.2 The proposed metamodel as a UML diagram

In Fig. 3.2 we depict the basic entities of the proposed metamodel in a UML diagram. All the constructs of our conceptual model which are introduced in the rest of this section refer to the entities of Fig. 3.2.

Database	Concept	Attributes
S_1	S_1.PARTSUPP	PKEY, SUPPKEY, QTY, COST
S_2	S_2.PARTSUPP	PKEY, SUPPKEY, DEPARTMENT, DATE, QTY, COST
DW	DW.PARTSUPP	PKEY, SUPPKEY, DATE, QTY, COST

Fig. 3.3 The schemata of the source databases and of the data warehouse

To motivate the discussion we will introduce an example involving two source databases S_1 and S_2 as well as a central data warehouse DW. The available concepts for these databases are listed in Fig. 3.3, along with their attributes. The scenario involves the propagation of data from the concept PARTSUPP of source S_1 as well as from the concept PARTSUPP of source S_2 to the data warehouse. Table DW.PARTSUPP stores daily(DATE) information for the available quantity(QTY) and cost(COST) of parts(PKEY) per supplier(SUPPKEY). We assume that the first supplier is European and the second is American, thus the data coming from the second source need to be converted to European values and formats.

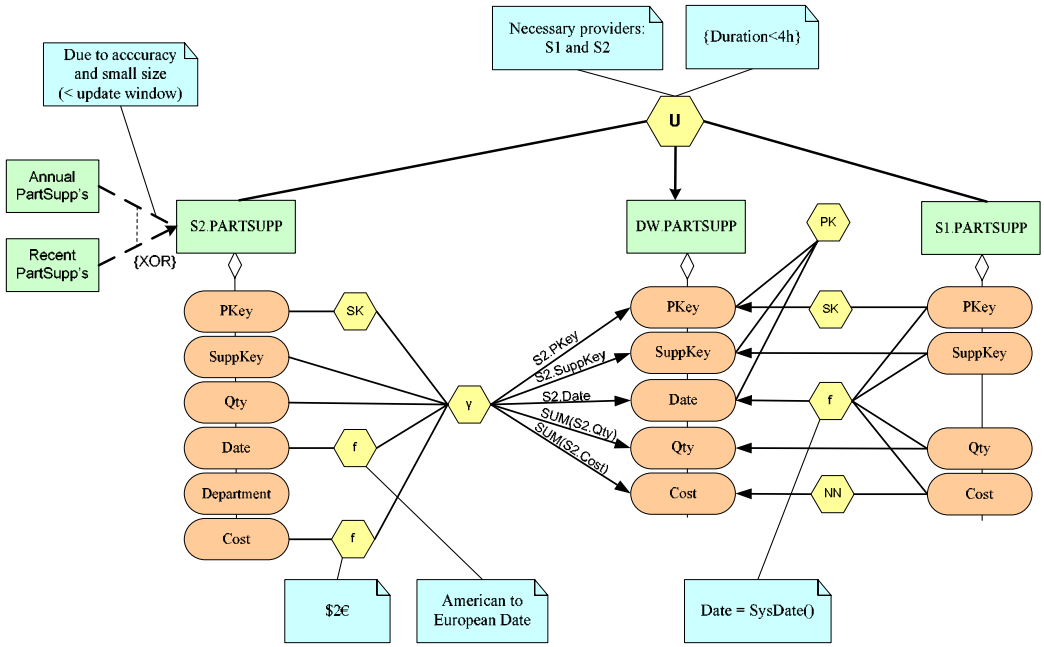


Fig. 3.4 The diagram of the conceptual model for our motivating example

In Fig. 3.4 we depict the full fledged diagram of our motivating example. In the rest of this section, we will explain each part of it.

3.1 Concepts and Attributes

Attributes. A granular module of information. The role of attributes is the same as in the standard ER/dimensional models. As in standard ER modeling, attributes are depicted with oval shapes.

Concepts. A concept represents an entity in the source databases or in the data warehouse. Concept instances are the files in the source databases, the data warehouse fact and dimension tables and so on. A concept is formally defined by a name and a finite set of attributes. In terms of the ER model, a concept is a generalization of entities and relationships; depending on the employed model (dimensional model or ER extension) all entities composed of a set of attributes are generally instances of class `Concept`.

As mentioned in [VQVJ01], we can treat several *physical* storage structures as finite lists of fields, including relational databases, COBOL or simple ASCII files, multidimensional cubes and dimensions. Concepts are fully capable of modeling this kind of structures, possibly through a generalization (ISA) mechanism. Let us take OLAP structures as an example. We should note that the interdependencies of levels and values, which are the core of all the approaches mentioned in Section 2, are not relevant for the case of ETL; thus, employing simply concepts is sufficient for the problem of ETL modeling. Still, we can refine the generic `Concept` structure to subclasses pertaining to the characteristics of any of the aforementioned approaches (e.g., classes `Fact Table` and `Dimension`), achieving thus a homogeneous way to treat OLAP and ETL modeling.

In our motivating example one can observe several concepts. The concepts `S1.PARTSUPP`, `S2.PARTSUPP` and `DW.PARTSUPP` are depicted in Fig. 3.4., along with their respective attributes.

3.2 Transformations, Constraints and Notes

Transformations. In our framework, transformations are abstractions that represent parts, or full modules of code, executing a single task. Two large categories of transformations include (a) filtering or data cleaning operations, like the check for primary or foreign key violations and (b) transformation operations, during which the schema of the incoming data is transformed (e.g., aggregation). Formally, a transformation is defined by (a) a finite set of

input attributes; (b) a finite set of output attributes and (c) a symbol that graphically characterizes the nature of the transformation. A transformation is graphically depicted as a hexagon tagged with its corresponding symbol.

In our motivating example of Fig. 3.4, one can observe several transformations. Note the ones pertinent to the mapping of `S1.PARTSUPP` to `DW.PARTSUPP`. One can observe a surrogate key assignment transformation (`SK`), a function application calculating the system date (`f`) and a Not Null (`NN`) check for attribute `Cost`. We will elaborate more on the functionality of transformations once provider relationships (that actually employ them) are introduced.

ETL Constraints. There are several occasions when the designer wants to express the fact that the data of a certain concept fulfill several requirements. For example, the designer might wish to impose a primary key or null value constraint over a (set of) attribute(s). This is achieved through the application of ETL constraints, which are formally defined as follows: (a) a finite set of attributes, over which the constraint is imposed and (b) a single transformation, which implements the enforcement of the constraint. Note, that despite the similarity in the name, ETL constraints are different modeling elements from the well known UML constraints. An ETL constraint is graphically depicted as a set of solid edges starting from the involved attributes and targeting the facilitator transformation. In our motivating example, observe that we apply a `Primary Key` ETL constraint to `DW.PARTSUPP` for the attributes `PKey`, `SuppKey`, `Date`.

Notes. Exactly as in UML modeling, notes are informal tags to capture extra comments that the designer wishes to make during the design phase or render UML constraints attached to an element or set of elements [4]. As in UML, notes are depicted as rectangles with a dog-eared corner. In our framework, notes are used for:

- Simple comments explaining design decisions.
- Explanations of the semantics of the applied transformations. For example, in the case of relational selections/joins this involves the specification of the respective selection/join condition, whereas in the case of functions this would involve the specification of the function signatures.

– Tracing of runtime constraints that range over different aspects of the ETL process, such as the time/event based scheduling, monitoring, logging, error handling, crash recovery etc.

For example, in the upper part of Fig. 5 we can observe a runtime constraint specifying that the overall execution time for the loading of `DW.PARTSUPP` (that involves the loading of `S1.PARTSUPP` and `S2.PARTSUPP`) cannot take longer than 4 hours.

3.3 Part-Of and Candidate Relationships

Part-of Relationships. We bring up part-of relationships in order to emphasize the fact that a concept is composed of a set of attributes. In general, standard ER modeling does not treat this kind of relationship as a first-class citizen of the model; UML modeling on the other hand, hides attributes inside classes and treats part-of relationships with a much broader meaning. We do not wish to redefine UML part-of relationships, but rather to emphasize the relationship of a concept with its attributes (since we need attributes as first class citizens in the inter-attribute mappings). Naturally, we do not preclude the usage of the part-of relationship for other purposes, as in standard UML modeling. As usually, a part-of relationship is denoted by an edge with a small diamond at the side of the container object.

Candidate relationships. In the case of data warehousing, it is most common a phenomenon, especially in the early stages of the project, to have more than one candidate source files/tables that could populate a target, data warehouse table. Thus, a set of candidate relationships captures the fact that a certain data warehouse concept can be populated by more than one candidate source concepts. Formally, a candidate relationship comprises (a) a single *candidate* concept and (b) a single *target* concept. Candidate relationships are depicted with bold dotted lines between the candidates and the target concepts. Whenever exactly one of them can be selected, we annotate the set of candidate relationships for the same concept with a UML `{XOR}` constraint.

Active candidate relationships. An active candidate relationship denotes the fact that out of a set of candidates, a certain one has been selected for the population of the target concept. Thus, an active candidate relationship is a specialization of candidate relationships, with the

same structure and refined semantics. We denote an active candidate relationship with a directed bold dotted arrow from the provider towards the target concept.

For the purpose of our motivating example, let us assume that source S_2 has more than one production systems (e.g., COBOL files), which are candidates for S2.PARTSUPP. Assume that the available candidates (depicted in the left upper part of Fig. 3.4) are:

– A concept AnnualPartSupp's (practically representing a file F1), that contains the full annual history about part suppliers; it is used basically for reporting purposes and contains a superset of fields than the ones required for the purpose of the data warehouse.
– A concept RecentPartSupp's (practically representing a file F2), containing only the data of the last month; it used on-line by end-users for the insertion or update of data as well as for some reporting applications. The diagram also shows that RecentPartSupp's was eventually selected as the active candidate; a note captures the details of this design choice.

3.4 Provider Relationships and Serial Composition of Transformations

Provider relationships. A provider relationship maps a set of input attributes to a set of output attributes through a relevant transformation. In the simple 1:1 case, provider relationships capture the fact that an input attribute in the source side populates an output attribute in the data warehouse side. If the attributes are semantically and physically compatible, no transformation is required. If this is not the case though, we pass this mapping through the appropriate transformation (e.g., European to American data format, not null check, etc.).

In general, it is possible that some form of schema restructuring takes place; thus, the formal definition of provider relationships comprises (a) a finite set of input attributes; (b) a finite set of output attributes; (c) an appropriate transformation (i.e., one whose input and output attributes can be mapped one to one to the respective attributes of the relationship). In the case of N:M relationships the graphical representation obscures the linkage between provider and target attributes. To compensate for this shortcoming, we annotate the link of a provider relationship with each of the involved attributes with a tag, so that there is no disambiguity for the actual provider of a target attribute.

In the 1:1 case, a provider relationship is depicted with a solid bold directed arrow from the input towards the output attribute, tagged with the participating transformation. In the general N:M case, a provider relationship is graphically depicted as a set of solid arrows

starting from the providers and targeting the consumers, all passing through the facilitator transformation.

Finally, we should also mention a syntactic sugar add-on of our model. It can be the case where a certain provider relationship involves all the attributes of a set of concepts. For example, in the case of a relational union operation, all the attributes of the input and the output concepts would participate in the transformation. To avoid overloading the diagram with too many relationships, we employ a syntactic sugar notation mapping the input to the output concepts (instead of attributes). This can also be treated as a zoom in/out operation on the diagram per se: at the coarse level, only concepts are depicted and an overview of the model is given; at the detailed level, the inter-concept relationships are expanded to the respective inter-attribute relationships, presenting the designer with all the available detail.

Let us examine now, the relationship between the attributes of concepts `S1.PARTSUPP`, `S2.PARTSUPP` and `DW.PARTSUPP`, as depicted in Fig. 3.4. For starters, we will ignore the aggregation that takes place for the rows of source `S2` and focus on the elementary transformations.

– Attribute `PKey` is directly populated from its homonymous attribute in `S1` and `S2`, through a surrogate key (`SK`) transformation. *Surrogate Key* assignment is common tactics in data warehousing, employed in order to replace the keys of the production systems with a uniform key[1]. For example, it could be the case that the part '`Steering Wheel`' has `PKEY=30` for source `S1`, `PKEY=40` for source `S2`, while at the same time source `S2` has `PKEY=30` for part '`Automobile Door`'. These conflicts can be easily resolved by a global replacement mechanism through the assignment of a uniform surrogate key.

– Attribute `SuppKey` is populated from the homonymous attributes in the sources.

– Attribute `Date` is directly populated from its homonymous attribute in `S2`, through an `American-To-European Date` transformation function. At the same time, the date for the rows coming from source `S1`, is determined through the application of a `Sysdate()` function (since `S1.PARTSUPP` does not have a corresponding attribute). Observe the

[1] In general, the basic reasons for a surrogate key transformation are both performance and semantic homogeneity. Textual attributes are not the best candidates for indexed keys and thus need to be replaced by integer keys. At the same time, different production systems might use different keys for the same object, or the same key for different objects, resulting in the need for a global replacement of these values in the data warehouse.

function applied for the rows coming from source S1: it uses as input all the attributes of S1.PARTSUPP (in order to determine that the produced value is a new attribute of the row), passes through a function application transformation calculating the system date, which, in turns, is directed to attribute DW.Date.

– Attribute Qty is directly populated from its homonymous attributes in the two sources, without the need for any transformation.
– Attribute Cost is populated from its homonymous attributes in the two sources. As far as source S2 is concerned, we need a $2€ transformation in order to convert the cost to European values. As far as source S1 is concerned, we apply a Not Null (NN) check, to avoid loading the data warehouse with rows having no cost for their parts.

Note also that there can be input attributes, like for example S2.PARTSUPP.Department, which are ignored during the ETL process.

Transformation Serial Composition. It is possible that we need to combine several transformations in a single provider relationship. For example, we would possibly like to group incoming data with respect to a set of attributes, having ensured at the same time that no null values are involved in this operation. In this case, we would need to perform a not null check for each of the attributes and propagate only the correct rows to the aggregation. In order to model this setting, we employ a *serial composition of the involved transformations*. The problem would be the requirement that a transformation has a set of attributes as inputs and attributes; thus, simply connecting two transformations would be inconsistent. To compensate this shortcoming, we employ a serial transformation composition. Formally, a serial transformation composition comprises (a) a single initiating transformation and (b) a single subsequent transformation. Serial transformation compositions are depicted as solid bold lines connecting the two involved transformations.

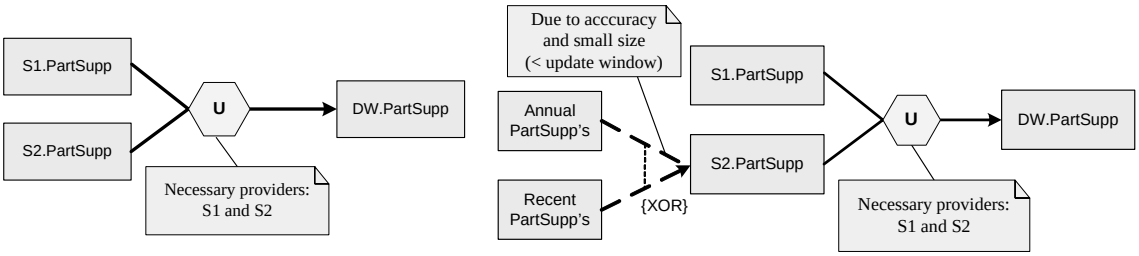
A rather complex part of the motivating example is the aggregation that takes place for the rows of source S2. Practically, source S2 captures information for part suppliers according to the particular department of the supplier organization. Loading this data to the data warehouse that ignores this kind of detail requires the aggregation of data per PKey, SuppKey, Date and the summation of Cost and Qty. This is performed from the aggregation transformation γ. Still, all the aforementioned elementary transformations are not ignored or banished; on the

contrary, they are linked to the aggregation transformation through the appropriate serial composition relationships. Note also the tags on the output of the aggregation transformation, determining their providers (e.g., `S2.PARTSUPP.PKey` for `DW.PARTSUPP.PKey` and `SUM[S2.PARTSUPP.Qty]` for `DW.PARTSUPP.Qty`).

4. Methodology for the usage of the conceptual model

In this section, we will give an overview of the assumed sequence of steps that a designer follows, during the construction of the data warehouse. Each step of this methodology will be presented in terms of our motivating example. As already mentioned, the ultimate goal of this design process is the production of inter-attribute mappings, along with any relevant auxiliary information.

Step 1: Identification of the proper data stores. The first thing that a designer faces during the requirements and analysis period of the data warehouse process is the identification of relative data sources. Assume that for a particular subset of the data warehouse, we have identified the concept `DW.PARTSUPP` which is a fact table of how parts where distributed according to their respective suppliers. Assume also that we have decided that for completeness reasons, we need to populate the target table with data from both sources S_1 and S_2, i.e., we need the union of the data of these two sources. Then, the diagram for the identified data sources and their interrelationships is depicted in Fig. 4.1a.



(a) (b)

Fig. 4.1 (a) Identification of the proper data stores; (b) Candidates and active candidates for the motivating example

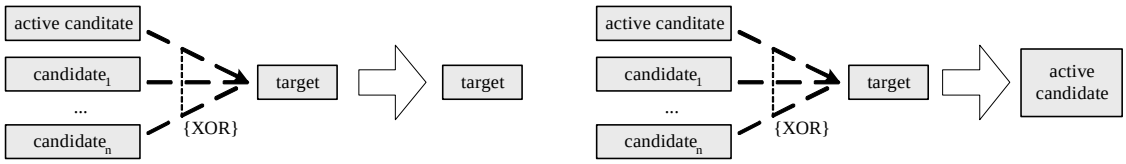
Step 2: Candidates and active candidates for the involved data stores. As already mentioned before, during the requirements and analysis stage, the designer can possibly find out that more than one data stores could be candidates to populate a certain concept. The way the decision is taken is outside the scope of this paper (although our personal experience suggests that, at least the size, the data quality and the availability of the sources play a major role in this kind of decision). For the purpose of our motivating example, let us assume that source S_2 has more than one production systems (e.g., COBOL files), which are candidates for concept `S2.PARTSUPP`. Assume that the available sources are:

– A concept `AnnualPartSupp's` (practically representing a file `F1`), that contains the full annual history about part suppliers; it is used basically for reporting purposes and contains a superset of fields than the ones required for the purpose of the data warehouse.

– A concept `RecentPartSupp's` (practically representing a file `F2`), containing only the data of the last month; it used on-line by end-users for the insertion or update of data as well as for some reporting applications.

The candidate concepts for concept `S2.PartSupp` and the (eventually selected) active candidate `RecentPartSupp's`, along with the tracing of the decision for the choice, are depicted in Fig. 4.1b.

Clearly, once a decision on the active candidate has been reached, a simplified 'working copy' of the scenario that eliminates all other candidates can also be produced. As shown in Fig.4.2, there are two ways to achieve this (a) by ignoring all the information on candidates for the target concept and (b) by replacing the target with the active candidate.



(a) (b)

Fig. 4.2 Working copy transformation (a) ignoring candidates; (b) replace targets with active candidates.

Step 3: Attribute mapping between the providers and the consumers. The most difficult task of the data warehouse designer is to determine the mapping of the attributes of the sources to the ones of the data warehouse. This task involves several discussions with the source administrators to explain the codes and implicit rules or values, which are hidden in the data and the source programs. Moreover, it involves quite a few 'data preview' attempts (in the form of sampling, or simple counting queries) to discover the possible problems of the provided data. Naturally, this process is interactive and error-prone. The support that a tool can provide the designer with, lies mainly in the area of inter-attribute mapping, with focus on the tracing of transformation and cleaning operations for this mapping.

For each target attribute, a set of provider relationships must be defined. In simple cases, the provider relationships are defined directly among source and target attributes. In the cases where a transformation is required, we pass the mapping through the appropriate transformation. In the cases where more than one transformations are required, we can insert a sequence of transformations between the involved attributes, all linked through composition relationships. ETL constraints are also specified in this part of the design process. An example of inter-attribute relationship for the concepts `S1.PartSupp`, `S2.PartSupp` and `DW.PartSupp`, is depicted in Fig. 4.3.

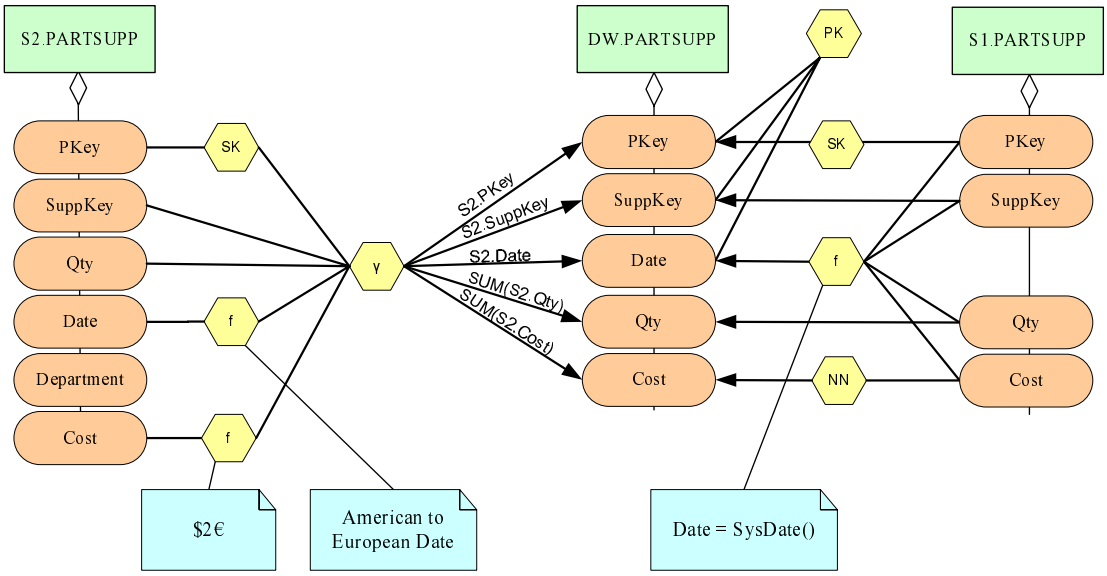


Fig. 4.3 Attribute mapping for the population of recordset `DW.PartSupp`

Step 4: Annotating the diagram with runtime constraints. Apart for the job definition for an ETL scenario, which specifies how the mapping from sources to the data warehouse is performed, along with the appropriate transformations, there are several other parameters that possibly need to be specified for the runtime environment. This kind of *runtime constraints* include:

- Time/Event based scheduling. The designer needs to determine the frequency of the ETL process, so that data are fresh and the overall process fits within the refreshment time window.
- Monitoring. On-line information about the progress/status of the process is necessary, so that the administrator can be aware of what step the load is on, its start time, duration, etc. File dumps, notification messages on the console, e-mail, printed pages, or visual demonstration can be employed for this purpose.
- Logging. Off-line information, presented at the end of the execution of the scenario, on the outcome of the overall process. All the relevant information (e.g., the previous tracing stuff) is shown, by employing the aforementioned techniques.
- Exception handling. The row-wise treatment of database/business rules violation is necessary for the proper function of the ETL process. Information like which rows are problematic, or how many rows are acceptable (acceptance rate, that is), characterizes this aspect of the ETL process.
- Error handling. Crash recovery and stop/start capabilities (e.g., committing a transaction every few rows) are absolutely necessary for the robustness and the efficiency of the process.

To capture all this information, we adopt *notes*, tagged to the appropriate concepts, transformations or relationships. In Fig. 4.4 we depict the diagram of Fig. 4.2b, annotated with a runtime constraint specifying that the overall execution time for the loading of `DW.PARTSUPP` (that involves the loading of `S1.PARTSUPP` and `S2.PARTSUPP`) cannot take longer than 4 hours.

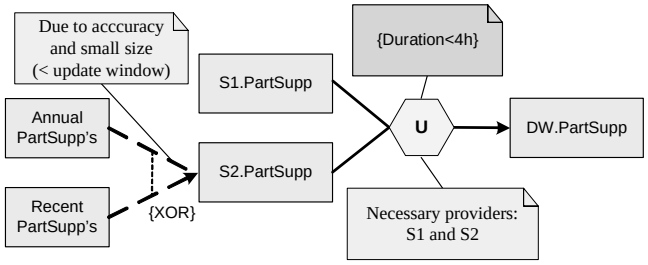


Fig. 4.4 Annotation of Fig. 4.2(b) with constraints on the runtime execution of the ETL scenario

5. Instantiation and Specialization Layers

We believe that the key issue in the conceptual representation of ETL activities lies (a) on the identification of a small set of *generic constructs* that are powerful enough to capture all cases and (b) on an extensibility mechanism that allows the construction of a 'palette' of *frequently used types* (e.g., for data stores and activities).

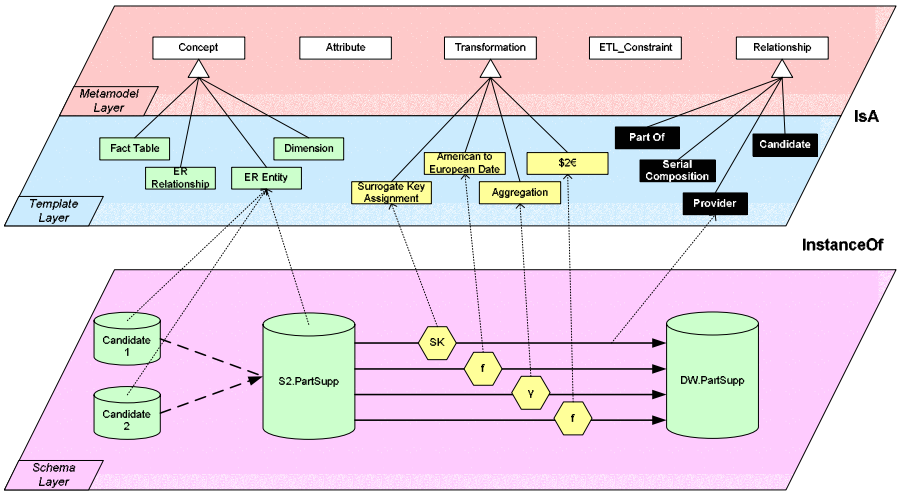


Fig. 5.1 The framework for the modeling of ETL activities

Our metamodeling framework is depicted in Fig. 5.1. The lower layer of Fig. 5.1, namely *Schema Layer*, involves a specific ETL scenario. All the entities of the Schema layer are *instances* of the classes `Concept`, `Attribute`, `Transformation`, `ETL Constraint` and `Relationship`. Thus, as one can see on the upper part of Fig. 5.1, we introduce a meta-class layer, namely *Metamodel Layer* involving the aforementioned classes. The linkage between the Metamodel and the Schema layers is achieved through instantiation ("`instanceOf`") relationships. The Metamodel layer implements the aforementioned genericity desideratum:

the five classes which are involved in the Metamodel layer are generic enough to model any ETL scenario, through the appropriate instantiation.

Still, we can do better than the simple provision of a meta- and an instance layer. In order to make our metamodel truly useful for practical cases of ETL processes, we enrich it with a set of ETL-specific constructs, which constitute a subset of the larger metamodel layer, namely the *Template Layer*. The constructs in the Template layer are also meta-classes, but they are quite customized for the regular cases of ETL processes. Thus, the classes of the Template layer as specializations (i.e., subclasses) of the generic classes of the Metamodel layer (depicted as "`IsA`" relationships in Fig. 5.1). Through this customization mechanism, the designer can pick the instances of the Schema layer from a much richer palette of constructs; in this setting, the entities of the Schema layer are instantiations, not only of the respective classes of the Metamodel layer, but also of their subclasses in the Template layer.

In the example of Fig. 5.1 the concept `DW.PARTSUPP` must be populated from a certain source S_2. Several operations must intervene during the propagation: for example, a surrogate key assignment and an aggregation take place in the scenario. Moreover, there are two candidates suitable for the concept S_2`.PARTSUPP`; out of them, exactly one (`Candidate`$_2$) is eventually selected for the task. As one can observe, the concepts that take part in this scenario are instances of class `Concept` (belonging to the metamodel layer) and specifically of its subclass `ER Entity` (assuming that we adopt an ER model extension). Instances and encompassing classes are related through links of type `instanceOf`. The same mechanism applies to all the transformations of the scenario, which are (a) instances of class `Transformation` and (b) instances of one of its subclasses, depicted in Fig. 5.1. Relationships do not escape the rule either: observe how the provider links from the concept `S2.PARTSUPP` towards the concept `DW.PARTSUPP` are related to class `Provider Relationship` through the appropriate `instanceOf` links.

As far as the class `Concept` is concerned, in the Template layer we can specialize it to several subclasses, depending on the employed model. In the case of the ER model, we have the subclasses `ER Entity`, and `ER Relationship`, whereas in the case of the Dimensional Model, we can have subclasses as `Fact Table` or `Dimension`.

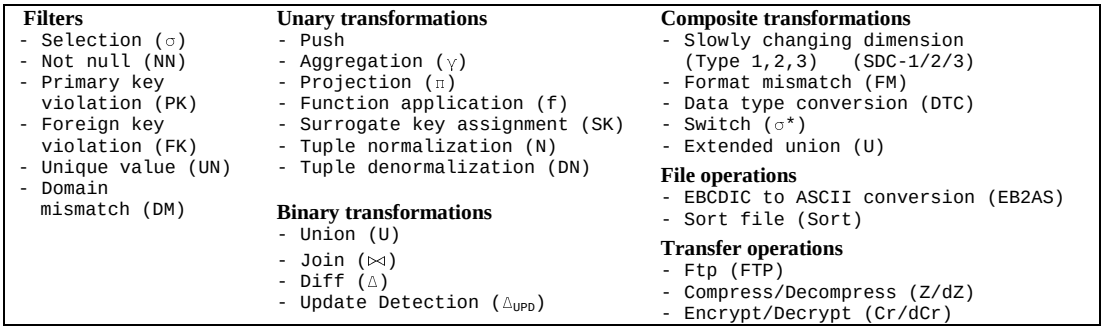


Fig. 5.2 Template transformations, along with their symbols, grouped by category

Following the same framework, class `Transformation` is further specialized to an extensible set of reoccurring patterns of ETL activities, depicted in Fig. 5.2. We now present each of the aforementioned classes in more detail. As one can see on the top side of Fig. 5.2, we group the template activities in six major logical groups. We do not depict the grouping of activities in subclasses in Fig. 5.1, in order to avoid overloading the figure; instead, we depict the specialization of class `Transformation` to three of its subclasses whose instances appear in the employed scenario of the Schema layer.

The first group, named *Filters*, provides checks for the respect of a certain condition. The semantics of these filters are the obvious (starting from a generic `selection` condition and proceeding to the check for `null values`, `primary` or `foreign key violation`, etc.). The second group of template activities is called *Unary Transformations* and except for the most generic `push` activity (which simply propagates data from the provider to the consumer), consists of the classical `aggregation` and `function application` operations along with three data warehouse specific transformations (`surrogate key assignment`, `normalization` and `denormalization`)[2]. The third group consists of classical *Binary Operations*, such as

[2] *Normalization/Denormalization*. It is common for source data to be long denormalized records, possibly involving more than a hundred attributes. This is due to the fact that bad database design or classical COBOL tactics led to the gathering of all the necessary information for an application in a single table/file. Although a data warehouse is also characterized by a carefully selected set of denormalized structures, sometimes it is imperative to normalize somehow the input data. Consider, for example, a table of the form R(**KEY**,TAX,DISCOUNT,PRICE), which we would like to transform to a table of the form R′(**KEY**,CODE,AMOUNT). For example, the input tuple t[**key**,30,60,70] is transformed into the tuples t1[**key**,1,30]; t2[**key**,2,60]; t3[**key**,3,70]. The operator NORMALIZATION performs this kind of transformation. DENORMALIZATION, at the same time is the inverse operation, particularly useful in the case of highly normalized production systems that act as sources. In this situation, we would like to denormalize the source data (e.g., consider the inverse of the aforementioned example) in order to achieve better performance in the answering of aggregate queries in the data warehouse.

union, join and difference of concepts as well as with a special case of difference involving the detection of updates.

Moreover, there is a set of *Composite/Special-Purpose* transformations, which can be derived from simple combinations of template activities. For example, slowly changing dimension is a technique for extracting the changes in the sources that populate dimension tables. Also, format mismatch is a special case of selection and function application (in the case when an attribute should fulfill a certain regular expression), data type conversion is also a special case of function application, a switch is the combination of several selections and extended union is a combination of binary unions to produce the respective n-ary operator.

Except for the aforementioned template activities, which mainly refer to logical transformations, we can also consider the case of physical operators that refer to the application of physical transformations to whole files/tables. Mainly, we discuss inter-concept physical operations like *Transfer Operations* (ftp, compress/decompress, encrypt/decrypt) and *File Operations* (EBCDIC to ASCII, sort file).

Summarizing, the Metamodel layer is a set of generic entities, able to represent any ETL scenario. At the same time, the genericity of the Metamodel layer is complemented with the extensibility of the Template layer, which is a set of "built-in" specializations of the entities of the Template layer, specifically tailored for the most frequent elements of ETL scenarios. Moreover, apart from this "built-in", ETL-specific extension of the generic metamodel, if the designer decides that several 'patterns', not included in the palette of the Template layer, occur repeatedly in his data warehousing projects, he/she can easily fit them into the customizable Template layer through a specialization mechanism.

6. Conclusions

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. In this paper, we have focused on the problem of the definition of ETL activities and provided foundations for their conceptual representation. More specifically, we

have proposed a novel conceptual model, which is customized for the tracing of inter-attribute relationships and the respective ETL activities in the early stages of a data warehouse project. The proposed model is constructed in a customizable and extensible manner, so that the designer can enrich it with his own re-occurring patterns for ETL activities, while, at the same time, we also offer a 'palette' of a set of frequently used ETL activities, like the assignment of surrogate keys, the check for null values, etc.

As far as future work is concerned, the first objective is the linkage of the proposed conceptual model to its logical and physical counterparts, with particular focus (a) on the relationship of the ETL activities to the underlying data stores; (b) the capturing of the composite workflow of the ETL scenario and (c) the optimization of its execution [VaSS02].

Acknowledgments

This research has been partially funded by the European Union's Information Society Technologies Programme (IST) under project EDITH (IST-1999-20722).

References

[Arde01] Ardent Software. DataStage Suite. Available at http://www.ardentsoftware.com/

[BaFM99] M. Bouzeghoub, F. Fabret, M. Matulovic. Modeling Data Warehouse Refreshment Process as a Workflow Application. In Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, (1999).

[BoDS00] V. Borkar, K. Deshmuk, S. Sarawagi. Automatically Extracting Structure from Free Text Addresses. Bulletin of the Technical Committee on Data Engineering, **23**(4), (2000).

[BoJR98] G. Booch, I. Jacobson, J. Rumbaugh. The Unified Modeling Language User Guide. Addison-Wesley Pub Co; ISBN: 0201571684; 1st edition (October 30, 1998).

[CDL+98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Information integration: Conceptual modeling and reasoning support. In Proc. Proceedings of the International Conference on Cooperative Information Systems (COOPIS), New York, USA, pp. 280-291, (August 1998)

[CDL+99] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. A principled approach to data integration and reconciliation in data warehousing. In Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, (1999).

[Data01] DataMirror Corporation. Transformation Server. Available at http://www.datamirror.com

[Dema97] M. Demarest. The politics of data warehousing. Available at http://www.hevanet.com/demarest/marc/dwpol.html (1997).

[ETI01] Evolutionary Technologies Intl. ETI*EXTRACT. Available at http://www.eti.com/

[GFSS00] H. Galhardas, D. Florescu, D. Shasha and E. Simon. Ajax: An Extensible Data Cleaning Tool. In Proc. ACM SIGMOD Intl. Conf. On the Management of Data, pp. 590, Dallas, Texas, (2000).

[GoMR98] M. Golfarelli, D. Maio, S. Rizzi. The Dimensional Fact Model: a Conceptual Model for Data Warehouses. Invited Paper, International Journal of Cooperative Information Systems, vol. 7, n. 2&3, 1998.

[GoRi98] M. Golfarelli, S. Rizzi: Methodological Framework for Data Warehouse Design. In ACM First International Workshop on Data Warehousing and OLAP (DOLAP '98), pp. 3-9, November 1998, Bethesda, Maryland, USA.

[HuLV00] B. Husemann, J. Lechtenborger, G. Vossen. Conceptual data warehouse modeling. In Proc. 2nd Intl. Workshop on Design and Management of Data Warehouses (DMDW), pp. 6.1 –6.11, Stockholm, Sweden (2000).

[Inmo97] B. Inmon. The Data Warehouse Budget. DM Review Magazine, January 1997. Available at http://www.dmreview.com/master.cfm?NavID=55&EdID=1315

[JeQJ98] M.A. Jeusfeld, C. Quix, M. Jarke: Design and Analysis of Quality Information for Data Warehouses. In Proc. of the 17th Intl. Conf. On Conceptual Modeling (ER'98), pp. 349-362, Singapore (1998).

[JJQV99] M. Jarke, M.A. Jeusfeld, C. Quix, P. Vassiliadis: Architecture and quality in data warehouses: An extended repository approach. Information Systems, **24**(3): 229-253 (1999). A previous version appeared in Proc. 10th Conf. of Advanced Information Systems Engineering (CAiSE '98), Pisa, Italy (1998).

[JLVV00] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis (eds.). Fundamentals of Data Warehouses. Springer, (2000).

[Kimb97] R. Kimball. A Dimensional Modeling Manifesto. DBMS Magazine. August 1997.

[KRRT98] R. Kimbal, L. Reeves, M. Ross, W. Thornthwaite. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses. John Wiley & Sons, February 1998.

[LWGG00] W. Labio, J.L. Wiener, H. Garcia-Molina, V. Gorelik. Efficient Resumption of Interrupted Warehouse Loads. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000), pp. 46-57, Dallas, Texas, USA (2000).

[Micr01] Microsoft Corp. MS Data Transformation Services. Available at www.microsoft.com/sq

[MoKo00] D.L. Moody, M.A.R. Kortink: From enterprise models to dimensional models: a methodology for data warehouse and data mart design. Proceedings of the Second Intl. Workshop on Design and Management of Data Warehouses, DMDW 2000, Stockholm, Sweden, June 5-6, 2000. Available at http: //sunsite.informatik.rwthaachen.de/Publications/ CEUR-WS/Vol-28/

[Mong00] A. Monge. Matching Algorithms Within a Duplicate Detection System. Bulletin of the Technical Committee on Data Engineering, **23**(4), (2000).

[NgTW00] T. B. Nguyen, A Min Tjoa, R. R. Wagner. An Object Oriented Multidimensional Data Model for OLAP. Proc. of the First International Conference on Web-Age Information Management (WAIM-00), Shanghai, China, June 2000.

[Orac01] Oracle Corp. Oracle9i™ Warehouse Builder User's Guide, Release 9.0.2. November 2001.

[RaHa00] E. Rahm, H. Do. Data Cleaning: Problems and Current Approaches. Bulletin of the Technical Committee on Data Engineering, **23**(4), (2000).

[RaHe01] V. Raman, J. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In Proceedings of 27th International Conference on Very Large Data Bases (VLDB), pp. 381-390, Roma, Italy, (2001).

[SBHD98] C. Sapia, M. Blaschka, G. Höfling, B. Dinter: Extending the E/R Model for the Multidimensional Paradigm. In ER Workshops 1998, pp. 105-116. Lecture Notes in Computer Science 1552 Springer 1999

[ShTy98] C. Shilakes, J. Tylman. Enterprise Information Portals. Enterprise Software Team. Available at http://www.sagemaker.com/company/downloads/eip/indepth.pdf (1998).

[TrBC99] N. Tryfona, F. Busborg, J.G.B. Christiansen. starER: A Conceptual Model for Data Warehouse Design. In ACM Second International Workshop on Data Warehousing and OLAP (DOLAP '99), pp. 3-8, November 1999, Kansas City, Missouri, USA.

[TrPG00] J.C. Trujillo, M. Palomar, J. Gómez: Applying Object-Oriented Conceptual Modeling Techniques to the Design of Multidimensional Databases and OLAP Applications. Proc. of the First International Conference on Web-Age Information Management (WAIM-00) pp. 83-94, Shanghai, China, June 2000.

[Tsoi01] A. Tsois. MAC: Conceptual data modeling for OLAP. In Proc. 3rd Intl. Workshop on Design and Management of Data Warehouses (DMDW), pp. 5.1–5.11, Interlaken, Switzerland (2001).

[Vass00] P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In Proc. 2nd Intl. Workshop on Design and Management of Data Warehouses (DMDW), pp. 12.1 –12.16, Stockholm, Sweden (2000).

[VaSS02] P. Vassiliadis, A. Simitsis, S. Skiadopoulos. Modeling ETL activities as graphs. In Proc. of Design and Management of Data Warehouses (DMDW'2002) 4th International Workshop in conjunction with CAiSE'02, pp. 52-61, Toronto, Canada, May 27, 2002.

[VQVJ01] P. Vassiliadis, C. Quix, Y. Vassiliou, M. Jarke. Data Warehouse Process Management. Information Systems, vol. 26, no.3, pp. 205-236, June 2001.

[VVS+01] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, T. Sellis. ARKTOS: Towards the modeling, design, control and execution of ETL processes. Information Systems, **26**(8), pp. 537-561, December 2001, Elsevier Science Ltd.