International Journal of Cooperative Information Systems
Vol. 10, No. 3 (2001) 237–271
© World Scientific Publishing Company

DATA INTEGRATION IN DATA WAREHOUSING

DIEGO CALVANESE*, GIUSEPPE DE GIACOMO†, MAURIZIO LENZERINI‡, DANIELE NARDI§ and RICCARDO ROSATI¶

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy

Information integration is one of the most important aspects of a Data Warehouse. When data passes from the sources of the application-oriented operational environment to the Data Warehouse, possible inconsistencies and redundancies should be resolved, so that the warehouse is able to provide an integrated and reconciled view of data of the organization. We describe a novel approach to data integration in Data Warehousing. Our approach is based on a conceptual representation of the Data Warehouse application domain, and follows the so-called local-as-view paradigm: both source and Data Warehouse relations are defined as views over the conceptual model. We propose a technique for declaratively specifying suitable reconciliation correspondences to be used in order to solve conflicts among data in different sources. The main goal of the method is to support the design of mediators that materialize the data in the Data Warehouse relations. Starting from the specification of one such relation as a query over the conceptual model, a rewriting algorithm reformulates the query in terms of both the source relations and the reconciliation correspondences, thus obtaining a correct specification of how to load the data in the materialized view.

Keywords: Data Warehousing, data integration, data reconciliation, local-as-view approach, query rewriting, automated reasoning.

1. Introduction

A Data Warehouse is a set of materialized views over the operational information sources of an organization, designed to provide support for data analysis and management's decisions. Information integration is at the heart of Data Warehousing.[1] When data passes from the application-oriented operational environment to the Data Warehouse, possible inconsistencies and redundancies should be resolved, so that the Warehouse is able to provide an integrated and reconciled view of data of the organization.

Generally speaking, information integration is the problem of acquiring data from a set of sources that are available for the application of interest.[2] This problem

*E-mail: Calvanese@dis.uniroma1.it
†E-mail: Giacomo@dis.uniroma1.it
‡E-mail: Lenzerini@dis.uniroma1.it
§E-mail: Nardi@dis.uniroma1.it
¶E-mail: Rosati@dis.uniroma1.it

has recently become a central issue in several contexts, including Data Warehousing, Interoperable and Cooperative Systems, Multi-database systems, and Web Information Systems. The typical architecture of an integration system is described in terms of two types of modules: wrappers and mediators.[3,4] The goal of a wrapper is to access a source, extract the relevant data, and present such data in a specified format. The role of a mediator is to collect, clean, and combine data produced by different wrappers (or mediators), so as to meet a specific information need of the integration system. The specification and the realization of mediators is the core problem in the design of an integration system.

The constraints that are typical of Data Warehouse applications restrict the large spectrum of approaches that have been proposed for integration.[1,5,6] First, while the sources are often external to the organization managing the integration system, in a Data Warehouse they are mostly internal to the organization. Second, a Data Warehouse should reflect the informational needs of the organization, and should therefore be defined in terms of a global, corporate view of data. Without such a global view, there is the risk of concentrating too much on what is in the sources at the operational level, rather than on what is really needed in order to perform the required analysis on data.[7] Third, such a corporate view should be provided in terms of representation mechanisms that are able to abstract from the physical and logical structure of data in the sources. It follows that the need and requirements for maintaining an integrated, conceptual view of the corporate data in the organization are stronger than in other contexts. A direct consequence of this fact is that the data in the sources and in the Data Warehouse should be defined in terms of the corporate view of data, and not the other way around. In other words, data integration in Data Warehousing should follow the *local-as-view* approach, where each table in a source and in the Data Warehouse is defined as a view of a global model of the corporate data. On the contrary, the *global-as-view* approach requires, for each information need, to specify the corresponding query in terms of the data at the sources, and is therefore suited when no global view of the data of the organization is available.

The above considerations motivate the local-as-view approach to information integration proposed in the context of the DWQ (Data Warehouse Quality) project.[8,9] The distinguishing features of the approach are: a rich *modeling language*, which extends an Entity-Relationship data model, in order to represent a Conceptual Data Warehouse Model; *reasoning tools* associated to the modeling language which support the Data Warehouse construction, maintenance and evolution.

Most of the work on integration has been concerned with the intensional/schema level.[10] Schema integration is nowadays a well-established discipline, which has also been addressed within DWQ.[11] It will not be discussed further in this paper. On the other hand, less attention has generally been devoted to the problem of data integration at the extensional/instance level. Data integration at the instance level is, nonetheless, crucial in Data Warehousing, where the process of acquiring data from the sources and making them available within the Data Warehouse is of paramount importance.

As a matter of fact, in the life time of a Data Warehouse the explicit representation of relationships between the sources and the materialized data in the Data Warehouse is useful in several tasks: from the initial loading, where the identification of the revelavant data within the sources is critical, to the refreshment process, which may require a dynamic adaptation depending on the availability of the sources, as well as on their reliability and quality that may change over time. Moreover, the extraction of data from a primary Data Warehouse for Data Mart applications, where the primary Warehouse is now regarded as a data source, can be treated in a similar way. In addition, even though the sources within an Enterprise are not as dynamic as in other information integration frameworks, they are nonetheless subject to changes; in particular, creation of new sources and deletion of existing ones must be taken into account. Consequently, the maintenance of the Data Warehouse requires several upgrades of the data flows towards the Data Warehouse. In other words, a Data Warehouse, especially in large organizations, should be regarded as an incremental system, which critically depends upon the relationships between the sources and the Data Warehouse.

Given a request for new data to be materialized in the Data Warehouse, which is formulated in terms of the conceptual, global view of the corporate data, (i.e. not the language of the sources, but of the enterprise), there are several steps that are required for the acquisition of data from the sources:

(1) Identification of the sources where the relevant information resides. Note that this task is typical of the local-as-view approach, and requires algorithms that are generally both sophisticated and costly.[4,12,13]
(2) Decomposition of the user request into queries to individual sources that are supposed to return the data of interest.
(3) Interpretation and merging of the data provided by a source. Interpreting data can be regarded as the task of casting them into a common representation. Moreover, the data returned by various sources need to be combined to provide the Data Warehouse with the requested information. The complexity of this reconciliation step is due to several problems, such as possible mismatches between data referring to the same real world object, possible errors in the data stored in the sources, possible inconsistencies between values representing the properties of the real world objects in different sources.[14]

In commercial environments for Data Warehouse design and management, the above tasks are taken care of through *ad hoc* components.[6] In general, such components provide the user with the capability of specifying the mapping between the sources and the Data Warehouse by browsing through a meta-level description of the relations of the sources. In addition, they generally provide both for automatic code generators and for the possibility of attaching procedures to accomplish *ad hoc* transformations and filtering of the data. Even though there are powerful and effective environments with the above features, their nature is inherently procedural, and close to the notion of global-as-view, where the task of relating the sources with the Data Warehouse is done on a query-by-query basis.

Several recent research contributions address the same problem from a more formal perspective.[5,15–21] Generally speaking, these works differ from our proposal since they follow the global-as-view approach. We refer to Sec. 6 for a comparison with our work. Research projects concerning the local-as-view approach have concentrated on the problem of reformulating a query in terms of the source relations.[12,13,22–34] However, none of them addresses both the problem of data reconciliation at the instance level, and the problem of query rewriting with respect to a conceptual model.

In this paper we present a novel approach to data integration in Data Warehousing, that builds upon and extends recent work within DWQ.[8,35,36] Compared with the existing proposals mentioned above, the novelty of our approach stems from the following features:

- It relies on the Conceptual Model of the corporate data, which is expressed in an Entity-Relationship formalism.
- It follows the local-as-view paradigm, in the sense that both the sources and the Data Warehouse relations are defined as views over the Conceptual Model.
- It allows the designer to declaratively specify several types of reconciliation correspondences between data in different schemas (either source schemas or Data Warehouse schema). Three types of Reconciliation Correspondences are taken into account, namely Conversion, Matching, and Merging Correspondences.
- It uses such correspondences in order to support the task of specifying the correct mediators for the loading of the materialized views of the Data Warehouse. For this purpose, our methodology relies on a query rewriting algorithm, whose role is to reformulate the query that defines the view to materialize in terms of both the source relations and the Reconciliation Correspondences. The characteristic feature of the algorithm is that it takes into account the constraints imposed by the Conceptual Model, and uses the Reconciliation Correspondences for cleaning, integrating, and reconciling data coming from different sources.

The paper is organized as follows. In Sec. 2, we summarize the general architecture we use in our approach to data integration in Data Warehousing. Section 3 illustrates the method we propose to describe the content of the sources and the Data Warehouse at the logical level. Section 4 is devoted to a discussion of the meaning and the role of Reconciliation Correspondences. Section 5 describes the query rewriting algorithm at the basis of our approach to the design of mediators. Section 6 compares our proposal with related work. Finally, Sec. 7 concludes the paper.

2. General Framework

We briefly summarize the general framework at the base of our approach to schema and data integration. Such a framework has been developed within the ESPRIT project DWQ, "Foundations of Data Warehouse Quality".[6,9] A Data Warehouse can be seen as a database which maintains an integrated, reconciled and materialized view of information residing in several data sources. In our approach we explicitly model the data in the sources and in the Data Warehouse at different levels of abstraction[37,8,38]:

- The *conceptual level*, which contains a conceptual representation of the corporate data.
- The *logical level*, which contains a representation in terms of a logical data model of the sources and of the data materialized in the Data Warehouse.
- The *physical level*, which contains a specification of the stored data, the wrappers for the sources, and the mediators for loading the data store.

The relationship between the conceptual and the logical, and between the logical and the physical level is represented explicitly by specifying mappings between corresponding objects of the different levels. In the rest of this section, we focus on the conceptual and logical levels, referring to the abstract architecture depicted in Fig. 1, the physical level is treated elsewhere.[6] In Sec. 5, we will explain in detail the construction of the specification of the mediators.

2.1. *Conceptual level*

In the overall Data Warehouse architecture, we explicitly conceive a *conceptual level*, which provides a conceptual representation of the data managed by the enterprise, including a conceptual representation of the data residing in sources, and of the global concepts and relationships that are of interest to the Data Warehouse application. Such a description, for which we use the term *Conceptual Model*, is independent from any system consideration, and is oriented towards the goal of expressing the semantics of the application. The Conceptual Model corresponds roughly to

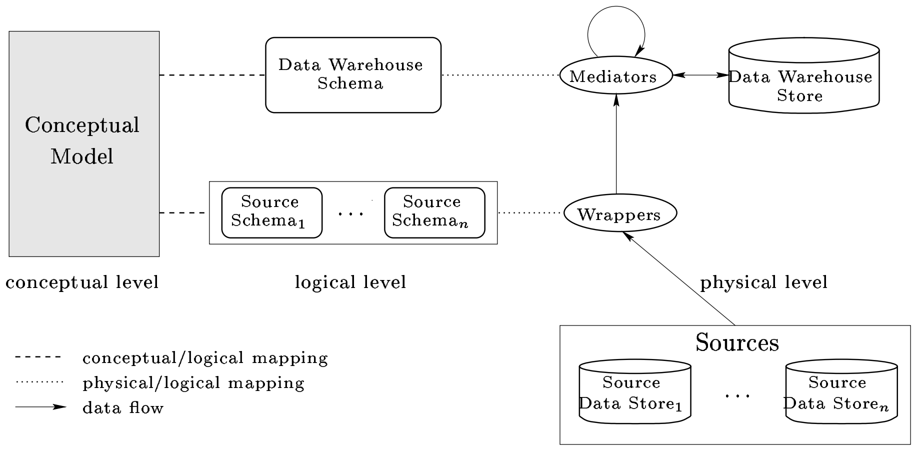

Fig. 1. Architecture for data integration.

the notion of integrated conceptual schema in the traditional approaches to schema integration, thus providing a consolidated view of the concepts and the relationships that are important to the enterprise, and have been currently analyzed. Such a view includes a conceptual representation of the portion of data, residing in the sources, currently taken into account. Hence, our approach is not committed to the existence of a fully specified Conceptual Model, but rather supports an incremental definition of such a model. Indeed, the Conceptual Model is subject to changes and additions as the analysis of the information sources proceeds.

An important aspect of the conceptual representation is the explicit specification of the set of interdependencies between objects in the sources and objects in the Data Warehouse. In this respect, data integration can be regarded as the process of understanding and representing the relationships between data residing in the information sources and the information contained in the Data Warehouse. Data reconciliation is also performed at this stage, instead of simply producing a unified data schema; moreover, such an integration activity is driven by automated reasoning tools, which are able to derive and verify several kinds of properties concerning the conceptual specification of information.

The formalization of information in the Conceptual Model is based on the distinction between *conceptual objects* and *values*. Reflecting such distinction, the Conceptual Model consists of two components:

(1) an enriched *Entity-Relationship model*, which formalizes the properties of conceptual objects; and
(2) a set of *domain assertions*, which model the properties of values.

We discuss the two components in the following.

2.1.1. *Enriched entity-relationship model*

The enriched Entity-Relationship model is formalized in terms of a logic based formalism, called $\mathcal{DLR}$.[37] Such a formalism allows us to capture the Entity-Relationship (ER) model augmented with several forms of constraints that cannot be expressed in the standard ER model. Moreover, it provides sophisticated automated reasoning capabilities, which can be exploited in verifying different properties of the Conceptual Model.

$\mathcal{DLR}$ belongs to the family of *Description Logics*, introduced and studied in the field of Knowledge Representation.[39,40] Generally speaking, Description Logics are class-based representation formalism that allow one to express several kinds of relationships and constraints (e.g. subclass constraints) holding among classes. Specifically, $\mathcal{DLR}$ includes:

- *concepts*, which are used to represent *entity types* (or simply *entities*), i.e. sets of conceptual objects having common properties;
- *n-ary relationships*, which are used to represent *relationship types* (or simply *relationships*), i.e. sets of tuples, each of which represents an association between

conceptual objects belonging to different entities. The participation of conceptual objects in relationships models properties corresponding to relations to other conceptual objects; and

- *attributes*, which are used to associate to conceptual objects (or tuples of conceptual objects) properties expressed by values belonging to one of several *domains*.

The Conceptual Model for a given application is specified by means of a set of logical *assertions* that express interrelationships between concepts, relationships, and attributes. In particular, the richness of $\mathcal{DLR}$ allows for expressing:

- disjointness and covering constraints, and more generally Boolean combinations between entities and relationships;
- universal and existential qualification of concepts and relationship components;
- participation and functionality constraints, and more complex forms of cardinality constraints;
- ISA relationships between entities and relationships; and
- definitions (expressing necessary and sufficient conditions) of entities and relationships in terms of other entities and relationships.

These features make $\mathcal{DLR}$ powerful enough to express not only the ER model, but also other conceptual, semantic, and object-oriented data models. Moreover, $\mathcal{DLR}$ assertions provide a simple and effective declarative mechanism to express the dependencies that hold between entities and relationships in different sources.[2,41] The use of inter-model assertions allows for an incremental approach to the integration of the conceptual models of the sources and of the enterprise.[37,42]

One distinguishing feature of $\mathcal{DLR}$ is that sound and complete reasoning algorithms are available.[43] Exploiting such algorithms, we gain the possibility of reasoning about the Conceptual Model. In particular, we can automatically verify the following properties:

- *Consistency of the Conceptual Model*, i.e. whether there exists a database satisfying all constraints expressed in the Conceptual Model.
- *Concept* (*relationship*) *satisfiability*, i.e. whether there exists a database that satisfies the Conceptual Model in which a certain entity (relationship) has a nonempty extension.
- *Entity* (*relationship*) *subsumption*, i.e. whether the extension of an entity (relationship) is a subset of the extension of another entity (relationship) in every database satisfying the Conceptual Model.
- *Constraint inference*, i.e. whether a certain constraint holds for all databases satisfying the Conceptual Model.

Such reasoning services support the designer in the construction process of the Conceptual Model: they can be used, for instance, for inferring inclusion between entities and relationships, and detecting inconsistencies and redundancies.

We show how to formalize in $\mathcal{DLR}$ a simple ER schema, which we will use as our running example. A full-fledged example of our methodology can be found in a case study from the telecommunication domain.[44,45]

Example 1. The schema shown in Fig. 2 represents persons divided in males and females and parent-child relationship. The following set of assertions exactly captures the ER schema in the figure.

$$\begin{aligned}
\mathsf{Person} &\sqsubseteq (= 1\ \mathsf{name}) \sqcap \forall \mathsf{name}.\mathsf{NameString} \sqcap (= 1 \mathsf{ssn}) \sqcap \forall \mathsf{ssn}.\mathsf{SSNString} \sqcap \\
&\quad\ (= 1 \mathsf{dob}) \sqcap \forall \mathsf{dob}.\mathsf{Date} \sqcap (= 1\ \mathsf{income}) \sqcap \forall \mathsf{income}.\mathsf{Money} \\
\mathsf{Person} &\equiv \mathsf{Female} \sqcup \mathsf{Male} \\
\mathsf{Female} &\sqsubseteq \neg \mathsf{Male} \\
\mathsf{CHILD} &\sqsubseteq (\$1 : \mathsf{Person}) \sqcap (\$2 : \mathsf{Person})
\end{aligned}$$

The first four assertions specify the existence and the domain of the attributes of Person. The next two assertions specify that persons are partitioned in females and males. The last assertion specifies the typing of the CHILD relationship.

We could also add constraints not expressible in the ER model, such as introducing a further entity MotherWith3Sons denoting mothers having at least three sons:

$$\mathsf{MotherWith3Sons} \ \equiv\ \mathsf{Female} \sqcap (\leq 3[\$1](\mathsf{CHILD} \sqcap (\$2 : \mathsf{Male})))$$

2.1.2. *Abstract domains*

Rather than considering concrete domains, such as strings, integers, and reals, our approach is based on the use of *abstract domains*. Abstract domains may have an underlying concrete domain, but their use allows the designer to distinguish between the different meanings that values of the concrete domain may have. The properties and mutual inter-relationships between domains can be specified by means of *domain assertions*, each of which is expressed as an inclusion between Boolean combinations of domains. In particular, domain assertions allow to express an ISA

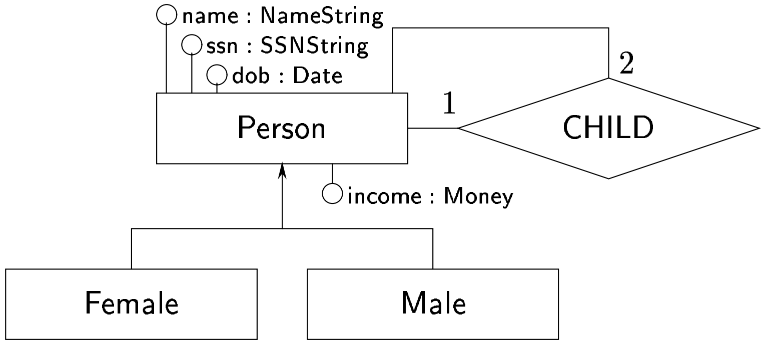

Fig. 2. Entity-relationship schema for parent-child relationship.

hierarchy between domains. We say that a domain assertion is *satisfied* if the inclusion between the corresponding Boolean combinations of domain extensions holds.

Example 2. Consider two attributes A_1 in a source and A_2 in the Data Warehouse, both representing amounts of money. Rather than specifying that both attributes have values of type Real, the designer may specify that the domain of attribute A_1 is $\mathsf{MoneyInLire}$ while the domain of attribute A_2 is $\mathsf{MoneyInEuro}$, which are both subsets of the domain Money (which possibly has Real as the underlying concrete domain). The relationship between the three domains can be expressed by means of the domain assertions:

$$\begin{aligned}\mathsf{MoneyInLire} &\sqsubseteq \mathsf{Money}\\ \mathsf{MoneyInEuro} &\sqsubseteq \mathsf{Money} \sqcap \neg\mathsf{MoneyInLire}\end{aligned}$$

In this way, it becomes possible to specify declaratively the difference between values of the two attributes, and take such knowledge into account for loading data from the source to the Data Warehouse.

Given a set of domain assertions, the designer may be interested in verifying several properties, such as:

- *Satisfiability* of the whole set of domain assertions, i.e. whether it is actually possible to assign to each domain an extension in such a way that all assertions are satisfied.
- *Domain satisfiability*, i.e. whether it is possible to assign to a domain a nonempty extension, under the condition that all assertions are satisfied.
- *Domain subsumption*, i.e. whether the extension of one domain is a subset of the extension of another domain, whenever all domain assertions are satisfied.

The presence of unsatisfiable domains reflects some error in the modeling process, and requires to remove the unsatisfiable domain or revise the set of domain assertions. Similarly, the presence of equivalent domains may be an indication for redundancy.

Typically, the designer is interested in automatically checking the above properties, and more generally, in automatically constructing a *domain hierarchy* reflecting the ISA relationships between domains that logically follow from the set of assertions.

Using $\mathcal{DLR}$ we can express and reason over the enriched Entity-Relationship model together with the domain hierarchy. However, if we are interested in reasoning about domains only, we can use a more direct approach. Indeed, by conceiving each domain as a unary predicate, we can correctly represent a domain assertion $D \sqsubseteq D'$ by the first-order logic formula $\forall x \cdot D(x) \supset D'(x)$. Since such formula does not contain any quantifier besides the outermost $\forall$, it can be captured correctly by the propositional formula $A \supset B$, where we consider each domain as a propositional symbol. Therefore we can exploit techniques developed for propositional reasoning[46–48] to perform inference on domains, and thus automatically check the desired properties resulting from the set of assertions.

2.2. *Logical level*

The logical level provides a description of the logical content of each source, called the *Source Schema*, and the logical content of the materialized views constituting the Data Warehouse, called the *Data Warehouse Schema* (see Fig. 1). Such schemas are intended to provide a structural description of the content of both the sources and the materialized views in the Data Warehouse.

A Source Schema is provided in terms of a set of relations using the relational model. The link between the logical representation and the conceptual representation of the source is formally defined by associating each relation with a query that describes its content in terms of a query over the Conceptual Model. In other words, the logical content of a source relation is described in terms of a view over the virtual database represented by the Conceptual Model, adopting the *local-as-view approach.* To map physical structures to logical structures we make use of suitable wrappers, which encapsulate the sources. The wrapper hides how the source actually stores its data, the data model it adopts, etc. and presents the source as a set of relations. In particular, we assume that all attributes in the relations are of interest to the Data Warehouse application (attributes that are not of interest are hidden by the wrapper). Relation attributes are thus modeled as either entity attributes or relationship attributes in the Conceptual Model.

The Data Warehouse Schema, which expresses the logical content of the materialized views constituting the Data Warehouse, is provided in terms of a set of relations. Similarly to the case of the sources, each relation of the Data Warehouse Schema is described in terms of a query over the Conceptual Model.

From a technical point of view such queries are unions of conjunctive queries. More precisely, a query q over the Conceptual Model has the form:

$$T(\vec{\mathrm{x}}) \leftarrow q(\vec{\mathrm{x}}, \vec{\mathrm{y}})$$

where the *head* $T(\vec{\mathrm{x}})$ defines the schema of the relation in terms of a name T, and its *arity*, i.e. the number of columns (number of components of $\vec{\mathrm{x}}$), and the *body* $q(\vec{\mathrm{x}}, \vec{\mathrm{y}})$ describes the content of the relation in terms of the Conceptual Model. The body has the form

$$conj_1(\vec{\mathrm{x}}, \vec{\mathrm{y}}_1) \ \mathsf{OR} \cdots \mathsf{OR} \ conj_m(\vec{\mathrm{x}}, \vec{\mathrm{y}}_m)$$

where each $conj_i(\vec{\mathrm{x}}, \vec{\mathrm{y}}_i)$ is a conjunction of *atoms*, and $\vec{\mathrm{x}}, \vec{\mathrm{y}}_i$ are all the variables appearing in the conjunct (we use $\vec{\mathrm{x}}$ to denote a tuple of variables $x_1, \ldots, x_n$, for some n). Each atom is of the form $E(t)$, $R(\vec{\mathrm{t}})$, or $A(t, t')$, where $\vec{\mathrm{t}}$, t, and t' are variables in $\vec{\mathrm{x}}, \vec{\mathrm{y}}_i$ or constants, and E, R, and A are respectively entities, relationships, and attributes appearing in the Conceptual Model.

In the following, we will also consider queries whose body may contain special predicates that do not appear in the conceptual model.

The semantics of queries is as follows. Given a database that satisfies the Conceptual Model, a query

$$T(\vec{x}) \leftarrow conj_1(\vec{x},\vec{y}_1)\mathsf{OR}\cdots\mathsf{OR}conj_m(\vec{x},\vec{y}_m)$$

of arity n is interpreted as the set of n-tuples $(d_1,\ldots,d_n)$, with each d_i an object of the database, such that, when substituting each d_i for x_i, the formula

$$\exists\vec{y}_1 \cdot conj_1(\vec{x},\vec{y}_1)\mathsf{OR}\cdots\mathsf{OR}\exists\vec{y}_m \cdot conj_m(\vec{x},\vec{y}_m)$$

evaluates to true.

Suitable inference techniques allow for carrying out the following reasoning services on queries by taking into account the Conceptual Model[43]:

- *Query containment.* Given two relational queries q_1 and q_2 (of the same arity n) over the Conceptual Model, we say that q_1 is *contained in* q_2, if the set of tuples denoted by q_1 is contained in the set of tuples denoted by q_2 in every database satisfying the Conceptual Model.
- *Query consistency.* A relational query q over the Conceptual Model is *consistent*, if there exists a database satisfying the Conceptual Model in which the set of tuples denoted by q is not empty.
- *Query disjointness.* Two relational queries q_1 and q_2 (of the same arity) over the Conceptual Model are *disjoint*, if the intersection of the set of tuples denoted by q_1 and the set of tuples denoted by q_2 is empty, in every database satisfying the Conceptual Model.

3. Source and Data Warehouse Logical Schema Descriptions

The notion of query over the Conceptual Model is a powerful tool for modeling the logical level of the Sources and the Data Warehouse. As mentioned above, we express the relational tables constituting both the Data Warehouse Schema and Source Schemas in terms of queries over the Conceptual Model, with the following characteristics:

- Relational tables are composed of tuples of values, which are the only kind of objects at the logical level. Therefore, each variable in the head of the query represents a value (not a conceptual object).
- Each variable appearing in the body of the query either denotes a conceptual object or a value, depending on the atoms in which it appears. Since, in each database that satisfies the Conceptual Model, conceptual objects and values are disjoint sets, no query can contain a variable which can be instantiated by both a conceptual object and a value.
- Each conceptual object is represented by a tuple of values at the logical level. Thus, a mechanism is needed to express this kind of correspondence between a tuple of values and the conceptual object it represents. This is taken into account by the notion of *adornment* introduced below.

3.1. *Source schema description*

We remind the reader that we assume that each source is encapsulated by a suitable wrapper, and this wrapper provides a logical view of the data stored in the source in terms of the relational model, i.e. in the form of a set of relations.

As we said before, the query associated with a source relation provides the glue between the conceptual and the logical representation. However, to capture in a precise way the data in the sources, more information is needed in order to describe the actual structure of the data in the relation. This is done by the *adornment* associated to the relation, whose role is to declare the domains of the columns of the table, and which are the attributes of the table that are used to identify the objects of the Conceptual Model. In other words, the adornment is used to make explicit how the objects of the conceptual representation are coded into values of the logical representation.

An adorned query is an expression of the form

$$T(\vec{\mathrm{x}}) \leftarrow q(\vec{\mathrm{x}}, \vec{\mathrm{y}}) | \alpha_1, \ldots, \alpha_n$$

where $\alpha_1, \ldots, \alpha_n$ constitutes the *adornment* in which each α_i is an *annotation* on variables appearing in $\vec{\mathrm{x}}$. In particular:

- For each $X \in \vec{\mathrm{x}}$, we have an annotation of the form

  $$X :: V$$

  where V is a domain expression. Such an annotation is used to specify how values bound to X are represented in the table at the logical level.
- For each tuple of variables $\vec{\mathrm{z}} \subseteq \vec{\mathrm{x}}$ that is used for identifying in T a conceptual object $Y \in \vec{\mathrm{y}}$ mentioned in $q(\vec{\mathrm{x}}, \vec{\mathrm{y}})$, we have an annotation of the form

  $$Identify([\vec{\mathrm{z}}], Y)$$

  where we have grouped the variables $\vec{\mathrm{z}}$ into a single argument $[\vec{\mathrm{z}}]$. Such annotation makes explicit that the tuple of values $\vec{\mathrm{z}}$ is a representation of the conceptual object Y.

We illustrate by means of an example how adorned queries are used to specify the information content of source tables. We use $X_1, \ldots, X_k :: V$ as an abbreviation for $X_1 :: V, \ldots, X_k :: V$.

Example 3. Suppose we store in two sources $\mathcal{S}_1$ and $\mathcal{S}_2$ parent-child relationships, according to the Conceptual Model shown in Fig. 2. Source $\mathcal{S}_1$ contains the information about fathers and their children, in terms of a relational table FATHER_1 which stores all such pairs. Similarly, source $\mathcal{S}_2$ contains the information about mothers and their children in terms of a relational table MOTHER_2. We assume that in source $\mathcal{S}_1$ persons (both fathers and children) are identified by their name and date of birth, while in source $\mathcal{S}_2$ persons are identified by their social security number. Hence, we can specify the information content of the two tables by the following adorned queries:

$$\begin{array}{lll}
\mathsf{FATHER}_1(N_f, D_f, N_c, D_c) & \leftarrow & \mathsf{Male}(F), \mathsf{Person}(C), \mathsf{CHILD}(F, C), \\
& & \mathsf{name}(F, N_f), \mathsf{dob}(F, D_f), \mathsf{name}(C, N_c), \mathsf{dob}(C, D_c) \\
& & \mid N_F, N_C :: \mathsf{NameString}, \\
& & \quad D_f, D_c :: \mathsf{Data}, \\
& & \quad \mathit{Identify}([N_f, D_f], F), \mathit{Identify}([N_c, D_c], C) \\
\mathsf{MOTHER}_2(S_m, S_c) & \leftarrow & \mathsf{Female}(M), \mathsf{Person}(C), \mathsf{CHILD}(M, C), \\
& & \mathsf{ssn}(M, S_m), \mathsf{ssn}(C, S_c) \\
& & \mid S_m, S_c :: \mathsf{SSNString}, \\
& & \quad \mathit{Identify}([S_m], M), \mathit{Identify}([S_c], C)
\end{array}$$

Example 4. Referring again to the Conceptual Model shown in Fig. 2, we want to model two sources storing the information about the income of persons. Source $\mathcal{S}_1$ stores the income per month of males in a table INCOME_1, while source $\mathcal{S}_2$ stores the income per year of females in a table INCOME_2, and the content of the tables is specified by the following adorned queries:

$$\begin{array}{lll}
\mathsf{INCOME}_1(S_m, I) & \leftarrow & \mathsf{Male}(M), \mathsf{ssn}(M, S_m), \mathsf{income}(M, I) \\
& & \mid S_m :: \mathsf{SSNString}, \\
& & \quad I :: \mathsf{IncomePerMonth}, \\
& & \quad \mathit{Identify}([S_m], M) \\
\mathsf{INCOME}_2(S_f, I) & \leftarrow & \mathsf{Female}(F), \mathsf{ssn}(F, S_f), \mathsf{income}(F, I) \\
& & \mid S_f :: \mathsf{SSNString}, \\
& & \quad I :: \mathsf{IncomePerYear}, \\
& & \quad \mathit{Identify}([S_f], F)
\end{array}$$

The adorned query associated to a table in a source contains a lot of information that can be profitably used in analyzing the quality of the Data Warehouse design process. Indeed, the adorned query precisely formalizes the content of a source table in terms of a query over the Conceptual Model, the domains of each attribute of the table, and the attributes used to identify entities at the conceptual level. One important check that we can carry out over the logical specification of a source is whether the adorned query associated with a table in a source is consistent or not. Let Q be an adorned query and let B be its body. The query B is said to be *inconsistent* with respect to the Conceptual Model $\mathcal{M}$, if for every database DB coherent with $\mathcal{M}$, the evaluation of B with respect to DB is empty. An adorned query Q may be inconsistent with respect to the Conceptual Model $\mathcal{M}$ because the body B of Q is inconsistent with respect to $\mathcal{M}$. Inference techniques allow us to check for these forms of inconsistency.[8] Another reason for inconsistency in specifying a table may be due to annotations that are incoherent with respect to what specified in $\mathcal{M}$. In particular, domain expressions used in the adornment can be inconsistent with respect to the set of domain assertions that are part of the

Conceptual Model. Standard propositional reasoning tools[46–48] can be adopted to detect such forms of inconsistency.

3.2. *Data Warehouse schema description*

Similarly to the case of source relations, the relations to be materialized in the Data Warehouse are described as adorned queries over the Conceptual model.

Note that the adorned query associated to a table in a source is the result of a reverse engineering analysis of the source, whereas in this case the adorned query is a high-level specification of what we want to materialize in the table of the Data Warehouse, and thus of the mediator for loading such a materialized view. Since we express the semantics of the Data Warehouse tables in terms of the Conceptual Model, also the relations in the Data Warehouse are seen as views of such a Conceptual Model.

Example 3. (Cont). Suppose we want to store in the Data Warehouse all pairs of male-female with a child in common. The pairs have to be stored in a table $\mathsf{COMMONCHILD}_{\mathsf{DW}}$, identifying each person by its social security number. We can specify the content of the table w.r.t. the Conceptual Model by the following adorned query:

$$\begin{aligned}\mathsf{COMMONCHILD}_{\mathsf{DW}}(S_f, S_m) \leftarrow\ & \mathsf{Male}(F), \mathsf{ssn}(F, S_f), \mathsf{CHILD}(F, C),\\ & \mathsf{Female}(M), \mathsf{ssn}(M, S_m), \mathsf{CHILD}(M, C)\\ & \mid S_f, S_m :: \mathsf{SSNString},\\ & \quad \mathit{Identify}([S_f], F), \mathit{Identify}([S_m], M)\end{aligned}$$

Example 4. (Cont). Suppose we want to store in the Data Warehouse pairs of persons with the same income. The pairs, together with the common income per year, have to be stored in a table $\mathsf{SAMEINCOME}_{\mathsf{DW}}$, identifying each person by its social security number. We can specify the content of the table w.r.t. the Conceptual Model by the following adorned query:

$$\begin{aligned}\mathsf{SAMEINCOME}_{\mathsf{DW}}(S_1, S_2, I) \leftarrow\ & \mathsf{Person}(P_1), \mathsf{ssn}(P_1, S_1), \mathsf{income}(P_1, I),\\ & \mathsf{Person}(P_2), \mathsf{ssn}(P_2, S_2), \mathsf{income}(P_2, I)\\ & \mid S_1, S_2 :: \mathsf{SSNString},\\ & \quad I :: \mathsf{IncomePerYear},\\ & \quad \mathit{Identify}([S_1], P_1), \mathit{Identify}([S_2], P_2)\end{aligned}$$

The reasoning services provided at the logical level make it possible to automatically generate the correct mediators for the loading. As illustrated in Sec. 5, this is realized by means of a query rewriting technique which uses query containment as its basic reasoning service.

The choice of the data to materialize, and how to organize them in relations, is an important step in the construction of the Data Warehouse. Several aspects have to be taken into account in making these choices, for example the required storage amount, the cost of loading, refreshment, and query processing, etc. A methodology for identifying the tables to materialize that is coherent with our framework has been developed.[49]

3.3. *Schematic differences*

The mechanism used in our framework for specifying adorned queries is able to cope with *schematic differences*.[50] The example below illustrates a case where there are various schematic differences, both among the sources, and between the sources and the Conceptual Model.

Example 5. Suppose that the Conceptual Model contains an entity Service with three attributes, date, type, and price, specifying respectively the date, the type, and the cost of the service. Suppose that source $\mathcal{S}_1$ represents information only on services of type t_1 and t_2, by means of two relations: t1 and t2, where $\mathsf{t1}(D, P)$ means that service t_1 costs P Italian Lira at date D, and $\mathsf{t2}(D, P)$ means that service t_2 costs P Italian Lira at date D. Suppose that source $\mathcal{S}_2$ represents information only on services t_3 and t_4 by means of a relation Serv_2, where $\mathsf{Serv}_{3,4}(D, P_3, P_4)$ means that services t_3 and t_4 cost P_3 and P_4 Euro respectively at date D. Finally, suppose that source $\mathcal{S}_3$ represents the information only for a certain date d by means of another relation Serv_d. The various relations in the three sources can be specified by means of the following adorned queries:

$$\begin{aligned}
\mathsf{t1}(D, P) \leftarrow\ & \mathsf{Service}(S), \mathsf{date}(S, D), \mathsf{type}(S, \text{`t1'}), \mathsf{price}(S, P)\\
& \mid D :: \mathsf{Date}, P :: \mathsf{ItalianLira}, \mathit{Identify}([\text{`t1'}, D], S)\\
\mathsf{t2}(D, P) \leftarrow\ & \mathsf{Service}(S), \mathsf{date}(S, D), \mathsf{type}(S, \text{`t2'}), \mathsf{price}(S, P)\\
& \mid D :: \mathsf{Date}, P :: \mathsf{ItalianLira}, \mathit{Identify}([\text{`t2'}, D], S)\\
\mathsf{Serv}_{3,4}(D, P_3, P_4) \leftarrow\ & \mathsf{Service}(S_3), \mathsf{date}(S_3, D), \mathsf{type}(S_3, \text{`t3'}), \mathsf{price}(S_3, P_3),\\
& \mathsf{Service}(S_4), \mathsf{date}(S_4, D), \mathsf{type}(S_4, \text{`t3'}), \mathsf{price}(S_4, P_4),\\
& \mid D :: \mathsf{Date}, P_3 :: \mathsf{Euro}, P_4 :: \mathsf{Euro},\\
& \quad \mathit{Identify}([\text{`t3'}, D], S_3), \mathit{Identify}([\text{`t4'}, D], S_4)\\
\mathsf{Ser}_d(T, P) \leftarrow\ & \mathsf{Service}(S), \mathsf{date}(S, \text{`d'}), \mathsf{type}(S, T), \mathsf{price}(S, P)\\
& \mid T :: \mathsf{TypeString}, P :: \mathsf{Euro}, \mathit{Identify}([T, \text{`d'}], S)
\end{aligned}$$

4. Reconciliation Correspondences

Assume that the decision of which data to materialize has been taken, and has resulted in the specification of a new Data Warehouse relation T expressed in terms of an adorned query. One crucial task is to design the mediator for T, i.e. the program that accesses the sources and loads the correct data into the relation T. As we said in the Introduction, designing the mediator for T requires first of all to reformulate the query associated with T in terms of the Source relations. However, such a reformulation is not sufficient. The task of mediator design is complicated by the possible heterogeneity between the data in the sources. We have already mentioned the most important ones, namely, mismatches between data referring to the same real world object, errors in the data stored in the sources, inconsistencies between values representing the properties of the real world objects in different sources.

Our idea to cope with this problem is based on the notion of Reconciliation Correspondence. Indeed, in order to anticipate possible errors, mismatches and inconsistencies between data in the sources, our approach allows the designer to declaratively specify the correspondences between data in different schemas (either source schemas or Data Warehouse schema). Such specification is done through special assertions, called *Reconciliation Correspondences*.

Reconciliation Correspondences are defined in terms of relations, similarly to the case of the relations describing the sources and the Data Warehouse at the logical level. The difference with source and Data Warehouse relations is that we conceive Reconciliation Correspondences as non-materialized relations, in the sense that their extension is computed by an associated program whenever it is needed. In particular, each Reconciliation Correspondence is specified as an adorned query with an associated *program* that is called to compute the extension of the virtual relation. Note that we do not consider the actual code of the program but just its input and output parameters.

We distinguish among three types of correspondences, namely Conversion, Matching, and Merging Correspondences.

4.1. *Conversion correspondences*

Conversion Correspondences are used to specify that data in one table can be converted into data of a different table, and how this conversion is performed. They are used to anticipate various types of data conflicts that may occur in loading data coming from different sources into the Data Warehouse.

Formally, a *Conversion Correspondence* has the following form:

$$\begin{aligned}\mathsf{convert}_C([\vec{\mathrm{x}}],[\vec{\mathrm{x}}_2]) \leftarrow{} & \mathit{Equiv}([\vec{\mathrm{x}}_2],[\vec{\mathrm{x}}_2]), \mathit{conj}(\vec{\mathrm{x}}_1,\vec{\mathrm{x}}_2,\vec{\mathrm{y}})\\ & \mid \alpha_1,\ldots,\alpha_n\\ & \mathsf{through}\ \mathit{Program}(\vec{\mathrm{x}}_1,\vec{\mathrm{x}}_2,\vec{\mathrm{y}})\end{aligned}$$

where $\mathsf{convert}_C$ is the *conversion predicate* defined by the correspondence; *Equiv* is a special predicate whose intended meaning is that $\vec{\mathrm{x}}_1$ and $\vec{\mathrm{x}}_2$ actually represent the same data[a]; *conj* is a conjunction of atoms, which specifies the conditions under which the conversion is applicable; $\alpha_1,\ldots,\alpha_n$ is an adornment of the query; *program* denotes a program that performs the conversion. In general, the program needs to take into account the additional parameters specified in the condition to actually perform the conversion. The conversion has a direction. In particular, it

[a]*Equiv* plays a special role during the construction of the rewriting for the synthesis of the mediator, as explained in Sec. 5.

operates from a tuple of values satisfying the conditions specified for $\vec{x}_1$ in *conj* and in the adornment to a tuple of values satisfying the conditions specified for $\vec{x}_2$. This means that the conversion program receives as input a tuple $\vec{x}_1$, and returns the corresponding tuple $\vec{x}_2$, possibly using the additional parameters $\vec{y}$ to perform the conversion. Notice that we will have to take into account that the conversion has a direction also when we make use of the correspondence for populating the Data Warehouse.

Example 3. (Cont). Since we know that persons in the Data Warehouse are identified by their social security number, while in source $\mathcal{S}_1$ they are identified by their name and date of birth, we have to provide a mean to convert the name and date of birth of a person to his social security number. Let name_dob_to_ssn be a suitable program that performs the conversion, taking as input name and date of birth, and returning the social security number. To give a declarative account of this ability, we provide the following Conversion Correspondence:

$$\begin{aligned} convert_{person}([N,D],[S]) \leftarrow\ & Equiv([N,D],[S]), \mathsf{Person}(P), \\ & \mathsf{name}(P,N), \mathsf{dob}(P,D), \mathsf{ssn}(P,S) \\ & \mid N :: \mathsf{NameString},\ D :: \mathsf{Date}, S :: \mathsf{SSNString}, \\ & \quad Identify([N,D],P), Identify([S],P) \\ & \mathsf{through}\ \mathsf{name\_dob\_to\_ssn}(N,D,S) \end{aligned}$$

The Conversion Correspondence specifies the condition under which a name N and date of birth D can be converted to a social security number S, by requiring the existence of a person with name N, date of birth D, and social security number S. Notice that the predicates in the body of the Conversion Correspondence play a fundamental role in restricting the applicability of the conversion to the proper attributes of persons.

In the example above we have used a correspondence that converts between different representations of the same object. Since a representation may consist of a tuple of values (e.g. $[N,D]$) this is a typical situation in which we compare tuples, possibly of different arity. Our approach requires indeed that, in order to compare tuples, we have a common underlying object, which is used to make explicit the relationship between the components of the two tuples. In the case where we do not have such an underlying object, we still allow to compare single values in different domains, as shown in the following example.

Example 4. (Cont.) To translate incomes per month to incomes per year we define the following Conversion Correspondence:

$$\begin{aligned} convert_{income}([I_m],[I_y]) \leftarrow\ & Equiv([I_m],[I_y]) \\ & \mid I_m :: \mathsf{IncomePerMonth}, I_y :: \mathsf{IncomePerYear} \\ & \mathsf{through}\ \mathsf{income\_month\_to\_year}(I_m, I_y) \end{aligned}$$

4.2. *Matching correspondences*

Matching Correspondences are used to specify how data in different source tables, typically referring to different sources, can match. Formally, a *Matching Correspondence* has the following form:

$$\begin{aligned} \mathit{match}_M([\vec{\mathrm{x}}_1],\ldots,[\vec{\mathrm{x}}_k]) \leftarrow\ & \mathit{Equiv}([\vec{\mathrm{x}}_1],[\vec{\mathrm{x}}_2]),\ldots,\mathit{Equiv}([\vec{\mathrm{x}}_{k-1}],[\vec{\mathrm{x}}_k]),\\ & \mathit{conj}(\vec{\mathrm{x}}_1,\ldots,\vec{\mathrm{x}}_k,\vec{\mathrm{y}})\\ & \mid \alpha_1,\ldots,\alpha_n\\ & \textsf{through}\ \mathit{program}(\vec{\mathrm{x}}_1,\ldots,\vec{\mathrm{x}}_k,\vec{\mathrm{y}}) \end{aligned}$$

where match_M is the *matching predicate* defined by the correspondence, *Equiv* is as before, *conj* specifies the conditions under which the matching is applicable, $\alpha_1,\ldots,\alpha_n$ is again an adornment of the query, and *program* denotes a program that performs the matching. The program receives as input k tuples of values satisfying the conditions (and possibly the additional parameters in the condition) and returns whether the tuples match or not.

Example 3. (Cont.) To compare persons in source $\mathcal{S}_1$ with persons in source $\mathcal{S}_2$, we have to provide a mean to compare persons identified by their name and date of birth with persons identified by their social security number. Let name_dob_matches_ssn be a suitable program that, given name, date of birth, and the social security number as input, returns whether they correspond to the same person or not. To give a declarative account of this program, we provide the following Matching Correspondence:

$$\begin{aligned} \mathit{match}_{person}([N,D],[S]) \leftarrow\ & \mathit{Equiv}([N,D],[S]),\mathsf{Person}(P),\\ & \mathsf{name}(P,N),\mathsf{dob}(P,D),\mathsf{ssn}(P,S)\\ & \mid N :: \mathsf{NameString}, D :: \mathsf{Date}, S :: \mathsf{SSNString},\\ & \quad \mathit{Identify}([N,D],P),\mathit{Identify}([S],P)\\ & \textsf{through}\ \mathsf{name\_dob\_matches\_ssn}(N,D,S) \end{aligned}$$

Notice that the Matching Correspondence match_{person} and the Conversion Correspondence $\mathit{convert}_{person}$ look the same. However there are fundamental differences between them. The associated programs behave differently. In particular, name_dob_to_ssn(N,D,S) is used to generate the social security number S of the person with name N and date of birth D, whereas name_dob_matches_ssn(N,D,S) is used to verify that N and D, and S refer to the same person. Consequently, when the Matching Correspondence and the Conversion Correspondence are used as atoms in a query Q that specifies how to populate a Data Warehouse table, they have to be used according to different binding patterns of the variables occurring in the atoms. In particular, in the Matching Correspondence all variables have to be bound, and therefore cannot occur in the head of the query Q. Instead, in the Conversion Correspondence, the variables in the first tuple need to be bound, while the ones in the second tuple are free. Hence the variables in the second tuple can occur in the head of Q.

4.3. *Merging correspondences*

Merging Correspondences are used to assert how we can merge data in different sources into data that populates the Data Warehouse. Formally, a *Merging Correspondence* has the following form:

$$\begin{aligned} marge_R([\vec{x}_1], \ldots, [\vec{x}_k], [\vec{x}_0]) \leftarrow\ & Equiv([\vec{x}_1], [\vec{x}_0]), \ldots, Equiv([\vec{x}_k], [\vec{x}_0]), \\ & conj(\vec{x}, \ldots, \vec{x}_k, \vec{x}_0, \vec{y}) \\ & \mid \alpha_1, \ldots, \alpha_n \\ & \textsf{through } program(\vec{x}_1, \ldots, \vec{x}_k, \vec{x}_0, \vec{y}) \end{aligned}$$

where *merge*$_R$ is the *merging predicate* defined by the correspondence, *Eqiv* is as before, *conj* specifies the conditions under which the merging is applicable, and *program* is a program that performs the merging. Such correspondence specifies that the k tuples of values $\vec{x}_1, \ldots, \vec{x}_k$ coming from the sources are merged into the tuple $\vec{x}_0$ in the Data Warehouse. Therefore, the associated program receives as input k tuples of values (and possibly the additional parameters in the condition) and returns a tuple which is the result of the merging. Example 7 below illustrates the use of a Merging Correspondence.

4.4. *Methodological guidelines*

The task of specifying suitable Reconciliation Correspondences is a responsibility of the designer. Once such Reconciliation Correspondences are specified, they are profitably exploited to automatically generate mediators, as described in Sec. 5. In the task of specifying Reconciliation Correspondences the system can assist the designer in various ways.

First of all, since each Reconciliation Correspondence is declaratively specified as an adorned query, all reasoning tasks available for such queries can be exploited to check desirable properties of the correspondence. In particular, the system can check the consistency of queries, rejecting inconsistent ones and giving the designer useful feedback. Also, the system can automatically detect whether the adorned queries associated with different correspondences are contained in each other (or are equivalent). This is an indication for redundancy in the specification. However, to determine whether a correspondence is actually redundant, one has to consider also the types of the correspondences and the programs associated with them. For example, a less general query, thus specifying stricter conditions for applicability, may still be useful in the case where the associated program takes advantage of the specialization and operates more efficiently.

In practice, the system automatically asserts several correspondences *by default*, thus simplifying the task of the designer.

- Several of the Reconciliation Correspondences that must be specified will have a very simple form, since they will correspond simply to equality. In particular, for each domain D in the Conceptual Model, the following Reconciliation

Correspondences are asserted by default:

$$\begin{aligned} convert_D([X],[Y]) \leftarrow\ & Equiv([X],[Y]) \\ & |X, Y :: D \\ & \mathsf{through\ identity}(X, Y) \\ match_D([X],[Y]) \leftarrow\ & Equiv([X],[Y]) \\ & \mid X, Y :: D \\ & \mathsf{through\ none} \\ merge_D([X],[Y],[Z]) \leftarrow\ & Equiv([X],[Z]), Equiv([Y],[Z]) \\ & \mid X, Y, Z :: D \\ & \mathsf{through\ identity}(X, Y, Z) \end{aligned}$$

where identity is the program that computes the identity function for values of domain D, and the matching correspondence has no associated program. When the designer provides an Reconciliation Correspondence referring to a domain D, then the automatic generation of the default correspondences for D is inhibited.

- Similarly, for each annotation of the form $Identify([X_1, \ldots, X_k], X)$ appearing in the adornment of a query q defining a source or Data Warehouse table T, and such that $X_1 :: D_1, \ldots, X_k :: D_k$ are the annotations in q specifying the domains associated with $X_1, \ldots, X_k$, the following Reconciliation Correspondences are asserted by default:

$$\begin{aligned} convert_{D_1,\ldots,D_k}([\vec{x}],[\vec{y}]) \leftarrow\ & Equiv([X_1],[Y_1]), \ldots, Equiv([X_k],[Y_k]) \\ & \mid X_1, Y_1 :: D_1, \ldots, X_k, Y_k :: D_k \\ & \mathsf{through\ identity}([\vec{x}],[\vec{y}]) \\ match_{D_1,\ldots,D_k}([\vec{x}],[\vec{y}]) \leftarrow\ & Equiv([X_1],[Y_1]), \ldots, Equiv([X_k],[Y_k]) \\ & \mid X_1, Y_1 :: D_1, \ldots, X_k, Y_k :: D_k \\ & \mathsf{through\ none} \\ merge_{D_1,\ldots,D_k}([\vec{x}],[\vec{y}],[\vec{z}]) \leftarrow\ & Equiv([X_1],[Z_1]), Equiv([Y_1],[Z_1]), \ldots, \\ & Equiv([X_k],[Y_k]), Equiv([Y_k],[Z_k]) \\ & \mid X_1, Y_1, Z_1 :: D_1, \ldots, X_k, Y_k, Z_k :: D_k \\ & \mathsf{through\ identity}([\vec{x}],[\vec{y}],[\vec{z}]) \end{aligned}$$

where $\vec{x}$ abbreviates $X_1, \ldots, X_k$, $\vec{y}$ abbreviates $Y_1, \ldots, Y_k$, and $\vec{z}$ abbreviates $Z_1, \ldots, Z_k$.

- For each Conversion Correspondence $convert_i$ asserted by the designer, the system automatically asserts the Matching Correspondence

$$\begin{aligned} match_i([\vec{x}_1],[\vec{x}_2]) \leftarrow\ & convert_i([\vec{x}_1],[\vec{y}]), Equiv(\vec{x}_2, \vec{y}) \\ & \mathsf{through\ identity}(\vec{x}_2, \vec{y}) \end{aligned}$$

Observe that a new tuple $\vec{y}$ in the body of the correspondence is necessary to respect the binding pattern for $convert_i$. The program associated with $convert_i$ instantiates $\vec{y}$, and Identity compares the obtained value with $\vec{x}_2$.

- For each Conversion Correspondence $convert_i$ asserted by the designer and for each Matching Correspondence $match_j$ asserted by the designer or by default, the system automatically asserts the Merging Correspondence

$$\begin{array}{l} merge_{i,j}([\vec{x}_1],[\vec{x}_2],[\vec{x}_0]) \leftarrow match_i([\vec{x}_1],[\vec{x}_2]), convert_j(\vec{x}_1,\vec{x}_0) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{through none} \end{array}$$

Example 3. (Cont). For the domain SSNString the system automatically asserts, e.g. the Conversion Correspondence

$$\begin{array}{rl} convert_{\mathsf{SSNString}}([S_1],[S_2]) \leftarrow & Equiv([S_1],[S_2]) \\ & \mid S_1, S_2 :: \mathsf{SSNString} \\ & \text{through identity}(S_1, S_2) \end{array}$$

Similarly, since the adornment of the query defining the table FATHER_1 contains the annotations $Identify([N_f, D_f], F)$, N_f :: NameString, and D_f :: Date, the system automatically asserts, e.g. the Matching Correspondence

$$\begin{array}{rl} match_{\mathsf{NameString,Date}}([N_1,D_1],[N_2,D_2]) \leftarrow & Equiv([N_1],[N_2]), Equiv([D_1],[D_2]) \\ & \mid N_1, N_2 :: \mathsf{NameString}, D_1, D_2 :: \mathsf{Date} \\ & \text{through none} \end{array}$$

In addition, the designer may use already specified Reconciliation Correspondences to define new ones.

Example 6. The designer may want to define a Matching Correspondence between two tuples by using two already defined Conversion Correspondences, which convert to a common representation, and then by comparing the converted values. In this case, she could provide the following definition of the Matching Correspondence:

$$\begin{array}{rl} match_{X,Y}([\vec{x}],[\vec{y}]) \leftarrow & convert_{X,Z}([\vec{x}],[\vec{z}_1]), convert_{Y,Z}([\vec{y}],[\vec{z}_2]), Equiv([\vec{z}_1],[\vec{z}_2]) \\ & \text{through identity}([\vec{z}_1],[\vec{z}_2]) \end{array}$$

Observe that, in this case, the program associated with the Matching Correspondence is used only to check whether the converted values are identical.

Similarly, the designer could define a Merging Correspondence by reusing appropriate Conversion or Matching Correspondences that exploit a common representation, as shown in the following example.

Example 7. Suppose we want to merge prices in Italian Lira and Deutsche Mark into prices in US Dollars, and we have programs that allowed us to define the Conversion Correspondences $convert_{L,E}$ from Italian Lira to Euro, $convert_{M,E}$ from Deutsche Mark to Euro, and $convert_{E,D}$ from Euro to US Dollar. Then we can obtain the desired Merging Correspondence as follows:

$$\begin{array}{rl} merge_{L,M,D}([L],[M],[D]) \leftarrow & convert_{L,E}([L],[E_1]), convert_{M,E}([M],[E_2]), \\ & Equiv([E_1],[E_2]), convert_{E,D}([E_1],[D]) \\ & \text{through identity}([E_1],[E_2]) \end{array}$$

5. Specification of Mediators

As we said before, our goal is to provide support for the design of the mediator for T, i.e. the program that accesses the sources and loads the correct data into the relation T. In general, the design of mediators requires a sophisticated analysis of the data, which aims at specifying, for every relation in the Data Warehouse Schema, how the tuples of the relation should be constructed from a suitable set of tuples extracted from the sources. Mediator design is typically performed by hand and is a costly step in the overall Data Warehouse design process. The framework presented here is also based on a detailed analysis of the data and of the information needs. However, the knowledge acquired by such an analysis is explicitly expressed in the description of source and Data Warehouse relations, and in the Reconciliation Correspondences. Hence, such a knowledge can be profitably exploited to support the design of the mediators associated to the Data Warehouse relations.

Suppose we have decided to materialize a new relation T in the Data Warehouse, and let q be the adorned query associated to T. Our technique requires to proceed as follows.

(1) We determine how the data in T can be obtained from the data stored in already defined Data Warehouse relations, the data stored in source relations, and the data returned by the programs that perform conversion, matching, and merging associated to the Reconciliation Correspondences. This is done by looking for a *rewriting* of q in terms of the available adorned queries, i.e. a new query q' contained in q whose atoms refer to (i) the already available Data Warehouse relations, (ii) the source relations, (iii) the available conversion, matching, and merging predicates.
(2) We specify how to deal with tuples computed by the rewriting and possibly representing the same information. Typically we will have to combine tuples coming from different sources to obtain the tuples to store in the Data Warehouse relations.

The resulting query, which will be a disjunction of conjunctive queries, is the specification for the design of the mediator associated to T. The above steps are discussed in more detail below.

5.1. *Construction of the rewriting*

The computation of the rewriting is the most critical step of our method. Compared with other approaches, our query rewriting algorithm is complicated by the fact that we must consider both the constraints imposed by the Conceptual Model, and the Reconciliation Correspondences.

We describe now a rewriting algorithm suitable for our framework, that takes into account the above observation. First of all, we assume that the designer can specify an upper bound on the size of the conjunctive queries that can be used to compose the automatically generated query defining the mediator. Such an assumption is justified by considering that the size of the query directly affects the cost of materializing the Data Warehouse relation, and such a cost has to be limited due to several factors, e.g. the available time windows for loading and refreshment of the Data Warehouse.

The rewriting algorithm is based on generating *candidate rewritings* that are conjunctive queries limited in size by the bound specified by the designer, and verifying for each candidate rewriting

- whether the binding patterns for the correspondence predicates are respected; and
- whether the rewriting is contained in the original query.

Each candidate rewriting that satisfies the conditions above contributes as a disjunct to the rewriting.

Checking whether the binding patterns are respected can be done in a standard way.[4,22] On the other hand, verifying whether the rewriting is contained in the original query requires more attention, since we have to take into proper account the need of exploiting Reconciliation Correspondences in comparing different attribute values.

First of all, we have to pre-process the original adorned query as follows:

(1) By introducing explicit equality atoms, we re-express the query (and the adornment) in such a way that all variables in the body and the head (excluding the adornment) are distinct. Then we replace each equality atom $X = Y$ between variables X and Y denoting attribute values with $Equiv([X], [Y])$. In this way we reflect the fact that we want to compare attribute values modulo representation in the relations. Notice that the case of an equality atom between two variables, one denoting an object and one denoting a value, can be excluded, since in this case the query would be inconsistent w.r.t. the Conceptual Model.
(2) We add to the query the atoms resulting from the adornment, considering each annotation as an atom: $X :: V$ is considered as the atom $V(X)$, and $Identify$ is considered as a binary predicate that will be treated in a special way.

Also when constructing the candidate rewriting, we have to take into account that the only way to compare attributes is through Reconciliation Correspondences. Therefore we require that in a candidate rewriting all variables are distinct, with the exception of those used in Reconciliation Correspondences, which may coincide with other variables. Notice that this is not a limitation, since the system provides by default the obvious Reconciliation Correspondences for equality.

We expand each atom in the candidate rewriting with the body of the query defining the predicate in the atom, including the adornment as specified above. We have to take into account that atoms whose predicate is a correspondence predicate defined in terms of other correspondences, have to be expanded recursively, until all correspondence predicates have been substituted. We call the resulting query the *pre-expansion* of the candidate rewriting.

Then we add to the pre-expansion additional atoms derived by combining *Equiv* and *Identify* predicates as follows:

(1) We add the symmetric and transitive closure of the *Equiv* atoms in the pre-expansion, i.e. we recursively add for each $Equiv([\vec{x}], [\vec{y}])$ its symmetric $Equiv([\vec{y}], [\vec{x}])$, and for each pair $Equiv([\vec{x}], [\vec{y}])$ and $Equiv([\vec{y}], [\vec{z}])$ the transitive composition $Equiv([\vec{x}], [\vec{z}])$. This reflects the fact that *Equiv* represents equality modulo representation in different relations.
(2) We add $Equiv([X], [X])$ for each variable X denoting an attribute value. This is necessary to take into account that in the expanded query we have substituted equality atoms between variables denoting attribute values with *Equiv* atoms. Note that for tuples $\vec{x}$ we do not need to consider atoms of the form $Equiv([\vec{x}], [\vec{x}])$.
(3) We introduce equality atoms between variables denoting conceptual objects (not values) by adding $X = Y$ whenever we have either $Identify([\vec{x}], X)$ and $Identify([\vec{x}], Y)$, or $Identify([\vec{x}], X)$, $Identify([\vec{y}], Y)$ and $Equiv([\vec{x}], [\vec{y}])$. Indeed, *Equiv* reflects equality modulo representation, and *Identify* maps representations to the conceptual objects.
(4) We propagate *Identify* through *Equiv*, by adding $Identify([\vec{x}_1], X)$ whenever all variables in $\vec{x}$ appear in the head and we have $Equiv([\vec{x}_1], [\vec{x}_2])$ and $Identify([\vec{x}_2], X)$. This reflects the fact that $\vec{x}_1$ and $\vec{x}_2$ are two different representations of the same conceptual object X.

We call the resulting query the *expansion* of the candidate rewriting.

Then, to decide whether the candidate rewriting is *correct* and hence can contribute to the final rewriting, we check if its expansion is contained in the preprocessed query, taking into account the Conceptual Model.[43]

The rewriting of the original query is the union of all correct candidate rewritings. It can be shown that such a rewriting is *maximal* (w.r.t. query containment) within the class of rewritings that are unions of conjunctive queries that respect the bound specified by the designer. This rewriting can be refined by using query containment to eliminate those correct candidate rewritings that are contained in others.

Observe that the query rewriting algorithm relies on the assumption that an upper bound for the length of the conjunctive query is defined: hence, it is possible that, by increasing the bound, one obtains a better rewriting.[b]

The computational complexity of the rewriting algorithm is dominated by the complexity of the containment test, which can be done in 2EXPTIME[43] w.r.t. the size of the queries. However, one has to take into account that the size of queries can be neglected w.r.t. the size of data at the sources and at the Data Warehouse, and therefore the above bound does not represent a severe problem in practice.

[b]Even without a bound on the length of the conjuncts in the rewriting, unions of conjunctive queries cannot exactly capture the original query in general. Indeed, to do so one would need to consider rewritings expressed in a query language that is at least NP-hard in data complexity.[51]

Example 3. (Cont). We would like to obtain a specification in terms of a rewriting of the mediator that populates the relation $\mathsf{COMMONCHILD_{DW}}$. First, in the query defining $\mathsf{COMMONCHILD_{DW}}$, we eliminate common variables denoting attribute values by introducing *Equiv* atoms. We add the atoms resulting from the adornment, obtaining the pre-processed query:

$$\begin{aligned}\mathsf{COMMONCHILD_{DW}}(S_f, S_m) \leftarrow\ & \mathsf{Male}(F), \mathsf{ssn}(F, S1_f), \mathsf{CHILD}(F, C),\\ & \mathsf{Female}(M), \mathsf{ssn}(M, S1_m), \mathsf{CHILD}(M, C),\\ & Equiv([S_f], [S1_f]), Equiv([S_m], [S1_m]),\\ & \mathsf{SSNString}(S_f), \mathsf{SSNString}(S_m),\\ & Identify[S_f], Identify([S_m], M)\end{aligned}$$

Now, to obtain a rewriting of the above query we can exploit the queries associated to the relations FATHER_1 in source $\mathcal{S}_1$ and MOTHER_2 in source $\mathcal{S}_2$, taking into account that persons are represented differently in the two sources. Indeed, consider the following candidate rewriting:

$$\begin{aligned}R(S_f, S_m) \leftarrow\ & \mathsf{FATHER}_1(N_f, D_f, N_c, D_c), \mathsf{MOTHER}_2(S2_m, S_c),\\ & match_{person}([N_c, D_c], S_c),\\ & convert_{person}([N_f, D_f], S_f), convert_{\mathsf{SSNString}}([S2_m], [S_m])\end{aligned}$$

To check that R is a correct rewriting, we first substitute each atom in the body of the query by its definition, considering also the adornments, obtaining:

$$\begin{aligned}R(S_f, S_m) \leftarrow\ & \mathsf{Male}(F1), \mathsf{Person}(C1), \mathsf{CHILD}(F1, C1),\\ & \mathsf{name}(F1, N_f), \mathsf{dob}(F1, D_f), \mathsf{name}(C1, N_c), \mathsf{dob}(C1, D_c),\\ & \mathsf{NameString}(N_f), \mathsf{NameString}(N_c), \mathsf{Date}(D_f), \mathsf{Date}(D_c),\\ & Identify([N_f, D_f], F1), Identify([N_c, D_c], C1),\\[6pt] & \mathsf{Female}(M2), \mathsf{Person}(C2), \mathsf{CHILD}(M2, C2),\\ & \mathsf{ssn}(M2, S2_m), \mathsf{ssn}(C2, S_c),\\ & \mathsf{SSNString}(S2_c), \mathsf{SSNString}(S_c),\\ & Identify([S2_m], M2), Identify([S_c], C2),\\[6pt] & Equiv([N_c, D_c], S_c]),\\ & \mathsf{Person}(P2), \mathsf{name}(P3, N_c), \mathsf{dob}(P3, D_c), \mathsf{ssn}(P3, S_c),\\ & \mathsf{NameString}(N_c), \mathsf{Date}(D_c), \mathsf{SSNString}(S_c),\\ & Identify([N_c, D_c], P3), Identify([S_c], P3),\\[6pt] & Equiv([N_f, D_f], [S_f]),\\ & \mathsf{Person}(P4), \mathsf{name}(P4, N_f), \mathsf{dob}(P4, D_f), \mathsf{ssn}(P4, S_f),\\ & \mathsf{NameString}(N_f), \mathsf{Date}(D_f), \mathsf{SSNString}(S_f),\\ & Identify([N_f, D_f], P4), Identify([S_f], P4),\\[6pt] & Equiv([S2_m], [S_m]), \mathsf{SSNString}(S2_m), \mathsf{SSNString}(S_m)\end{aligned}$$

Then we add to the body of the query the following atoms resulting from propagating *identify* and *Equiv*

$$F_1 = P4, C1 = C2, C1 = P3, Identify([S_m], M2), Identify([S_f], F1)$$

plus an atom $Equiv([X],[X])$, for each variable X denoting an attribute value.

It is easy to check that the expanded candidate rewriting is indeed contained in the preprocessed query, which shows that the candidate rewriting is correct.

Example 4. (Cont). To obtain a specification of the mediator that populates the relation $\mathsf{SAMEINCOME_{DW}}$, we can pre-process the query defining such a relation obtaining:

$$\begin{aligned}\mathsf{SAMEINCOME_{DW}}(S_1,S_2,I) \leftarrow\ & \mathsf{Person}(P_1),\mathsf{ssn}(P_1,S_3),\mathsf{income}(P_1,I_1),\\ & \mathsf{Person}(P_2),\mathsf{ssn}(P_2,S_4),\mathsf{income}(P_2,I_2),\\ & Equiv([S_1],[S_3]),Equiv([S_2],[S_4]),Equiv([I_1],I_2]),\\ & \mathsf{SSNString}(S_1),\mathsf{SSNString}(S_2),\mathsf{IncomePerYear}(I),\\ & Identify([S_1],P_1),Identify([S_2],P_2)\end{aligned}$$

A candidate rewriting in terms of the queries defining INCOME_1 in source $\mathcal{S}_1$ and INCOME_2 in source $\mathcal{S}_2$ is:

$$\begin{aligned}R_1(S_1,S_2,I) \leftarrow\ & \mathsf{INCOME}_1(S_m,I_m),\mathsf{INCOME}_2(S_y,I_y),\\ & match_{income}([I_m],[I_y]),\\ & convert_{\mathsf{IncomePerYear}}([I_Y],[I]),\\ & convert_{\mathsf{SSNString}}([S_m],[S_1]),convert_{\mathsf{SSNString}}([S_y],[S_2])\end{aligned}$$

where $match_{income}$ is the Matching Correspondence automatically generated from $convert_{income}$, and the other Conversion Correspondences are automatically generated for the proper domains. We can again check that the expansion of the rewriting is contained in the pre-processed query, by taking into account that the schema, which the queries refers to, implies that Male and Female are sub-entities of Person.

The above correct candidate rewriting corresponds to perform the join between relations INCOME_1 and INCOME_2. Considering also the other possible joins between the two relations we obtain the following rewriting R:

$$\begin{aligned}R_1(S_1,S_2,I) \leftarrow\ & \mathsf{INCOME}_1(S_m,I_m),\mathsf{INCOME}_2(S_y,I_y),\\ & match_{income}([I_m],[I_y]),\\ & convert_{\mathsf{IncomePerYear}}([I_y],[I]),\\ & convert_{\mathsf{SSNString}}([S_m],[S_1]),convert_{\mathsf{SSNString}}([S_y],S_2])\\ & \mathsf{OR}\\ & \mathsf{INCOME}_2(S_y,I_y),\mathsf{INCOME}_1(S_m,I_m),\\ & match_{income}([I_m],[I_y]),\\ & convert_{\mathsf{IncomePerYear}}([I_y],[I]),\\ & convert_{\mathsf{SSNString}}([S_m],[S_1]),convert_{\mathsf{SSNString}}([S_y],[S_2])\\ & \mathsf{OR}\\ & \mathsf{INCOME}_1(S_m,I_m),\mathsf{INCOME}_1(S1_m,I1_m),\\ & match_{\mathsf{IncomePerYear}}([I_m],[I1_m]),\\ & convert_{income}([I_m],[I]),\\ & convert_{\mathsf{SSNString}}([S_m],[S_1]),convert_{\mathsf{SSNString}}([S1_m],[S_2])\end{aligned}$$

$$\begin{array}{l}
\mathsf{OR} \\
\mathsf{INCOME}_2(S_y, I_y), \mathsf{INCOME}_2(S1_y, I1_y), \\
match_{\mathsf{IncomePerYear}}([I_y], [I1_y]), \\
convert_{\mathsf{IncomePerYear}}([I_y], [I]), \\
convert_{\mathsf{SSNString}}([S_y], [S_1]), convert_{\mathsf{SSNString}}([S1_y], [S_2])
\end{array}$$

Observe that it may happen that no rewriting for the query exists. One reason for this may be that the available relations do not contain enough information to populate the relation defined by the query. In this case a further analysis of the source is required. It may also be the case that the relations do in fact contain the information needed, but the available reconciliation programs are not able to convert the data in a representation suitable to answer the query. Formally, this is reflected in the fact that appropriate Reconciliation Correspondences are missing. In this case our rewriting algorithm can be used by the designer to acquire indications on which are the required Reconciliation Correspondences, and hence the associated programs that must be added to generate the mediator.

5.2. *Combining tuples coming from different sources*

Since the rewriting constructed as specified above is in general a disjunction, we must address the problem of combining the results of several queries. A similar problem arises in other approaches,[14] where the result of approximate joins may require a specification of a construction operation. In order to properly define the result of the query, we introduce the notion of *combine-clause*. In particular, if the query r computed by the rewriting is constituted by more than one disjunct, then the algorithm associates to r a suitable set of so-called merging clauses, taking into account that the answers to the different disjuncts of the query may contain tuples that represent the same real world entity or the same value. A *combine-clause* is an expression of the form

$$\begin{array}{l}
\mathsf{combine}\ \textit{tuple-spec}_1\ \mathsf{and} \cdots \mathsf{and}\ \textit{tuple-spec}_n \\
\mathsf{such\ that}\ \textit{combination-condition} \\
\mathsf{into}\ \textit{tuple-spec}_1\ \mathsf{and} \cdots \mathsf{and}\ \textit{tuple-spec}_{t_m}
\end{array}$$

where *tuple-spec*$_i$ denotes a tuple returned by the ith disjunct of r, *combination-condition* specifies how to combine the various tuples denoted by *tuple-spec*$_1$,..., *tuple-spec*$_n$, and *tuple-spec*$_{t_1}$,..., *tuple-spec*$_{t_m}$ denote the tuples resulting from the combination that are inserted in the relation defined by the query.

We observe that the rewriting algorithm is able to generate one combine-clause template for each pair of disjuncts that are not disjoint. Starting from such templates, the designer may either specify the `such that` and the `into` parts, depending on the intended semantics, or change the templates in order to specify a different combination plan (for example for combining three disjuncts, rather than three pairs of disjuncts).

5.3. *Refining the rewriting*

As already mentioned, the rewriting returned by the algorithm can be refined by eliminating certain conjuncts from the union of conjunctive queries according to suitable criteria for populating Data Warehouse relations. In particular, such criteria may be determined by factors that affect the quality of the data in the source relations and in the Data Warehouse, such as completeness, accuracy, confidence, freshness, etc. In practice, a convenient way to characterize such quality factors is by providing *ad hoc* information in a meta-repository.[52–54]

While the goal of the present work is to provide a language to represent transformations, thus not requiring an explicit introduction of *meta-level descriptions*, there are certain aspects of the data integration process that can take advantage of a meta-information approach. More specifically, together with the actual data stored in the source, the wrapper can provide metadata, both at relation-level and at tuple-level, which can be suitably exploited in refining the rewriting. As an example, consider the case where a materialized Data Warehouse is available. The materialized view can be described as any other source, being always preferred with respect to other sources because of its higher accuracy and confidence.

6. Related Work

In this section, we address related approaches, pointing out the novel features of the approach presented in this paper.

To put the approach presented in this paper in perspective with the literature, a couple of preliminary remarks are in order. The work on information integration has rapidly evolved in recent years from the target of multiple heterogenous databases to the more general framework comprising information sources available from the Web or through the intranets (HTML, XML and textual data). While in principle the proposed approach could be applied also to these semistructured and unstructured kind of sources, provided that suitable wrappers are available, we have not dealt with this kind of sources, assuming that each source is wrapped into relational data. More specifically, we do not address or propose a new (semistructured) data model as done by others.[20,55]

The second remark concerns the need for distinguishing the conceptual level, where the semantics of data are captured, from the logical level, where the actual structure of the relational database is described in Ref. 42. In this way, the request for data is formalized at the conceptual level, namely without referring to the actual representation and format of the data objects involved in the query (except for the result tuples). This capability, which has been emphasized in the DWQ project, both to meet the requirements of the target application environments and to exploit the capabilities of knowledge representation languages, is generally not provided, also in the declarative approaches characterized below.

Various software architectures and system organizations have been proposed to deal with the heterogeneity of the sources and the process for integrating the data.[6]

In particular, we refer to mediated architectures,[3] which have been very successful in separating the aspects that are specific of each source from the integration process consisting in reconciling and combining the information extracted from different sources.[4]

Several other elements for classification of the approaches to information integration and, more specifically, mediated query systems, can be considered.[56]

The first feature for classification is whether the data are materialized in a "universal" database or Data Warehouse, or else the data are kept in the sources, in which case the approach is called virtual. Our approach to information integration can be applied to acquire the data from the sources, as we have assumed in the examples presented in the paper. However, it can also be applied to handle data requests issued to the Data Warehouse (to populate Data Marts or Secondary Data Warehouses). In this case, the materialized views can be treated as an additional source, that is generally preferred in case it can fulfill the request.

In the introduction we have mentioned another feature for classifying the proposed approaches to data integration: global-as-view versus local-as-view. In particular, we have argued that typical data integration constraints in a Data Warehouse context lead to consider the local-as-view as more appropriate, given the need to provide the Data Warehouse with an enterprise integrated view of the data. In fact, the local-as-view approach is another distinguishing feature of the present work, since the local-as-view is not pursued by the majority of the proposals for information integration, whose focus is in providing answers to specific information needs, without explicit introduction of an enterprise view of the data also in a Data Warehouse framework.[16–17]

One feature that we consider especially significant is whether the relationship between the Data Warehouse and the data sources is procedurally defined, namely specific to each query, or else the system can provide for mediators for arbitrary queries. We have called the latter approach declarative, since, as we have shown in the paper, mediators can be generated by the system from the specification.

An example of the procedural approach can be found in a recent methodology for extracting, comparing and matching data and objects located in different sources.[20] The proposed methodology is based on the Object Exchange Model, which requires the explicit semantic labeling of the objects, to support object exchange, and emphasizes the need for a tight interaction between the system and the user. However, the method remains of procedural nature, since it requires the user to build and maintain the relationship between the sources and the Data Warehouse on a query-by-query basis. The same remark applies also to the subsequent work,[57] where a rule-based language for Mediator Specification is proposed, but each rule defining a mediator specifically refers to the sources to be used to materialize the data.

The approach to information integration developed in the H2O project[5,18,19] provides a wide spectrum of solutions from the fully materialized to the hybrid, to the fully virtual ones, while still providing a global-as-view approach. Integration in H20 is also based on a object-model, but aims at a more declarative specification of data integration. This is achieved through the Integration Specification Language (ISL) which allows one to specify portions of the source schemas, the criteria for matching objects from different source classes and the specification of the classes to be associated with a mediator. The system can generate mediators from the specification by implementing the specified matching criteria. The specification of mediators, as well as the mediator generation process, is centered on the notion of object matching, requiring the definition of specific matching criteria for all the sources. The need for explicitly providing the matching criteria makes the addition of a new source a non-trivial task, since it requires an adaptation of the matching criteria for all the mediators that can access the new source.

Another recent approach is the rule-based one.[58] This approach is actually pursued both in dealing with complex transformations due to diversity in the data model, such as for example HTML and OODB,[55] and to schema matching.[59] In the former case, the goal of the rule specification is to generate a customized instantiation of general purpose translation tools. In the latter case, the idea is that the system automatically finds the matching between two schemas, based on a set of rules that specify how to perform the matching. As in previously cited approaches, the rules provide a pattern specifying the sources where the data to be gathered can be found.

Approaches more declarative in nature have also been proposed.[60] Suitable data structures for reconciling different representations of the same data are represented in a *context theory*, which is used by the system to transform the queries for gathering the data from the various sources. In such a declarative approach, the user is not directly concerned with the identification and resolution of semantic conflicts when formulating the requests for data. Rather, once the specification of the sources is available, conflicts are detected by the system, and conversion and filtering are automatically enforced. However, the method still follows the global-as-view approach, and the context theory is used as a description of reconciled data structures, rather than as the conceptual model of the corporate data.

Compared with the above cited approaches, the present work shows several interesting features. Mediator generation is accomplished by first rewriting the query in terms of the sources, and then providing for the necessary transformations and merging of the data. A tight connection between the process for identifying data sources, and the definition of the actual extraction procedure is very relevant from a practical viewpoint, since integration cannot be realized without considering both aspects at the same time. Secondly, the specification can be provided incrementally with respect to the addition of new sources, since it is not centered on the criteria for matching the objects in each source, but it aims at providing description of relationships between the sources and the enterprise model.

Finally, we mention two recent works that are specific to conflict resolution and data cleaning, two aspects that are related to data integration but are not specifically addressed by the present work.

Recently, a method for resolving conflicts that arise when integrating data coming from different sources has been proposed.[61] The approach, developed within the framework of the AURORA system, is based on a declarative specification of the relationship between source schemas and a global schema, by means of so-called registrations. Moreover, a set of attributes, called plug-in identifiers, is used for object-matching. Integration is accomplished in two steps: first, all the matching tuples are collected; afterwards, the conflicts are either solved with a specific conflict resolution function or by means of the conflict tolerant query model which can provide answers with different levels of confidence. While this approach focuses on the notion of tolerance to define the transformations and optimize them, it does not address the identification and selection of the sources that are relevant for a given query.

Finally, the overall process of data integration has also been analyzed from a data cleaning perspective.[14] In particular, a distinction is made between the so-called object identity problem,[62] which amounts to finding matching tuples in the presence of various kinds of errors and anomalous data, and the verification and merging of the resulting tuples. Since the emphasis is on data cleaning rather than on schematic differences, the specification of the sources that must be used to construct a specific table is done in and *ad hoc* way. On the other hand, we do not deal with techniques for finding and verifying the matches, which could be embodied in our framework as well.

7. Conclusions

We have described a new approach to data integration in Data Warehousing. The approach is based on the availability of a Conceptual Model of the corporate data, and on the idea of declaratively specifying several types of reconciliation correspondences between data in different sources. Such correspondences are used by a query rewriting algorithm that supports the task of designing the correct mediators for the loading of the materialized views of the Data Warehouse.

The proposed framework for Data Integration has been implemented within a tool delivered by the DWQ project.[6,9] The overall tool supports the Data Warehouse design process according to the DWQ methodology.[42] In all phases of the design process, the tool allows for the insertion, retrieval, and update of the meta-data relevant for the design. The tool makes use of Concept Base[63] as the central repository. The query and update functionalities of Concept Base are used to facilitate access to the information in the repository. The tool has been developed under Solaris as a Java client of the Concept Base system, and has been used in a case study carried out in cooperation with Telecom Italia.[44]

The work reported in this paper concentrates on the integration of sources in Data Warehousing. Although not addressing all aspects of loading and updating the Data Warehouse, it provides a principled approach to one of the key aspects of information integration, namely supporting all the tasks that require access to the sources. The approach is suited for a range of scenarios from virtual warehouses to materialized ones.

Regarding the realm of the proposed approach, the following aspects deserve further investigation and we plan to address them in future work.

Our methodology does not provide support as how to realize the actual programs matching objects in different sources. Note, however, that techniques for this task[14,64] can be easily integrated with our approach.

The query derived by our rewriting algorithm is a complex query, where usual database operations are interleaved with programs performing reconciliation operations. Optimizing such a query is a very interesting and important aspect, that we have not addressed in this paper.[14]

In our approach we have concentrated on sources that are expressed in the relational model, and that contain elementary data. It is interesting to extend the framework to more complex scenarios, such as semistructured sources and sources with aggregate data. Basic results for query rewriting in semistructured data have been obtained recently.[32]

Acknowledgments

This work was supported in part by the ESPRIT LTR Project No. 22469 DWQ (Foundations of Data Warehouse Quality). We thank all participants of the DWQ project, who have contributed to the basic ideas reported in this paper, and Telecom Italia for the support in validating the approach. Also, we are grateful to Antonio Baffigo and Domenico Lembo for their work on the development of the data integration tool.

References

1. W. H. Inmon, *Building the Data Warehouse*, 2nd end. (John Wiley & Sons, 1996).
2. R. Hull, Managing semantic heterogeneity in databases: A theoretical perspective, *Proc. PODS'97*, 1997.
3. G. Wiederhold, Mediators in the architecture of future information systems, *IEEE Comput.* **25**, 3 (1992) 38–49.
4. J. D. Ullman, Information integration using logical views, *Proc. ICDT'97*, *LNCS* (Springer-Verlag, 1997) 19–40.
5. R. Hull and G. Zhou, A framework for supporting data integration using the materialized and virtual approaches, *Proc. ACM SIGMOD* (1996) 481–492.
6. M. Jarke, M. Lenzerini, Y. Vassiliou and P. Vassiliadis (eds.), *Fundamentals ofData Warehouses* (Springer-Verlag, 1999).
7. B. Devlin, *Data Warehouse: From Architecture to Implementation* (Addison Wesley Publ. Co., Reading, MA, 1997).
8. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Source integration in data warehousing, *Proc. DEXA'98* (IEEE Computer Society Press, 1998) 192–197.
9. *The Data Warehouse Quality Project* http://www.dbnet.ece.ntua.gr/~dwq/, 1999.
10. C. Batini, M. Lenzerini and S. B. Navathe, A comparative analysis of methodologies for database schema integration, *ACM Computing Surveys* **18**, 4 (1986) 323–364.
11. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Source integration in data warehousing, Technical Report DWQ-UNIROMA-002, DWQ Consortium, October 1997.

12. A. Y. Levy, A. O. Mendelzon, Y. Sagiv and D. Srivastava, Answering queries using views, *Proc. PODS'95* (1995) 95–104.
13. S. Abiteboul and O. Duschka, Complexity of answering queries using materialized views, *Proc. PODS'98* (1998) 254–265.
14. H. Galhardas, D. Florescu, D. Shasha and E. Simon, An extensible framework for data cleaning, Technical Report 3742, INRIA, Rocquencourt, 1999.
15. J. Hammer, H. Garcia-Molina, J. Widom, W. Labio and Y. Zhuge, The Stanford data warehousing project, *IEEE Bull. Data Eng.* **18**, 2 (1995) 41–48.
16. J. Widom, Special issue on materialized views and data warehousing, *IEEE Bull. Data Eng.* **18**, 2, 1995.
17. A. Gupta and I. S. Mumick, Maintenance of materialized views: Problems, techniques and applications, *IEEE Bull. Data Eng.* **18**, 2 (1995) 3–18.
18. G. Zhou, R. Hull and R. King, Generating data integration mediators that use materializations, *J. Intell. Inf. Syst.* **6** (1996) 199–221.
19. G. Zhou, R. Hull, R. King and J.-C. Franchitti, Using object matching and materialization to integrate heterogeneous databases, *Proc. CoopIS'95* (1995) 4–18.
20. Y. Papakonstantinou, H. Garcia-Molina and J. Widom, Object exchange across heterogeneous information sources, *Proc. ICDE'95* (1995) 251–260.
21. C. H. Goh, S. Bressan, S. E. Madnick and M. D. Siegel, Context interchange: New features and formalisms for the intelligent integration of information, *ACM Trans. Inf. Sys.* **17**, 3 (1999) 270–293.
22. A. Rajaraman, Y. Sagiv and J. D. Ullman, Answering queries using templates with binding patterns, *Proc. PODS'95*, 1995.
23. D. Srivastava, S. Dar, H. V. Jagadish and A. Levy, Answering queries with aggregation using views, *Proc. VLDB'96* (1996) 318–329.
24. F. N. Afrati, M. Gergatsoulis and T. Kavalieros, Answering queries using materialized views with disjunction, *Proc. ICDT'99*, *Lecture Notes in Computer Science* (Springer-Verlag, 1999) 435–452.
25. O. M. Duschka and M. R. Genesereth, Answering recursive queries using views, *Proc. PODS'97* (1997) 109–116.
26. C. Beeri, A. Y. Levy and M.-C. Rousset, Rewriting queries using views in description logics, *Proc. PODS'97* (1997) 99–108.
27. J. Gryz, Query folding with inclusion dependencies, *Proc. ICDE'98* (1998) 126–133.
28. O. M. Duschka and A. Y. Levy, Recursive plans for information gathering, *Proc. IJCAI'97* (1997) 778–784.
29. S. Cohen, W. Nutt and A. Serebrenik, Rewriting aggregate queries using views, *Proc. PODS'99* (1999) 155–166.
30. S. Grumbach, M. Rafanelli and L. Tininini, Querying aggregate data, *Proc. PODS'99* (1999) 174–184.
31. G. Grahne and A. O. Mendelzon, Tableau techniques for querying information sources through global schemas, *Proc. ICDT'99*, *Lectures Notes in Computer Science* (Springer-Verlag, 1999) 332–347.
32. D. Calvanese, G. De Giacomo, M. Lenzerini and M. Y. Vardi, Rewriting of regular expressions and regular path queries, *Proc. PODS'99* (1999) 194–204.
33. D. Calvanese, G. De Giacomo, M. Lenzerini and M. Y. Vardi, Answering regular path queries using views, *Proc. ICDE 2000* (2000) 389–398.
34. D. Calvanese, G. De Giacomo, M. Lenzerini and M. Y. Vardi, Query processing using views for regular path queries with inverse, *Proc. PODS 2000* (2000) 58–66.
35. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati. A principled approach to data integration and reconciliation in data warehousing, *Proc. DMDW'99*,

CEUR Electronic Workshop Proceedings. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-19/, 1999.

36. D. Calvanese, G. De Giacomo and R. Rosati, Data integration and reconciliation in data warehousing: Conceptual modeling and reasoning support, *Network Inf. Sys.* **2** (1999) 4.
37. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Description logic framework for information integration, *Proc. KR'98* (1998) 2–13.
38. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Schema and data integration methodology for DWQ, Technical Report DWQ-UNIROMA-004, DWQ Consortium, September 1998.
39. A. Borgida, Description logics in data management, *IEEE Trans. Knowledge Data Eng.* **7**, 5 (1995) 671–682.
40. F. M. Donini, M. Lenzerini, D. Nardi and A. Schaerf, Reasoning in description logics, ed. G. Brewka, *Principles of Knowledge Representation*, Studies in Logic, Language and Information (CSLI Publications, 1996) 193–238.
41. T. Catarci and M. Lenzerini, Representing and using interschema knowledge in cooperative information systems, *J. Intell. Cooperative Inf. Sys.* **2**, 4 (1993) 375–398.
42. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Information integration: Conceptual modeling and reasoning support, *Proc. CoopIS'98* (1998) 280–291.
43. D. Calvanese, G. De Giacomo and M. Lenzerini, On the decidability of query containment under constraints, *Proc. PODS'98* (1998) 149–158.
44. S. M. Trisolini, M. Lenzerini and D. Nardi, Data integration and warehousing in Telecom Italia, *Proc. ACM SIGMOD* (1999) 538–539.
45. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Use of the reconciliation tool at Telecom Italia, Technical Report DWQ-UNIROMA-007, DWQ Consortium, October 1999.
46. R. Bayardo and R. Schrag, Using CSP look-back techniques to solve real-world SAT instances, *Proc. AAAI'97* (AAAI Press/The MIT Press, 1997) 203–208.
47. H. Zhang, SATO: An efficient propositional prover, *Proc. CADE'97*, Lectures Notes in Computer Science, 1997.
48. J. Crawford and L. Auton, Experimental results on the crossover point in random 3SAT, *Artificial Intelligence* **81**, 1&2 (1996) 31–57.
49. D. Theodoratos, S. Ligoudistianos and T. Sellis, Designing the global Data Warehouse with SPJ views, *Proc. CAiSE'99* (1999).
50. A. Sheth and V. Kashyap, So far (schematically) yet so near (semantically) *Proc. IFIP DS-5 Conf. Semantics Interoperable Database Sys.* (Elsevier, 1992).
51. D. Calvanese, G. De Giacomo, M. Lenzerini and M. Y. Vardi, View-based query processing and constraint satisfaction, *Proc. LICS 2000* (2000) 361–371.
52. P. A. Bernstein and T. Bergstraesser, Meta-data support for data transformations using microsoft repository, *IEEE Bull. Data Eng.* **22**, 1 (1999) 9–14.
53. S. Heiler, W.-C. Lee and G. Mitchell, Repository support for metadata-based legacy migration, *IEEE Bull. Data Eng.* **22**, 1 (1999) 37–42.
54. J. M. Hellerstein, M. Stonebraker and R. Caccia, Independent, open enterprose data integration, *IEEE Bull. Data Eng.* **22**, 1 (1999) 43–49.
55. S. Abiteboul, S. Cluet and T. Milo, Correspondence and translation for heterogeneous data, *Proc. ICDT'97* (1997) 351–363.
56. R. Domenig and K. R. Dittrich, An overview and classification of mediated query systems, *SIGMOD Record* **28**, 3 (1999) 63–72.
57. Y. Papakonstantinou, H. Garcia-Molina and J. D. Ullman, MedMaker: A mediation

system based on declarative specifications, ed. S. Y. W. Su, *Proc. ICDE'96* (1996) 132–141.
58. S. Abiteboul, S. Cluet, T. Milo, P. Mogilevsky, J. Simeéon and S. Zohar, Tools for data translation and integration, *IEEE Bull. Data Eng.* **22**, 1 (1999) 3–8.
59. S. Cluet, C. Delobel, J. Simeéon and K. Smaga, Your mediators need data conversion!, *Proc. VLDB'98* (1999) 279–290.
60. C. H. Goh, S. E. Madnick and M. Siegel, Context interchange: Overcoming the challenges of large-scale interoperable database systems in a dynamic environment, *Proc. CIKM'94* (1994) 337–346.
61. L. L. Yang and M. T. Ozsu, Conflict tolerant queries in AURORA, *Proc. CoopIS'99* (1999) 279–290.
62. W. W. Cohen, Some practical observations on the integration of web information, *Proc. WebDB'99* (1999) 55–60.
63. M. Jarke, R. Gallersdoerfer, M. Jeusfeld, M. Staudt and S. Eherer, Conceptbase — A deductive object manager for meta databases, *J. Intell. Inf. Sys.* **4** (1995) 2.
64. Z. Kedad and E. Métais, Dealing with semantic heterogeneity during data integration, *ER99*, *Lecture Notes in Computer Science*, **1728** (Springer-Verlag, 1999) 325–339.