RESEARCH

Data integration and reconciliation in Data Warehousing: Conceptual modeling and reasoning support

Diego Calvanese — Giuseppe De Giacomo — Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
Email: {calvanese,degiacomo,rosati}@dis.uniroma1.it
Web: http://www.dis.uniroma1.it/~{calvanese,degiacomo,rosati}

ABSTRACT. *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 and reconciliation, based on a conceptual representation of the Data Warehouse application domain. The main idea is to declaratively specify suitable matching, conversion, and reconciliation operations to be used in order to solve possible conflicts among data in different sources. Such a specification is provided in terms of the conceptual model of the application, and is effectively used during the design of the software modules that load the data from the sources into the Data Warehouse.*

KEY WORDS : *conceptual modeling, Data Warehousing, data integration, data reconciliation.*

1. Introduction

Information Integration is the problem of acquiring data from a set of sources that are available for the application of interest. The typical architecture of an integration system is described in terms of two types of modules: wrappers and mediators [WIE 92, ULL 97]. 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 merge 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. This problem has recently become a central issue in several contexts, including multi-database systems, Data Warehousing and information gathering from the Web.

The constraints that are typical of Data Warehouse applications restrict the large spectrum of approaches that are being proposed [HUL 97, INM 96, JAR 99]. First, while the sources on the Web are often external, in a Data Warehouse they are mostly internal to the organization. Second, a Data Warehouse should reflect the informational needs of an organization, and should therefore be defined in terms of a global, corporate view of data. Third, such a view should be provided in terms of conceptual representation mechanism that is able to abstract from the physical and logical organization 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 with respect to 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 conceptual model, 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, integrated view of the data of the organization is available.

The above considerations motivate the approach to information integration proposed in [CAL 98d], whose distinguishing feature is to exploit the possibility of representing the conceptual level of a Data Warehouse in a very expressive language and use reasoning tools to support the Data Warehouse construction, maintenance and evolution. In fact, the idea is to balance the effort of building a conceptual model of the Data Warehouse by improving the capabilities of the system in maintaining the Data Warehouse and support the incremental addition of information sources. The proposed approach follows a local as view paradigm, by explicitly requiring an enterprise conceptual model which is therefore regarded as a unified view of the data available within the organization.

Most of the work on integration has been concerned with the intensional/schema level, while less attention has been devoted to the problem of data integration at the extensional level. Integration of data is, nonetheless, at the heart of Data Warehousing [INM 96]. When data passes from the application-oriented operational environment to the 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. Thus, in the context of a Data Warehouse, data integration and reconciliation is the process of acquiring data from the sources and making them available within the Warehouse.

Given a request for data (e.g., for materializing a new relation in the Data Warehouse), which is formulated in terms of the global view of the corporate data, (i.e., not the language of the sources, but of the enterprise), there are several steps that enable 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 [ABI 98, LEV 95].

2. Decomposition of the user request into queries to individual sources that would return the data of interest.

3. Interpretation of the data provided by a source. Interpreting data can be regarded as the task of casting them into a common representation, which can thereafter be used to manipulate the data.

4. Merging of the data. The data returned by various sources need to be combined to provide the Data Warehouse with the requested information.

In commercial environments for Data Warehouse design and management the above tasks are taken care of through ad-hoc components [JAR 99]. In general, such a component provides 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, it generally provides 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 [HAM 95, WID 95, GUP 95, HUL 96, ZHO 96, ZHO 95, PAP 95, GOH 94]. For example, a methodology for extracting, comparing and matching data and objects located in different sources is described in [PAP 95]. The 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 approach proposed in [GOH 94] is more declarative in nature. 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 as appropriate 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.

In this paper we present an approach to data integration and reconciliation based on a conceptual representation of the Data Warehouse application domain. Specifically, the method for data integration and reconciliation builds upon and extends the work in [CAL 98d], therefore relying on the availability of a Conceptual Model to declaratively represent the relationship between the sources and the Data Warehouse. The declarative approach is further pursued in the task of data integration and reconciliation, where the system is given a declarative description of the data in the sources and provides automatic support in satisfying the data requests for populating the Data Warehouse.

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.

— It allows the designer to declaratively specify several types of correspondences between data in different schemas (either source schemas or Data Warehouse schema). Three types of Interschema Correspondences are taken into account, namely Conversion, Matching, and Reconciliation Correspondences.

— It uses such correspondences for supporting the task of specifying the correct mediators for the loading of the materialized views of the Data Warehouse.

Our methodology relies on a novel 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 Interschema Correspondences.

The paper is organized as follows. In Section 2, we summarize the relevant features of the proposed approach to information integration. Section 3 illustrates the method we use to describe the content of the sources at the logical level. Section 4 is devoted to a discussion of the meaning and the role of Interschema Correspondences. Section 5 describes the query rewriting algorithm at the basis of our approach to the design of mediators. Section 6 concludes the paper.

2. Integration framework

In this section we briefly describe the proposed data integration framework which allows one to explicitly model data and information needs – i.e., a specification of the data that the Data Warehouse provides to the user – at various levels [CAL 98b, CAL 98d, CAL 98c]:

— The *conceptual level* contains a conceptual representation of the corporate data.

— The *logical level* 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* contains a store for the materialized data, wrappers for the sources and mediators for loading the materialized 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.

We briefly describe the conceptual and logical levels, referring to the abstract architecture depicted in Figure 1.

The Conceptual Model is 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. The conceptual model is expressed in terms of an enriched *Entity-Relationship model* in which complex entity and relationship expressions can be constructed and used, and in which interdependencies between elements of different sources and of the enterprise are captured using *intermodel assertions* [CAL 98b, CAT 93]. Intermodel assertions provide a simple and effective declarative mechanism to express the dependencies that hold between entities (i.e. classes and relationships) in different models [HUL 97]. The use of intermodel assertions allows for an incremental approach to the integration of the conceptual models of the sources and of the enterprise. Due to space limitations, we cannot consider this aspect in further detail, and refer the interested reader to [CAL 98b].

The conceptual representation contains, besides entities and relationships, also a description of *domains*, which are used to typify the attributes of entities and relationships. Rather than considering only concrete domains, such as strings, integers, and reals, our approach is based on the use of *abstract domains*. An abstract domain may have an underlying concrete domain, but allows the designer to distinguish between the different meanings that a value of the concrete domain may have. Additionally, also Boolean combinations of domains and the possibility to construct an ISA hierarchy between domains are supported.

Example 1 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 `MoneyInLire` while the domain of attribute A_2 is `MoneyInEuro`, both of which have `real` as the underlying concrete domain. In this way, it becomes possible to specify declaratively the difference between values of the two attributes, and take into account such knowledge for loading data from the source to the Data Warehouse. ■

We provide an example of the form of the Conceptual Model, and refer to [CAL 98b] for a more detailed description of the adopted formalism.

Example 2 As our running example we consider an enterprise and two sources containing information about contracts between customers and departments for services, and about registration of customers at departments. Source 1 contains information about customers registered at public-relations departments. Source 2 contains information about contracts and complete information about services. Such situation

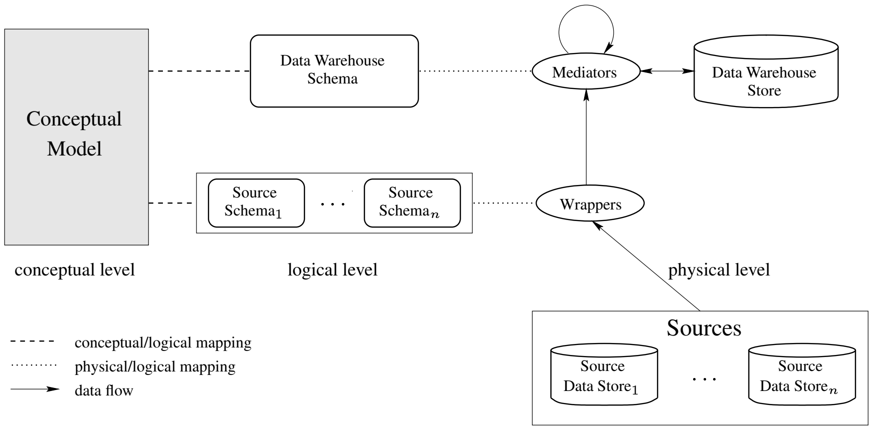

Figure 1. *Architecture for Data Integration*

can be represented by means of the ER diagrams shown in Figure 2, together with the following intermodel assertions ($\sqsubseteq$ represents ISA while $\equiv$ represents equivalence):

$$\begin{array}{rcl}
\texttt{Department}_1 & \equiv & \texttt{PrDept}_0 \\
\texttt{REG-AT}_1 & \sqsubseteq & \texttt{REG-AT}_0 \\
\texttt{Customer}_1 & \equiv & \texttt{Customer}_0 \sqcap \\
 & & (\geq 1\,[\$1](\texttt{REG-AT}_0 \sqcap (\$2\!:\texttt{PrDept}_0))) \\
\texttt{Customer}_0 \sqcap (\geq 1\,[\$1]\texttt{CONTRACT}_0) & \sqsubseteq & (\geq 1\,[\$1]\texttt{PROMOTION}_1) \\
\texttt{Customer}_2 & \sqsubseteq & \texttt{Customer}_0 \sqcap (\geq 1\,[\$1]\texttt{CONTRACT}_0) \\
\texttt{Department}_2 & \sqsubseteq & \texttt{Department}_0 \\
\texttt{Service}_2 & \equiv & \texttt{Service}_0 \\
\texttt{CONTRACT}_2 & \sqsubseteq & \texttt{CONTRACT}_0 \\
\texttt{Customer}_1 & \equiv & \texttt{Customer}_2 \\
\texttt{Department}_1 & \equiv & \texttt{Department}_2
\end{array}$$

and the following domain hierarchy:

$$\begin{array}{rcl}
\texttt{PersNameString} & \sqsubseteq & \texttt{String} \\
\texttt{DeptNameString} & \sqsubseteq & \texttt{String} \\
\texttt{SSNString} & \sqsubseteq & \texttt{String} \\
\texttt{DeptCodeInteger} & \sqsubseteq & \texttt{Integer} \\
\texttt{ServNoInteger} & \sqsubseteq & \texttt{Integer}
\end{array}$$

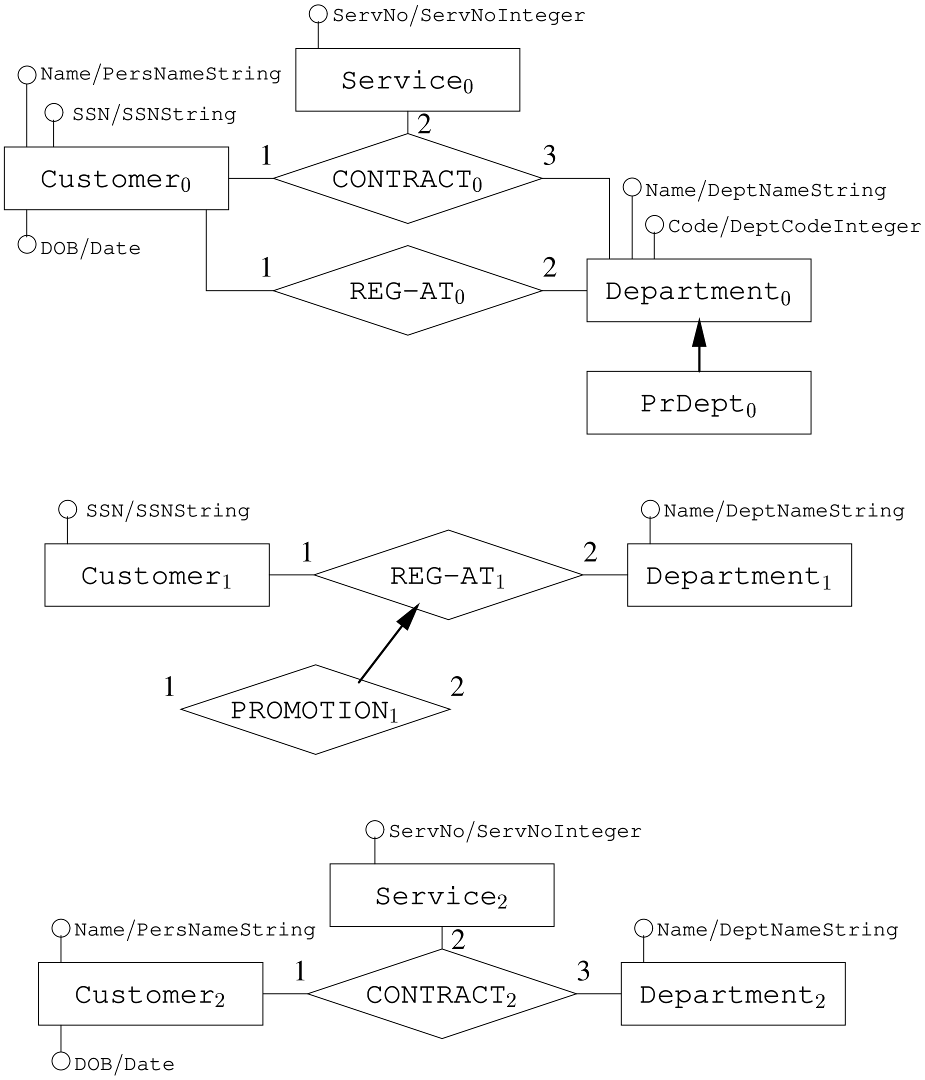

Figure 2. *Conceptual model of the application of Example 2*

■

At the logical level, the logical content of each source, called the *Source Schema* (see Section 3), is provided in terms of a set of relational tables using the relational model. The link between the logical representation and the conceptual representation of the source is formally defined by associating with each table a query that describes its content in terms of a query over the Conceptual Model. In other words, the logical content of a source table is described in terms of a view over the Conceptual Model.

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 relational tables. In particular, we assume that all attributes in the tables are of interest to the Data Warehouse application (attributes that are not of interest are hidden by the wrapper). The logical content of the materialized views constituting the Data Warehouse, called the *Data Warehouse Schema* (see Section 4), is provided in terms of a set of relational tables. Similarly to the case of the sources, each table of the Data Warehouse Schema is described in terms of a view over the Conceptual Model. As we said before, the way in which a view is actually materialized, starting from the data in the sources, is specified by means of mediators.

In such a framework, we have devised suitable inference techniques, which allow for carrying out several reasoning services on both the conceptual representation, such as inferring inclusion between entities and relationships, satisfiability of entities, etc. [CAL 98d], and the logical representation, such as query containment [CAL 98a], which is at the basis of query rewriting. The possibilities offered by such reasoning tools are used in the accomplishment of several activities concerning both the design and the operation of the Data Warehouse.

3. Source schema description

In this section we focus on the specification of the logical schemas of the sources. Such schemas are intended to provide a structural description of the content of the sources, which are encapsulated by suitable wrappers.

We describe a source S by associating to each relational table T of S an *adorned query* that is constituted by a head, a body, and an adornment:

— The *head* defines the relational schema of the table in terms of a name, and the number of columns.

— The *body* describes the content of the table in terms of a query over the Conceptual Model.

— The *adornment* declares the domains of the columns of the table, and also which are the attributes of the table that are used to identify an entity of the Conceptual Model.

We now present in detail the notions introduced above.

3.1. *Query over the conceptual model*

Generally speaking, the connection to the Conceptual Model is established by defining each table as a relational query over the elements of the Conceptual Model.

A query q for a Conceptual Model $\mathcal{M}$ is a non-recursive Datalog query, written in the form:

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

where each $conj_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i)$ is a conjunction of *atoms* or *negated atoms*, and $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i$ are all the variables appearing in the conjunct. Each atom is either of the forms $E(t)$ or of the form $\mathbf{R}(\vec{\mathbf{t}})$, where $\vec{\mathbf{t}}$, t, and t' are variables in $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i$ or constants, and E and $\mathbf{R}$, and entities and relationships of $\mathcal{M}$ respectively.

The semantics of queries is as follows. Given an interpretation $\mathcal{I}$ of a Conceptual Model $\mathcal{M}$ with interpretation domain $\Delta^{\mathcal{I}}$, a query q of arity n is interpreted as the set $q^{\mathcal{I}}$ of n-tuples $(d_1, \ldots, d_n)$, with each $d_i \in \Delta^{\mathcal{I}}$, such that, when substituting each d_i for x_i, the formula

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

evaluates to true in $\mathcal{I}$.

The fact that a relation in a source is defined in terms of a query over the Conceptual Model confirms that we are following the local-as-view approach: each table is seen as a view of the virtual database represented by the Conceptual Model.

3.2. *Adornment*

To make the connection to the Conceptual Model precise, it is not sufficient to define each table as a relational query over the elements of the Conceptual Model. We need to make it explicit how the objects of the conceptual representation are coded into values of the logical representation. The notion of *adorned query* is introduced exactly for this purpose.

An *adorned query* is an expression of the form:

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

where T is the name of the relational table, $\vec{\mathbf{x}}$ are its attributes (observe that attributes denote *values* and not *objects*), $q(\vec{\mathbf{x}}, \vec{\mathbf{y}})$ is a query as defined above, and each α_i is an *annotation* on variables appearing in $\vec{\mathbf{x}}$. In particular:

1. For each $X \in \vec{\mathbf{x}}$, 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 example, which currency is used for a real value denoting an amount of money.

2. For each tuple of variables $\vec{\mathbf{z}} \subseteq \vec{\mathbf{x}}$ that is used for identifying in T an object $Y \in \vec{\mathbf{y}}$ mentioned in $q(\vec{\mathbf{x}}, \vec{\mathbf{y}})$, we have an annotation of the form:

$$identify([\vec{\mathbf{z}}], Y)$$

For example, the designer may assert that the attributes `first_name`, `last_name`, and `date_of_birth` in a table are used to identify students.

We point out that our method is able to cope with several *schematic differences* that may be present in the sources [SHE 92]. We illustrate this point with the help of an example.

Example 3 Suppose that the Conceptual Model contains a relationship `Service` with three attributes, `Date`, `ServiceNo`, and `Price`, where $\texttt{Service}(D, S, P)$ means that at the date D the service S costs P Euro. Suppose that Source S_1 represents the same kind of information only on Services v_1 and v_2, by means of two tables: `v1` and `v2`, where $\texttt{v1}(D, P)$ means that service v_1 costs P Italian Lira at date D, and $\texttt{v2}(D, P)$ means that service v_2 costs P Italian Lira at date D. Suppose that Source S_2 represents the same kind of information only on Services v_3 and v_4 by means of a table `Serv`, where $\texttt{Serv}(X, Y, D)$ means that services v_3 and v_4 cost X and Y Euro respectively at date D. Finally, suppose that Source S_3 represents the information only for a certain date d by means of another table $\texttt{Serv}_3$. The various tables in the three sources can be specified by means of the following adorned queries:

$$
\begin{array}{rcl}
\texttt{v1}(D,P) & \leftarrow & \texttt{Service}(D,'\texttt{v1}',P) \mid \\
 & & P :: \texttt{ItalianLira},\ D :: \texttt{Date} \\
\texttt{v2}(D,P) & \leftarrow & \texttt{Service}(D,'\texttt{v2}',P) \mid \\
 & & P :: \texttt{ItalianLira},\ D :: \texttt{Date} \\
\texttt{Serv}(X,Y,D) & \leftarrow & \texttt{Service}(D,'\texttt{v3}',X), \\
 & & \texttt{Service}(D,'\texttt{v4}',Y) \mid \\
 & & X :: \texttt{Euro},\ Y :: \texttt{Euro},\ D :: \texttt{Date} \\
\texttt{Serv}_3(S1,P) & \leftarrow & \texttt{Service}(d,S1,P), \\
 & & \texttt{Code}(S,S1) \mid \\
 & & P :: \texttt{Euro}, \\
 & & \mathit{identify}([S1],S),\ S1 :: \texttt{String}
\end{array}
$$

■

The above example illustrates a case where there are various schematic differences, both among the sources, and between the sources and the Conceptual Model. The mechanisms used in our methodology for specifying adorned queries is able to cope with such differences.

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 is inconsistent with respect to the Conceptual Model $\mathcal{M}$ either because the body B of Q is inconsistent with respect to $\mathcal{M}$, or because the annotations are incoherent with respect to what specified in $\mathcal{M}$. The inference techniques described in [CAL 98d] allow us to check the consistency of the relational tables defined for describing a source.

Example 2 (cont.) Assuming that in Source 1 a customer is actually identified by its social security number, and a department by its name, we can specify the relational table $\mathtt{TABLE}_1$ by the following adorned query:

$$
\begin{array}{lll}
\mathtt{TABLE}_1(S,M,P) & \leftarrow & \mathtt{REG\text{-}AT}_1(X,D),\ \neg\mathtt{PROMOTION}_1(X,D),\ P = \mathtt{false}, \\
 & & \mathtt{SSN}(X,S),\ \mathtt{Name}(D,M) \\
 & & \mathsf{OR} \\
 & & \mathtt{PROMOTION}_1(X,D),\ P = \mathtt{true},\ \mathtt{SSN}(X,S), \\
 & & \mathtt{Name}_1(D,M)\ \mid \\
 & & \mathit{identify}([S],X),\ S :: \mathtt{SSNString}, \\
 & & \mathit{identify}([M],D),\ M :: \mathtt{DeptNameString}, \\
 & & P :: \mathtt{Boolean}
\end{array}
$$

Additionally, we assume that in Source 2 the actual data can be described in terms of a relational table $\mathtt{TABLE}_2$ with four columns, two for the customer, one for the service the customer has registered, and one for the department. As in Source 1, in Source 2 departments are still identified by their name, but, differently from Source 1, customers are identified by their name and date of birth. Services are identified by a unique service number. Hence the following adorned query is used to specify $\mathtt{TABLE}_2$:

$$
\begin{array}{lll}
\mathtt{TABLE}_2(N,B,I,M) & \leftarrow & \mathtt{CONTRACT}_2(X,S,D),\ \mathtt{Name}(X,N),\ \mathtt{DOB}(X,B), \\
 & & \mathtt{ServNo}(S,I),\ \mathtt{Name}(D,M)\ \mid \\
 & & \mathit{identify}([N,B],X),\ N :: \mathtt{PersNameString}, \\
 & & \qquad\qquad\qquad\quad B :: \mathtt{Date} \\
 & & \mathit{identify}([I],S),\ I :: \mathtt{ServNoInteger}, \\
 & & \mathit{identify}([M],D),\ M :: \mathtt{DeptNameString}
\end{array}
$$

■

4. Interschema Correspondences

We now describe how to define *Interschema Correspondences*, which are used to declaratively specify the correspondences between data in different schemas (either source schemas or data warehouse schema).

In our approach, Interschema Correspondences are defined in terms of relational tables, similarly to the case of the relations describing the sources at the logical level. The difference with source relations is that we conceive Interschema Correspondences as non-materialized relational tables, in the sense that their extension is computed by an associated program whenever it is needed. It follows that, to each Interschema

Correspondence, we associate a *head*, a *body*, and an *adornment*. Differently from the case of a source relation, the adornment specifies which is the program that is able to compute the extension of the virtual table.

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

Conversion Correspondences are used to specify that data in one source can be converted into data of a different source or of the data warehouse, and how this conversion is performed. They are used to anticipate several types of data conflicts that may occur in loading data.

As an example, suppose that in a table of a source costs are represented in Italian Lira, while in a table of the Data Warehouse we want to express them in Euro. Then, in order to use the source table in the rewriting of a query that defines the Data Warehouse table, it is necessary to know about the possibility of converting each amount in Italian Lira into an amount in Euro.

A Conversion Correspondence $convert$ has the following form:

$$\begin{array}{rcl} convert([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) & \leftarrow & conj(\vec{\mathbf{x}},\vec{\mathbf{y}},\vec{\mathbf{z}}) \\ & & \text{through } program(\vec{\mathbf{x}},\vec{\mathbf{y}},\vec{\mathbf{z}}) \end{array}$$

where $conj$ is a conjunctive query, which specifies the conditions under which the conversion is applicable, and $program$ is a predicate that we assume associated to 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 operates from a tuple of values satisfying the conditions specified for $\vec{\mathbf{x}}$ in $conj$ to a tuple of values satisfying the conditions specified for $\vec{\mathbf{y}}$. This means that the conversion program receives as input a tuple $\vec{\mathbf{x}}$, and returns the corresponding tuple $\vec{\mathbf{y}}$, possibly using the additional parameter $\vec{\mathbf{z}}$ to perform the conversion.

Matching Correspondences are used to specify how data in different sources can match. A Matching Correspondence $match$ has the following form:

$$\begin{array}{rcl} match([\vec{\mathbf{x}}_1],\ldots,[\vec{\mathbf{x}}_k]) & \leftarrow & conj(\vec{\mathbf{x}}_1,\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{z}}) \\ & & \text{through } program(\vec{\mathbf{x}},\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{z}}) \end{array}$$

where $conj$ specifies the conditions under which the matching is applicable, and $program$ is a predicate that we assume associated to 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 they match or not.

Note that already specified Interschema Correspondences may be used to define new ones. As an example, 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 using equality. In this case, he could provide the following definition of the Matching Correspondence:

$$\begin{array}{rcl} \mathit{match}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) & \leftarrow & \mathit{convert}_1([\vec{\mathbf{x}}],[\vec{\mathbf{z}}]),\ \mathit{convert}_2([\vec{\mathbf{y}}],[\vec{\mathbf{z}}]), \\ & & \mathit{conj}(\vec{\mathbf{x}},\vec{\mathbf{y}},\vec{\mathbf{z}},\vec{\mathbf{w}}) \\ & & \textsf{through}\ \mathit{none} \end{array}$$

Observe that, in this case, the program associated to the Matching Correspondence is empty, since the actual conversions are performed by the programs associated to the Conversion Correspondences.

Reconciliation Correspondences are used to assert how we can reconcile data in different sources into data of the data warehouse. A Reconciliation Correspondence $\mathit{reconcile}$ has the following form:

$$\begin{array}{rcl} \mathit{reconcile}([\vec{\mathbf{x}}_1],\ldots,[\vec{\mathbf{x}}_k],[\vec{\mathbf{z}}]) & \leftarrow & \mathit{conj}(\vec{\mathbf{x}},\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{z}},\vec{\mathbf{w}}) \\ & & \quad\quad \textsf{through}\ \mathit{program}(\vec{\mathbf{x}}_1,\ldots,\vec{\mathbf{x}}_k,\vec{\mathbf{z}},\vec{\mathbf{w}}) \end{array}$$

where conj specifies the conditions under which the reconciliation is applicable, and $\mathit{program}$ is a predicate that we assume associated to a program that performs the reconciliation. Such correspondence specifies that the k tuples of values $\vec{\mathbf{x}}_1,\ldots,\vec{\mathbf{x}}_k$ coming from the sources are reconciled to the tuple $\vec{\mathbf{z}}$ 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 reconciled tuple.

Again, a Reconciliation Correspondence could simply be defined as a combination of appropriate Matching and Conversion Correspondences, e.g.:

$$\begin{array}{rcl} \mathit{reconcile}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}],[\vec{\mathbf{z}}]) & \leftarrow & \mathit{convert}_1([\vec{\mathbf{x}}],[\vec{\mathbf{w}}_1]),\ \mathit{convert}_2([\vec{\mathbf{y}}],[\vec{\mathbf{w}}_2]), \\ & & \mathit{match}_1([\vec{\mathbf{w}}_1],[\vec{\mathbf{w}}_2]),\ \mathit{convert}_3([\vec{\mathbf{w}}_1],[\vec{\mathbf{z}}]), \\ & & \mathit{conj}(\vec{\mathbf{x}},\vec{\mathbf{y}},\vec{\mathbf{w}}_1,\vec{\mathbf{w}}_2,\vec{\mathbf{z}}) \\ & & \textsf{through}\ \mathit{none} \end{array}$$

In practice, several of the Interschema Correspondences that must be specified will have a very simple form, since they will correspond simply to equality. Therefore, in order to simplify the task of the designer in specifying the various Interschema Correspondences, we assume that several correspondences are automatically asserted *by default* by the system. In particular, for each domain D in the conceptual model, the following Interschema Correspondences are specified by default:

$$\begin{array}{rcl} \mathit{convert}([X],[Y]) & \leftarrow & D(X),\ D(Y),\ X = Y \\ & & \textsf{through}\ \mathit{identity}(X,Y) \\ \mathit{match}([X],[Y]) & \leftarrow & D(X),\ D(Y),\ X = Y \\ & & \textsf{through}\ \mathit{none} \\ \mathit{reconcile}([X],[Y],[Z]) & \leftarrow & D(X),\ D(Y),\ D(Z),\ X = Y,\ X = Z \\ & & \textsf{through}\ \mathit{identity}(X,Z) \end{array}$$

where $identity$ is the program that computes the identity function for values of domain D, and the matching correspondence has no associated program.

The system allows the designer to inhibit the default correspondences for a certain domain, simply by providing an alternative Interschema Correspondence referring to that domain.

Moreover, we assume that for each Conversion Correspondence $convert_i$ asserted by the designer, the system automatically asserts a new Matching Correspondence $match_i$ as follows:

$$\begin{array}{rcl} match_i([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) & \leftarrow & convert_i([\vec{\mathbf{x}}],[\vec{\mathbf{z}}]),\ \vec{\mathbf{y}} = \vec{\mathbf{z}} \\ & & \text{through } none \end{array}$$

Moreover, 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 a new Reconciliation Correspondence $reconcile_{i,j}$ as follows:

$$\begin{array}{rcl} reconcile_{i,j}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}],[\vec{\mathbf{z}}]) & \leftarrow & match_i([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]), \\ & & convert_j([\vec{\mathbf{x}}],[\vec{\mathbf{z}}]) \\ & & \text{through } none \end{array}$$

Example 2 (cont.) The following Conversion Correspondence specifies that the name and date of birth of a customer can be converted into a Social Security Number through the program `name_to_SSN`:

$$\begin{array}{rcl} convert_1([N,B],[S]) & \leftarrow & \texttt{PersNameString}(N),\ \texttt{Date}(B),\ \texttt{SSNString}(S), \\ & & \texttt{Customer}_0(X),\ \texttt{Name}(X,N),\ \texttt{DOB}(X,B),\ \texttt{SSN}(X,S) \\ & & \text{through } \texttt{name\_to\_SSN}(N,B,S) \end{array}$$

Moreover, we add the following Conversion Correspondence, which represents the fact that a department name can be converted into a department code through the program `dept_name_to_code`:

$$\begin{array}{rcl} convert_2([M],[C]) & \leftarrow & \texttt{DeptNameString}(M),\ \texttt{DeptCodeInteger}(C), \\ & & \texttt{Department}_0(D),\ \texttt{Name}(D,M),\ \texttt{Code}(D,C) \\ & & \text{through } \texttt{dept\_name\_to\_code}(M,C) \end{array}$$

According to the above rules, the system asserts automatically (among others) the Matching Correspondence and Conversion Correspondences:

$$\begin{array}{rcl} match_1([N,B],[S]) & \leftarrow & convert_1([N,B],[S_1]),\ S = S_1 \\ & & \text{through } none \\ match_2([M],[C]) & \leftarrow & convert_2([M],[C_1]),\ C = C_1 \\ & & \text{through } none \\ match_3([M_1],[M_2]) & \leftarrow & \texttt{DeptNameString}(M_1), \\ & & \texttt{DeptNameString}(M_2),\ M_1 = M_2 \\ & & \text{through } none \end{array}$$

$$\begin{array}{rcl} \mathit{convert}_4([S_1],[S_2]) & \leftarrow & \texttt{SSNString}(S_1),\ \texttt{SSNString}(S_2),\ S_1 = S_2 \\ & & \text{through } \mathit{identity}(S_1,S_2) \\ \mathit{convert}_5([P_1],[P_2]) & \leftarrow & \texttt{Boolean}(P_1),\ \texttt{Boolean}(P_2),\ P_1 = P_2 \\ & & \text{through } \mathit{identity}(P_1,P_2) \\ \mathit{convert}_6([I_1],[I_2]) & \leftarrow & \texttt{ServNoInteger}(I_1),\ \texttt{ServNoInteger}(I_2), \\ & & I_1 = I_2 \\ & & \text{through } \mathit{identity}(I_1,I_2) \end{array}$$

and the Reconciliation Correspondences

$$\begin{array}{l} \mathit{reconcile}_{1,1}([N,D],[S_1],[S_2]) \\ \mathit{reconcile}_{3,2}([M_1],[M_2],[C]) \end{array}$$

■

5. Specification of mediators

As we said in the introduction, the problem of data integration and reconciliation is crucial for the task of designing the mediators that load the data in the Data Warehouse. Such a task 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.

Suppose we have decided to materialize a new relation T in the Data Warehouse. Our goal is to support the designer in providing a formal specification for the design of the mediator used to extract the correct data from the sources, and to load such data in T. The methodology we propose is based on the following steps.

1. We apply the method described in Section 3 to provide the specification of the relation T. In other words, we specify T in terms of an adorned query:

$$q \leftarrow q' \mid c_1, \ldots, c_n.$$

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 specification of what we want to materialize in the table of the Data Warehouse. Note also that we express the semantics of T again in terms of the conceptual model. Not only the sources, but also the relations in the Data Warehouse are seen as views of such a conceptual model.

2. We look for a rewriting of q in terms of the queries $q_1, \ldots, q_s$ that correspond to the materialized views in the Data Warehouse. If a complete, equivalent rewriting exists, then the new table can be derived from the existing tables in the Data Warehouse. Otherwise, the algorithm is able to single out the part that cannot be derived from the Data Warehouse, and that must be loaded from the sources. In the following, q denotes such part.

3. We look for a rewriting of q in terms of the queries corresponding to the tables in the sources. The rewriting aims at expressing the data in T in terms of a disjunction of conjunctive queries where each atom refers to

— a table in a source, or

— a matching, conversion, or reconciliation predicate defined in the Interschema Correspondences.

In other words, we are trying to reformulate q in terms of the relations in the sources, and possibly in terms of the matching, conversion, and reconciliation predicates. If there are different rewritings, then we choose the best rewriting r with respect to suitable quality parameters. There are several criteria to be taken into consideration when evaluating the quality of a rewriting, such as:

— Completeness of the rewriting. Obviously, the best situation is the one where the rewriting is complete, in the sense that the rewritten query is equivalent to the original query. Such a check can be done by exploiting the algorithm for query containment.

— Accuracy, confidence, freshness, and availability of data in the source relations that the rewriting requires to access.

The resulting query is the specification for the design of the mediator associated to T. The most critical step of the above method is the computation of the rewriting. Our rewriting algorithm is based on the method presented in [DUS 97], modified to take into account the following aspects:

— We deal with queries whose atoms refer to a conceptual model that includes ISA assertions and a limited form of functional dependencies. Such constraints have to be considered in the computation of the rewriting.

— We deal with queries that are disjunctions of conjunctions. It follows that the rewriting itself is in general a disjunction, and therefore, we need to deal with the problem of merging the results of several queries. This problem is addressed by the notion of merging clause. In particular, if the query r computed by the rewriting is an *or*-query (i.e., it 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 *or*-parts of the query may contain objects and values that represent the same real world entity or the same value. A merging clause is an expression of the form

merging *tuple-spec*$_1$ and $\cdots$ and *tuple-spec*$_n$
such that *matching-condition*
into *tuple-spec*$_{t_1}$ and $\cdots$ and *tuple-spec*$_{t_m}$

where *tuple-spec*$_i$ denotes a tuple returned by the i-th disjunct of r, *matching-condition* specifies how to merge the various tuples denoted by *tuple-spec*$_1$,...,*tuple-spec*$_n$, and *tuple-spec*$_{t_1}$,...,*tuple-spec*$_{t_m}$ denote the tuples in T resulting from the merging.

We observe that the rewriting algorithm is able to generate one merging 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 merging plan (for example for merging three disjuncts, rather than three pairs of disjuncts).

— The algorithm computes the maximally contained rewriting (i.e., every other rewriting is included in the one computed by the query), but we also want to inform the designer whether such a rewriting is equivalent or not to the original query. Indeed, we have devised an effective method for checking equivalence between the original query and the computed rewriting [CAL 98a].

— Besides the relational tables in the sources, our rewriting algorithm takes into account the matching, conversion, and reconciliation predicates defined in the Interschema Correspondences.

— Even when no rewriting exists for the query (i.e., when the maximally contained rewriting is empty), we want to provide the designer with useful indications on whether there is a method for enriching the Interschema Correspondences to get a non-empty rewriting. Indeed, our rewriting algorithm adopts a form of abductive reasoning that enables to single out the specification of which matching, conversion and reconciliation operations would allow to get a non-empty rewriting. This indication can be profitably used by the designer to check whether she/he can add new Interschema Correspondences in order to make the computed rewriting complete.

Example 2 (cont.) Suppose we want to store in the Data Warehouse a relation containing the information about customers that have a contract for a certain service with a department at which they are also registered, or that are eligible for a promotion. Independently from the fact that the customer has a contract, we want to include the information on whether he is eligible for a promotion. We can make use of a relational table `TDW` with four components, defined by the following adorned query, where we have assumed that in the Data Warehouse we want to identify customers by their SSN, services by their service number, and departments by their code:

$$
\begin{array}{lll}
\mathtt{TDW}(S,I,C,P) & \leftarrow & \mathtt{CONTRACT}_0(X,R,D),\ \mathtt{PROMOTION}_1(X,D),\\
& & \mathtt{SSN}(X,S),\ \mathtt{ServNo}(R,I),\ \mathtt{Code}(D,C),\ P=\mathtt{true}\\
& & \mathsf{OR}\\
& & \mathtt{CONTRACT}_0(X,R,D),\ \mathtt{REG\text{-}AT}_1(X,D),\\
& & \neg\mathtt{PROMOTION}_1(X,D),\ \mathtt{SSN}(X,S),\ \mathtt{ServNo}(R,I),\\
& & \mathtt{Code}(D,C),\ P=\mathtt{false}\\
& & \mathsf{OR}\\
& & \mathtt{PROMOTION}_1(X,D),\ \mathtt{SSN}(X,S),\ \mathtt{Code}(D,C),\\
& & P=\mathtt{true},\ I=\mathtt{NULL}\ \mid\\
& & \mathit{identify}([S],X),\ S :: \mathtt{SSNString},\\
& & \mathit{identify}([I],R),\ I :: \mathtt{ServNoInteger},\\
& & \mathit{identify}([C],D),\ C :: \mathtt{DeptCodeInteger},\\
& & P :: \mathtt{Boolean}
\end{array}
$$

Using the asserted and automatically derived Interschema Correspondences, the system is able to rewrite the above query in terms of $\mathtt{TABLE}_1$ in Source 1 and $\mathtt{TABLE}_2$ in Source 2 (see Section 3) as follows:

$$
\begin{array}{l}
\mathtt{TDW}(S_0, I_0, C, P_0) \leftarrow \\
\quad \mathtt{TABLE}_1(S_1, M_1, P_1),\ \mathtt{TABLE}_2(N_2, B, I_2, M_2), \\
\quad \mathit{reconcile}_{1,1}([N_2, B], [S_1], [S_0]), \\
\quad \mathit{reconcile}_{3,2}([M_1], [M_2], [C]), \\
\quad \mathit{convert}_5([P_1], [P_0]),\ \mathit{convert}_6([I_2], [I_0]) \\
\quad \mathsf{OR} \\
\quad \mathtt{TABLE}_1(S_1, M_1, P_1),\ I_0 = \mathtt{NULL} \wedge P_1 = \mathtt{true}, \\
\quad \mathit{convert}_2([M_1], [C]),\ \mathit{convert}_4([S_1], [S_0]), \\
\quad \mathit{convert}_5([P_1], [P_0]) \\
\quad \textsf{merging} \quad (S_a, I_a, C_a, P_a) \textsf{ and } (S_b, I_b, C_b, P_b) \\
\quad \textsf{such that} \quad S_a = S_b,\ C_a = C_b,\ P_a = P_b,\ I_a \neq \mathtt{NULL},\ I_b = \mathtt{NULL} \\
\quad \textsf{into} \quad (S_a, I_a, C_a, P_a)
\end{array}
$$

Notice that, in the above query, each customer eligible for a promotion that has a contract, and thus is selected by the first disjunct, is also selected by the second disjunct. In this case the merging clause merges the two tuples into the one selected by the first disjunct, which contains the information on the service number missing in the other one. ∎

6. Conclusions

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

Based on the described methodology, we are currently implementing a design tool within the ESPRIT LTR project DWQ, "Foundations of Data Warehouse Quality" [JAR 99]. The tool exploits the Concept Base System [JAR 92] as a meta-repository, and provides support for both schema and data integration in Data Warehousing.

7. References

[ABI 98] ABITEBOUL S. and DUSCHKA O., « Complexity of Answering Queries Using Materialized Views ». In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, p. 254–265, 1998.

[CAL 98a] CALVANESE D., DE GIACOMO G. and LENZERINI M., « On the Decidability of Query Containment under Constraints ». In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, p. 149–158, 1998.

[CAL 98b] CALVANESE D., DE GIACOMO G., LENZERINI M., NARDI D. and ROSATI R., « Description Logic Framework for Information Integration ». In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, p. 2–13, 1998.

[CAL 98c] CALVANESE D., DE GIACOMO G., LENZERINI M., NARDI D. and ROSATI R., « Schema and Data Integration Methodology for DWQ ». Technical report DWQ-UNIROMA-004, DWQ Consortium, September 1998.

[CAL 98d] CALVANESE D., DE GIACOMO G., LENZERINI M., NARDI D. and ROSATI R., « Source Integration in Data Warehousing ». In *Proc. of the 9th Int. Workshop on Database and Expert Systems Applications (DEXA'98)*, p. 192–197. IEEE Computer Society Press, 1998.

[CAT 93] CATARCI T. and LENZERINI M., « Representing and using Interschema Knowledge in Cooperative Information Systems ». *J. of Intelligent and Cooperative Information Systems*, vol. 2, n. 4, p. 375–398, 1993.

[DUS 97] DUSCHKA O. M. and LEVY A. Y., « Recursive Plans for Information Gathering ». In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, p. 778–784, 1997.

[GOH 94] GOH C. H., MADNICK S. E. and SIEGEL M., « Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment ». In *Proc. of the 3rd Int. Conf. on Information and Knowledge Management (CIKM'94)*, p. 337–346, 1994.

[GUP 95] GUPTA A. and MUMICK I. S., « Maintenance of Materialized Views: Problems, Techniques, and Applications ». *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 18, n. 2, p. 3–18, 1995.

[HAM 95] HAMMER J., GARCIA-MOLINA H., WIDOM J., LABIO W. and ZHUGE Y., « The Stanford Data Warehousing Project ». *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 18, n. 2, p. 41–48, 1995.

[HUL 96] HULL R. and ZHOU G., « A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches ». In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, p. 481–492, 1996.

[HUL 97] HULL R., « Managing Semantic Heterogeneity in Databases: A Theoretical Perspective ». In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'97)*, 1997.

[INM 96] INMON W. H., *Building the Data Warehouse*. John Wiley & Sons, second edition, 1996.

[JAR 92] JARKE M., « ConceptBase V3.1 User Manual ». Technical report 92–17, Aachener Informatik-Berichte, Aachen (Germany), 1992.

[JAR 99] JARKE M., LENZERINI M., VASSILIOU Y. and VASSILIADIS P., *Fundamentals of Data Warehouses*. Springer-Verlag, 1999.

[LEV 95] LEVY A. Y., MENDELZON A. O., SAGIV Y. and SRIVASTAVA D., « Answering Queries Using Views ». In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'95)*, p. 95–104, 1995.

[PAP 95] PAPAKONSTANTINOU Y., GARCIA-MOLINA H. and WIDOM J., « Object Exchange Across Heterogeneous Information Sources ». In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95)*, p. 251–260, 1995.

[SHE 92] SHETH A. and KASHYAP V., « So Far (Schematically) yet So Near (Semantically) ». In *Proc. of the IFIP DS-5 Conf. on Semantics of Interoperable Database Systems*. Elsevier Science Publishers (North-Holland), Amsterdam, 1992.

[ULL 97] ULLMAN J. D., « Information Integration using Logical Views ». In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, vol. 1186 of *Lecture Notes in Computer Science*, p. 19–40. Springer-Verlag, 1997.

[WID 95] WIDOM J., « Special Issue on Materialized Views and Data Warehousing ». *IEEE Bulletin on Data Engineering*, vol. 18, n. 2, 1995.

[WIE 92] WIEDERHOLD G., « Mediators in the Architecture of Future Information Systems ». *IEEE Computer*, vol. 25, n. 3, p. 38–49, 1992.

[ZHO 95] ZHOU G., HULL R., KING R. and FRANCHITTI J.-C., « Using Object Matching and Materialization to Integrate Heterogeneous Databases ». In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95)*, p. 4–18, 1995.

[ZHO 96] ZHOU G., HULL R. and KING R., « Generating Data Integration Mediators that Use Materializations ». *J. of Intelligent Information Systems*, vol. 6, p. 199–221, 1996.

Diego Calvanese has completed his PhD in Computer Science at the University of Rome "La Sapienza" in 1996. He is currently research associate at the Dipartimento di Informatica e Sistemistica of the same university. His research interests include logics for knowledge representation and their relationship to data models, information integration and data warehousing, and semistructured data.

Riccardo Rosati has completed his PhD in Computer Science at the University of Rome "La Sapienza" in 1997. He is currently research associate at the Dipartimento di Informatica e Sistemistica of the same university. His research interests include knowledge representation, nonmonotonic reasoning, database and knowledge base applications, planning and reasoning about actions.

Giuseppe De Giacomo has completed his PhD in Computer Science at the University of Rome "La Sapienza" in 1995. He is currently assistant professor at the Faculty of Engineering of the same university. His research interests include knowledge representation, databases, information integration and data warehousing, semistructured data, reasoning about actions, cognitive robotics, and program verification.