Document Number:	DWQ-UNIROMA-006
Project Name:	Foundations of Data Warehouse Quality (DWQ)
Project Number:	EP 22469
Title:	**Experimental Prototype Supporting Data Reconciliation**
Authors:	Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, Riccardo Rosati
Workpackage:	WP4
Document Type:	Deliverable (D4.2)
Classification:	Public
Distribution:	DWQ Consortium
Status:	Final
Document file:	wp4-2.tex
Version:	1.3
Date:	October 18, 1999
Number of pages:	40

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	September 15, 1999	First draft
v.1.1	October 6, 1999	First revision
v.1.2	October 12, 1999	Second revision
hline v.1.3	October 18, 1999	Final version

1 Introduction

According to [Inm96], integration is the most important aspect of a data warehouse. 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.

In DWQ, the ultimate goal of source integration and data reconciliation is to represent the migration of the data from the sources to the Data Warehouse, in order to support the design of (materialized) views that meet user requirements, and have high quality with respect to correctness, interpretability, usefulness, believability.

The present report illustrates the work carried out in the context of *WP4 - Data Reconciliation* of DWQ during the last two years of the project. In the rest of this section we recall the main notions regarding the general architecture of DWQ and the structure of the methodology for source and data integration (see [CDGL+97b, CDGL+98b] for the reports on the first and second year of the project on this subject), and then we illustrate the organization of the remaining sections of the report.

1.1 Summary of the Architecture

Generally speaking, if we refer to the DWQ framework illustrated in Deliverable 2.1, source and data integration and reconciliation deals with the objects named "Enterprise Model", "Source Model", "Data Warehouse Schema", and "Source Schema", and is characterized by the following elements (see Figure 1):

- The *Enterprise Model* is a conceptual representation of the global concepts and relationships that are of interest to the Data Warehouse application. In our context, the Enterprise Model is not necessarily a complete representation of all the data of the sources. Being the integration process incremental, the Enterprise Model provides a consolidated view of the concepts and the relationships that are important to the enterprise, and have been currently analyzed. Such a view is subject to changes and additions as the analysis of the information sources proceeds. We use a logic-based formalism for describing the Enterprise Model, as well as the other models at the conceptual level. The formalism is general and powerful enough to express the usual database models, such as the Entity-Relationship Model, and the Relational Model, which are the database formalisms of interest in the present report. Moreover, there are suitable inference techniques associated with the formalism, which allow for carrying out several reasoning services on the representation.

- For a given information source S, the *Source Model* of S is a conceptual representation of the data residing in S, or at least of the portion of data currently taken into account. Again, our approach does not require a source to be fully analyzed and conceptualized. Source Models are expressed by means of the same formalism used for the Enterprise Model.

- The Enterprise Model, the various Source Models, and the interdependencies between elements of different Source Models and between Source Models and the Enterprise Model, constitute what we call the *Conceptual Data Warehouse Model*. The notion of interdependency is a central one in our approach. Since the sources are of interest in the overall architecture, integration does not simply mean producing the Enterprise Model, but rather to be able to establish the correct relationships both between the Source Models and the Enterprise Model, and between the various Source Models. We formalize the notion of interdependency by means of interschema assertions (here called intermodel assertions), introduced in [CL93]. An *intermodel assertion* states that one object (i.e., class, entity, or relation) belonging to a certain Model (either the Enterprise or a Source Model) is always a subset of an object belonging to another Model. This simple declarative mechanisms has been showed to be extremely effective in establishing relationships among different database schemas (see also [Hul97]). We use again a logic-based formalism to express intermodel assertions, and the associated inference techniques provide a means to reason about interdependencies among models.

- The logical content of each source S, called the *Source Schema*, is provided in terms of a set of definitions of relations, each one expressed in terms of a query over the Source Model of S. The

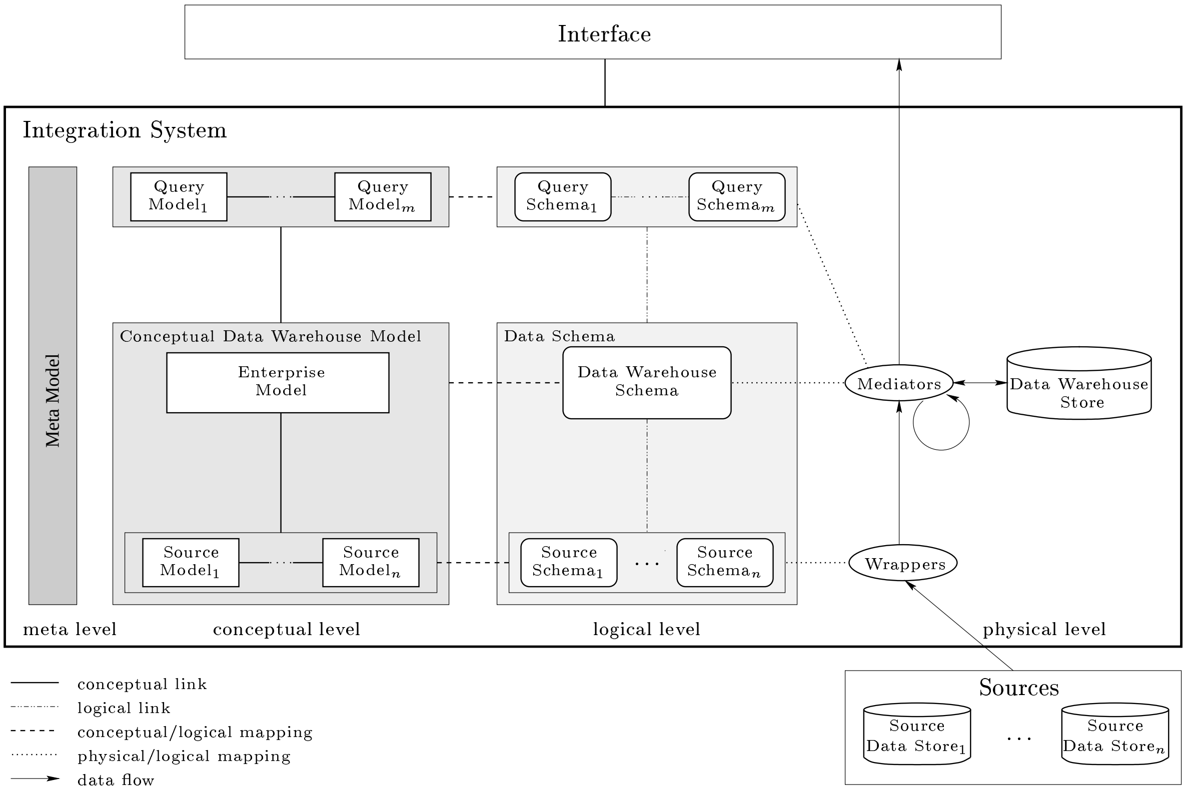

Figure 1: Architecture for Data Integration

logical content of a source represents the structure of data expressed in terms of a logical data model, in this case the Relational Model. In our framework, such a structure is provided by specifying the connection with the conceptual representation of the source. In other words, the logical content of a source S, or of a portion thereof, is described in terms of a view over the Source Model associated with S (and, therefore, of the Conceptual Data Warehouse Model). We envision the use of suitable mechanisms, called wrappers, to map physical structures to logical structures (see Figure 1).

- The logical content of the materialized views constituting the Data Warehouse, called the *Data Warehouse Schema*, is provided in terms of a set of definitions of relations, each one expressed in terms of a query over the Conceptual Data Warehouse Model. Similarly to the case of the sources, each portion of the Data Warehouse Schema is described in terms of a view over the Conceptual Data Warehouse Model. The way in which a view is actually materialized starting from the data in the sources is specified by means of suitable mechanisms, called mediators (see Figure 1).
- Each element of the integration process has a corresponding representation in the DWQ Repository (see Deliverable 2.1).

1.2 Summary of the Methodology

The proposed approach to integration is incremental: no "Complete Enterprise Model"[1] is prescribed. Instead, whenever a new portion of a source is taken into account, the new information is integrated with the Enterprise Model, and the necessary new relationships are added. Thus, the Enterprise Model provides a consolidated view of the concepts and the relationships that are important to the enterprise, and have been currently analyzed. Such a view is subject to changes and additions as the analysis of the information sources proceeds. This approach would also allow the notion of Complete Enterprise Model

[1]The notion of Complete Enterprise Model corresponds to the one of global schema used in Database integration.

(in particular in the extreme case where all aspects of all sources have been considered), but is more flexible.

The methodology identifies the following situations that need to be dealt with in a systematic manner in order to build and maintain the Conceptual Data Warehouse Model, the Source Schemas and the Data Warehouse Schema:

1. A new source (or, a new portion of a source) is analyzed and added to the whole architecture; the data integration design scenario corresponding to this situation is called *source-driven*.

2. A new query is formulated by a client; the design scenario corresponding to this situation is called *client-driven*.

Note that, although the two scenarios refer to a dynamic situation where a new piece of information or a new request is considered, one can in practice iterate the process to design the Data Warehouse from scratch.

Source-driven integration is constituted by the following main phases (we do not consider here client-driven integration, which is discussed in [CDGL$^+$97b]):

1. *Enterprise and Source Model construction.* The Source Model corresponding to the new source is produced, if not available. Analogously, the conceptual model of the enterprise is produced, if not available.

2. *Source Model integration.* The Source Model is *integrated into the Conceptual Data Warehouse Model*. This can lead to changes both to the Source Models, and to the Enterprise Model. Moreover, intermodel assertions between the Enterprise Model and the Source Models and between the new source and the existing sources are added to the Conceptual Data Warehouse Model.

3. *Source and Data Warehouse Schema specification.* The Source Schema corresponding to the new source (or, corresponding to a new portion of the source) is produced. On the basis of the new analyzed source, an analysis is carried out on whether the Data Warehouse Schema should be restructured and/or modified.

4. *Data Integration and Reconciliation.* The problem of data integration and reconciliation arises when data passes from the application-oriented environment to the Data Warehouse. During the transfer of data, possible inconsistencies and redundancies should be resolved, so that the warehouse is able to provide and integrated and reconciled view of data of the organization. In our methodology, the problem of data reconciliation is addressed after the phase of Data Schema specification, and is based on (1) specifying through the notion of *Interschema Correspondences* how the relations in the Data Warehouse Schema are linked to the relations in the Source Schemas, and (2) designing suitable *mediators* for every relation in the Data Warehouse Schema. In step (1), interschema correspondences are used to declaratively specify the correspondences between data in different schemas (either source schemas or data warehouse schema). Interschema Correspondences are defined in terms of relational tables, similarly to the case of the relations describing the sources at the logical level. We distinguish among three types of correspondences, namely *Conversion*, *Matching*, and *Reconciliation Correspondences*. By virtue of such correspondences, the designer can specify different forms of data conflicts holding between the source, and can anticipate methods for solving such conflicts when loading the Data Warehouse. In step (2), the methodology aims at producing, for every relation in the Data Warehouse Schema, a specification of the corresponding mediator, which determines how the tuples of such a relation should be constructed from a suitable set of tuples extracted from the relations stored in the sources.

In all the above phases, specific steps are performed in order both to store the values of the quality factors involved in source and data integration, and to analyze the quality of the design choices. The *Quality Factors* of the Conceptual Data Warehouse Model and the various schemas are evaluated and a restructuring of the Models and the schemas is accomplished to match the required criteria. This step requires also the use of the reasoning techniques associated with our formalisms. Moreover, during the whole design phase the metadata repository [JGJ$^+$95] is exploited for storing and and manipulating the representation of the conceptual model, and for querying the metadata in order to get various information about the design choices.

Phases 1 and 2 of source driven integration are described in [CDGL+98b]. In this report we illustrate in detail Phase 3 (Source and Data Warehouse Schema specification) and Phase 4 (data integration and reconciliation), and, describe a prototype tool developed in the context of DWQ supporting the integration methodology and the access to the metadata repository.

1.3 Organization of the Report

The remaining sections of the report are organized as follows. In Section 2, we summarize the relevant features of the formalisms that are used both at the conceptual and at the logical level. Section 3 is devoted to a discussion on the reasoning services associated to the formalism. Section 4 describes the Source and Data Warehouse Schema specification phase, while Section 5 discusses the data integration and reconciliation phase. Section 6 describes the basic features of the tool supporting data integration and reconciliation. Section 7 illustrates the work done in WP4 in the context of semi-structured data sources. Finally, Section 8 concludes the report, and relevant papers on the topics addressed in the report are included in the Appendices.

2 The Formalisms

In this section we summarize the relevant features of the formalisms that are used at the logical and conceptual levels in the general Data Warehouse architecture. The formalisms are essentially those that were already introduced in Deliverable 3.3, extended with the possibility of expressing disjunctive queries, and using domain constants at the logical level. Additionally we show how the general formalism at the conceptual level can be specialized to the Entity-Relationship model.

2.1 The Formalism for the Conceptual Level

The formalism we use for the conceptual level belongs to the family of *Description Logics*, introduced and studied in the field of Knowledge Representation (see e.g. [Bor95, DLNS96]). Generally speaking, Description Logics are based on objects, classes (or concepts) and relationships. In many fields of Computer Science we find formalisms for the representation of objects and classes. A *class* (or *concept*) denotes a subset of the domain of discourse, and a class-based representation formalism allows one to express several kinds of relationships and constraints (e.g., subclass constraints) holding among classes. Moreover, class-based formalisms aim at taking advantage of the class structure in order to provide various information, such as whether an element belongs to a class, whether a class is a subclass of another class, and more generally, whether a given constraint holds for a given set of classes.

In traditional Description Logics, structured knowledge is described by means of so called *concepts* and *roles*, which denote unary and binary predicates, respectively. Starting from a set of atomic symbols one can build complex concept and role expressions by applying suitable constructors which characterize a Description Logic. Formally, concepts are interpreted as subsets of a domain and roles as binary relations over that domain, and all constructors are equipped with a precise set-theoretic semantics. The most common constructors include boolean operators on concepts, and quantification over roles. For example, the concept $\mathtt{Person} \sqcap \forall\mathtt{child.Male}$, denotes the set of individuals that are instances of the concept `Person` and are connected through the role `child` only to instances of the concept `Male`, while the concept $\exists\mathtt{child}$ denotes all individuals that are connected through role `child` to some individual. Further constructors that have been considered important include more general forms of quantification, number restrictions, which allow one to state limits on the number of connections that an individual may have via a certain role, and constructors on roles, such as intersection, concatenation and inverse. A Description Logic knowledge base, expressing the intensional knowledge about the modeled domain, is built by stating inclusion and/or equality assertions between concepts, which have to be satisfied by the models of the knowledge base. The assertions are used to specify necessary and/or necessary and sufficient conditions for individuals to be instances of certain concepts. Reasoning on such knowledge bases includes the detection of inconsistencies in the knowledge base itself, determining whether a concept can be populated in a model of the knowledge base, and checking subsumption, i.e., whether all instances of a concept are necessarily also instances of another concept in all models of the knowledge base.

For the conceptual level of a Data Warehouse, we propose here a new Description Logic, called $\mathcal{DLR}_{reg}$, which includes *concepts*, *n-ary relationships*, and *regular expressions*. $\mathcal{DLR}_{reg}$ is inspired by the languages introduced in [CDGL97a, CDGL95, DGL95, DGL94, CL93], where the notion of n-ary relationship was already present. We believe that $\mathcal{DLR}_{reg}$ is a natural extension of Description Logics towards n-ary relationships, that are extremely important in our context.

2.1.1 Syntax of $\mathcal{DLR}_{reg}$

In order to define the syntax of $\mathcal{DLR}_{reg}$ we consider a fixed finite set $\Sigma_\mathcal{D}$ of *domains* (ranged over by D), where each domain $D \in \Sigma_\mathcal{D}$ has an associated set of *domain values* (ranged over by v).

Concepts, relationships, and regular expressions are defined over $\Sigma_\mathcal{D}$ and over an alphabet $\Sigma = \Sigma_\mathcal{C} \dot\cup \Sigma_\mathcal{R}$ in which:

1. $\Sigma_\mathcal{R}$ is a finite set of *atomic relationships* (ranged over by $\mathbf{P}$);
2. $\Sigma_\mathcal{C}$ is a finite set of *atomic concepts* (ranged over by A).

Each atomic relationship $\mathbf{P}$ has an associated *signature* $\tau = \langle t_1, \ldots, t_n \rangle$, where $n \geq 2$ is the *arity* of $\mathbf{P}$ and t_i ($i \in \{1, \ldots, n\}$) is the *type* of the i-th component which can be either *obj* (denoting objects) or *val* (denoting values).

We assume that for each signature τ, $\Sigma_\mathcal{R}$ contains a special atomic relationship $\top_\tau$ of signature τ. Similarly, we assume that $\Sigma_\mathcal{C}$ contains two special atomic concepts $\top_\mathcal{C}$ and $\top_\mathcal{D}$.

We use $\mathbf{R}^{(\tau)}$ to denote arbitrary relationships of signature τ, C to denote arbitrary concepts, E to denote regular expressions, and V to denote domains, which are built starting from atomic relationships, atomic concepts, domains, and domain values by applying *constructors* according to the following syntax:

$$
\begin{array}{rcl}
\mathbf{R}^{(\tau)} & ::= & \top_\tau \mid \mathbf{P}^{(\tau)} \mid (\$i^o{}_\tau\!: C) \mid (\$i^v{}_\tau\!: V) \mid \neg\mathbf{R}^{(\tau)} \mid \mathbf{R}_1{}^{(\tau)} \sqcap \mathbf{R}_2{}^{(\tau)} \\
E & ::= & \varepsilon \mid \mathbf{R}^{(\tau)}|_{\$i^o,\$j^o} \mid E_1 \circ E_2 \mid E_1 \sqcup E_2 \mid E^* \\
C & ::= & \top_\mathcal{C} \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid \exists E.C \mid \exists[\$i^o]\mathbf{R}^{(\tau)} \mid (\leq k\,[\$i^o]\mathbf{R}^{(\tau)}) \\
V & ::= & \top_\mathcal{D} \mid D \mid \{v_1, \ldots, v_h\} \mid \neg V \mid V_1 \sqcap V_2
\end{array}
$$

where i^o and j^o are components in τ of type *obj*, i^v is a component in τ of type *val*, each v_i in $\{v_1, \ldots, v_h\}$ is a value associated with some domain D_i in $\Sigma_\mathcal{D}$, and k denotes a nonnegative integer.

Before providing the formal semantics we briefly discuss the intuitive meaning of the introduced constructors:

- $\mathbf{R}_1{}^{(\tau)} \sqcap \mathbf{R}_2{}^{(\tau)}$ denotes *conjunction of relationships* with identical signature. Notice that only relationships of the same signature can be combined to form expressions of type $\mathbf{R}_1{}^{(\tau)} \sqcap \mathbf{R}_2{}^{(\tau)}$ (which inherit the signature τ). Therefore we omit the signature of a relationship whenever it is apparent from the context.
- $(\$i^o{}_\tau\!: C)$ denotes a relationship of signature τ whose component i^o is of type C (similarly for $(\$i^v{}_\tau\!: V)$). These are used essentially to impose conditions on *single components of relationships*.
- $\neg\mathbf{R}^{(\tau)}$ is used essentially together with conjunction to denote *difference of relationships*. We consider *disjunction* of relationships $\mathbf{R}_1{}^{(\tau)} \sqcup \mathbf{R}_2{}^{(\tau)}$ as an abbreviation for $\neg(\neg\mathbf{R}_1{}^{(\tau)} \sqcap \neg\mathbf{R}_2{}^{(\tau)})$.
- $\mathbf{R}^{(\tau)}|_{\$i^o,\$j^o}$ denotes the *projection* of a relationship on its two components i^o and j^o. It is used to construct regular expressions which represent "navigations" through a conceptual model. These are then used inside the $\exists E.C$ constructor to impose complex conditions on how objects are connected to each other via relationships. We may use $\forall E.C$ as an abbreviation for $\neg\exists E.\neg C$.
- $\neg C$ and $C_1 \sqcap C_2$ denote *negation* and *conjunction of concepts*. We consider *disjunction* of concepts $C_1 \sqcup C_2$ as an abbreviation for $\neg(\neg C_1 \sqcap \neg C_2)$.

- $\exists[\$i^o]\mathbf{R}^{(\tau)}$ denotes *existential quantification* over relationships and is used to impose the participation of objects in relationships. Notice that this constructor cannot be used with components of type *val*. We may use $\forall[\$i^o]\mathbf{R}^{(\tau)}$ as an abbreviation for $\neg\exists[\$i^o]\neg\mathbf{R}^{(\tau)}$, and use it to impose conditions on the set of all tuples of which an object is component i^o.

- $(\leq k\,[\$i^o]\mathbf{R}^{(\tau)})$ denotes a *qualified number restriction* over relationships and imposes a limit on the maximum number of tuples of a relationship in which an object is component i^o. In particular, when k is equal to 1 it can be used to denote functional dependencies and, together with negation, existence dependencies. We consider $(\geq k\,[\$i^o]\mathbf{R}^{(\tau)})$ as an abbreviation for $\neg(\leq k{-}1\,[\$i^o]\mathbf{R}^{(\tau)})$, and $(= k\,[\$i^o]\mathbf{R}^{(\tau)})$ as an abbreviation for $(\geq k\,[\$i^o]\mathbf{R}^{(\tau)}) \sqcap (\leq k\,[\$i^o]\mathbf{R}^{(\tau)})$

2.1.2 Semantics of $\mathcal{DLR}$

The semantics of $\mathcal{DLR}_{reg}$ is specified through the notion of interpretation. An *interpretation* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of a $\mathcal{DLR}_{reg}$ schema $\mathcal{S}$ is constituted by an *interpretation domain* $\Delta^\mathcal{I}$ and an *interpretation function* $\cdot^\mathcal{I}$ that assigns

- to each domain $D \in \Sigma_\mathcal{D}$ a subset $C^\mathcal{I}$ of $\Delta^\mathcal{I}$,
- for each domain $D \in \Sigma_\mathcal{D}$, to each domain value v in D an element $v^\mathcal{I}$ of $\Delta^\mathcal{I}$ under the unique name assumption,
- to each concept C a subset $C^\mathcal{I}$ of $\Delta^\mathcal{I}$,
- to each regular expression E a subset $E^\mathcal{I}$ of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$,
- to each relationship $\mathbf{R}$ of arity n a subset $\mathbf{R}^\mathcal{I}$ of $(\Delta^\mathcal{I})^n$,

such that the following conditions are satisfied ($\mathbf{P}$, $\mathbf{R}$, $\mathbf{R}_1$, and $\mathbf{R}_2$ have signature $\tau = \langle t_1, \ldots, t_n\rangle$):

$$
\begin{array}{rcl}
\top_\tau^\mathcal{I} & \subseteq & (\Delta^\mathcal{I})^n \\
\mathbf{P}^\mathcal{I} & \subseteq & \top_\tau^\mathcal{I} \\
(\$i^o{}_\tau\!: C)^\mathcal{I} & = & \{(d_1,\ldots,d_n) \in \top_\tau^\mathcal{I} \mid d_i \in C^\mathcal{I}\} \\
(\$i^v{}_\tau\!: V)^\mathcal{I} & = & \{(d_1,\ldots,d_n) \in \top_\tau^\mathcal{I} \mid d_i \in V^\mathcal{I}\} \\
(\neg\mathbf{R})^\mathcal{I} & = & \top_\tau^\mathcal{I} \setminus \mathbf{R}^\mathcal{I} \\
(\mathbf{R}_1 \sqcap \mathbf{R}_2)^\mathcal{I} & = & \mathbf{R}_1^\mathcal{I} \cap \mathbf{R}_2^\mathcal{I} \\
\\
\varepsilon^\mathcal{I} & = & \{(x,x) \mid x \in \Delta^\mathcal{I}\} \\
(\mathbf{R}|_{\$i,\$j})^\mathcal{I} & = & \{(x_i,x_j) \mid (x_1,\ldots,x_n) \in \mathbf{R}^\mathcal{I}\} \\
(E_1 \circ E_2)^\mathcal{I} & = & E_1^\mathcal{I} \circ E_2^\mathcal{I} \\
(E_1 \sqcup E_2)^\mathcal{I} & = & E_1^\mathcal{I} \cup E_2^\mathcal{I} \\
(E^*)^\mathcal{I} & = & (E^\mathcal{I})^* \\
\\
\top_\mathcal{C}^\mathcal{I} & \subseteq & \Delta^\mathcal{I} \\
A^\mathcal{I} & \subseteq & \top_\mathcal{C}^\mathcal{I} \\
(\neg C)^\mathcal{I} & = & \top_\mathcal{C}^\mathcal{I} \setminus C^\mathcal{I} \\
(C_1 \sqcap C_2)^\mathcal{I} & = & C_1^\mathcal{I} \cap C_2^\mathcal{I} \\
(\exists E.C)^\mathcal{I} & = & \{d \in \top_\mathcal{C}^\mathcal{I} \mid \exists d' \in C^\mathcal{I}.(d,d') \in E^\mathcal{I}\} \\
(\exists[\$i]\mathbf{R})^\mathcal{I} & = & \{d \in \top_\mathcal{C}^\mathcal{I} \mid \exists(d_1,\ldots,d_n) \in \mathbf{R}^\mathcal{I}.d_i = d\} \\
(\leq k\,[\$i]\mathbf{R})^\mathcal{I} & = & \{d \in \top_\mathcal{C}^\mathcal{I} \mid |\{(d_1,\ldots,d_n) \in \mathbf{R}^\mathcal{I} \mid d_i = d\}| \leq k\} \\
\\
\top_\mathcal{D}^\mathcal{I} & \subseteq & \Delta^\mathcal{I} \\
D^\mathcal{I} & \subseteq & \top_\mathcal{D}^\mathcal{I} \\
v^\mathcal{I} & \in & \top_\mathcal{D}^\mathcal{I} \\
\{v_1,\ldots,v_h\}^\mathcal{I} & = & \{v_1^\mathcal{I},\ldots,v_h^\mathcal{I}\} \\
\neg V^\mathcal{I} & = & \top_\mathcal{D}^\mathcal{I} \setminus V^\mathcal{I} \\
(V_1 \sqcap V_2)^\mathcal{I} & = & V_1^\mathcal{I} \cap V_2^\mathcal{I}
\end{array}
$$

Notice that in the semantics we have enforced that the extension of every relationship of signature τ is contained in the extension of $\top_\tau$. However the extension of $\top_\tau$ is *not* forced to be the n-Cartesian product of the extensions of the single components in τ. Hence negated relationships essentially are able to capture difference of relationships, since the extension of $\neg\mathbf{R}^{(\tau)}$ is obtained as the difference between the extension of $\top_\tau$ and the extension of $\mathbf{R}^{(\tau)}$.

In order to take into account concrete domains (such as strings, integers, reals, ...) we restrict our interest to interpretation of a special form, called *admissible interpretations*, in which each domain and each domain value have a fixed extension. Formally, we define a function $dom(\cdot)$ which associates a concrete domain $dom(D)$ with each domain name $D \in \Sigma_{\mathcal{D}}$, and define

$$\Delta_{\mathcal{D}} \;=\; \bigcup_{D \in \Sigma_{\mathcal{D}}} dom(D).$$

Moreover, we define a function $\alpha(\cdot)$ which associates a distinct element of the concrete domain $dom(D)$ with each domain value v of the domain D, i.e., $\alpha(v) \in dom(D)$, and $\alpha(v_1) \neq \alpha(v_2)$ if $v_1 \neq v_2$.

The interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over the alphabet Σ is *admissible* if the following conditions are satisfied:

1. $\Delta^{\mathcal{I}} = \top_{\mathcal{C}}^{\mathcal{I}} \,\dot\cup\, \Delta_{\mathcal{D}}$;
2. $\top_{\mathcal{D}}^{\mathcal{I}} = \Delta_{\mathcal{D}}$;
3. $D^{\mathcal{I}} = dom(D)$, for each $D \in \Sigma_{\mathcal{D}}$;
4. $v^{\mathcal{I}} = \alpha(v)$, for each value of each $D \in \Sigma_{\mathcal{D}}$;
5. $\top_{\langle t_1,\ldots,t_n\rangle}^{\mathcal{I}} \subseteq \Delta_1 \times \cdots \times \Delta_n, \quad \Delta_i = \begin{cases} \top_{\mathcal{C}}^{\mathcal{I}}, & \text{if } t_i = obj \\ \Delta_{\mathcal{D}}, & \text{if } t_i = val \end{cases}$ for each $\top_{\langle t_1,\ldots,t_n\rangle} \in \Sigma_{\mathcal{R}}$.

In the following we will implicitly consider only admissible interpretations.

2.1.3 The Enterprise and Source Models

We consider two kinds of conceptual models in source integration, namely, the Enterprise Model, and the Source Models, one for each relevant source. We use the term *conceptual model* to denote either the Enterprise Model or one of the Source Models.

A $\mathcal{DLR}_{reg}$ *conceptual model* $\mathcal{M}$ over the alphabet Σ and domains $\Sigma_{\mathcal{D}}$ is constituted by a finite set of *assertions*, of the form

$$L_i \;\sqsubseteq\; L_j \qquad\qquad L_i \;\not\sqsubseteq\; L_j \qquad\qquad L_i \;\equiv\; L_j \qquad\qquad L_i \;\not\equiv\; L_j$$

where L_i and L_j are either two relationships with the same signature, or two concepts over $\Sigma \cup \Sigma_{\mathcal{D}}$, or two domains.

In order to define the semantics of assertions we specify when an interpretation $\mathcal{I}$ satisfies an assertion as follows: $\mathcal{I}$ satisfies

$$\begin{array}{lcl} L_i \sqsubseteq L_j & \text{iff} & L_i^{\mathcal{I}} \subseteq L_j^{\mathcal{I}} \\ L_i \not\sqsubseteq L_j & \text{iff} & L_i^{\mathcal{I}} \not\subseteq L_j^{\mathcal{I}} \\ L_i \equiv L_j & \text{iff} & L_i^{\mathcal{I}} = L_j^{\mathcal{I}} \\ L_i \not\equiv L_j & \text{iff} & L_i^{\mathcal{I}} \neq L_j^{\mathcal{I}} \end{array}$$

An interpretation $\mathcal{I}$ is a *model* of a conceptual model $\mathcal{M}$ if and only if:

- $\mathcal{I}$ is an admissible interpretation and
- $\mathcal{I}$ satisfies all assertions in $\mathcal{M}$.

$\mathcal{DLR}_{reg}$ can be specialized to the *Entity-Relationship* (ER) Model, since each ER diagram can be translated into a $\mathcal{DLR}_{reg}$ conceptual model capturing its semantics. For the details of the translation, and a discussion of the advantages of adopting an expressive conceptual model like $\mathcal{DLR}_{reg}$, we refer to [CDGL+98b]. Here we only illustrate the translation on an example, which we will extend in the following sections.

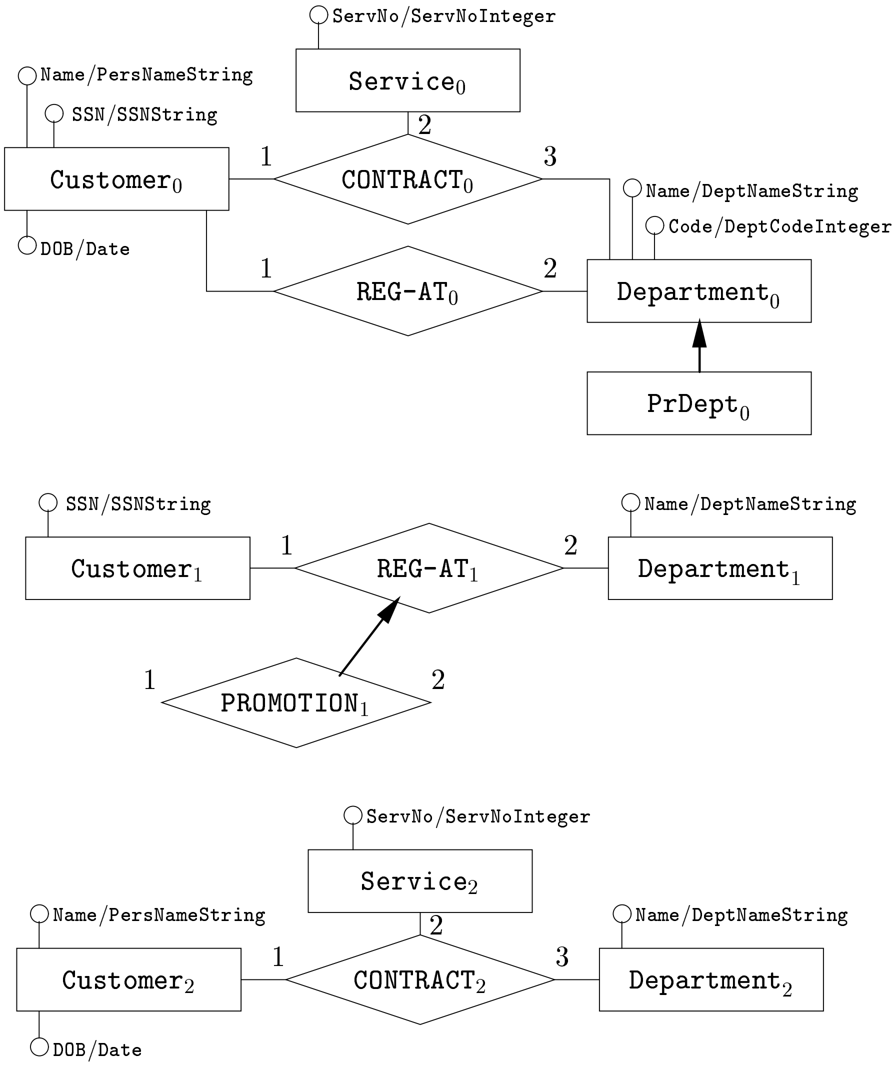

Figure 2: Enterprise and source models as Entity-Relationship diagrams

Example 1 We would like to model 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 can be represented by means of the ER diagrams shown in Figure 2, where for simplicity we omit the specification of the attributes, except for LOCATION.

By applying the translation we obtain the following $\mathcal{DLR}_{reg}$ conceptual models: an Enterprise Model $\mathcal{M}_0$, and two Source Models $\mathcal{M}_1$ and $\mathcal{M}_2$. The alphabets $\Sigma_0, \Sigma_1, \Sigma_2$ of the three models are defined as follows:

- $\Sigma_0 = \{\texttt{CONTRACT}_0, \texttt{REG-AT}_0, \texttt{Customer}_0, \texttt{Service}_0, \texttt{Department}_0, \texttt{PrDept}_0\}$
- $\Sigma_1 = \{\texttt{REG-AT}_1, \texttt{PROMOTION}_1, \texttt{Customer}_1, \texttt{Department}_1, \texttt{LOCATION}_1\}$
- $\Sigma_2 = \{\texttt{CONTRACT}_2, \texttt{Customer}_2, \texttt{Department}_2, \texttt{Service}_2\}$

The various relationships in the alphabets have the following signatures:

$\texttt{CONTRACT}_0$, $\texttt{CONTRACT}_2$, have signature $\tau = \langle obj, obj, obj \rangle$;
$\texttt{REG-AT}_0$, $\texttt{REG-AT}_1$, $\texttt{PROMOTION}_1$, have signature $\tau' = \langle obj, obj \rangle$;
$\texttt{LOCATION}_1$ has signature $\tau'' = \langle obj, val \rangle$.

Finally, the assertions defining the three models are as follows:

- $\mathcal{M}_0$ contains the assertions:

$$\begin{array}{rcl}
\mathtt{CONTRACT}_0 & \sqsubseteq & (\$1\!:\mathtt{Customer}_0) \sqcap (\$2\!:\mathtt{Department}_0) \sqcap (\$3\!:\mathtt{Service}_0) \\
\mathtt{REG\text{-}AT}_0 & \sqsubseteq & (\$1\!:\mathtt{Customer}_0) \sqcap (\$2\!:\mathtt{Department}_0) \\
\mathtt{PrDept}_0 & \sqsubseteq & \mathtt{Department}_0
\end{array}$$

- $\mathcal{M}_1$ contains the assertions:

$$\begin{array}{rcl}
\mathtt{REG\text{-}AT}_1 & \sqsubseteq & (\$1\!:\mathtt{Customer}_1) \sqcap (\$2\!:\mathtt{Department}_1) \\
\mathtt{PROMOTION}_1 & \sqsubseteq & \mathtt{REG\text{-}AT}_1 \\
\mathtt{LOCATION}_1 & \sqsubseteq & (\$1\!:\mathtt{Department}_1) \sqcap (\$2\!:\mathtt{String}) \\
\mathtt{Department}_1 & \sqsubseteq & (\leq 1\, [\$1]\mathtt{LOCATION}_1)
\end{array}$$

- $\mathcal{M}_2$ contains the assertions:

$$\mathtt{CONTRACT}_2 \sqsubseteq (\$1\!:\mathtt{Customer}_2) \sqcap (\$2\!:\mathtt{Department}_2) \sqcap (\$3\!:\mathtt{Service}_2)$$

■

2.1.4 The Conceptual Data Warehouse Model

The Conceptual Data Warehouse Model provides a conceptual representation of the enterprise and the various data sources. Formally, we consider a Conceptual Data Warehouse Model to be a set of conceptual models $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_m$, where $\mathcal{M}_0$ is the Enterprise Model, and each $\mathcal{M}_i$, with $i \in \{1, \ldots, m\}$, is the Source Model for source S_i. Each conceptual model $\mathcal{M}_i$ ($i \in \{0, \ldots, m\}$) is over an alphabet Σ_i and a set $\Sigma_{\mathcal{D}}$ of common domains. In the following we use subscript i when denoting symbols of the alphabet Σ_i.

Inclusion assertions in a conceptual model $\mathcal{M}_i$ express knowledge on the relationships, concepts, and domains in $\mathcal{M}_i$. For this reason such assertions are also called *intramodel assertions*. Additionally, to specify knowledge on the conceptual interrelationships among the various sources and/or the enterprise, we introduce special assertions called intermodel assertions [CL93]. An *intermodel assertion* over Σ_i, Σ_j ($i \neq j$) is an assertion of one of the following forms

$$\begin{array}{llllllll}
L_i \sqsubseteq_{ext} L_j & \quad L_i \not\sqsubseteq_{ext} L_j & \quad L_i \equiv_{ext} L_j & \quad L_i \not\equiv_{ext} L_j \\
L_i \sqsubseteq_{int} L_j & \quad L_i \not\sqsubseteq_{int} L_j & \quad L_i \equiv_{int} L_j & \quad L_i \not\equiv_{int} L_j
\end{array}$$

in which L_i and L_j are either two relationships with compatible signatures or two concepts over Σ_i and Σ_j respectively. Notice that since all domains are common to all conceptual model we do not allow to express intermodel assertions between domains. In fact, we can assume for simplicity that all assertions on domains are included in the Enterprise Model.

Intermodel assertions are interpreted by defining when an interpretation satisfies such an assertion as follows:

- If $\mathbf{R}_i$ and $\mathbf{R}_j$ are two relationships over the alphabets Σ_i and Σ_j respectively, with the same signature τ, then $\mathcal{I}$ satisfies

$$\begin{array}{lcc}
\mathbf{R}_i \sqsubseteq_{ext} \mathbf{R}_j & \text{iff} & \mathbf{R}_i^{\mathcal{I}} \subseteq \mathbf{R}_j^{\mathcal{I}} \\
\mathbf{R}_i \not\sqsubseteq_{ext} \mathbf{R}_j & \text{iff} & \mathbf{R}_i^{\mathcal{I}} \not\subseteq \mathbf{R}_j^{\mathcal{I}} \\
\mathbf{R}_i \equiv_{ext} \mathbf{R}_j & \text{iff} & \mathbf{R}_i^{\mathcal{I}} = \mathbf{R}_j^{\mathcal{I}} \\
\mathbf{R}_i \not\equiv_{ext} \mathbf{R}_j & \text{iff} & \mathbf{R}_i^{\mathcal{I}} \neq \mathbf{R}_j^{\mathcal{I}} \\
\mathbf{R}_i \sqsubseteq_{int} \mathbf{R}_j & \text{iff} & \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_i^{\mathcal{I}} \subseteq \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_j^{\mathcal{I}} \\
\mathbf{R}_i \not\sqsubseteq_{int} \mathbf{R}_j & \text{iff} & \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_i^{\mathcal{I}} \not\subseteq \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_j^{\mathcal{I}} \\
\mathbf{R}_i \equiv_{int} \mathbf{R}_j & \text{iff} & \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_i^{\mathcal{I}} = \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_j^{\mathcal{I}} \\
\mathbf{R}_i \not\equiv_{int} \mathbf{R}_j & \text{iff} & \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_i^{\mathcal{I}} \neq \top_{\tau_i}^{\mathcal{I}} \cap \top_{\tau_j}^{\mathcal{I}} \cap \mathbf{R}_j^{\mathcal{I}}
\end{array}$$

- If C_i and C_j are two concepts over the alphabets Σ_i and Σ_j respectively, then $\mathcal{I}$ satisfies

$$
\begin{array}{lll}
C_i \sqsubseteq_{ext} C_j & \text{iff} & C_i^{\mathcal{I}} \subseteq C_j^{\mathcal{I}} \\
C_i \not\sqsubseteq_{ext} C_j & \text{iff} & C_i^{\mathcal{I}} \not\subseteq C_j^{\mathcal{I}} \\
C_i \equiv_{ext} C_j & \text{iff} & C_i^{\mathcal{I}} = C_j^{\mathcal{I}} \\
C_i \not\equiv_{ext} C_j & \text{iff} & C_i^{\mathcal{I}} \neq C_j^{\mathcal{I}} \\
C_i \sqsubseteq_{int} C_j & \text{iff} & \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_i^{\mathcal{I}} \subseteq \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_j^{\mathcal{I}} \\
C_i \not\sqsubseteq_{int} C_j & \text{iff} & \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_i^{\mathcal{I}} \not\subseteq \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_j^{\mathcal{I}} \\
C_i \equiv_{int} C_j & \text{iff} & \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_i^{\mathcal{I}} = \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_j^{\mathcal{I}} \\
C_i \not\equiv_{int} C_j & \text{iff} & \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_i^{\mathcal{I}} \neq \top_{\mathcal{C}i}^{\mathcal{I}} \cap \top_{\mathcal{C}j}^{\mathcal{I}} \cap C_j^{\mathcal{I}}
\end{array}
$$

A *Conceptual Data Warehouse Model* $\mathcal{W}$ over the alphabet $\Sigma_{\mathcal{W}} = \Sigma_0 \,\dot\cup\, \Sigma_1 \,\dot\cup \ldots \dot\cup\, \Sigma_m$ and domains $\Sigma_{\mathcal{D}}$ is an $(m+2)$-tuple $\langle \mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_m, \mathcal{G}\rangle$ such that:

- $\mathcal{M}_0$ is the Enterprise Model, and each $\mathcal{M}_i$ is a Source Model (for each $i \in \{1, \ldots, m\}$).
- $\mathcal{M}_i$ is a $\mathcal{DLR}$ conceptual model (i.e., a finite set of intramodel assertions) over the alphabet Σ_i and domains $\Sigma_{\mathcal{D}}$, for each $i \in \{0, \ldots, m\}$. We assume that Σ_0 contains, for each atomic relationship $\top_{\tau i}$ appearing in some Σ_i ($i \in \{1, \ldots, m\}$), a corresponding atomic relationship $\top_{\tau 0}$.
- $\mathcal{G}$ (for "glue") is a finite set of intermodel assertions, such that each element of $\mathcal{G}$ is an intermodel assertion over Σ_i, Σ_j ($i \neq j$), for some $i, j \in \{0, \ldots, m\}$. We assume that $\mathcal{G}$ always includes for each $i \in \{1, \ldots, m\}$ the following assertions:

$$
\begin{array}{llll}
\top_{\mathcal{C}i} & \sqsubseteq_{ext} & \top_{\mathcal{C}0} & \\
\top_{\tau i} & \sqsubseteq_{ext} & \top_{\tau 0} & \text{for each signature } \tau \text{ in } \mathcal{M}_i
\end{array}
$$

In order to take into account concrete domains, we restrict again our interest to admissible interpretations of a Conceptual DW Model $\mathcal{W}$. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over the alphabet $\Sigma_{\mathcal{W}}$ is *admissible* if the following conditions are satisfied:

1. $\Delta^{\mathcal{I}} = \top_{\mathcal{C}0}^{\mathcal{I}} \,\dot\cup\, \Delta_{\mathcal{D}}$;
2. $\top_{\mathcal{D}}^{\mathcal{I}} = \Delta_{\mathcal{D}}$;
3. $D^{\mathcal{I}} = dom(D)$, for each $D \in \Sigma_{\mathcal{D}}$;
4. $v^{\mathcal{I}} = \alpha(v)$, for each value of each $D \in \Sigma_{\mathcal{D}}$;
5. $\top_{\langle t_1,\ldots,t_n\rangle i}^{\mathcal{I}} \subseteq \Delta_1 \times \cdots \times \Delta_n, \qquad \Delta_j = \begin{cases} \top_{\mathcal{C}i}^{\mathcal{I}}, & \text{if } t_j = obj \\ \Delta_{\mathcal{D}}, & \text{if } t_j = val \end{cases}$ for each $\top_{\langle t_1,\ldots,t_n\rangle i} \in \Sigma_i$.

Finally, an interpretation $\mathcal{I}$ for a Conceptual DW Model $\mathcal{W}$ is a *model* for $\mathcal{W}$ if and only if the following conditions hold:

1. $\mathcal{I}$ is an admissible interpretation;
2. $\mathcal{I}$ satisfies all the intramodel assertions in $\mathcal{M}_i$ ($i \in \{0, \ldots, m\}$);
3. $\mathcal{I}$ satisfies all the intermodel assertions in $\mathcal{G}$.

Observe that, in order to extract from $\mathcal{I}$ an interpretation $\mathcal{I}_i = (\Delta^{\mathcal{I}_i}, \cdot^{\mathcal{I}_i})$ for each conceptual model $\mathcal{M}_i$ ($i \in \{1, \ldots, m\}$), it suffices to define:

- $\Delta^{\mathcal{I}_i} = \top_{\mathcal{C}i}^{\mathcal{I}} \cup \Delta_{\mathcal{D}}$;
- $A_i^{\mathcal{I}_i} = A_i^{\mathcal{I}}$;
- $\mathbf{P}_i^{\mathcal{I}_i} = \mathbf{P}_i^{\mathcal{I}}$.

Example 1 (continued) We illustrate the method by enriching the Enterprise and Source Models with suitable intermodel assertions. The whole situation can be modeled by means of a Conceptual DW Model $\mathcal{W} = \langle \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \mathcal{G} \rangle$, where the alphabets Σ_0, Σ_1 and Σ_2, and the signatures τ, τ', and τ'' are defined as in Section 2.1.3. The intermodel assertions representing the various interdependencies between relationships and concepts in different models are collected in $\mathcal{G}$, which is constituted by the following assertions:

$$
\begin{array}{rcl}
\mathtt{Department}_1 & \equiv_{ext} & \mathtt{PrDept}_0 \\
\mathtt{REG\text{-}AT}_1 & \sqsubseteq_{ext} & \mathtt{REG\text{-}AT}_0 \\
\mathtt{Customer}_1 & \equiv_{ext} & \mathtt{Customer}_0 \sqcap (\geq 1\,[\$1](\mathtt{REG\text{-}AT}_0 \sqcap (\$2 \colon \mathtt{PrDept}_0))) \\
\mathtt{Customer}_0 \sqcap (\geq 1\,[\$1]\mathtt{CONTRACT}_0) & \sqsubseteq_{ext} & (\geq 1\,[\$1]\mathtt{PROMOTION}_1) \\
\\
\mathtt{Customer}_2 & \sqsubseteq_{ext} & \mathtt{Customer}_0 \sqcap (\geq 1\,[\$1]\mathtt{CONTRACT}_0) \\
\mathtt{Department}_2 & \sqsubseteq_{ext} & \mathtt{Department}_0 \\
\mathtt{Service}_2 & \equiv_{ext} & \mathtt{Service}_0 \\
\\
\mathtt{Customer}_1 & \equiv_{int} & \mathtt{Customer}_2 \\
\mathtt{Department}_1 & \equiv_{int} & \mathtt{Department}_2
\end{array}
$$

■

2.2 The Formalism for the Logical Level and for the Mapping to the Conceptual Level

In this section we present the formalism we use for expressing both the Source Schemas and the Data Warehouse Schema, and for specifying the mapping between the Conceptual and the Logical Levels. Such schemas are intended to provide a structural description of the content of both the sources and the materialized views in the Data Warehouse. The design of the formalism was driven by the following requirements:

- The Source Schemas and the Data Warehouse Schema should be described in terms of a formalism positioned at the logical level in the database terminology.
- The connection between the formulation of the logical structure of the schemas and the conceptual level of the representation should be explicit.
- The formalism should be able to support reasoning over both the logical and the conceptual level of the representation.

In order to meet the above requirements, we propose to express the logical schema (of both the sources and the Data Warehouse) in terms of a set of relation schemas. Each relation schema describes either an element of the Source Schema or the logical definition of a materialized view in the Data Warehouse. The connection to the Conceptual Data Warehouse Model is established by defining each relation as a relational query over the elements of the Conceptual Data Warehouse Model. In defining such queries the objects of the conceptual representation need to be coded into values of the logical representation. The problems related to this issue are dealt with in Section 4. Furthermore, we designed suitable techniques for reasoning over relational queries in the formalism (see Section 3).

We allow for the use of constants in queries, each of which is either

- a domain value v, or
- a constant belonging to an alphabet $\mathcal{C}$ of constants, and denoting an object of the interpretation domain.

A query q for a $\mathcal{DLR}_{reg}$ conceptual model $\mathcal{M}$ (either the Enterprise Model or a Source Model) over $\Sigma_{\mathcal{M}}$ is a non-recursive Datalog query, written in the form:

$$q(\vec{\mathbf{x}}) \;\leftarrow\; conj_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee conj_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)$$

where each $conj_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i, \vec{\mathbf{c}}_i)$ is a conjunction of *atoms*, and $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i$ (resp. $\vec{\mathbf{c}}_i$) are all the variables (resp. constants) appearing in the conjunct. Each atom has one of the forms $\mathbf{R}(\vec{\mathbf{t}})$, $C(t)$, or $E(t, t')$, where

- $\vec{\mathbf{t}}$, t, and t' are constants or variables in $\vec{\mathbf{x}}, \vec{\mathbf{y}}_i, \vec{\mathbf{c}}_i$
- $\mathbf{R}$, C, and E are relations, concepts, and regular expressions over $\mathcal{S}$.

The number of variables of $\vec{\mathbf{x}}$ is called the *arity* of q, i.e., the arity of the relation denoted by the query q.

In order to define the semantics of queries we need to extend the interpretation function by assigning to each constant $c \in \mathcal{C}$ (i.e., a constant that is not a domain value) an element of $\top_{\mathcal{C}}^{\mathcal{I}}$. Given such an interpretation $\mathcal{I}$ of a conceptual model $\mathcal{M}$, a query q for $\mathcal{M}$ of arity n is interpreted as the set $q^{\mathcal{I}}$ of n-tuples $(o_1, \ldots, o_n)$, with each $o_i \in \Delta^{\mathcal{I}}$, such that, when substituting each o_i for x_i, the formula

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

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

We observe that the atoms in the queries are arbitrary $\mathcal{DLR}_{reg}$ relations and concepts, freely used in the assertions of the schema. This distinguishes our approach with respect to [DLNS98, LR96], where no constraints can be expressed in the conceptual model on the relations that appear in the queries. The mechanism we adopt for describing the logical content of a source is similar to the one adopted in Information Manifold [LSK95, KLSS95, LRO96], where sources are described in terms of views over the World view. However, we note that our conceptual formalisms is more powerful than the corresponding one in Information Manifold. Moreover, we have defined suitable reasoning techniques over relational queries, as illustrated in Section 3.

We conclude the section with an example of relational query, referring again to the ongoing example.

Example 1 (continued) Suppose that the form of (part of) the actual data in Source 1 can be described in terms of a relational table $\mathtt{TABLE}_1$ with three columns, one for the customer, one for the department which the customer is registered at, and one containing a flag which is `true` if the customer is eligible for a promotion and `false` otherwise. Such a table can be specified by means of the following query:

$$\begin{array}{rcl}\mathtt{TABLE}_1(X, D, P) & \leftarrow & \mathtt{REG\text{-}AT}_1(X, D) \wedge \neg \mathtt{PROMOTION}_1(X, D) \wedge P = \mathtt{false} \vee \\ & & \mathtt{PROMOTION}_1(X, D) \wedge P = \mathtt{true}\end{array}$$

■

3 Reasoning

Providing a formalization of the Conceptual DW Model in terms of the logic $\mathcal{DLR}_{reg}$ and of expressing the logical schemas in terms of relational queries on the Conceptual DW Model allows for exploit the reasoning services in $\mathcal{DLR}_{reg}$ can support several forms of reasoning on the Conceptual DW Model and the logical schemas. This section is devoted to a discussion on such reasoning services.

3.1 Reasoning on the Conceptual Data Warehouse Model

The typical reasoning tasks needed at the conceptual level in order to support the designer (see [CDGL+97b]) consist in verifying the following properties with respect to a Conceptual DW Model $\mathcal{W}$:

- *Consistency of the Conceptual DW Model*, i.e. whether $\mathcal{W}$ admits a model; a special case of this service concerns checking the consistency of the Enterprise Model or one Source Model in isolation.
- *Concept satisfiability*, i.e. whether $\mathcal{W}$ admits a model in which a certain concept has a nonempty extension.
- *Relation satisfiability*, i.e. whether $\mathcal{W}$ admits a model in which a certain relation has a nonempty extension.
- *Concept subsumption*, i.e. whether the extension of one concept is a subset of the extension of another concept in every model of $\mathcal{W}$.

- *Relation subsumption*, i.e. whether the extension of one relation is a subset of the extension of another relation in every model of $\mathcal{W}$.

Checking *concept satisfiability* can be considered as the most basic reasoning task, since all reasoning tasks mentioned above can be reduced to it as follows:

- Checking whether $\mathcal{W}$ is consistent amounts to verify whether the concept $\top_{\mathcal{C}}[0]$ (which denotes the set of all objects in the interpretation domain) is satisfiable in $\mathcal{W}$. Checking whether a Source Model $\mathcal{M}_i$ is consistent in $\mathcal{W}$ amounts to verify whether the concept $\top_{\mathcal{C}}[i]$ is satisfiable in $\mathcal{W}$.
- Checking whether a relation $\mathbf{R}^{(\tau)}$ is satisfiable in $\mathcal{W}$ amounts to verify whether the concept $\exists[\$i^o]\mathbf{R}^{(\tau)}$ is satisfiable.
- Checking whether a concept C is subsumed by a concept C' in $\mathcal{W}$ amounts to verify whether $C \sqcap \neg C'$ is unsatisfiable in $\mathcal{W}$.
- Checking whether a relation $\mathbf{R}^{(\tau)}$ is subsumed by a relation $\mathbf{R}'^{(\tau)}$ in $\mathcal{W}$, amounts to verify whether $\mathbf{R}^{(\tau)} \sqcap \neg\mathbf{R}'^{(\tau)}$ is unsatisfiable in $\mathcal{W}$.

These reasoning tasks can in particular be exploited for computing the concept and relation lattice of the Conceptual DW Model, or more generally the lattice of all concept and relation expressions: this can be performed in an incremental manner by verifying subsumption between all pairs of concepts (relations) in the Conceptual DW Model. Observe that such lattice can be maintained in an incremental manner (see [CDGL+97b] for more details).

Example 1 (continued) we can prove for instance that the assertion

$$\texttt{PROMOTION}_1 \sqsubseteq_{ext} \texttt{REG-AT}_0 \sqcap (\$2\colon \texttt{PrDept}_0) \tag{1}$$

is a logical consequence of the Conceptual DW Model (the reasoning service involved is relation subsumption). Indeed, observe that the Conceptual DW Model includes the following assertions:

$$\begin{aligned}
\texttt{REG-AT}_1 &\sqsubseteq (\$1\colon \texttt{Customer}_1) \sqcap (\$2\colon \texttt{Department}_1) && (2)\\
\texttt{PROMOTION}_1 &\sqsubseteq \texttt{REG-AT}_1 && (3)\\
\texttt{Department}_1 &\equiv_{ext} \texttt{PrDept}_0 && (4)\\
\texttt{REG-AT}_1 &\sqsubseteq_{ext} \texttt{REG-AT}_0 && (5)
\end{aligned}$$

The intramodel assertions (2) and (3) imply that $\texttt{PROMOTION}_1 \sqsubseteq_{ext} (\$2\colon \texttt{Department}_1)$, which, by the intermodel assertions (4) and (5), implies (1). ∎

The expressiveness required by a data model to be able to express meaningful properties in the Conceptual DW Model, and which is featured by $\mathcal{DLR}_{reg}$, makes reasoning a complex task. We have devised a sound and complete procedure to decide the satisfiability of a concept with respect to a Conceptual DW Model which works in worst-case deterministic exponential time in the size of the Conceptual DW Model and the concept. Indeed, it is also possible to show that such worst-case complexity is inherent to the problem, proving that reasoning with respect to a Conceptual DW Model is EXPTIME complete (see [CDGL+97b] for more details).

Theorem 1 ([CDGL+97b]) *Checking concept satisfiability (and thus Conceptual DW Model consistency, Source and Enterprise Model consistency, concept subsumption, relation satisfiability, and relation subsumption) with respect to a Conceptual DW Model is an EXPTIME complete problem.*

3.2 Reasoning Taking into Account the Logical Level

The basic reasoning tasks on the logical schema involve reasoning on relational queries.

- *Query containment.* Given two relational queries q_1 and q_2 (of the same arity n) for a Conceptual DW Model $\mathcal{W}$, we say that q_1 is *contained in* q_2 with respect to $\mathcal{W}$, if the set of tuples denoted by q_1 is contained in the set of tuples denoted by q_2 in every model of $\mathcal{W}$.
- *Query consistency.* A relational query q is *consistent* with respect to a $\mathcal{W}$, if there exists a model of $\mathcal{W}$ where the set of tuples denoted by q is not empty.
- *Query disjointness.* Two relational queries q_1 and q_2 (of the same arity) are *disjoint* with respect to a $\mathcal{W}$, if there exists no model of $\mathcal{W}$ where the set of tuples denoted by q_1 overlaps with the set of tuples denoted by q_2.

In fact, query disjointness can be reduced to query consistency, which can be in turn reduced to query containment. A relational query q_1 is disjoint from q_2 with respect to $\mathcal{W}$ if and only if the relational query $test(\vec{x}) \leftarrow q_1(\vec{x}) \wedge q_2(\vec{x})$ is inconsistent with respect to $\mathcal{W}$. On the other hand, a relational query q is consistent with respect to $\mathcal{W}$ if and only if q is not contained in the relational query $empty(\vec{x}) \leftarrow R(\vec{x}) \wedge \neg R(\vec{x})$ with respect to $\mathcal{W}$, where the arity of *empty* is that of q.

Example 1 (continued) Consider the following two queries posed to the Conceptual DW Model

$$\begin{array}{lcl} q_1(X,Y) & \leftarrow & \texttt{Customer}_0(X) \wedge \texttt{CONTRACT}_0(X,Y,Z) \\ q_2(X,Y) & \leftarrow & \texttt{Customer}_0(X) \wedge \texttt{CONTRACT}_0(X,Y,Z) \wedge \texttt{REG-AT}_0(X,W) \wedge \texttt{PrDept}_0(W) \end{array}$$

It is immediate to see that q_2 is contained in q_1, even without taking the Conceptual DW Model into account. However, also the converse containment can be proved. Indeed, from assertion (1), which we proved above, it follows that

$$(\geq 1\, [\$1]\texttt{PROMOTION}_1) \sqsubseteq_{ext} (\geq 1\, [\$1](\texttt{REG-AT}_0 \sqcap (\$2{:}\,\texttt{PrDept}_0))).$$

This, together with the intermodel assertion

$$\texttt{Customer}_0 \sqcap (\geq 1\, [\$1]\texttt{CONTRACT}_0) \sqsubseteq_{ext} (\geq 1\, [\$1]\texttt{PROMOTION}_1)$$

in the Conceptual DW Model implies that q_1 is contained in q_2. ■

We have devised sound and complete procedures to decide query containment with respect to a Conceptual DW Model that work in worst-case exponential time in the size of the Conceptual DW Model (see [CDGL+97b, CDGL98a] for more details). The results are summarized in the theorem below.

Theorem 2 [CDGL98a] *Deciding whether a relational query q_1 is contained in a relational query q_2 with respect to a Conceptual DW Model $\mathcal{W}$, where either q_1 does not contain regular expressions or both q_1, q_2, and $\mathcal{W}$ do not contain number restrictions, can be done in time $O(2^{p(|\mathcal{W}|\cdot \ell_1^{\ell_2})})$, where $|\mathcal{W}|$ is the size of $\mathcal{W}$, ℓ_1 is the sum of the number of variables in q_1 and the number of constants and values in q_1, q_2, and $\mathcal{W}$, and ℓ_2 is bounded by the number of existentially quantified variables in q_2.*

4 Source and Data Warehouse Schema Specification

In this section we focus on the phase devoted to the specification of the logical schemas of the sources and the logical schema of the Data Warehouse. We analyze the important case where schemas are expressed in terms of the relational model. The section is organized as follows. We first present the notion of adorned query, which is at the basis of our approach. Then, we discuss the various steps required by the methodology both in Source and in Data Warehouse Schema specification. Finally, we illustrate how quality analysis should be performed in the phase of Source and Data Warehouse Schema Specification.

4.1 The Notion of *Adorned Query*

As we said in Section 2.2, the connection between the conceptual level and the logical level is established by associating with each relation at the logical level, a query over the elements of the Conceptual DW Model.

When dealing with the phase of Source Schema and Data Warehouse Schema specification, in the definition of the relations, it should be made 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 a 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 in Section 2.2, 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

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

   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.

Example 1 (continued) 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:

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

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 $\mathtt{TABLE}_2$:

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

■

The adorned query associated to a relational table (in a source or in the DW) 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 relational 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 is whether the adorned query associated with a relational table 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 [CDGL$^+$98c] allow us to check the consistency of the relational tables defined for describing a source.

4.2 Source Schema Specification

The goal of this step is to produce or enrich the Source Schemas, i.e., the logical view of the sources, so as to characterize the actual data in the sources in terms of a logical model of data. As we said before, we concentrate our attention on the case of sources expressed at the logical level in terms of the relational model. Notice that, in order to guarantee the access to the actual data of the source, a specific wrapper for the source has to be provided, if not already available.

In our methodology, the main activities to be carried out during the phase of Source Schema specification can be summarized as follows. Suppose we are analyzing Source S_1, and suppose we want to provide the specification of a relation T contained in S_1. For this purpose, the following steps should be carried out:

1. In order to link the content of T to the Conceptual DW Model, associate to T a query q, using the formalism described in Section 2.2. During this step, special attention should be devoted to the problem of null values. We remind the reader that null values in a relation T in a source are represented by means of a special constant appearing in the query associated to T.

2. Check how the values and the objects are represented in T, and specify the annotation on the variables accordingly, thus deriving the adorned query associated with T.

3. Check the quality of the adorned query, following the suggestions specified in Section 4.4. In particular, if the adorned query is inconsistent, then change the original query so as to eliminate the inconsistency, and go back to the previous step.

4. If the adorned query is consistent, then obtain from it the *specification of the relational schema* T, i.e., the specification of:

   - the name of the relation,
   - the number of columns,
   - the domain of each column,
   - whether each column admits null values or not,
   - the associated query, which specifies conceptually the content of the relation in the source,
   - the associated adorned query, which specifies the actual data stored in the relation in the source.

We point out that our method is able to cope with several *schematic differences* that may be present in the sources [SK92]. We illustrate this point with the help of an example.

Example 2 Suppose that the Conceptual Model contains a relationship `Service` with three attributes, `Date`, `ServiceNo`, and `Price`, where `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 `v1`(D, P) means that service v_1 costs P Italian Lira at date D, and $\mathtt{v2}(D,P)$ 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 $\mathtt{Serv}(X,Y,D)$ 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 $\mathtt{Serv}_3$. The various tables in the three sources can be specified by means of the following adorned queries:

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

■

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.

4.3 Data Warehouse Schema Specification

This step concerns the task of building, enriching, or restructuring the Data Warehouse Schema, i.e., the logical view of the data that are actually materialized in the Data Warehouse. Also in this case, the logical view is constituted by a set of relations, each one specified as a query over the Conceptual DW Model. This step is also included in the sequence of actions to be performed in the client-driven methodology. In this case, the restructuring may be needed on the basis of new data available after the analysis of a new source (or a new portion of a source). Indeed, the availability of new concepts and relations in the Conceptual DW Model, and the possibility of characterizing how such new concepts are represented in the sources, may result in a new organization of the materialized data in the Data Warehouse, in order to better meet quality requirements. For example, the designer may want to reconsider design questions such as

1. What is appropriate to materialize in the Data Warehouse?
2. What kind of aggregations are needed in the Data Warehouse?
3. How should the logical schema be organized in order to effectively represent the multidimensional modeling requirements?

Finally, notice that a restructuring of the Data Warehouse Schema may require the design of new mediators.

In our methodology, the main activities to be carried out during the phase of Data Warehouse Schema specification can be summarized as follows. Suppose we want to provide the specification of a relation T contained in the Data Warehouse Schema. For this purpose, analogously to the case of Source Schema specification, the following steps should be carried out:

1. In order to link the content of T to the Conceptual DW Model, associate to T a query q, using the formalism described in Section 2.2. During this step, special attention should be devoted to the problem of null values.

2. Check how the values and the objects should be represented in T, and specify the annotation on the variables accordingly, thus deriving the adorned query associated with T.

3. Check the quality of the adorned query, following the suggestion specified in Section 4.4. In particular, if the adorned query is inconsistent, then change the original query so as to eliminate the inconsistency, and go back to the previous step.

4. If the resulting adorned query is consistent, then obtain the specification of the relational schema T, i.e., the specification of:
   - the name of the relation,
   - the number of columns,
   - the domain of each column,
   - whether each column admits null values or not,
   - the associated query, which specifies conceptually the content of the relation in the Data Warehouse,
   - the associated adorned query, which specifies the actual data stored in the relation of the Data Warehouse.

Example 1 (continued) 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 query:

$$\begin{array}{lll} \mathtt{TDW}(X,S,D,P) & \leftarrow & \mathtt{CONTRACT}_0(X,S,D) \wedge \mathtt{PROMOTION}_1(X,D) \wedge F = \mathtt{true} \vee \\ & & \mathtt{CONTRACT}_0(X,S,D) \wedge \mathtt{REG\text{-}AT}_1(X,D) \wedge \neg\mathtt{PROMOTION}_1(X,D) \wedge P = \mathtt{false} \vee \\ & & \mathtt{PROMOTION}_1(X,D) \wedge P = \mathtt{true} \wedge S = \mathtt{NULL} \end{array}$$

Assuming that in the Data Warehouse we want to identify customers by their SSN, services by their service number, and departments by their code, we can adorn the above query as follows:

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

■

4.4 Quality Analysis

During the phase of Source and Data Warehouse Schema specification, the metadata repository is exploited for storing the representation of the logical schemas, inserting the values of the corresponding quality factors, and querying the metadata in order to get various information (see Section 6).

Checking the quality of the logical specification is a fundamental aspect in this phase. The most important checks are related to the need of verifying whether the specification of the various relation schemas (both Source and Data Warehouse schemas) is consistent, according to the notion of *consistency* of an adorned query. As said in Section 4.1, the consistency check can be done by exploiting the algorithm for query containment (see Section 3).

Another interesting check is the one related to the need of verifying whether there are redundancies in the Data Warehosue Schema. For this purpose, for every relation r in the Data Warehouse, we can check whether there is another relation r' such that r is contained in r'. If the answer is yes, it is obvious that there is redundancy in the Data Warehouse, in the sense that all tuples of r are represented in r'. Note that checking containment between relations in the Data Warehouse means checking containment of the associated queries. Again, such check can be done by exploiting the algorithm for query containment.

5 Data Integration and Reconciliation

In the context of Data Warehousing, the problem of data integration and reconciliation arises when data passes from the application-oriented environment to the Data Warehouse. During the transfer of data, possible inconsistencies and redundancies should be resolved, so that the warehouse is able to provide and integrated and reconciled view of data of the organization.

As reported in [CDGL+97b] most of the work on database integration concerns the schema level, whereas instance level integration have received less attention. Nevertheless, recent works, such as [ZHK96, HZ96, ZHKF95, CW91, PGMW95] aim at providing methodological guidelines for the problem of comparing, matching, and merging data and objects residing in different sources.

In our methodology, the problem of data reconciliation is addressed after the phase of Data Warehouse Schema specification, and is based on specifying how the relations in the Data Warehouse Schema are linked to the relations in the Source Schemas. In particular, data reconciliation is based on the following principles:

- The designer specifies how data in one source can be matched with or converted into data in another source, and how data in different sources can be reconciled into data in the data warehouse. The specification is provided in terms of the so-called interschema correspondences (See Subsection 5.1).
- Once interschema correspondences are specified, the problem is addressed of how the tuples of the data warehouse relations should be constructed from a suitable set of tuples extracted from the relations stored in the sources. This kind of information is crucial for the task of data acquisition in the Data Warehouse, and in particular for understanding
  - how the loading of data in the Data Warehouse should be carried on;
  - how the refreshment of the Data Warehouse with data suitably extracted from the sources should be performed.

5.1 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:

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

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:

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

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:

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

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 *reconcile* has the following form:

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

where *conj* specifies the conditions under which the reconciliation is applicable, and *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.,

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

In practice, several of the Interschema Correspondences that must be specified will have a very simple form, since they will correspond simply to equality in the case of a matching and to identity in the case of a conversion. 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:

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

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 $\mathit{convert}_i$ asserted by the designer, the system automatically asserts a new Matching Correspondence match_i as follows:

$$\begin{array}{rcl} \mathit{match}_i([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) & \leftarrow & \mathit{convert}_i([\vec{\mathbf{x}}],[\vec{\mathbf{z}}]) \wedge \vec{\mathbf{y}} = \vec{\mathbf{z}} \\ & & \textsf{through } \mathit{none} \end{array}$$

Moreover, for each Conversion Correspondence $\mathit{convert}_i$ 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 $\mathit{reconcile}_{i,j}$ as follows:

$$\begin{array}{rcl} \mathit{reconcile}_{i,j}([\vec{\mathbf{x}}],[\vec{\mathbf{y}}],[\vec{\mathbf{z}}]) & \leftarrow & \mathit{match}_i([\vec{\mathbf{x}}],[\vec{\mathbf{y}}]) \wedge \mathit{convert}_j([\vec{\mathbf{x}}],[\vec{\mathbf{z}}]) \\ & & \textsf{through } \mathit{none} \end{array}$$

Example 1 (continued) The following Conversion Correspondence specifies that the name and date of birth of a person can be converted into a Social Security Number through the program `name_to_SSN`:

$$\begin{array}{rcl} \mathit{convert}_1([N,B],[S]) & \leftarrow & \texttt{Customer}_0(X) \wedge \texttt{PersNameString}(N) \wedge \texttt{Date}(B) \wedge \texttt{Name}(X,N) \wedge \texttt{DOB}(X,B) \wedge \\ & & \texttt{SSNString}(S) \wedge \texttt{SSN}(X,S) \\ & & \textsf{through } \texttt{name\_to\_SSN}(N,B,S) \end{array}$$

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`:

$$\begin{array}{rcl} \mathit{convert}_2([M],[C]) & \leftarrow & \texttt{DeptNameString}(M) \wedge \texttt{DeptCodeInteger}(C) \\ & & \textsf{through } \texttt{dept\_name\_to\_code}(M,C) \end{array}$$

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

$$\begin{array}{rcl} \mathit{match}_1([N,B],[S]) & \leftarrow & \mathit{convert}_1([N,B],[S_1]) \wedge S = S_1 \\ & & \textsf{through } \mathit{none} \\ \mathit{match}_2([M],[C]) & \leftarrow & \mathit{convert}_2([M],[C_1]) \wedge C = C_1 \\ & & \textsf{through } \mathit{none} \\ \mathit{match}_3([M_1],[M_2]) & \leftarrow & \texttt{DeptNameString}(M_1) \wedge \texttt{DeptNameString}(M_2) \wedge \\ & & M_1 = M_2 \\ & & \textsf{through } \mathit{none} \end{array}$$

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

and the Reconciliation Correspondences

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

■

5.2 Specification of mediators

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.[2] 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 4.3 to provide the specification of the relation T. In other words, we specify T in terms of an adorned query

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

   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 DW 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 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:

[2] To see how DWQ addresses the issue of deciding what to materialize in the Data Warehouse, we refer to [TLS99].

- 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 techniques presented in [DL97, CDGL99a], 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 $\mathit{tuple\text{-}spec}_1$ and $\cdots$ and $\mathit{tuple\text{-}spec}_n$
  such that *matching-condition*
  into $\mathit{tuple\text{-}spec}_{t_1}$ and $\cdots$ and $\mathit{tuple\text{-}spec}_{t_m}$

  where $\mathit{tuple\text{-}spec}_i$ denotes a tuple returned by the i-th disjunct of r, *matching-condition* specifies how to merge the various tuples denoted by $\mathit{tuple\text{-}spec}_1, \ldots, \mathit{tuple\text{-}spec}_n$, and $\mathit{tuple\text{-}spec}_{t_1}, \ldots, \mathit{tuple\text{-}spec}_{t_m}$ 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 [CDGL98a].
- 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.

Perhaps, the most interesting aspect of the rewriting technique in our context is how the rewriting algorithm takes into account the interschema correspondences. Here we elaborate more on this aspect.

The basic idea of the rewriting algorithm is to generate candidate rewritings, and for each candidate check whether it is really a rewriting, i.e., whether it is contained in the original query. The presence of interschema correspondences and the form of queries (adorned queries) that we use to describe Source and Data Warehouse tables have the following implications on the algorithm:

- Candidate rewritings to be generated are queries containing interschema correspondence predicates, besides source table predicates.
- While in the traditional case the containment check is easy, here we must take into account that the candidate rewriting contains interschema correspondence predicates. More precisely, in the traditional case the containment is carried out by first expanding the candidate rewriting (i.e., substituting every atom in the candidate rewriting with its view definition) and then check if the expansion is a subset of the original query. In our case, the expansion contains the adornments, and we have devised a technique to take into account such adornments when checking containment.

The main problem to address in order to take into account the adornments when checking containment is related to the need of computing joins between different source tables. Suppose that query Q has a join of two relations R_1 and R_2 on two attributes of type *obj*. The question is whether we can rewrite such a join in terms of a join on two attributes, say A and B, of tables T_1 and T_2. To answer this question, we exploit the fact that the join between objects α and β can be represented by a match (through a *match* predicate) between attribute A of T_1 and attribute B of T_2 only if A identifies α in table T_1, and B identifies β in table T_2. It is clear that the *identify* clause in the adornment of the table definition is crucial for this kind of check. Indeed, our technique introduces in the candidate rewriting an equality atom $X = Y$ between two objects whenever A and B match, and $\mathit{identify}([A], X)$ and $\mathit{identify}([B], Y)$ hold. A similar technique allows us to take into account reconciliation correspondences.

Example 1 (continued) Using the asserted and automatically derived Interschema Correspondences, the system is able to rewrite the query defining the Data Warehouse table TDW (see Section 4.3) in terms of $\mathtt{TABLE}_1$ in Source 1 and $\mathtt{TABLE}_2$ in Source 2 (see Section 4.2) as follows:

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

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. ■

5.3 Quality analysis

During the phase of Data Integration quality factors are taken into account in several ways. First, there is the need of checking the quality of the interschema correspondences. The most important checks are related to the need of verifying whether the interschema correspondences are consistent. Again, the consistency check can be done by exploiting the algorithm for query containment (see Section 3).

As we said before, our method aims at reformulating a query q corresponding to a Data Warehouse table, 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 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.

It is interesting to observe that interschema correspondences may be used to reflect design decisions based on quality parameters. For example, suppose that the single-value attribute A of the enterprise model is represented by attribute A_1 in Source 1 and attribute A_2 in Source 2, and suppose that the level of reliability and believability for values of attribute A in Source 1 is clearly higher than the corresponding value for Source 2. To reflect the fact that we prefer to load in the Data Warehouse the value of A_1 than the value of A_2, we can specify the following interschema correspondence:

$$\textit{reconcile}([X],[Y],[X]) \leftarrow A_1(W,X) \wedge A_2(W,Y) \wedge X \neq \texttt{NULL}$$

With this correspondence, we automatically drive the rewriting algorithm to choose the value of Source 1 for attribute A when loading the data in the Data Warehouse. Indeed, with this interschema correspondence, whenever a data conflict arises between the values of A in sources 1 and 2, the rewriting algorithm will prefer the value stored in Source 1, provided that it is not null.

6 Description of the Prototype Tool for Data Integration and Reconciliation

In this section we describe the prototype tool we have developed within DWQ that supports the Data Warehouse design process according to the DWQ methodology (see [CDGL+98b] and Section 5).

In all phases of the design process the tool allows for the insertion, retrieval and update of the metadata relevant for the design. The tool makes use of Concept Base [JGJ+95] as the metadata repository. The query and update functionalities of Concept Base are used by the tool to facilitate access to the information in the repository. Moreover the tool supports the designer in his activity, with special attention to the aspects related to quality analysis, by means of several kinds of checks on the specification, including:

- consistency and redundancy checks on Source models,

- consistency and redundancy checks on Enterprise model,

- consistency and redundancy checks on adorned queries defining Source schemas,

- consistency and redundancy checks on adorned queries defining Data Warehouse schema,

- computation of the rewriting for the specification of mediators.

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 (see Deliverable D4.3 - DWQ-UNIROMA-007).

We illustrate now the structure of the metaclasses that constitute the repository. At a high abstraction level, all objects and classes in the metadata repository are grouped in very general metaclasses, according to whether they belong to the conceptual, logical, or physical level of the DWQ architecture (see Figure 1). This is accomplished by the following class definitions:

```
Class ObjectType end
Class DW_Object in ObjectType end
Class ConceptualObject in ObjectType isA DW_Object end
Class LogicalObject in ObjectType isA DW_Object end
Class PhysicalObject in ObjectType isA DW_Object end
```

In the following we concentrate only on the structure of the metaclasses that allow for storing and accessing the information that is specific for the data integration and reconciliation phase. For the sake of completeness, we include as an appendix (Appendix 1) the definition of the Concept Base metaclasses concerning the conceptual level of the Data Warehouse architecture for DWQ.

All the relevant classes are situated in the logical level of the architecture, and refer to the classes used to store the Conceptual Data Warehouse Model, which are situated at the conceptual level of the architecture, and whose specification is not included here.

Logical Schema: Each schema is either a Source Schema or a Data Warehouse Schema.

```
Class Schema in ObjectType isA LogicalObject end

Class SourceSchema in ObjectType isA Schema with
  attribute, single, necessary
    isSchemaOf : Source
end

Class DWSchema in ObjectType isA Schema end
```

Materialized Tables: The class `Table` is used to represent materialized tables. Hence each table belongs to a specific schema, has an arity and an adorned query defining it.

```
Class Table in ObjectType isA LogicalObject with
  attribute,single,necessary
    refersTo : Schema;
    arity : Integer;
    hasAdornedQuery : AdornedQuery
  constraint
    all_value_Adornment_constraint :
      $ forall r/Table aq/AdornedQuery i,a/Integer
        (r arity a) and (r hasAdornedQuery aq) and (i <= a) ==>
        (exists vc/DomainAnnotation (aq hasAdornment vc) and
                                    (vc variable i)) $;
    onlyConceptRelationship&Attrib_name :
      $ forall r/Table a/AdornedQuery q/Query cq/ConjunctiveQuery c/Conjunct
        (r hasAdornedQuery a) and
        (a hasQuery q) and (q hasConjunct cq) and (cq containsSyntactically c)
        ==> (c in ConceptRelationship) or (c in Attrib_name) $
end
```

Moreover, a table in the Data Warehouse may have an associated mediator (see below).

```
Class DWTable in ObjectType isA Table with
  attribute
    hasMediator : Mediator
end
```

Adorned Query: An adorned query is a query with an adornment, which is a set (multivalued attribute) of annotations (either `DomainAnnotations` or `IdentityAnnotations`).

```
Class AdornedQuery in ObjectType isA LogicalObject with
  attribute, necessary,single
    hasQuery : Query
  attribute,necessary
    hasAdornment : Adornment
```

```
end

Class Adornment in ObjectType isA LogicalObject with
  constraint
    unionAdornment:
      $ forall a/Adornment
        (a in DomainAnnotation) or (a in IdentifyAnnotation) $;
    xorAdornment:
      $ forall v/DomainAnnotation not (v in IdentifyAnnotation) $
end

Class DomainAnnotation in ObjectType isA Adornment with
  attribute,single,necessary
    variable : Integer;
    value : Domain
end

Class IdentifyAnnotation in ObjectType isA Adornment with
  attribute, necessary
    ident : Integer
  attribute,single,necessary
    object : Integer
end
```

Query: Each query is a set of conjunctive queries, to be interpreted as a union of its members. Each conjunctive query is a set of conjuncts, to be interpreted as a conjunction of its members. Each conjunct is either a table, a concept, a relationship, an attribute, a correspondence (see below) or a domain.

```
Class Query in ObjectType with
  attribute,necessary
    hasConjunct : ConjunctiveQuery
end

Class ConjunctiveQuery in ObjectType isA Query with
  attribute,single,necessary
    body : String
  attribute
    containsSyntactically : Conjunct
end

Class Conjunct in ObjectType  isA LogicalObject with
  constraint
    unionConjunct :
      $ forall c/Conjunct
        (c in Table) or (c in ConceptRelationship) or (c in Attrib_name) or
        (c in Correspondence) or (c in Domain) $;
    xorConjunct1 :
      $ forall c/Table
        not (c in ConceptRelationship) and not (c in Attrib_name) and
        not (c in Correspondence) and not (c in Domain) $;
    xorConjunct2 :
      $ forall c/ConceptRelationship
        not (c in Attrib_name) and not (c in Correspondence) and
        not (c in Domain) $;
    xorConjunct3 :
```

```
      $ forall c/Attrib_name
        not (c in Correspondence) and not (c in Domain) $;
    xorConjunct4 :
      $ forall c/Domain not (c in Correspondence) $
end

Table isA Conjunct end
ConceptRelationship isA Conjunct end
Domain isA Conjunct end
Attrib_name isA Conjunct end
```

Interschema Correspondences: Each interschema correspondence is a conjunctive query and has an associated program. We may have conversion, matching, and reconciliation correspondences.

```
Class Correspondence in ObjectType isA Conjunct with
  attribute,single,necessary
    condition : ConjunctiveQuery;
    arity     : Integer
  attribute,necessary
    argument : Integer
  attribute,single
    program : String
  constraint
    notTable :
      $ forall cr/Correspondence q/Query cq/ConjunctiveQuery c/Conjunct
        (cr condition q) and (q hasConjunct cq) and
        (cq containsSyntactically c) ==>
        not (c in Table) $;
    position_constraint :
       $ forall c/Correspondence a1,a2/Correspondence!argument i,p1,p2/Integer
         (Ai(c,argument,a1) and Ai(c,argument,a2) and (a1 position p1) and
          (a2 position p2) and not (a1 == a2) and A(c,arity,i)) ==>
         (not (p1 ==p2) and (p1 <= i) and (p2 <= i)) $;
    all_positions_constraint :
       $ forall c/Correspondence i,a/Integer
         A(c,arity,a) and (i <= a) ==>
         (exists arg/Correspondence!argument Ai(c,argument,arg) and
                                             (arg position i)) $
end

Correspondence!argument in Class with
  attribute,single,necessary
    position : Integer
end

Class Convert in ObjectType isA Correspondence with
  constraint
    c : $ forall c/Convert i/Integer (c arity i) ==> i=2 $
end

Class Reconcile in ObjectType isA Correspondence end
Class Match in ObjectType isA Correspondence end
```

Mediator: A mediator is an adorned query made up of materialized tables and predicates specifying interschema correspondences, with possibly an associated merging clause.

```
Class Mediator in ObjectType isA AdornedQuery with
  attribute
    hasMergingClause : MergingClause
    hasQuery : RewrittenQuery
end

Class RewrittenQuery in ObjectType isA Query with
  attribute
    hasConjunct : RewrittenConjunctiveQuery
end

Class RewrittenConjunctiveQuery in ObjectType isA
      ConjunctiveQuery,RewrittenQuery with
  constraint
  onlyTableCorrespondence :
    $ forall r/RewrittenConjunctiveQuery c/Conjunct
      (r containsSyntactically c) ==>
      (c in Table) or (c in Correspondence) $
end

Class MergingClause in ObjectType isA LogicalObject with
  attribute, necessary
    mergingPart : RewrittenConjunctiveQuery
  attribute, single
    suchThatPart : RewrittenConjunctiveQuery
  attribute,single, necessary
    intoPart : RewrittenConjunctiveQuery
  constraint
    onlyCorrespondence_for_suchThatPart :
      $ forall m/MergingClause r/RewrittenConjunctiveQuery c/Conjunct
        (m suchThatPart r) and (r containsSyntactically c) ==>
        (c in Correspondence) $;
    onlyTable_for_intoPart :
      $ forall m/MergingClause r/RewrittenConjunctiveQuery c/Conjunct
        (m intoPart r) and (r containsSyntactically c) ==>
        (c in Table) $
end
```

7 Semi-structured data sources

In the last year of the project, the case of source schemata expressed in terms of a semistructured data model has also been addressed in more detail.

While in WP3 and WP4 most of the work assumes that sources are expressed in terms of the relational model, the case of semistructured data sources is becoming more and more interesting. Indeed, semistructured data schemata are becoming more and more popular in databases, due to their ability to formalize typical information sources on the World Wide Web. Consequently, it is a common opinion that, in the near future, Data Warehouse applications will need to deal with semistructured data sources.

Unfortunately, basic research problems have still to be solved in this context. One fundamental question when dealing with semistructured data sources is the formalism used to express that data schema and the associated queries. For this purpose, we have defined a data model and a query mechanism for semistructured data which is fully compatible with the data model used in DWQ, and we have shown how to use the formalism to describe XML data sources [CDGL99b, CDGL99c].

Another important research question when integrating semistructured data sources is the one related to query answering and query rewriting using views. In this context, we have defined algorithms for, and we have characterized the computational complexity of query rewriting and query answering using views, in the case of schemata expressed in terms of a semistructured data model [CDGLV99, CDGLV00]. Both queries and views are expressed in terms of a language providing the ability to express regular path queries, and therefore may contain a form of recursion. It is worth noticing that our algorithms are the first ones dealing with queries and views containing recursion.

8 Conclusions

The results reported in the present document show that the tasks within workpackage WP4 (Data Integration and Reconciliation) have been accomplished according to the workplan and significant progress has been made towards the objectives. In particular, we have achieved the following goals:

- We have specified a methodology within the DWQ framework for integrating and reconciling data from different sources. The methodology provides precise guidelines for using the metadata repository during the phases of data integration and reconciliation, in particular for storing the representation of the logical schemas, and querying the metadata in order to get various information.
- The notions of adorned queries and interschema correspondences are the basis for a comprehensive treatment of data integration and reconciliation. Such notions provide the formal support for generating mediators associated to the relation schemas in the Data Warehouse, and which are responsible for loading and maintaining the corresponding relations.
- Based on the described methodology, we have implemented a prototype of a data integration and reconciliation tool, which exploits the Concept Base System as metadata repository. The tool has been used in a case study developed in cooperation with Telecom Italia.
- The case of source schemata expressed in terms of a semistructured data model has also been addressed. We have defined a data model and a query mechanism for semistructured data which is fully compatible with the data model used in DWQ, and we have shown how to use the formalism to describe XML data sources. Moreover, we have defined algorithms for, and we have characterized the computational complexity of query rewriting and query answering using views, in the case of schemata expressed in terms of a semistructured data model.

References

[Bor95] Alexander Borgida. Description logics in data management. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[CDGL95] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD'95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 229–246. Springer-Verlag, 1995.

[CDGL97a] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Conjunctive query containment in Description Logics with n-ary relations. In *Proc. of the 1997 Description Logic Workshop (DL'97)*, pages 5–9, 1997.

[CDGL+97b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Source integration in data warehousing. Technical Report DWQ-UNIROMA-002, DWQ Consortium, October 1997.

[CDGL98a] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 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)*, pages 149–158, 1998.

[CDGL+98b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Schema and data integration methodology for dwq. Technical Report DWQ-UNIROMA-004, DWQ Consortium, September 1998.

[CDGL+98c] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Source integration in data warehousing. In *Proc. of the 9th Int. Workshop on Database and Expert Systems Applications (DEXA'98)*, pages 192–197. IEEE Computer Society Press, 1998.

[CDGL99a] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views in description logics. In *Proc. of the 1999 Description Logic Workshop (DL'99)*, pages 9–13. CEUR Electronic Workshop Proceedings `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-22/`, 1999.

[CDGL99b] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Modeling and querying semi-structured data. *Network and Information Systems*, 2(2), 1999.

[CDGL99c] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on XML documents: A description logic approach. *J. of Logic and Computation*, 9(3):295–318, 1999.

[CDGL+99d] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. A principled approach to data integration and reconciliation in data warehousing. In *Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'99)*. CEUR Electronic Workshop Proceedings `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-19/`, 1999.

[CDGLV99] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. of the 18th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'99)*, pages 194–204, 1999.

[CDGLV00] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, 2000. To appear.

[CL93] Tiziana Catarci and Maurizio Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[CW91] Stefano Ceri and Jennifer Widom. Deriving production rules for incremental view maintenance. In *Proc. of the 17th Int. Conf. on Very Large Data Bases (VLDB'91)*, pages 577–589, 1991.

[DGL94] Giuseppe De Giacomo and Maurizio Lenzerini. Description logics with inverse roles, functional restrictions, and n-ary relations. In *Proc. of the 4th European Workshop on Logics in Artificial Intelligence (JELIA'94)*, volume 838 of *Lecture Notes In Artificial Intelligence*, pages 332–346. Springer-Verlag, 1994.

[DGL95] Giuseppe De Giacomo and Maurizio Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pages 801–807, 1995.

[DL97] Oliver M. Duschka and Alon Y. Levy. Recursive plans for information gathering. In *Proc. of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, pages 778–784, 1997.

[DLNS96] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, Studies in Logic, Language and Information, pages 193–238. CSLI Publications, 1996.

[DLNS98] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: Integrating datalog and description logics. *J. of Intelligent Information Systems*, 10(3):227–252, 1998.

[Hul97] Richard Hull. 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.

[HZ96] Richard Hull and Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 481–492, 1996.

[Inm96] W. H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, second edition, 1996.

[JGJ+95] M. Jarke, R. Gallersdoerfer, M. Jeusfeld, M. Staudt, and S. Eherer. Conceptbase - A deductive object manager for meta databases. *J. of Intelligent Information Systems*, 4(2), 1995.

[KLSS95] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, pages 85–91, 1995.

[LR96] Alon Y. Levy and Marie-Christine Rousset. CARIN: A representation language combining Horn rules and description logics. In *Proc. of the 12th European Conf. on Artificial Intelligence (ECAI'96)*, pages 323–327, 1996.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI'96)*, pages 40–47, 1996.

[LSK95] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *J. of Intelligent Information Systems*, 5:121–143, 1995.

[PGMW95] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE'95)*, pages 251–260, 1995.

[SK92] Amit Sheth and Vipul Kashyap. 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.

[TLS99] Dimitri Theodoratos, Spyros Ligoudistianos, and Timos Sellis. Designing the global Data Warehouse with SPJ views. In *Proc. of the 11th Conf. on Advanced Information Systems Engineering (CAiSE'99)*, 1999.

[ZHK96] Gang Zhou, Richard Hull, and Roger King. Generating data integration mediators that use materializations. *J. of Intelligent Information Systems*, 6:199–221, 1996.

[ZHKF95] Gang Zhou, Richard Hull, Roger King, and Jean-Claude Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95)*, pages 4–18, 1995.

Appendix 1

For the sake of completeness, we include as an appendix the definition of the Concept Base metaclasses relative to the conceptual level of the Data Warehouse architecture for DWQ.

```
Class Model in ObjectType isA ConceptualObject end

Class EnterpriseModel in ObjectType isA Model end

Class SourceModel in ObjectType isA Model with
  attribute,single,necessary
    isModelOf : Source
end

Class ConceptRelationship in ObjectType isA ConceptualObject with
  attribute,single,necessary
    refersTo : Model
  attribute
    isSubsumedBy : ConceptRelationship;
    isNotSubsumedBy : ConceptRelationship
  constraint
    xorConceptRelationship :
     $ forall c/ConceptRelationship
       (c in Concept)  ==> not (c in Relationship) $;
    unionConceptRelationship :
     $ forall c/ConceptRelationship
       (c in AtomicConcept) or (c in ComplexConcept) or
       (c in AtomicRelationship) or (c in ComplexRelationship) $
end

Class AtomicConceptRelationship in ObjectType isA ConceptRelationship with
  attribute
    attrib : Attrib_name
  constraint
    exists_destination:
     $ forall c/AtomicConceptRelationship  at/Attrib_name
       (c attrib at) ==>
       (exists ca/AtomicConceptRelationship!attrib
               Ai(c,attrib,ca) and To(ca,at) ) $
end

Class Attrib_name in ObjectType isA ConceptualObject with
  attribute,single,necessary
    refersTo: Model
end


{ An atomic concepts or relationship is simply a name }

AtomicConceptRelationship!attrib in Class with
  attribute,necessary,single
    destination : ConceptDomain;
    functional : Boolean;
    mandatory : Boolean
end
```

```
{ A complex concept or relationship is an expression, which is simply stored
  as a string.  Parsing of the string is done by the client tool.
  The object has a link to the concepts which appear syntactically
  in the expression. }

Class ComplexConceptRelationship in ObjectType isA ConceptRelationship with
  attribute,single,necessary
    definition : String
  attribute
    containsSyntactically : ConceptualObject
  constraint
    atomicContainmentConstraint :
      $ forall c/ComplexConceptRelationship d/ConceptualObject
        (c containsSyntactically d) ==>
        ((d in AtomicConceptRelationship) or (d in AtomicDomain) or
         (d in Attrib_name)) $;
  sameModelConstraint :
    $ forall c1/ComplexConceptRelationship c2/AtomicConceptRelationship m/Model
      ((c1 refersTo m) and (c1 containsSyntactically c2)) ==> (c2 refersTo m) $
end

Class Concept in ObjectType isA ConceptRelationship,ConceptDomain with
  constraint
    subsumeConcept :
      $ forall c1/Concept c2/ConceptRelationship
        ((c1 isSubsumedBy c2) or (c1 isNotSubsumedBy c2)) ==> (c2 in Concept) $
end

Class AtomicConcept in ObjectType isA Concept,AtomicConceptRelationship end

Class ComplexConcept in ObjectType isA Concept,ComplexConceptRelationship end

Class ConceptDomain in ObjectType isA ConceptualObject with
  constraint
    xorDomainConceptRelationship :
      $ forall c/ConceptRelationship not (c in Domain) $;
    unionDomainConceptRelationship :
      $ forall cd/ConceptDomain (cd in ConceptRelationship) or (cd in Domain) $
end

Class Relationship in ObjectType isA ConceptRelationship with
  attribute,single,necessary
    arity : Integer
  attribute
    component : ConceptDomain
  constraint
    arity_constraint :
      $ forall r/Relationship i/Integer (r arity i) ==> (i >= 2 ) $;
    position_constraint :
      $forall r/Relationship c1,c2/Relationship!component a/Integer
              p1,p2/Integer
       (Ai(r,component,c1) and Ai(r,component,c2) and (c1 position p1) and
```

```
       (c2 position p2) and not(c1 == c2) and A(r,arity,a)) ==>
       (not (p1 == p2) and (p1 <= a) and (p2 <= a)) $;
    all_positions_constraint :
      $ forall r/Relationship i/Integer a/Integer
        A(r,arity,a) and (i<= a) ==>
        (exists c/Relationship!component
                Ai(r,component,c) and (c position i)) $;
    subsumeRelationship :
      $ forall r1/Relationship r2/ConceptRelationship
        ((r1 isSubsumedBy r2) or (r1 isNotSubsumedBy r2)) ==>
        (r2 in Relationship) $;
    subsumeArityRelationship :
      $ forall r1,r2/Relationship
        (exists i/Integer
                (((r1 isSubsumedBy r2) or (r1 isNotSubsumedBy r2)) and
                (r1 arity i) ==> (r2 arity i))) $
end

Class AtomicRelationship in ObjectType isA Relationship,
                                           AtomicConceptRelationship
end

Class ComplexRelationship in ObjectType isA Relationship,
                                            ComplexConceptRelationship with
constraint
  topConceptDomain:
    $ forall c/ComplexRelationship d/ConceptDomain
      (c component d) ==> ((d == TopConcept) or (d == TopDomain)) $
end

Relationship!component in Class with
  attribute,single,necessary
    position : Integer
end

Class Domain in ObjectType isA ConceptDomain with
  attribute
    isSubsumedBy : Domain;
    isNotSubsumedBy : Domain
  constraint
    xorDomain :
      $ forall ad/AtomicDomain  not (ad in ComplexDomain) $;
    unionDomain :
      $ forall d/Domain (d in AtomicDomain) or (d in ComplexDomain) $
end

Class AtomicDomain in ObjectType isA Domain end

Class ComplexDomain in ObjectType isA Domain with
  attribute,single,necessary
    domainDefinition : String
  attribute
    containsSyntactically : AtomicDomain
end
```

```
Class Assertion in ObjectType isA ConceptualObject with
  attribute,single,necessary
    equivalence: Boolean
  constraint
    unionAssertion :
     $ forall a/Assertion (a in DomainAssertion) or (a in ModelAssertion) $;
  conceptComponentAssertion :
    $ forall a/ModelAssertion l,r/Relationship i/Integer
      (exists lc,rc/Relationship!component cdl,cdr/ConceptDomain
              ((a left l) and (a right r) and From(lc, l) and To(lc, cdl) and
               From(rc, r) and To(rc, cdr) and (lc position i) and
               (rc position i) and (cdl in Concept) ==>
               (cdr in Concept))) $;

  domainComponentAssertion :
    $ forall a/ModelAssertion l,r/Relationship i/Integer
      (exists lc,rc/Relationship!component cdl,cdr/ConceptDomain
              ((a left l) and (a right r) and From(lc, l) and To(lc, cdl) and
               From(rc, r) and To(rc, cdr) and (lc position i) and
               (rc position i) and (cdl in Domain) ==>
               (cdr in Domain))) $
end

Class DomainAssertion in ObjectType isA Assertion with
  attribute,single,necessary
    left : Domain;
    right : Domain
end

Class ModelAssertion in ObjectType isA Assertion with
  attribute,single,necessary
    left : ConceptRelationship;
    right : ConceptRelationship
  constraint
    conceptAssertion:
      $ forall a/ModelAssertion l,r/ConceptRelationship
        (a left l) and (a right r) and (l in Concept) ==> (r in Concept) $;
    relationshipAssertion:
      $ forall a/ModelAssertion l,r/ConceptRelationship
        (a left l) and (a right r) and (l in Relationship) ==>
        (r in Relationship) $;
    relationshipArityAssertion:
      $ forall a/ModelAssertion l,r/Relationship (exists i/Integer
        ((a left l) and (a right r) and (l arity i) ==> (r arity i))) $;
    unionModelAssertion:
      $ forall a/ModelAssertion
        (a in InterModelAssertion) or (a in IntraModelAssertion) $
end

Class IntraModelAssertion in ObjectType isA ModelAssertion with
  constraint
    intraModel:
      $ forall a/IntraModelAssertion l,r/ConceptRelationship m/Model
```

```
        (a left l) and (a right r) and (l refersTo m) ==> (r refersTo m) $
end

Class InterModelAssertion in ObjectType isA ModelAssertion with
  attribute,single,necessary
    intensional: Boolean
  constraint
    interModel:
      $ forall a/InterModelAssertion l,r/ConceptRelationship m1,m2/Model
       (a left l) and (a right r) and (l refersTo m1) and (r refersTo m2) ==>
       not (m1 == m2) $
end

ModelEnt in EnterpriseModel end

TopConcept in AtomicConcept with
  refersTo r: ModelEnt
end

TopDomain in AtomicDomain end
```

Appendix 2

We include as appendix the publications [CDGL+99d, CDGL99b, CDGL99c, CDGLV99, CDGL99a, CDGLV00].