Document Number:	DWQ-UNIROMA-004
Project Name:	Foundations of Data Warehouse Quality (DWQ)
Project Number:	EP 22469
Title:	**Schema and Data Integration Methodology for DWQ**
Authors:	Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, Riccardo Rosati
Workpackage:	WP4
Document Type:	Deliverable (D.4.1)
Classification:	Confidential
Distribution:	DWQ Consortium
Status:	Final
Document file:	wp4-1.tex
Version:	1.3
Date:	October 5, 1998
Number of pages:	36

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	September 2, 1998	First draft
v.1.1	September 16, 1998	Formalism and resoning completed
v.1.2	September 22, 1998	Model construction and integration completed
v.1.3	September 30, 1998	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 meets 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 *WP3 - Source Integration*, and *WP4 - Data Riconciliation* of DWQ during the first 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] for the report on the first 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 the Deliverable 2.1, source and data integration 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

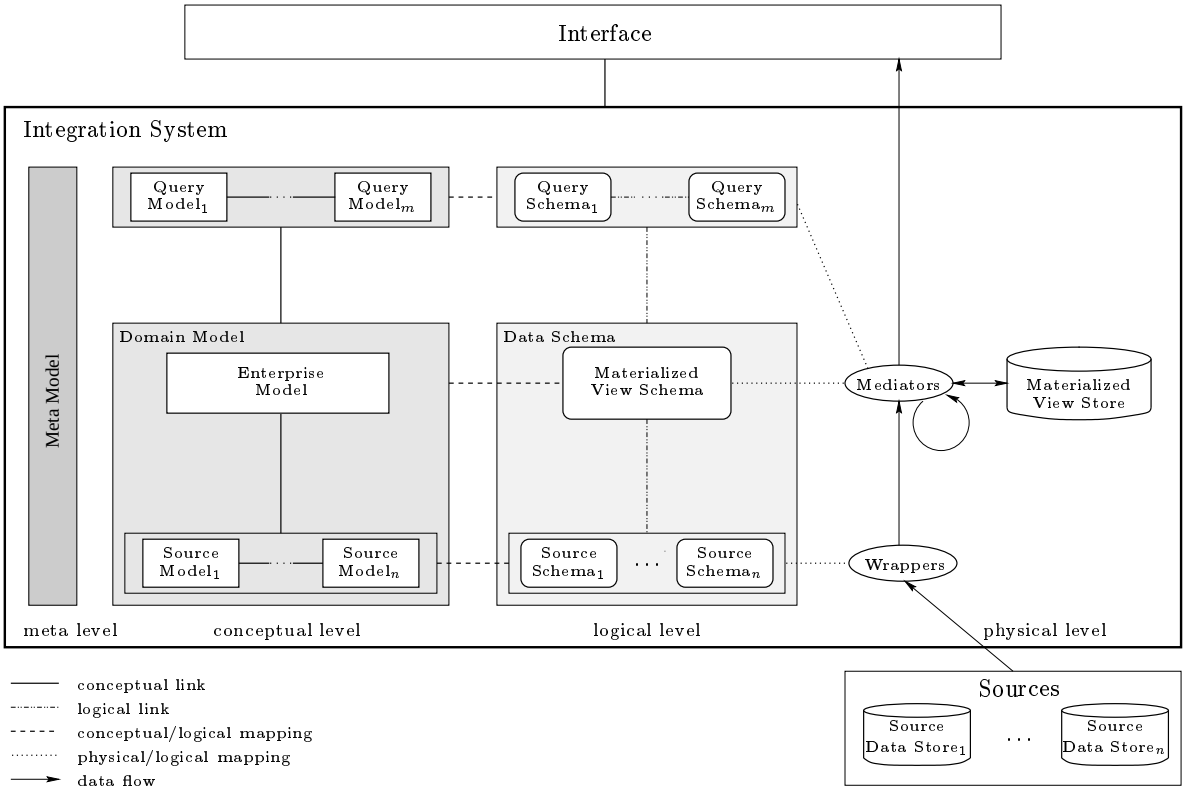

Figure 1: Architecture for Data Integration

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 will be called *source-driven*.

2. A new query is formulated by a client; the design scenario corresponding to this situation will be 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.

In this report we will delve into the details of source-driven integration. Client-driven integration is discussed in [CDGL+97b], and will be further investigated in the future activities within workpackage WP4. Source-driven integration is constituted by the following main phases:

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 enterpise is produced, if not avaliable.

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 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, the methodology aims at producing, for every relation in the Data Warehouse Schema, a specification on 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.

1.3 Organization of the Report

The remining 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 the logical level. Section 3 is devoted to a discussion on the reasoning services associated to the formalism. Sections 4, 5, 6, and 7 discuss the

main phases of the methodology for schema and data integration. Section 8 concludes the report, by pointing out the future work within WP4. Finally, the Appendices include relevant papers on the topic addressed in the report.

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 $\texttt{Person} \sqcap \forall \texttt{child.Male}$, 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 $\exists \texttt{child}$ 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 $\tau = \langle t_1, \ldots, t_n \rangle$):

$$
\begin{array}{rcl}
\top_{\tau}^{\mathcal{I}} & \subseteq & (\Delta^{\mathcal{I}})^n \\
\mathbf{P}^{\mathcal{I}} & \subseteq & \top_{\tau}^{\mathcal{I}} \\
(\$i^{o}{}_{\tau}\!: C)^{\mathcal{I}} & = & \{(d_1, \ldots, d_n) \in \top_{\tau}^{\mathcal{I}} \mid d_i \in C^{\mathcal{I}}\} \\
(\$i^{v}{}_{\tau}\!: V)^{\mathcal{I}} & = & \{(d_1, \ldots, d_n) \in \top_{\tau}^{\mathcal{I}} \mid d_i \in V^{\mathcal{I}}\} \\
(\neg \mathbf{R})^{\mathcal{I}} & = & \top_{\tau}^{\mathcal{I}} \setminus \mathbf{R}^{\mathcal{I}} \\
(\mathbf{R}_1 \sqcap \mathbf{R}_2)^{\mathcal{I}} & = & \mathbf{R}_1^{\mathcal{I}} \cap \mathbf{R}_2^{\mathcal{I}}
\end{array}
$$

$$
\begin{array}{rcl}
\varepsilon^{\mathcal{I}} & = & \{(x, x) \mid x \in \Delta^{\mathcal{I}}\} \\
(\mathbf{R}|_{\$i,\$j})^{\mathcal{I}} & = & \{(x_i, x_j) \mid (x_1, \ldots, x_n) \in \mathbf{R}^{\mathcal{I}}\} \\
(E_1 \circ E_2)^{\mathcal{I}} & = & E_1^{\mathcal{I}} \circ E_2^{\mathcal{I}} \\
(E_1 \sqcup E_2)^{\mathcal{I}} & = & E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}} \\
(E^*)^{\mathcal{I}} & = & (E^{\mathcal{I}})^*
\end{array}
$$

$$
\begin{array}{rcl}
\top_{\mathcal{C}}^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\
A^{\mathcal{I}} & \subseteq & \top_{\mathcal{C}}^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} & = & \top_{\mathcal{C}}^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(\exists E.C)^{\mathcal{I}} & = & \{d \in \top_{\mathcal{C}}^{\mathcal{I}} \mid \exists d' \in C^{\mathcal{I}}.(d, d') \in E^{\mathcal{I}}\} \\
(\exists [\$i]\mathbf{R})^{\mathcal{I}} & = & \{d \in \top_{\mathcal{C}}^{\mathcal{I}} \mid \exists (d_1, \ldots, d_n) \in \mathbf{R}^{\mathcal{I}}.d_i = d\} \\
(\leq k\ [\$i]\mathbf{R})^{\mathcal{I}} & = & \{d \in \top_{\mathcal{C}}^{\mathcal{I}} \mid |\{(d_1, \ldots, d_n) \in \mathbf{R}^{\mathcal{I}} \mid d_i = d\}| \leq k\}
\end{array}
$$

$$
\begin{array}{rcl}
\top_{\mathcal{D}}^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\
D^{\mathcal{I}} & \subseteq & \top_{\mathcal{D}}^{\mathcal{I}} \\
v^{\mathcal{I}} & \in & \top_{\mathcal{D}}^{\mathcal{I}} \\
\{v_1, \ldots, v_h\}^{\mathcal{I}} & = & \{v_1^{\mathcal{I}}, \ldots, v_h^{\mathcal{I}}\} \\
\neg V^{\mathcal{I}} & = & \top_{\mathcal{D}}^{\mathcal{I}} \setminus V^{\mathcal{I}} \\
(V_1 \sqcap V_2)^{\mathcal{I}} & = & V_1^{\mathcal{I}} \cap V_2^{\mathcal{I}}
\end{array}
$$

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. $\top_{\langle t_1,\ldots,t_n\rangle}^{\mathcal{I}} \subseteq \Delta_1 \times \cdots \times \Delta_n$, $\Delta_i = \begin{cases} \top_{\mathcal{C}}^{\mathcal{I}}, & \text{if } t_i = obj \\ \Delta_{\mathcal{D}}, & \text{if } t_i = val \end{cases}$ 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

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

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

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

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 $\mathcal{M}$.

2.1.4 Specialization of a $\mathcal{DLR}_{reg}$ Conceptual Model to the Entity-Relationship Model

We now show how $\mathcal{DLR}_{reg}$ can be specialized to the *Entity-Relationship* (ER) Model. This is done by showing how an ER diagram can be translated into a $\mathcal{DLR}_{reg}$ conceptual model capturing its semantics. The correctness of the translation follows from the work reported in [CLN94], which defines a similar translation to the Description Logic $\mathcal{ALCQI}$, a restriction of the logic $\mathcal{DLR}_{reg}$. Indeed, the conceptual model derived from an ER diagram has a special restricted form which allows for a simplification of the associated reasoning procedures.

The Entity-Relationship Model The basic elements of the ER Model are entities, relationships, and attributes. An *entity set* (or simply *entity*) denotes a set of objects, called its *instances*, that have common properties. Elementary properties are modeled through *attributes*, whose values belong to one of several predefined domains, such as `Integer`, `String`, or `Boolean`. Properties that are due to relations to other entities are modeled through the participation of the entity in relationships. A *relationship set* (or simply *relationship*) denotes a set of tuples (also called its instances), each of which represents an association among a different combination of instances of the entities that participate in the relationship. Since each entity can participate in a relationship more than once, the notion of *ER-role* is introduced, which represents such a participation and to which a unique name is assigned. The *arity* of a relationship is the number of its ER-roles. *Cardinality constraints* can be attached to an ER-role in order to restrict the number of times each instance of an entity is allowed to participate via that ER-role in instances of

the relationship. Such constraints can be used to specify both existence dependencies and functionality of relations [CK86]. They are often used only in a restricted form, where the minimum cardinality is either 0 or 1 and the maximum cardinality is either 1 or ∞. Additionally, so called *ISA* relations are used to represent inclusion assertions between entities and between relationships, and therefore the inheritance of properties from a more general entity or relationship to a more specific one.

The $\mathcal{DLR}_{reg}$ Conceptual Model Corresponding to an Entity-Relationship Diagram We now show how an ER diagram can be correctly represented by a $\mathcal{DLR}_{reg}$ conceptual model which fully captures its semantics. This means that all properties holding for the entities and relationships in the ER diagram are reflected in analogous properties of the corresponding concepts and relationships in the conceptual model. Indeed, we present a straightforward translation from ER diagrams to $\mathcal{DLR}_{reg}$ conceptual models which has the following property: every database coherent with the ER diagram corresponds to a model of the derived conceptual model and vice-versa.

The conceptual model $\mathcal{M}_\mathcal{S}$ coresponding to an ER diagram $\mathcal{S}$ is defined as follows: The set of atomic concept of $\mathcal{M}_\mathcal{S}$ consists of the set of entity symbols in $\mathcal{S}$. The set of atomic relationships of $\mathcal{M}_\mathcal{S}$ consists of the set of ER-relationships and attributes in $\mathcal{S}$. All relationships corresponding to attributes have the same signature $\tau_{att} = \langle obj, val \rangle$ in $\mathcal{M}_\mathcal{S}$, and all relationships corresponding to ER-relationships have signature $\tau_R = \langle obj, \ldots, obj \rangle$, where the arity is equal to the number of entities connected to the corresponding ER-relationship in the ER diagram. The set of domains of $\mathcal{M}_\mathcal{S}$ equals the set of attribute domains of $\mathcal{S}$.

The $\mathcal{DLR}_{reg}$ conceptual model $\mathcal{M}_\mathcal{S}$ consists of the following assertions:

- For each pair of entities E_1, E_2 such that E_1 *ISA* E_2 holds in $\mathcal{S}$, the assertion
$$E_1 \sqsubseteq E_2$$
- For each pair of relationships R_1, R_2 such that R_1 *ISA* R_2 holds in $\mathcal{S}$, the assertion
$$R_1 \sqsubseteq R_2$$
- For each attribute A with domain D of an entity E,
$$E \sqsubseteq \forall[\$1](A \sqcap (\$2_{\tau_{att}} : D)) \sqcap (= 1\,[\$1]A)$$
- For each relationship R of arity n between entities $E_1, \ldots, E_n$, the assertion
$$R \sqsubseteq (\$1 : E_1) \sqcap \cdots \sqcap (\$n : E_n)$$
- For each cardinality constraint of the form (m, n) on the participation of entity E in relationship R as component i,
  - if $m \neq 0$, the assertion
  $$E \sqsubseteq (\geq m\,[\$i]R)$$
  - if $n \neq \infty$, the assertion
  $$E \sqsubseteq (\leq n\,[\$i]R)$$

Based on the results presented in [CLN94], we are able to prove the correctness of the translation presented above. More specifically, let $\mathcal{S}$ be an ER diagram. Then, there is a one-to-one correspondence between legal database states of $\mathcal{S}$ and models of the $\mathcal{DLR}_{reg}$ conceptual model $\mathcal{M}_\mathcal{S}$. For example, an entity E can be populated in a legal database state for $\mathcal{S}$ if and only if $\mathcal{M}_\mathcal{S}$ admits a model in which E has a non-empty extension. This allows us to exploit the reasoning techniques developed for $\mathcal{DLR}_{reg}$ (see Section 3) in order to reason on ER diagrams.

We illustrate the translation presented above on an example, which we will extend in the following sections. We would like to model an enterprise and two sources containing information about contracts

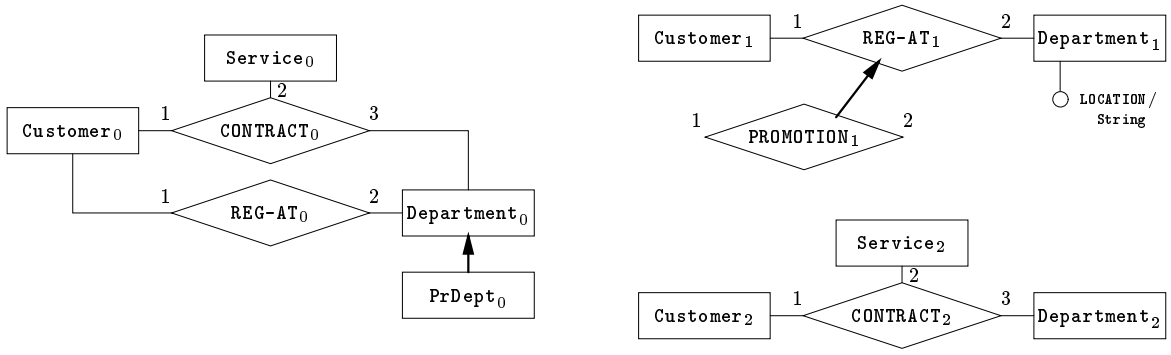

Figure 2: Enterprise and source models as Entity-Relationship diagrams

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:

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

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

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

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

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

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

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

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

Additions to the Entity-Relationship Model The ER model does not provide several features and modeling primitives which would prove useful in order to represent complex dependencies between data. The richness of constructs of $\mathcal{DLR}_{reg}$, and the correspondence between the two formalisms established in the previous section makes it however possible to add such constructs to the basic model and take them fully into account when reasoning on an ER diagram. We provide now several meaningful examples of possible additions to the basic ER model that arise as a natural consequence of the correspondence with $\mathcal{DLR}_{reg}$.

- *Arbitrary boolean constructs on entities.* The only direct relationship between entities that can be expressed in the basic ER Model is the ISA relation. A common extension is by so called *generalization hierarchies* (see e.g. [BCN92]), which allow one to express that the extension of an entity should be the disjoint union of the extensions of other entities. Such construct can easily be translated by making use of union and negation of $\mathcal{DLR}_{reg}$.
- *Refinement of properties along an ISA hierarchy.* Another important extension that should be considered is the possibility to specify more complex forms of refinement of properties of entities along ISA hierarchies, than the mere addition of attributes. This is already an essential feature of the more recent Object-Oriented Models. In particular, cardinality constraints could be refined by restricting the range of values, and the participation in relationships can be restricted. One may require for specific instances of an entity that the objects they are related to via a certain relationship belong to a more specific entity than the one directly associated to the ER-role. Such forms of constraints can be naturally expressed in $\mathcal{DLR}_{reg}$ by making use of universal quantification over relationships.
- *Definitions of classes by means of complex properties.* In the ER model (and more generally in Semantic Data Models) one can specify only necessary conditions that the instances of entities must satisfy. This means that in a database that conforms to an ER diagram one cannot deduce that a certain object is an instance of an entity unless this fact is explicitly stated. By using the different types of assertions of $\mathcal{DLR}_{reg}$, necessary and sufficient (and even just sufficient) conditions can be easily imposed and become part of the conceptual model.

2.1.5 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 alphabel 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

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

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

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

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

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

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:

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

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. $\Delta^{\mathcal{I}} = \top_{c_0}^{\mathcal{I}} \dot\cup \Delta_{\mathcal{D}}$;
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. $\top_{\langle t_1, \ldots, t_n \rangle_i}^{\mathcal{I}} \subseteq \Delta_1 \times \cdots \times \Delta_n$, $\Delta_j = \begin{cases} \top_{c_i}^{\mathcal{I}}, & \text{if } t_j = obj \\ \Delta_{\mathcal{D}}, & \text{if } t_j = val \end{cases}$ 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:

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

We illustrate the method by enriching the Enterprise and Source Models of the example of Section 2.1.4 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:

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

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 6. 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:

$$q(\vec{\mathbf{x}}) \leftarrow body_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee body_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)$$

where each $body_i(\vec{\mathbf{x}}, \vec{\mathbf{y}}_i, \vec{\mathbf{c}}_i)$ 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

$$\exists \vec{\mathbf{y}}_1 . body_1(\vec{\mathbf{x}}, \vec{\mathbf{y}}_1, \vec{\mathbf{c}}_1) \vee \cdots \vee \exists \vec{\mathbf{y}}_m . body_m(\vec{\mathbf{x}}, \vec{\mathbf{y}}_m, \vec{\mathbf{c}}_m)$$

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. We refer again to the ongoing example. Suppose that the form of (part of) the actual data in Source 1 can be described in terms of a relational table $\texttt{TABLE}_1$ 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:

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

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).

Referring to the ongoing example, we can prove for instance that the assertion

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

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:

$$\texttt{REG-AT}_1 \sqsubseteq (\$1{:}\texttt{Customer}_1) \sqcap (\$2{:}\texttt{Department}_1) \tag{2}$$

$$\texttt{PROMOTION}_1 \sqsubseteq \texttt{REG-AT}_1 \tag{3}$$

$$\texttt{Department}_1 \equiv_{ext} \texttt{PrDept}_0 \tag{4}$$

$$\texttt{REG-AT}_1 \sqsubseteq_{ext} \texttt{REG-AT}_0 \tag{5}$$

The intramodel assertions (2) and (3) imply that $\texttt{PROMOTION}_1 \sqsubseteq_{ext} (\$2{:}\texttt{Department}_1)$, 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.

Referring again to the ongoing example consider the following two queries posed to the Conceptual DW Model

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

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 Enterprise Model Construction

In this section we focus on the phase devoted to the construction of the conceptual model of a source or of the enterprise expressed in $\mathcal{DLR}_{reg}$. In principle, during this phase, we allow the designer to take advantage of the full expressive power of $\mathcal{DLR}_{reg}$, and to exploit the associated reasoning techniques. However, we provide a construction methodology and a set of guidelines which aim at specializing the conceptual model construction towards the Entity-Relationship model, thus allowing for an easy visualization of the data dependencies by exploiting design tools equipped with a graphical interface. For this reason, in the rest of the section we use the term *entity* to denote atomic concepts that appear in the conceptual model.

4.1 Guidelines for the Conceptual Model Construction

We present some guidelines for the construction of the conceptual model of a source or of the enterprise which aim at rationalizing its design by

- Imposing certain restrictions on the form of the assertions in the conceptual model. This allows for extracting from the conceptual model an ER diagram by means of which the most important semantic relationships introduced by the designer can be visualized.
- Leading to the introduction of several natural defaults in the construction of the conceptual model. This allows the designer to concentrate on the significant properties leaving the specification of the more routinely ones to the system.

In particular, the guidelines impose the following restrictions which should be obeyed by the designer when constructing the conceptual model.

- Each name of an entity and of a relationship in the conceptual model of a source is *prefixed* by the name of the source. Similarly for the enterprise model. For example, the entity `Department` of a source `Source` is in fact named by `Source_Department`. In this way the conceptual model to which an entity or a relationship belongs is immediately apparent.

  Observe that this convention for names is not enforced on domains which are fixed once and for all for the conceptual models of all the sources and of the enterprise.

  When we come to specify relations between different conceptual models, this naming convention allows us to assume by default that two entities/relationships with the *same name* (apart from the prefix) belonging to two different conceptual models are *intensionally equivalent* (see Section 5).
- Each conceptual model contains a certain number of most general entities and relationship (of given arity) which are assumed to be disjoint from each other. Every time a new entity/relationship is introduced, it must be declared (explicitly or implicitly via an ISA chain) as a sub-entity/sub-relationship of one of the most general entities/relationships. Observe that in this way the entities/relationships are partitioned into disjoint collections, since every entity/relationship is subsumed by one of the most general ones, and these are all disjoint from each other. In other words, several disjointness constraints can be inferred without having to specify them explicitly.
- Every relationship has an entity or a domain associated to each of its components, called the *primary entity/domain* for that component. Such an entity/domain establishes the type of the component to which it is associated.
- In addition, any form of intramodel assertion is allowed, including definitions of entities and relationships. Such assertions express requirements that cannot be captured by an ER diagram.

The methodology presented above allows for extracting from a conceptual model an ER diagram, by dropping all intermodel assertions that do not correspond to typing of relationships, ISA between entities or relationships, or generalization hierarchies. We call such an ER diagram, the *outline* of the conceptual model.

In a well designed conceptual model, the outline points out the general structure of the interrelations among entities and relationships, which allows for a graphical visualization of the semantics of the domain of interest. Obviously, the designer should aim at exposing such structure as much as possible, avoiding to hide it by using arbitrary complex assertions not representable in the outline.

4.2 Quality Analysis

Once the conceptual representation of the source or enterprise under analysis in provided in terms of a conceptual model, the inference techniques associated to $\mathcal{DLR}_{reg}$ are used to determine the values of the quality factors associated to the conceptual design, and to adjust the design accordingly. In particular, the following formal properties are used in assessing the quality factors associated to the conceptual model:

- *Consistency of the conceptual model.* A conceptual model is *consistent*, if it admits a model, i.e. an interpretation that satisfies all (intramodel) assertions that constitute the conceptual model. In other words, the database of which the conceptual model provides a conceptual view, can in fact be populated without violating any of the conditions specified at the conceptual level. It is important to be able to discover inconsistencies in a conceptual model directly during the design phase, since handling them during the data base population stage can be rather inconvenient.
- *Consistency of entities and relationships.* An entity/relationship is *consistent*, if the conceptual model admits a model in which the entity/relationship has a nonempty extension. The inconsistency of an entity/relationship may be due to a design error or due to over-constraining. In any case, the designer needs to remove the cause of inconsistency, either by correcting the error, or by relaxing some constraints, or by deleting the entity/relationship, thus removing redundancy from the schema. Observe also that while the consistency of the conceptual model follows from consistency of all its entities and relationships (in fact from consistency of at least one entity or relationship), the converse is in general not true.
- *Equivalence of entities and relationships.* Two entities/relationships are *equivalent* if they denote the same set of instances in all models of the conceptual model. Determining equivalence of two entities/relationships allows for their merging, thus reducing the complexity of the conceptual model and eliminating redundancy.
- *Subsumption of entities and relationships.* An entity/relationship L_1 is *subsumed by* an entity/relationship L_2 if in all models of the conceptual model the extension of L_1 is a subset of the extension of L_2. Subsumption allows one to deduce properties for one entity/relationship from those of another one. It is also the basis for a *classification* of all the entities/relationships that appear in a schema within a lattice, which can be usefully exploited in several ways within the modeling process [BN94]. Observe also that subsumption between two entities/relationships may have consequences on other entities/relationships in the schema (e.g. inconsistency).

4.3 Method for the Conceptual Model Construction

The above observations suggest the following methodology for the construction of the conceptual model of a source or of the enterprise, which exploits the analysis of the quality factors:

1. Construct the ER diagram which constitutes the outline of the conceptual model. Such outline is always consistent, since it does not involve any complex intramodel assertions.
2. For each entity/relationship in turn, add, if necessary, an equivalence assertion defining the entity/relationship in terms of the other entities/relationships in the conceptual model, and one or more inclusion assertions relating the entity/relationship to the other entities/relationships in the conceptual model.
3. After the insertion of each assertion, check the consistency of the resulting conceptual model. In case the model is inconsistent, determine the assertions (one or more) that cause the inconsistency and remove them.

4. After the insertion of each assertion, check the consistency of all entities and relationships. If an entity/relationship is inconsistent, either remove the cause of inconsistency, or remove the entity/relationship from the conceptual model. If an entity is removed, remove also all relationships that have the entity or a subentity of it as the primary entity.

5. After the insertion of each assertion, check for every pair of entities/relationships whether they are equivalent, and when such a pair is found, remove one of the entities/relationships and replace every occurrence of it in the conceptual model by the other one.

6. After the insertion of each assertion, check for every pair of entities/relationships whether one is subsumed by the other, and when such a pair is found, add the ISA link to the outline of the conceptual model, thus making the subsumption explicit.

We observe that the proposed methodology can be effectively supported by a design tool equipped with reasoning procedures over a $\mathcal{DLR}_{reg}$ conceptual model, and which provides a graphical interface used to display the outline of the conceptual model.

4.4 Link to the Repository

During the whole design phase the metadata repository [JGJ+95] is exploited for the following three main tasks:

1. to store and manipulate the representation of the conceptual model, which is inserted directly by the designer;

2. to insert and store the values of the quality factors associated to the elements of the conceptual model;

3. to query the metadata in order to get various information; the process of answering such queries may trigger the reasoning facilities of the system, for example to deduce information about the conceptual model.

4.4.1 Storing the Conceptual Model

The metadata in the repository comprises all information about entities and relationships and the intramodel assertions stating their properties. Such metadata is inserted in the repository during the design process and maintained during the complete life-cycle of the Data Warehouse. The metadata repository [JGJ+95] has been enriched with suitable classes in order to facilitate the process of inputing all relevant aspects of the conceptual models.

4.4.2 Storing Quality Factors

The organization of the data in the repository, as presented in [JJQV98] allows for associating the values of the quality factors both to the single entities/relationships and to the whole conceptual model. In particular, the factors of interest for the conceptual modeling of a Source or of the Enterprise are the following (see [JJQV98]):

- *Quality factors for entities/relationships - description level:*
  - Consistency: coherence of the description
  - Redundancy: equivalence with other entities/relationships
  - Completeness: level of completeness of the description
  - Interpretability: existence and readability of explicit description
  - Accuracy: level of granularity and preciseness of the description
  - Confidence: degree of believability of the process/designer producing the description of the entity/relationship

- *Quality factors for entities/relationships - data instance level:*
  - Consistency: nonemptiness of the extension
  - Redundancy: equivalence with other instances
  - Completeness: percentage of instances represented in the data of the source/enterprise
  - Interpretability: understandability of the format the instances
  - Accuracy: level of preciseness of the representation of the properties of the instances
  - Confidence: degree of believability of the process producing the instances and establishing their properties
  - Freshness: degree of timeliness and updatedness of instances
  - Availability: degree of accessibility of the data in the source/enterprise for retrieving instances and their properties
- *Quality factors for the Conceptual Model:*
  - Consistency: coherence of the model
  - Redundancy: percentage of redundant entities, relationships, and intramodel assertions
  - Completeness: level of completeness with respect to the data in the source
  - Interpretability: level of readability of the outline of the model and readability of the additional intramodel assertions, existence of design history
  - Accuracy: level of detail of the intramodel assertions describing entities and relationships
  - Confidence: degree of believability of the process/designer producing the model

4.4.3 Querying the Metadata

The system allows general queries to be issued on the metadata. The most interesting kinds of queries are those requiring the activation of the reasoning procedures over the conceptual model, and storing the results of the reasoning process. A notable example of such an activation is the one occurring when the designer asks whether one entity is a subset of another entity. In this case, the subsumption algorithm of the reasoning tool is triggered, and the result is stored in the repository. On the other hand, when a conceptual model is modified, the system retracts all the deduced information that are no longer valid. In particular, the links `isSubsumedBy` and `isNotSubsumedBy` of the metedata model are maintained as follows.

- Whenever a new subsumption relation between C_i and C_j is computed, the pair C_1, C_2 is stored in `isSubsumedBy`.
- Whenever a subsumption check between C_i and C_j fails, the pair C_1, C_2 is stored in `isNotSubsumedBy`.
- Whenever a new assertion is stored in the Conceptual Data Warehouse Model, all links of type `isNotSubsumedBy` that can be affected by the new subsumption relation are canceled.
- Whenever an assertion is removed from the Conceptual Data Warehouse Model, all links of type `isSubsumedBy` that were dependent from the removed assertion are canceled.

Note that the ability to interface directly with the reasoner is exploited in the design phase of the conceptual model to assess and improve the values of the quality factors, as described in Section 4.3.

5 Source Model Integration

In this section we focus on the Source Model Integration phase, i.e. the phase where the Source Model is integrated into the Conceptual Data Warehouse Model. We remind the reader that our approach to integration is mainly declarative: the most important efforts during source integration are indeed devoted to single out and to specify the intermodel assertions relating the Enterprise Model and the Source Models, rather than producing a unified conceptual representation.

5.1 Method

The methodology for source integration is based on the specification of intermodel assertions. As we said in Section 2.1.5, there are two types of intermodel assertions, namely, assertions relating concepts in one Source Model with concepts in the Enterprise Model, and assertions relating concepts in different Source Models. Although the vast majority of assertions will be of the first type, we stress the importance of specifying inter-relationships between sources, when available. For example, specifying that the set of instances of concept C_i of source S_i is always a subset of those of C_j in source S_j can be important in order to infer that accessing source S_j for retrieving instances of C is useless. We point out that the possibility of expressing relationships between concepts in different sources is a distinguished feature of our approach.

The first observation is that the methodology requires that the designer specify the relationships between the most general entities and relationships in a source model and the most general entities and relationships in the enterprise model. In other words, for every pair of "top" entities in a source S and in the enterprise model, the designer is required to specify one intermodel assertion. Tipically, such an intermodel assertion will specify either that the entity in the source is a subset of the entity in the enterprise model, or that the two entities are disjoint, or that they are overlapping. Note that from the disjointness assertion the system is able to derive one disjointness assertion for every pairs of entities that are subset of the two most general entities. The same considerations apply to relationships.

The second observation regarding the methodology is that the naming convention discussed in Section 4 allows us to assume by default that two entities/relationships with the *same name* (apart from the prefix) belonging to two different conceptual models are *intensionally equivalent*. Obviously, the designer may override this default by means of synonym assertions (see later).

Further intermodel assertions that the designer can specify can be classified as follows:

- *Subsetting assertions*, that are used to state that a certain concept or relation in the Source Model is a subset of another concept or relation in the Enterprise Model (or in another source). These assertion have the form $L_i \sqsubseteq_{ext} L'_j$.
- *Definition assertions*, that are used to completely characterize the set of instances of one concept in a model in terms of the set of instances of a concept in another model.
- *Completeness assertions*, that are used to state that the set of instances of a concept or relation in the Enterprise Model can be obtained as the union of different concepts or relations in the various sources. A special case of this type of assertions is the one stating that a certain concept in the Enterprise Model is fully captured by a concept in one Source Model. This assertion formalizes the notion of "proprietary entity" used e.g. in the method adopted in Telecom Italia for source integration in their Data Warehouse project IBDA.
- *Synonym assertions*, that are used to state that different symbols in two models denote in fact the same concept. These assertions have the form $L_i \equiv_{int} L_j$.
- *Homonym assertions*, that are used to state that the same symbol is used to denote different concepts in different models. These assertions have the form $L_i \not\equiv_{int} L_j$.

It is important to observe that the possibility of using complex concept and relation expressions in the context of intermodel assertions greatly enhances the expressive power of such assertions, and is another distinguished feature of our approach.

The specification of intermodel assertion implicitely requires to carry out a conflict resolution step. Indeed, the most important type of conflicts that may arise during Source integration is the so-called structural conflict. In our methodology, a structural conflict arises when the designer tries to enforce an intermodel assertion between an entity and a relationship. In other words, the same concept is represented as an entity in one model (either the source model or the enterprise model) and a relationship in the other model.

There are three main strategies for solving this type of conflicts:

1. The conflict is solved by preferring the design choice made in the enterprise model. This means that suitable transformations are applied to the source model in order to conform to the enterprise

model. Such transformations are described, for example, in [BCN92]. Observe that this strategy keeps the enterprise model, as well as the orher source models unchanged.

2. The conflict is solved by transforming the relationship into an entity. So, in this case, either the source or the enterprise model is modified. Again, the transformation is the one reported in in [BCN92]. It is important to note that, if the modification regards the enterprise model, it may trigger new structural conflicts with different sources. Such conflicts are in turn solved by means of the same kind of transformation applied to the sources.

3. The conflict is solved by selecting the design choice of the model that is more reliable with respect to the concept represented by the entity and by the relationship. This strategy heavily relies on the quality factors associated with the sources and to the enterprise.

5.2 Quality Analysis and Link to the Repository

The quality analysis and the link to the repository are performed in a way similar to the one discussed in Section 4. Here we detail only the additional aspects to be considered when dealing with the Conceptual DW Model.

In addition to the ones presented in Section 4.2, the following formal property is relevant when assessing the quality factors:

- *Consistency of the Conceptual DW Model.* The *Conceptual DW Model* is *consistent*, if it admits a model, i.e. an interpretation that satisfies all intramodel assertions in the constituent Source Models and in the Enterprise Model and all intermodel assertions. Inconsistencies can arise due to the integration of a new source or a new portion of an already partially integrated source. Discovering such inconsistencies during the integration phase can lead to restructuring parts of the Conceptual DW Model.

Due to the need of combining several conceptual models in the Conceptual DW Model, in addition to the quality factors presented in Section 4.4, the following ones need to be considered (see [JJQV98]):

- *Quality factors for the Conceptual DW Model:*
  - Consistency: coherence of the Conceptual DW Model
  - Redundancy: percentage of redundant intermodel assertions
  - Completeness: level of completeness of the Enterprise Model with respect to the data in the sources
  - Interpretability: level of readability of the intermodel assertions, existence of design history
  - Accuracy: level of detail of the intermodel assertions describing the interrelations between entities and relationships
  - Confidence: degree of believability of the Source Models and the Enterprise Model and of the process/integrator producing the intermodel assertions

6 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 only analyze the case where schemas are expressed in terms of the relational model. The case of sources organized in terms of a semi-structured data model will be the subject of future work (see Section 8). The section is organized as follows. We first present the notion of adorned query, which is 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.

6.1 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

$$q \mid c_1, \ldots, c_n$$

where

- q is a query as defined in Section 2.2,
- each c_i is a *constraint* on a variable appearing in q, i.e. an expression of the form:

  $$X :: c$$

  which is used to specify how the values or the objects bound to X are represented at the logical level in the table associated with the query,
- there is exactly one constraint for each variable in the head of q.

Taking into account the form of the Conceptual DW Model, there are two possibilities for a variable appearing in a constraint:

1. The variable is bound to values. In this case the constraint has the form

   $$X :: V$$

   where V is a domain expression, and the constraint is used to specify how values bound to X are represented in the relation at the logical level (for example, which scale is used for an integer value denoting a temperature).

2. The variable is bound to objects. In this case the constraint has the form

   $$X :: rec(Y_1, \ldots, Y_m), R_1, \ldots, R_h, F_1, \ldots, F_k \tag{6}$$

   where $Y_1, \ldots, Y_m$ are variables not appearing anywhere else in the query, $R_1, \ldots, R_h$ are atoms on the variables $X, Y_1, \ldots, Y_m$ asserting how $Y_1, \ldots, Y_m$ are related to X, and $F_1, \ldots, F_k$ are atomos on the variables $Y_1, \ldots, Y_m$, asserting how they are bound to values. This kind of constraint is used to specify which are the values that are used in the relation at the logical level for denoting each abstract object. In particular, constraint (6) states that objects bound to X are represented by m values (for example, a student may be represented by its name and its date of birth), and these values are represented by the variables $Y_1, \ldots, Y_m$.

Referring to our ongoing example, we assume that the Source Models contain additional attributes, such as $\texttt{SSN}_1$, $\texttt{Name}_1$, $\texttt{NameC}_2$, $\texttt{ServNo}_2$, and $\texttt{Name}_2$ with the intuitive meaning.

Assuming that in Source 1 a a customer is actually identified by its social security number, and a department by its name, we can adorn the query defined in Section 2.2, specifying the relational table $\texttt{TABLE}_1$ as follows:

$$\begin{aligned}\texttt{TABLE}_1(X, D, P) \leftarrow\ & \texttt{REG-AT}_1(X, D) \wedge \neg\texttt{PROMOTION}_1(X, D) \wedge P = \texttt{false} \vee \\ & \texttt{PROMOTION}_1(X, D) \wedge P = \texttt{true} \mid \\ & X :: rec(S),\ \texttt{SSN}_1(X, S),\ \texttt{String}(S), \\ & D :: rec(N),\ \texttt{Name}_1(D, N),\ \texttt{String}(N), \\ & P :: \texttt{Boolean}\end{aligned}$$

Additionally, we assume that in Source 2 the actual data can be described in terms of a relational table $\texttt{TABLE}_2$ with three columns, one 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, also customers are identified by their name. Services are identified by a unique service number. Hence the following adorned query is used to specify $\texttt{TABLE}_2$:

$$\begin{aligned}\texttt{TABLE}_2(X,S,D) \leftarrow\ & \texttt{CONTRACT}_2(X,S,D) \mid \\ & X :: rec(N),\ \texttt{NameC}_2(X,N),\ \texttt{String}(N), \\ & S :: rec(I),\ \texttt{ServNo}_2(S,I),\ \texttt{Integer}(I), \\ & D :: rec(M),\ \texttt{Name}_2(D,M),\ \texttt{String}(M)\end{aligned}$$

Note that an adorned query may be *inconsistent* (for instance when the constraints on one variable are inconsistent), even if the original query is consistent.

Given an adorned query, it is important to derive from it what we call the *canonical adorned query*, i.e. the adorned query which is conceptually equivalent to the original one, and in which all variables *appearing in the head* are bound to values. The method for deriving the canonical adorned query starting from the schema and the adorned query q is as follows.

1. For each variable X bound to objects in the head of q, let

   $$X :: rec(Y_1,\ldots,Y_m), R_1,\ldots,R_h, F_1,\ldots,F_k$$

   be the constraint associated with X in q. Then the variable X in the head of q is replaced by $Y_1,\ldots,Y_m$, and $R_1,\ldots,R_h,F_1,\ldots,F_k$ are added to the body of the query.

2. For a variable X bound to values in the head of q, let V_X be the *most specific domain for* X, i.e. the expression in disjunctive normal form such that M', which is the Conceptual DW Model M in which all domains have been substituted with $\top_{\mathcal{D}}$, implies that X is an instance of V_X, and there is no other V' such that V' is a subset of V_X in M, and X is an instance of V'. Then for each variable X bound to values in the head of q, replace the constraint on X with $X :: V_X$.

As an example, the canonical adorned query corresponding to the above adorned query specifying $\texttt{TABLE}_1$ is the following:

$$\begin{aligned}\texttt{TABLE}_1(S,N,P) \leftarrow\ & \texttt{SSN}_1(X,S) \wedge \texttt{Name}_1(D,N) \wedge \texttt{REG-AT}_1(X,D) \wedge \neg\texttt{PROMOTION}_1(X,D) \wedge P = \texttt{false} \vee \\ & \texttt{SSN}_1(X,S) \wedge \texttt{Name}_1(D,N) \wedge \texttt{PROMOTION}_1(X,D) \wedge P = \texttt{true} \mid \\ & S :: \texttt{String},\ N :: \texttt{String},\ P :: \texttt{Boolean}\end{aligned}$$

A canonical adorned query $q \mid c_1,\ldots,c_n$ is *consistent* with respect to M if q is consistent with respect to M, and every constraint c_i is consistent.

We observe that, given a relation T at the logical level, the associated adorned query and the corresponding canonical adorned query play different roles. The former specifies conceptually the content of the relation, the latter specifies the form of the relation as actually found at the logical level. The fact that the two notions are different is testified by the fact that it may be the case that the arity of the two queries differ.

6.2 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, in this report 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. This aspect will be covered in the future activities within the workpackage.

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 constraints on the variables accordingly, thus deriving the adorned query associated with T. Since such an adorned query represents a relation in a source, it must conform to the restriction that each constraint is on a variable appearing in q.
3. Obtain the canonical adorned query associated with T, following the method described in Section 6.1.
4. Check the quality of the adorned query, following the suggestions specified in Section 6.4. In particular, if the adorned query is inconsistent, then change the original adorned query so as to eliminate the inconsistency, and go back to the previous step.
5. If the resulting canonical 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 adorned query, which specifies conceptually the content of the relation in the source,
   - the associated canonical 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.

Suppose that the Conceptual DW Model contains a relationship `Service` with three attributes, `Date`, `ServiceNo`, and `Price`, where $\texttt{Service}(D, S, P)$ means that at the date D the service S costs P Lire. Suppose that Source S_1 represents the same kind of information only on Services v_1 and v_2, by means of two relations: `v1` and `v2`, where $\texttt{v1}(D, P)$ means that service v_1 costs P thousand Lire at date D, and $\texttt{v2}(D, P)$ means that service v_2 costs P thousand Lire at date D. Finally, suppose that Source S_2 represents the same kind of information only on Services v_3 and v_4 by means of a relation `Serv`, where $\texttt{Serv}(X, Y, D)$ means that services v_3 and v_4 cost X and Y hundred Lire respectively at date D. The various relations in the two sources can be specified by means of the following adorned queries:

$$
\begin{array}{rcl}
\texttt{v1}(D, P) & \leftarrow & \texttt{Service}(D, '\texttt{v1}', P) \mid P :: \texttt{Integer} \sqcap \texttt{ThousandLire},\ D :: \texttt{Date} \\
\texttt{v2}(D, P) & \leftarrow & \texttt{Service}(D, '\texttt{v2}', P) \mid P :: \texttt{Integer} \sqcap \texttt{ThousandLire},\ D :: \texttt{Date} \\
\texttt{Serv}(X, Y, D) & \leftarrow & \texttt{Service}(D, '\texttt{v3}', X) \wedge \texttt{Service}(D, '\texttt{v4}', Y) \mid \\
 & & X :: \texttt{Integer} \sqcap \texttt{HundredLire},\ Y :: \texttt{Integer} \sqcap \texttt{HundredLire},\ D :: \texttt{Date}
\end{array}
$$

The above example illustrates a case where there are various schematic differences, both between source S_1 and S_2, and between the two sources and the Conceptual DW Model. We observe that the mechanism used in our methodology for specifying adorned queries is able to cope with such differences.

6.3 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. The problems related to the client-driven methodology will be further detailed and addressed in the future activities of the workpackage.

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 constraints on the variables accordingly, thus deriving the adorned query associated with T. Since such an adorned query represents a relation in the Data Warehouse, it must conform to the restriction that each constraint is on a variable appearing in q.
3. Obtain the canonical adorned query associated with T, following the method described in the previous section.
4. Check the quality of the adorned query, following the suggestion specified in Section 6.4. In particular, if the adorned query is inconsistent, then change the original adorned query so as to eliminate the inconsistency, and go back to the previous step.
5. If the resulting canonical 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 adorned query, which specifies conceptually the content of the relation in the Data Warehouse,
   - the associated canonical adorned query, which specifies the actual data stored in the relation of the Data Warehouse.

Referring to our ongoing example, 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:

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

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:

$$\begin{aligned}\texttt{TDW}(X,S,D,P) \leftarrow\ & \texttt{CONTRACT}_0(X,S,D) \wedge \texttt{PROMOTION}_1(X,D) \wedge F = \texttt{true} \vee \\ & \texttt{CONTRACT}_0(X,S,D) \wedge \texttt{REG-AT}_1(X,D) \wedge \neg\texttt{PROMOTION}_1(X,D) \wedge P = \texttt{false} \vee \\ & \texttt{PROMOTION}_1(X,D) \wedge P = \texttt{true} \wedge S = \texttt{NULL} \ \mid \\ & X :: \mathit{rec}(N),\ \texttt{SSN}(X,N),\ \texttt{String}(N), \\ & S :: \mathit{rec}(I),\ \texttt{ServNo}(S,I),\ \texttt{Integer}(I), \\ & D :: \mathit{rec}(C),\ \texttt{Code}(D,C),\ \texttt{Integer}(C), \\ & P :: \texttt{Boolean}\end{aligned}$$

6.4 Quality Analysis

Analogously to the case of Source Model Contruction and Integration, 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. With respect to the first two tasks, we refer to future work within WP4 (see Section 8).

The need of checking the quality of the logical specification was mentioned several times in this section. The most important checks are related to need of verifying whether the specification of the various relation schemas (both Source and Data Warehouse schemas) is consistent. For this purpose, we use the notion of *consistency* of an adorned query. As we said before, an adorned query

$$q \mid c_1, \ldots, c_n$$

is *consistent* with respect to M if q is consistent with respect to M, and every constraint c_i is consistent. Therefore, 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 schema 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.

7 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, it is crucial that, for every relation in the Data Warehouse Schema, a specification is provided on how the tuples of such a relation should be constructed from a suitable set of tuples extracted from the relations stored in the sources. Indeed, the information about the link between the Data Warehouse Schema and the Source Schemas is useful 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.

7.1 Linking Relational Schemas

We describe here the methodology we use for linking the relation schemas in the Data Warehouse with the relation schemas in the sources. Let T be a relation in the Data Warehouse Schema, let $q \mid c_1, \ldots, c_n$ the adorned query associated to T, and let $q' \mid c'_1, \ldots, c'_m$ be the canonical adorned query associated to T. The methodology proceeds through the following steps.

1. Look for a rewriting r of q by means of a suitable set of adorned queries $q_1, \ldots, q_s$ associated to the relations in the sources. If there are different rewritings, then choose the best rewriting r with respect to suitable quality parameters.
2. The rewriting r is a specification of how the tuples of T can be conceptually obtained from the relations in the sources, ignoring for the moment their constraint parts. Note that the constraints $c_1, \ldots, c_n$ are maintained, so that the rewriting has actually the form

   $$r \mid c_1, \ldots, c_n.$$

   However, in order to deal with data heterogeinity, several transformations on the query need to be undertaken. To address this issue, during this step, the variables in r are renamed so that atoms deriving from different sources have different variables, and suitable new predicates are introduced representing matching and conversion functions. More precisely, for each disjunct d appearing in r, we do the following

   - For each variable X not appearing in the head of r, and appearing in atoms $T_1, \ldots, T_h$ belonging to sources $S_1, \ldots, S_h$, respectively, introduce new variables $X_1, \ldots, X_h$, and add to d the atom

     $$match(X_1, \ldots, X_h).$$

     Intuitively, such an atom represents the conditions under which different values or objects in different sources match, taking into account that, at the conceptual level, such a matching condition is simply an equality.
   - For each variable X appearing in the head of r, and appearing in atoms $R_1, \ldots, R_h$ belonging to sources $S_1, \ldots, S_h$, respectively, introduce new variables $X_1, \ldots, X_h$, and add to d the atom

     $$convert(X_0, X_1, \ldots, X_h).$$

     Intuitively, such an atom represents the conditions under which different values or objects in different sources match and are converted into a value or an object in the Data Warehouse, taking into account that, at the conceptual level, such a matching and conversion condition is simply an equality.

   Finally, we add to the constraint part associated to r the information on how the relevant values are represented in the sources, i.e. suitable constraints deriving from the contraint parts of $q_1, \ldots, q_s$.

3. If the query obtained from the previous step is an *or*-query (i.e. it is constituted by more than one disjunct), then associate to the query a suitable set of so-called *merging clauses*, taking into account that the answers to the different *or*-parts of the query may contain objects and values that represent the same real world entity or the same value. A merging clause is an expression of the form:

   merging *tuple-spec* and ... and *tuple-spec*
   such that *matching-cond*
   into *tuple-spec* and ... and *tuple-spec*

   We observe that the system 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 changing the templates in order to specify a different merging plan (for example for merging three disjuncts, rather than a pair).

   The resulting query is called the *data acquisition plan* for T. Indeed, the resulting query is a conceptual specification of how we can construct the tuples of T starting from a suitable set of tuples extracted from the sources.

4. Consider the query obtained in the previous step, with the associated merging clauses, and apply the same transformation used for producing a canonical adorned query from an adorned query:

   (a) For each variable X bound to objects, let

   $$X :: rec(Y_1, \ldots, Y_m), R_1, \ldots, R_h, F_1, \ldots, F_k$$

   be the constraint associated with X. Then the variable X is replaced by $Y_1, \ldots, Y_m$, and $R_1, \ldots, R_h, F_1, \ldots, F_k$ are added to the body of the query.

   (b) For each variable X bound to values, add $X :: V_X$ to the constraints, where V_X is the most specific domain for X.

   The canonical adorned query so obtained is then simplified as follows. For every *match* atom whose arguments $X_1, \ldots, X_n$ are subject to exactly the same constraints, eliminate the atom, and replace all variables $X_1, \ldots, X_n$ with a new variable. Analogously, for every *convert* atom whose arguments $X_0, X_1, \ldots, X_n$ are subject to exactly the same constraints, eliminate the atom, and replace all variables $X_0, X_1, \ldots, X_n$ with a new variable. Intutively, the simplification reflects the fact there is no clue that we need complex matching and conversion functions for values belonging to exactly the same domains.

5. Finally, obtain the so-called *complete data acquisition plan* for T, i.e. the specification of:

   - the name of the relation,
   - the number of columns,
   - the type of each column,
   - whether each column admits null values or not,
   - the associated adorned query, which specifies conceptually the content of the relation in the Data Warehouse, in terms of the relations in the sources,
   - the associated canonical adorned query, which specifies the actual data stored in the relation of the Data Warehouse, again in terms of the relations in the sources,
   - the merging clauses associated to T.

Observe that the complete data acquisition plan for T provides the designer with the specification of the matching, conversion, and merging functions that should be designed and implemented in order for T to acquire the correct tuples from the sources. The system is also able to provide the designer with the following information associated to each function:

- the type of the various arguments,
- the meaning of the various arguments at the conceptual level.

Referring to our ongoing example, a possible rewriting of the query that defines table TDW (see Section 6.3) in the Data Warehouse in terms of $\texttt{TABLE}_1$ in Source 1 and $\texttt{TABLE}_2$ in Source 2 (see Section 6.1) is the following:

$$\begin{aligned}\texttt{TDW}(X,S,D,P) \leftarrow\ & \texttt{TABLE}_2(X,S,D) \wedge \texttt{TABLE}_1(X,D,P) \vee \\ & \texttt{TABLE}_1(X,D,P) \wedge S = \texttt{NULL} \wedge P = \texttt{true} \ \mid \\ & X :: rec(N),\ \texttt{SSN}(X,N),\ \texttt{String}(N), \\ & S :: rec(I),\ \texttt{ServNo}(S,I),\ \texttt{Integer}(I), \\ & D :: rec(C),\ \texttt{Code}(D,C),\ \texttt{Integer}(C), \\ & P :: \texttt{Boolean}\end{aligned}$$

Notice that the adornment of the above rewritten query is the one coming from the original query.

To obtain the data acquisition plan for TDW we rename the variables in the body of the query, introduce appropriate *convert* clauses, add the constraints deriving from the adornments of $\texttt{TABLE}_1$ and $\texttt{TABLE}_2$, and finally add appropriate merging clauses:

$$\begin{aligned}\texttt{TDW}(X,S,D,P) \leftarrow\ & \texttt{TABLE}_2(X_2,S_2,D_2) \wedge \texttt{TABLE}_1(X_1,D_1,P_1) \wedge \\ & convert_1(X,X_2,X_1) \wedge convert_2(S,S_2) \wedge convert_3(D,D_2,D_1) \wedge convert_4(P,P_1) \\ & \vee \\ & \texttt{TABLE}_1(X_3,D_3,P_3) \wedge S = \texttt{NULL} \wedge P_3 = \texttt{true} \wedge \\ & convert_5(X,X_3) \wedge convert_6(D,D_3) \wedge convert_7(P,P_3) \ \mid \\ & X :: rec(N),\ \texttt{SSN}(X,N),\ \texttt{String}(N), \\ & S :: rec(I),\ \texttt{ServNo}(S,I),\ \texttt{Integer}(I), \\ & D :: rec(C),\ \texttt{Code}(D,C),\ \texttt{Integer}(C), \\ & P :: \texttt{Boolean}, \\ & X_1 :: rec(S_1),\ \texttt{SSN}_1(X_1,S_1),\ \texttt{String}(S_1), \\ & D_1 :: rec(N_1),\ \texttt{Name}_1(D_1,N_1),\ \texttt{String}(N_1), \\ & P_1 :: \texttt{Boolean}, \\ & X_2 :: rec(N_2),\ \texttt{NameC}_2(X_2,N_2),\ \texttt{String}(N_2), \\ & S_2 :: rec(I_2),\ \texttt{ServNo}_2(S_2,I_2),\ \texttt{Integer}(I_2), \\ & D_2 :: rec(M_2),\ \texttt{Name}_2(D_2,M_2),\ \texttt{String}(M_2), \\ & X_3 :: rec(S_3),\ \texttt{SSN}_1(X_3,S_3),\ \texttt{String}(S_3), \\ & D_3 :: rec(N_3),\ \texttt{Name}_1(D_3,N_3),\ \texttt{String}(N_3), \\ & P_3 :: \texttt{Boolean} \\ & \text{merging } \texttt{TABLE}_2(X_2,S_2,D_2) \wedge \texttt{TABLE}_1(X_1,D_1,P_1) \wedge \\ & \qquad convert_1(X,X_2,X_1) \wedge convert_2(S,S_2) \wedge convert_3(D,D_2,D_1) \wedge convert_4(P,P_1) \\ & \qquad \text{and} \\ & \qquad \texttt{TABLE}_1(X_3,D_3,P_3) \wedge S = \texttt{NULL} \wedge P_3 = \texttt{true} \wedge \\ & \qquad convert_5(X,X_3) \wedge convert_6(D,D_3) \wedge convert_7(P,P_3) \\ & \text{such that } convert_8(X,X_2,X_3) \wedge convert_9(D,D_2,D_3) \\ & \text{into } \texttt{TDW}(X,S_2,D,P_1)\end{aligned}$$

Note that the data acquisition plan for TDW also provides the designer with relevant information about the various conversion and matching predicates. For example, $convert_1(\alpha,\beta,\gamma)$ is true if and only if the customer γ in Source 1, represented by its social security number, can be matched with the customer β in Source 2 represented by its name, and both can be converted to α in the Data Warehouse, represented again by its social security number.

7.2 Quality Analysis

The most important checks related to quality analysis are related to the need of verifying the quality of the rewriting. If T is a relation in the Data Warehouse Schema, and $q \mid c_1,\ldots,c_n$ the adorned query

associated to T, the first step required by the methodology is to look for a rewriting r of q by means of a suitable set of adorned queries $q_1, \ldots, q_s$ associated to the relations in the sources. If there are different rewritings, then the best rewriting r with respect to suitable quality parameters should be chosen. There are several criteria to be taken into considerations 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 check can be done by exploiting the algorithm for query containment. If no equivalent rewriting r exists, then it could of interest to isolate the part r of the query that can be answered by using the data at the source, and report on the part e of the query that cannot be answered from the sources. We can then use the information on e to trigger a new integration cycle (Source Analysis, Source Model specification, Data Warehouse specification etc.).
- Accuracy, confidence, freshness, and availability of data in the source relations that the rewriting requires to access.

8 Conclusions

The results reported in the present document show that the tasks within workpackages WP3 and WP4 (Source and Data Integration) are being accomplished according to the workplan and significant progress has been made towards the objectives. In particular, we have achieved the following goals:

- We have refined the architecture for source integration, which is based on a well-established structure with both a conceptual and a logical level, but embodies several novel features as compared with conventional database design.
- We have specialized the modeling language to the case where the conceptual models are expressed in terms of an Entity-Relationship Model, which is the most popular data model nowadays.
- We have proposed reasoning techniques that provide for the basic reasoning services both at the conceptual and at the logical level, by exploiting previous results on reasoning in expressive Description Logics and extending them to the case of disjunctions of conjunctive queries.
- We have outlined a methodology for systematically adding, characterizing, and integrating schemas and data concerning different information sources.

The remaining part of WP4 will be devoted to a further development of the methodology, with special emphasis on:

1. More precise guidelines for using the metadata repository during the phases of Source and DW Schema Specification and Data reconciliation, in particular 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.
2. The case of source schema expressed in terms of a semistructured data model; this case requires an extension of the formalism at the logical level, in the line of [CDGL98b].
3. Algorithms for query rewriting in our setting. This is the basis of a more complete treatment of Client-driven integration, and for a more detailed treatment of quality factors in the phase of Data integration and reconciliation.
4. How to exploit the complete data acquisition plan associated to a relation schema in the Data Warehouse in order to produce an effective specification of the mediators that are responsible of loading and maintaining the corresponding relation.

References

[BCN92] Carlo Batini, Stefano Ceri, and Sham B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach.* Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.

[BN94] Sonia Bergamaschi and Bernhard Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence*, 4(2):185–203, 1994.

[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)*, number 1013 in 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.

[CDGL98b] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. What can knowledge representation do for semi-structured data? In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI-98)*, pages 205–210, 1998.

[CDGL+98c] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS-98)*, pages 280–291, 1998.

[CDGL+98d] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-98)*, pages 2–13, 1998.

[CDGL+98e] 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.

[CK86] S. S. Cosmadakis and P. C. Kanellakis. Functional and inclusion dependencies - A graph theoretical approach. In P. C. Kanellakis and F. P. Preparata, editors, *Advances in Computing Research, Vol. 3*, pages 163–184. JAI Press, 1986.

[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.

[CLN94] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. A unified framework for class based representation formalisms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 109–120, Bonn, 1994. Morgan Kaufmann, Los Altos.

[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.

[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*, 1998. To appear.

[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.

[JJQV98] Matthias Jarke, Manfred A. Jeusfeld, Christoph Quix, and Panos Vassiliadis. Quality oriented evolution of dw designs. Technical Report D.9.1, DWQ Consortium, 1998.

[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)*, 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 Publishers, 1992.

[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.

Appendices

We include as appendices the following publications: [CDGL98a, CDGL+98c, CDGL+98d, CDGL+98e, CDGL98b].