Document Number:	DWQ-UNIROMA-007
Project Name:	Foundations of Data Warehouse Quality (DWQ)
Project Number:	EP 22469
Title:	**Use of the Reconciliation Tool at Telecom Italia**
Authors:	Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, Riccardo Rosati
Workpackage:	WP4
Document Type:	Deliverable (D4.3)
Classification:	Public
Distribution:	DWQ Consortium
Status:	Final
Document file:	wp4-3.tex
Version:	1.2
Date:	October 11, 1999
Number of pages:	33

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

1 Introduction

This document describes the experimentation of the DWQ methodology within Telecom Italia. The experimentation consisted in developing a case study based on an actual data sources at Telecom Italia. The case study has been entirely modeled making use of the reconciliation tool based on the DWQ Concept Base repository (see Deliverable D4.2 - DWQ-UNIROMA-006 for a description of the tool). In this document we do not present the schema expressed in Concept Base but, to simplify the exposition, we describe the case study discursively.

For more details about the methodology and the tool for source and data integration in DWQ, we refer to [CDGL+97, CDGL+99b, CDGL+99a], and for more details about the collaboration between the University of Rome "La Sapienza" and Telecom Italia within DWQ, we refer to [TLN99].

1.1 Organization of the Report

The document is organized as follows. In this section, we summarize the architecture and the methodology developed within DWQ, and then we provide a general description of the case study.

In Section 2, we describe the application at the conceptual level. We illustrate the Enterprise Model, and the Conceptual Model of 5 data sources. In Section 3, we describe the application at the logical level. In particular, we illustrate the Source Schema of each of the 5 sources. In Section 4, we present the interschema correspondence used to declaratively specify the correspondences between data in different sources, and we illustrate by means of an example how mediators to answer queries of interest are specified.

1.2 Summary of the Architecture

Generally speaking, if we refer to the DWQ framework illustrated in the Deliverable D2.1, 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 intermodel assertions. 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. We use again a logic-based formalism

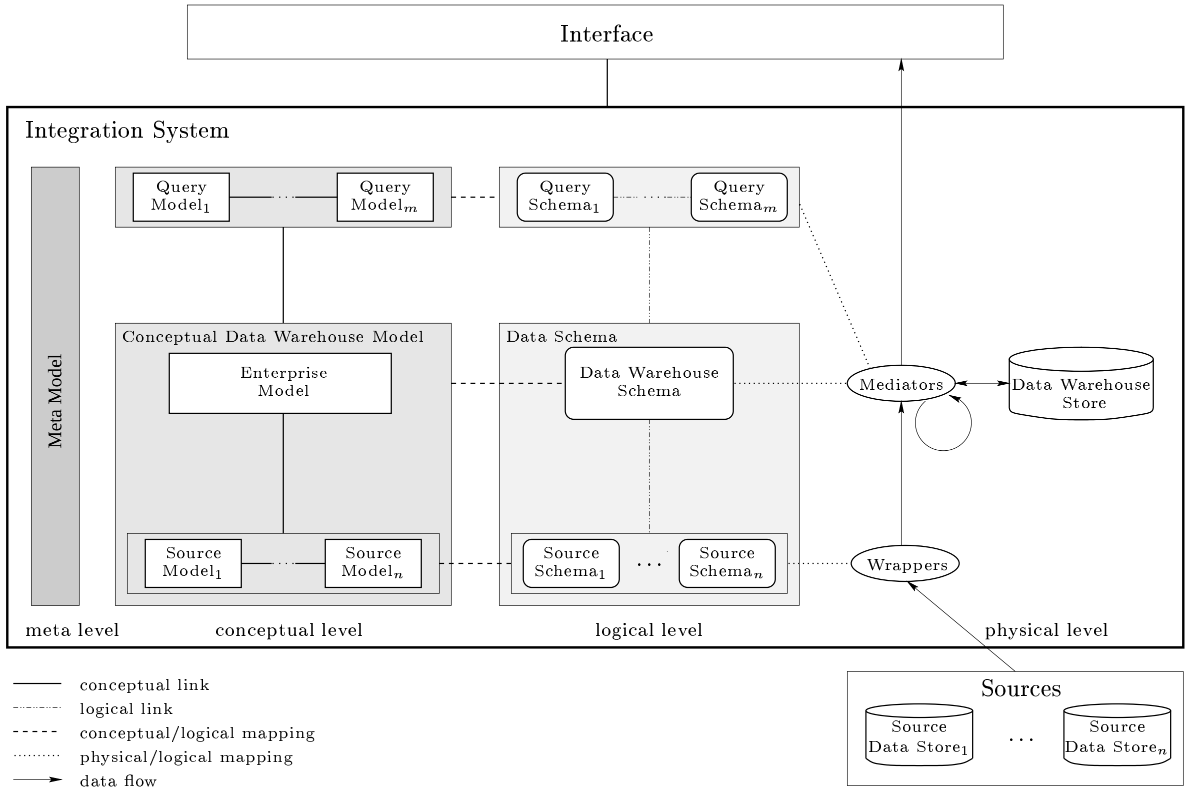

Figure 1: Architecture for Data Integration

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 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). Collectively the sources schemas and the materialized view schemas constitute the *Data Schema*. In addition to the Source schemata and the Data Warehouse Schema, the Data Schema contains the *interschema correspondences* that are used to relate data belonging to the different schemas.

- Each element of the integration process has a corresponding representation in the DWQ Repository (see Deliverable D2.1).

1.3 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 (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.

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 enterprise is produced, if not available.
2. *Source Model integration.* The Source Model is *integrated into the Conceptual Data Warehouse Model.* This can lead to changes both to the Source Models, and to the Enterprise Model. Moreover, intermodel assertions between the Enterprise Model and the Source Models and between the new source and the existing sources are added to the Conceptual Data Warehouse Model.
3. *Source and Data Warehouse Schema specification.* The Source Schema corresponding to the new source (or, corresponding to a new portion of the source) is produced. On the basis of the new analyzed source, an analysis is carried out on whether the Data Warehouse Schema should be restructured and/or modified.
4. *Data Integration and Reconciliation.* The problem of data integration and reconciliation arises when data passes from the application-oriented environment to the Data Warehouse. During the transfer of data, possible inconsistencies and redundancies should be resolved, so that the warehouse is able to provide and integrated and reconciled view of data of the organization. In our methodology, the problem of data reconciliation is addressed after the phase of Data Schema specification, and is based on (1) specifying through the notion of *Interschema Correspondences* how the relations in the Data Warehouse Schema are linked to the relations in the Source Schemas, and (2) designing suitable *mediators* for every relation in the Data Warehouse Schema. In step (1), interschema correspondences are used to declaratively specify the correspondences between data in different schemas (either source schemas or data warehouse schema). Interschema Correspondences are defined in terms of relational tables, similarly to the case of the relations describing the sources at the logical level. We distinguish among three types of correspondences, namely *Conversion*, *Matching*, and *Reconciliation Correspondences.* By virtue of such correspondences, the designer can specify different forms of data conflicts holding between the source, and can anticipate methods for solving such conflicts when loading the Data Warehouse. In step (2), the methodology aims at producing, for every relation in the Data Warehouse Schema, a specification of the corresponding mediator, which determines how the tuples of such a relation should be constructed from a suitable set of tuples extracted from the relations stored in the sources.

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

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 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.

1.4 General Description of the Case Study

The case study developed with Telecom Italia refers to a Telecom application, where the information of interest concern the customers, the contracts, and the services provided by different Telecom Companies in Europe. We refer to [TLN99] for a description of both the general approach to Data Warehousing in Telecom Italia, and the collaboration between Telecom Italia and UNIROMA in the context of the DWQ project. The case study is based on the following 5 sources:

- Source 1: **Contracts**

  This source contains data about contracts between customers and Telecom companies. The description of the source is based on a real source at Telecom Italia.

- Source 2: **Business Contracts**

  This source contains data about customers that are companies, together with their contracts for various telecommunication services. The description of the source is based on a real source at Telecom Italia.

- Source 3: **Private Contracts**

  This source contains data about customers that are persons, together with their contracts. These contracts typically concern phone line services. The description of the source is based on a real source at Telecom Italia.

- Source 4: **French Calls**

  This sources contains data about phone calls issued from France. The description of the source is simplified.

- Source 5: **German Calls**

  This sources contains data about phone calls issued from Germany. The description of the source is simplified.

Note that all names of the objects in the various source conceptual models and the various source schemas will be followed by the number identifying the source. For the objects in the Enterprise model we will use names without subscripts.

2 Conceptual Models

In this section we provide a description of all the conceptual models of our case study. We will describe the Enterprise Model first, and then the conceptual models of each of the 5 sources.

2.1 Enterprise Model

The diagrammatic representation of the Enterprise Model is shown in the figure. In the diagram, we do not show the attributes of entities and relationships. We provide separately an explanation for each of the objects in the Enterprise Model, including the list of attributes. We remind the reader that in our formalism an attribute is seen as a binary relation (binary predicate) linking an entity and a value domain. The value domain (e.g., integer, string, etc.) of the attributes is not explicitly specified: it can be easily derived from the context.

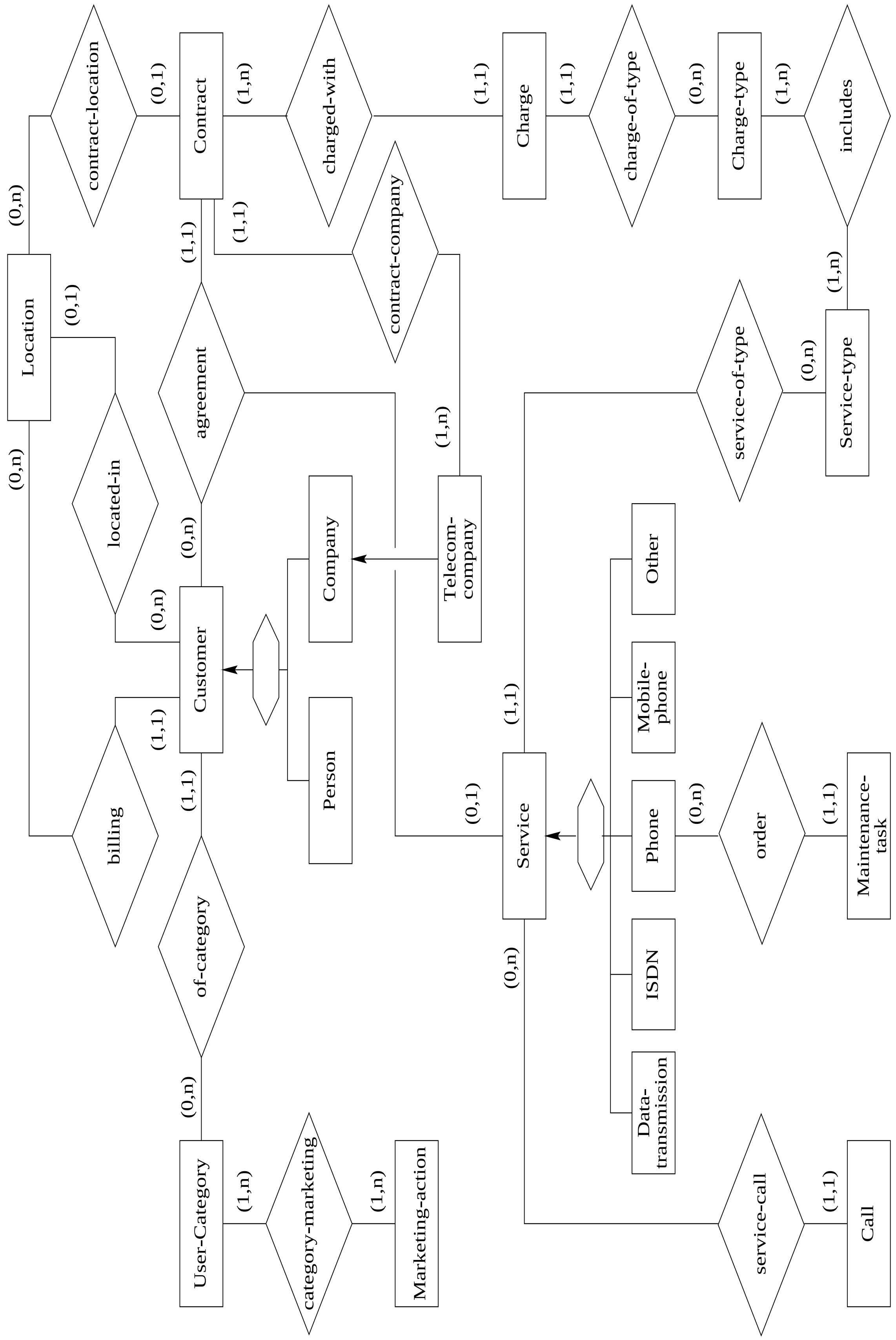

Figure 2: Enterprise Model

- The entity `Customer` represents the set of customers of the Telecom companies, partitioned into `Person` and `Company`. The attributes of the entity are:
  - `customer-code`
  - `customer-description`
- The entity `Person` represents the set of customers which are persons. The attributes of the entity are those of `Customer`, plus:
  - `ssn`
  - `family-name`
  - `first-name`
  - `date-of-birth`
  - `sex`
  - `place-of-birth`
- The entity `Company` represents the set of customers which are company. The attributes of the entity are those of `Customer`, plus:
  - `tax-code`
  - `number-of-employee`
  - `company-name`
  - `company-type`
- The entity `Telecom-company` represents the set of Telecom companies providing services to customers. The attributes of the entity are those of `Company`.
- The entity `Location` represents the set of locations of customers. The attributes of the entity are:
  - `apart-no` (optional)
  - `street`
  - `no`
  - `zip-code`
  - `city`
  - `state` (optional)
  - `country`
- The entity `Contract` represents the set of contracts of customers. The attributes of the entity are:
  - `contract-code`
  - `date`
  - `validity-period`
- The entity `Service` represents the set of services provided by Telecom companies to customers, partitioned into `Data-transmission`, `Isdn`, `Phone`, `Mobile-phone`, and `Other`. The only attribute of the entity is
  - `service-code`

  It is important not to confuse `Service` with `Service-type`. An instance s of `Service` is a service which concerns a certain contract with a certain customer. The connection between services and service types is that every service has a type: for example, s could be of type "ISDN" (and in this case s will also be an instance of the entity `Isdn`).

- The entity `Service-type` represents the types of services provided by Telecom companies to customers. Examples of instances of such entity are: "data-transmission", "ISDN", "phone", "mobile-phone", etc. The attributes of the entity are:
  - `service-type-code`
  - `service-type-description`
- The entity `Data-transmission` represents the set of services of type "data transmission". The attributes of the entity are:
  - `protocol`
  - `speed`
  - `line-number`
- The entity `Isdn` represents the set of services of type "ISDN". The attributes of the entity are:
  - `isdn-number`
  - `isdn-type`
- The entity `Phone` represents the set of services of type "phone". The only attribute of the entity is:
  - `phone-number`
- The entity `Mobile-phone` represents the set of services of type "mobile phone". The attributes of the entity are:
  - `mobile-ph-number`
  - `mobile-ph-type`
- The entity `Other` represents the set of services of other type. The only attribute of the entity is:
  - `sub-type`
- The entity `User-category` represents the set of category used for customer classification. The attributes of the entity are:
  - `category-code`
  - `category-description`
- The entity `Marketing-action` represents the set of marketing actions that the Telecom companies undertake towards customers. The attributes of the entity are:
  - `action-code`
  - `action-description`
- The entity `Call` represents the set of calls issued by a certain customer under a certain service. The attribute of the entity are:
  - `date`
  - `destination`
  - `start-time`
  - `end-time`
  - `cost`
- The entity `Charge-type` represents the types of charge that the company adopts. The attribute of the entity are:
  - `charge-type-code`

  - `charge-type-description`
  - `amount-of-money`
  - `frequency`

- The entity `Charge` represents the charges concerning a certain contract. The attribute of the entity are:
  - `charge-code`
  - `time`
  - `date-of-payment` (optional)

  If `date-of-payment` is missing, then the corresponding payment has not been done. It is important not to confuse `Charge` with `Charge-type`. An instance c of `Charge` is a charge which concerns a certain contract. The connection between charges and charge type is that every charge has a type: for example, c could be of type t, which means "fixed", "monthly", etc.

- The entity `Maintenance-task` represents the set of tasks carried out for maintenance purposes by the Telecom companies. The attribute of the entity are:
  - `task-code`
  - `task-description`
  - `date-of-request`
  - `date-of-execution` (optional)

  If `date-of-execution` is missing, then the corresponding task has not been executed.

- The relationship `agreement` represents an agreement between a customer and a Telecom company for a certain service, formalized by a specified contract. Notice that the Telecom company involved in an agreement is represented by means of the relationship `contract-company`.
- The relationship `contract-company` represents the connection between a contract and the Telecom company involved.
- The relationship `located-in` links a customer with his location. The relationship `billing` links a customer and the location chosen for billing.
- The relationship `contract-location` links a contract and the location where the services specified in the contract are to be provided.
- The relationship `category-marketing` links each customer category with the marketing actions that the Telecom companies aim at carrying out for the customers of that category.
- The relationships `service-of-type` links the service with the corresponding type.
- The relationships `charge-of-type` links the charge with the corresponding type.
- The relationship `includes` links service types and the corresponding charge types. The relationship has two attributes:
  - `start-date`
  - `end-date`

  to indicate the period of time in which the association between a certain service type and a certain charge type is valid. Note that several charge types may exist for the same service type, and, conversely, many service types may be associated with the same charge types.

- The relationship `service-call` links each call to the corresponding service. Note that such service determines the customer who issued the call, and the contract. In turn the contract determines the Telecom company who managed the call.
- The relationship `order` links every maintenance task with the phone line for which such a maintenance task was requested.

2.2 Source 1: Conceptual Model

We describe the conceptual model of Source 1. The diagrammatic representation of the model is shown in the figure. We provide separately an explanation for each of the objects in the conceptual model of Source 1, illustrating the connection with the objects in the Enterprise Model.

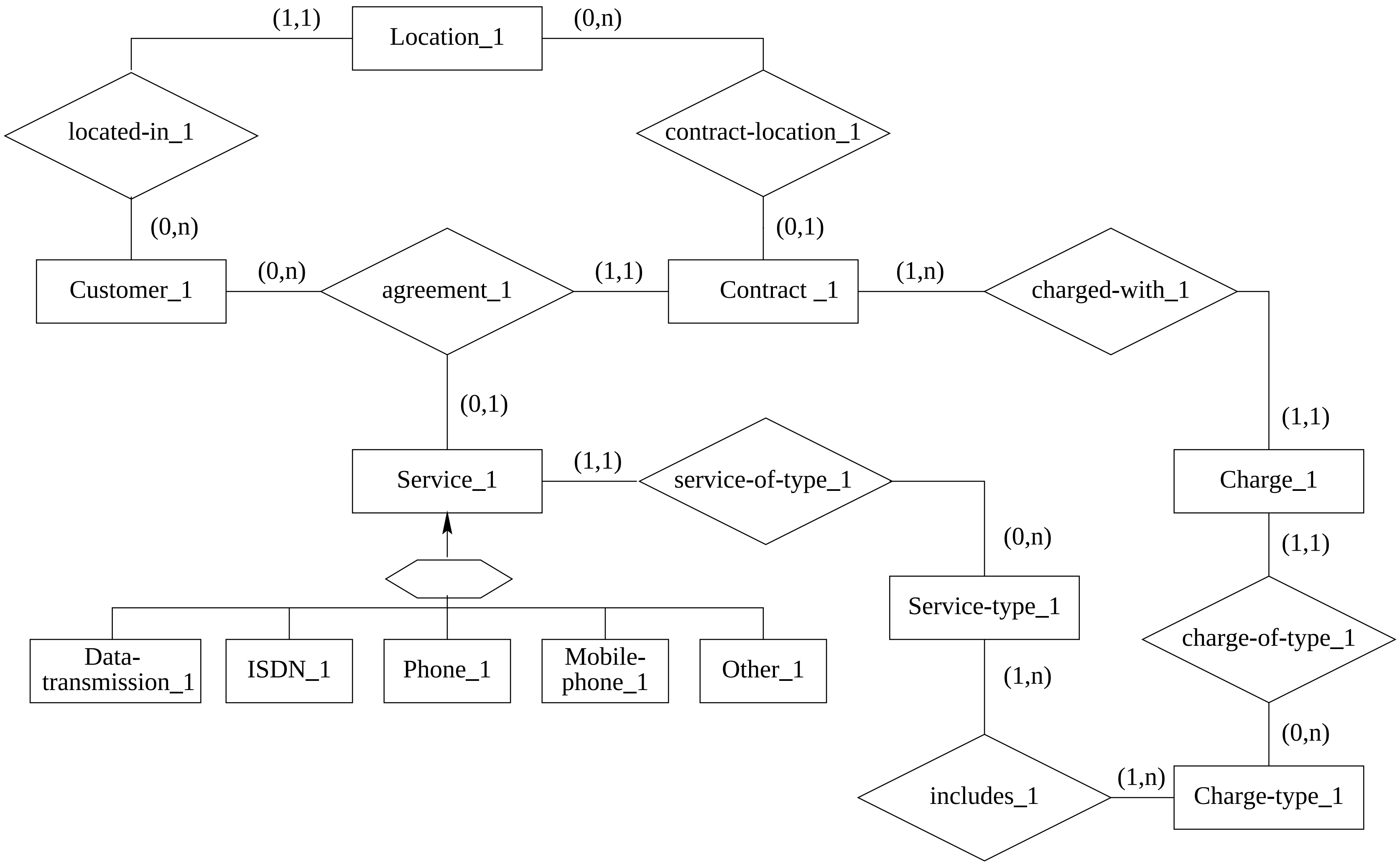

Figure 3: Conceptual model of Source 1: Contracts

- The entity `Customer_1` represents the set of customers of Telecom Italia. The attributes of the entity are:
  - `customer-code_1`
  - `customer-description_1`

  This entity corresponds to the entity `Customer` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity `Location_1` represents the set of locations of customers of Telecom Italia. The attributes of the entity are:
  - `loc-code_1`
  - `apart-no_1` (optional)
  - `street_1`
  - `no_1`
  - `zip-code_1`
  - `city_1`
  - `state_1` (optional)
  - `country_1`

  This entity corresponds to the entity `Location` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity `Contract_1` represents the set of contracts of customers of Telecom Italia. The attributes of the entity are:
  - `contract-code_1`
  - `date_1`
  - `validity-period_1`

  This entity corresponds to the entity `Contract` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity `Service_1` represents the set of services provided by Telecom Italia to customers, partitioned into `Data-transmission_1`, `Isdn_1`, `Phone_1`, `Mobile-phone_1`, and `Other_1`. The only attribute of the entity is
  - `service-code_1`

  This entity corresponds to the entity `Service` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity `Service-type_1` represents the types of services provided by Telecom Italia to customers. The attributes of the entity are:
  - `service-type-code_1`
  - `service-type-description_1`

  This entity corresponds to the entity `Service-type` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity `Data-transmission_1` represents the set of Telecom Italia services of type "data transmission". The attributes of the entity are:
  - `speed_1`

  - line-number_1

  This entity corresponds to the entity Data-Transmission of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Isdn_1 represents the set of Telecom Italia services of type "ISDN". The attributes of the entity are:
  - isdn-number_1
  - isdn-type_1

  This entity corresponds to the entity Isdn of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Phone_1 represents the set of Telecom Italia services of type "phone". The only attribute of the entity is:
  - phone-number_1

  This entity corresponds to the entity Phone of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Mobile-phone_1 represents the set of Telecom Italia services of type "mobile phone". The attributes of the entity are:
  - mobile-ph-number_1
  - mobile-ph-type_1

  This entity corresponds to the entity Mobile-phone of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Other_1 represents the set of Telecom Italia services of other type. The only attribute of the entity is:
  - sub-type_1

  This entity corresponds to the entity Other of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Charge-type_1 represents the types of charge that Telecom Italia adopts. The attribute of the entity are:
  - charge-type-code_1
  - amount-of-money_1
  - frequency_1

  This entity corresponds to the entity Charge-type of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Charge_1 represents the charges concerning a certain contract of Telecom Italia. The attribute of the entity are:
  - charge-code_1
  - time_1
  - date-of-payment_1 (optional)

  This entity corresponds to the entity Charge of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The relationship `agreement_1` represents an agreement between a customer and Telecom Italia for a certain service, formalized by a specified contract.

  This relationship corresponds to the relationship `agreement` of the Enterprise Model.
- The relationship `located-in_1` corresponds to the relationship `located-in` of the Enterprise Model.
- The relationship `contract-location_1` corresponds to the relationship `contract-location` of the Enterprise Model.
- The relationship `service-of-type_1` corresponds to the relationship `service-of-type` of the Enterprise Model.
- The relationship `charge-of-type_1` corresponds to the relationship `charge-of-type` of the Enterprise Model.
- The relationship `includes_1` corresponds to the relationship `includes` of the Enterprise Model, with the corresponding attributes.

2.3 Source 2: Conceptual Model

We describe the conceptual model of Source 2. The diagrammatic representation of the model is shown in the figure. We provide separately an explanation for each of the objects in the conceptual model of Source 2, illustrating the connection with the objects in the Enterprise Model.

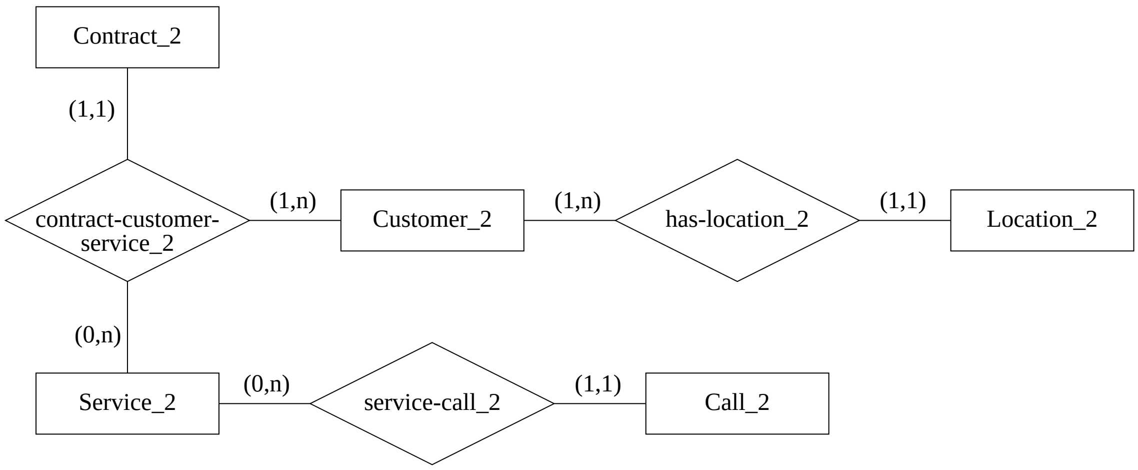

Figure 4: Conceptual model of Source 2: Business contracts

- The entity `Customer_2` represents the set of companies that are customers of Telecom Italia. The attributes of the entity are:
  - `tax-code_2`
  - `number-of-employee_2`
  - `company-name_2`
  - `company-type_2`

  This entity corresponds to the entity `Company` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.
- The entity `Location_2` represents the set of locations of customers of Telecom Italia. The attributes of the entity are:
  - `loc-code_2`

  - apart-no_2 (optional)
  - street_2
  - no_2
  - zip-code_2
  - city_2
  - state_2 (optional)
  - country_2

  This entity corresponds to the entity Location of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Contract_2 represents the set of contracts of company customers of Telecom Italia. The attributes of the entity are:
  - contract-code_2
  - date_2
  - validity-period_2

  This entity corresponds to the entity Contract of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Service_2 represents the set of services provided by Telecom Italia to company customers. The attributes of the entity are:
  - service-code_2
  - service-type_2
  - service-number_2 (optional)

  This entity corresponds to the entity Service of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Call_2 represents the set of calls issued by a certain Telecom Italia customer under a certain service. The attribute of the entity are:
  - date_2
  - destination_2
  - start-time_2
  - end-time_2
  - cost_2

  This entity corresponds to the entity Call of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The relationship contract-customer-service_2 represents an agreement between a customer and Telecom Italia for a certain service, formalized by a specified contract.

  This relationship corresponds to the relationship agreement of the Enterprise Model.

- The relationship has-location_2 corresponds to the relationship located-in of the Enterprise Model.

- The relationship service-call_2 corresponds to the relationship service-call of the Enterprise Model.

2.4 Source 3: Conceptual Model

We describe the conceptual model of Source 3. The diagrammatic representation of the model is shown in the figure. We provide separately an explanation for each of the objects in the conceptual model of Source 3, illustrating the connection with the objects in the Enterprise Model.

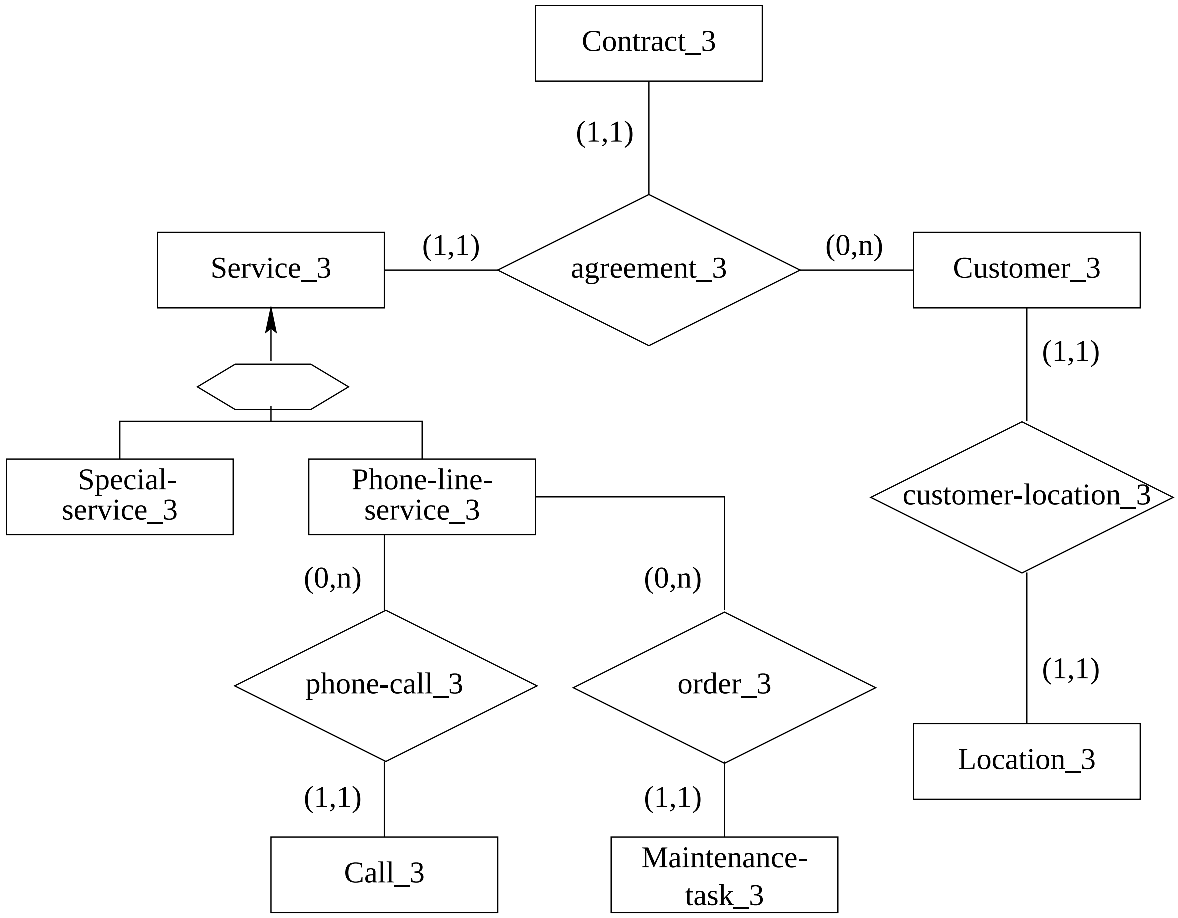

Figure 5: Conceptual model of Source 3: Private contracts

- The entity `Customer_3` represents the set of persons that are customers of Telecom Italia. The attributes of the entity are:
  - `ssn_3`
  - `family-name_3`
  - `first-name_3`
  - `date-of-birth_3`
  - `sex_3`
  - `place-of-birth_3`

  This entity corresponds to the entity `Person` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity `Location_3` represents the set of locations of customers of Telecom Italia. The attributes of the entity are:
  - `loc-code_3`

  - apart-no_3 (optional)
  - street_3
  - no_3
  - zip-code_3
  - city_3
  - state_3 (optional)
  - country_3

  This entity corresponds to the entity Location of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Contract_3 represents the set of contracts of private customers of Telecom Italia. The attributes of the entity are:
  - contract-code_3
  - date_3
  - validity-period_3

  This entity corresponds to the entity Contract of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Service_3 represents the set of services provided by Telecom Italia to private customers, partitioned into Phone-line-service_3, and Special-service_3. The only attribute of the entity is
  - service-code_3

  This entity corresponds to the entity Service of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Phone-line-service_3 represents the set of Telecom Italia services of type "phone" provided to private customers. The only attribute of the entity is:
  - phone-number_3

  This entity corresponds to the entity Phone of the Enterprise Model. Analogously, the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Special-service_3 represents the set of Telecom Italia services of special type. The only attribute of the entity is:
  - sub-type_3

  This entity corresponds to (in the sense that it is a subset of) the entity Other of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Maintenance-task_3 represents the set of tasks carried out by Telecom Italia for maintenance purposes. The attribute of the entity are:
  - task-code_3
  - task-description_3
  - date-of-request_3
  - date-of-execution_3

  This entity corresponds to the entity Maintenance-task of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity Call_3 represents the set of calls issued by a certain Telecom Italia customer under a certain service. The attribute of the entity are:
  - date_3
  - destination_3
  - start-time_3
  - end-time_3
  - cost_3

  This entity corresponds to the entity Call of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.
- The relationship agreement_3 represents an agreement between a customer and Telecom Italia for a certain service, formalized by a specified contract.

  This relationship corresponds to the relationship agreement of the Enterprise Model.
- The relationship customer-location_3 corresponds to the relationship located-in of the Enterprise Model.
- The relationship order_3 corresponds to the relationship order of the Enterprise Model.
- The relationship phone-call_3 corresponds to the relationship service-call of the Enterprise Model.

2.5 Source 4: Conceptual Model

We describe the conceptual model of Source 4. The diagrammatic representation of the model is shown in the figure. We provide separately an explanation for each of the objects in the conceptual model of Source 4, illustrating the connection with the objects in the Enterprise Model.

Figure 6: Conceptual model of Source 4: French calls

- The entity Call_4 represents the set of calls issued by a certain French Telecom customer under a certain service. The attribute of the entity are:
  - date_4
  - destination_4
  - start-time_4
  - end-time_4
  - cost_4

  This entity corresponds to the entity Call of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.
- The entity Service_4 represents the set of service provided by French Telecom. The attribute of the entity are:
  - service-code_4
  - service-description_4

  – `number_4`

  This entity corresponds to the entity `Service` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The relationship `service-call_4` corresponds to the relationship `service-call` of the Enterprise Model.

2.6 Source 5: Conceptual Model

We describe the conceptual model of Source 5. The diagrammatic representation of the model is shown in the figure. We provide separately an explanation for each of the objects in the conceptual model of Source 5, illustrating the connection with the objects in the Enterprise Model.

Figure 7: Conceptual model of Source 5: German calls

- The entity `Call_5` represents the set of calls issued by a certain Deutsche Telecom customer under a certain service. The attribute of the entity are:
  - `date_5`
  - `destination_5`
  - `start-time_5`
  - `end-time_5`
  - `cost_5`

  This entity corresponds to the entity `Call` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The entity `Service_5` represents the set of service provided by Deutsche Telecom. The attribute of the entity are:
  - `service-code_5`
  - `service-description_5`
  - `number_5`

  This entity corresponds to the entity `Service` of the Enterprise Model, and the attributes correspond to the analogous attributes of the Enterprise Model.

- The relationship `service-call_5` corresponds to the relationship `service-call` of the Enterprise Model.

3 Logical Schemas

In this section, we describe the logical schemas of the 5 sources of the application. We describe a source S by associating to each relational table T of S an *adorned query* that is constituted by a head, a body, and an adornment:

- The *head* defines the relational schema of the table in terms of a name, and the number of columns.
- The *body* describes the content of the table in terms of a query over the conceptual model.

- The *adornment* declares the domains of the columns of the table, and also which are the attributes of the table that are used to identify an entity of the conceptual model.

Let us recall the main details of the notions above (see Deliverables D4.1 - DWQ-UNIROMA-005, and D4.2 - DWQ-UNIROMA-006, for more details).

Query over the Conceptual Model

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

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

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

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

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

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

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

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

Adornment

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

An *adorned query* is an expression of the form

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

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

1. For each $X \in \vec{\mathbf{x}}$, we have an annotation of the form

$$X :: V$$

   where V is a domain expression. Such an annotation is used to specify how values bound to X are represented in the table at the logical level. For example, which currency is used for a real value denoting an amount of money.

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

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

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

The adorned query associated to a table in a source contains a lot of information that can be profitably used in analyzing the quality of the Data Warehouse design process. Indeed, the adorned query precisely formalizes the content of a source table in terms of a query over the conceptual model, the domains of each attribute of the table, and the attributes used to identify entities at the conceptual level. One important check that we can carry out over the logical specification of a source is whether the adorned query associated with a table in a source is consistent or not. Let Q be an adorned query and let B be its body. The query B is said to be *inconsistent* with respect to the conceptual model $\mathcal{M}$, if for every database DB coherent with $\mathcal{M}$, the evaluation of B with respect to DB is empty. An adorned query Q is inconsistent with respect to the conceptual model $\mathcal{M}$ either because the body B of Q is inconsistent with respect to $\mathcal{M}$, or because the annotations are incoherent with respect to what specified in $\mathcal{M}$. The inference techniques allow us to check the consistency of the relational tables defined for describing a source.

3.1 Source 1: Logical Schema

The logical schema of Source 1 (Contracts) is constituted by the following relational tables.

```
Contract-table_1(Code, Date, Validity-period, Location-code)
Contract-table_1(CC,D,V,LC) :- Contract_1(C), has-code_1(C,CC), date_1(C,D),
                               validity_1(C,V), contract-location_1(C,L),
                               Location(L), loc-code_1(L,LC)
                               OR
                               Contract_1(C), has-code_1(C,CC), date_1(C,D),
                               validity_1(C,V), LC=NULL
                               | CC::String, D::DateValue, V::Months,
                                 LC::(String OR NULL),
                                 identify([CC],C), identify([LC],L)

Location-table_1(Code, Apart-No, Street, No, Zip-code, City, State, Country, Cust)
Location-table_1(D,A,S,Z,C,ST,T,MC) :- Location_1(L), loc-code_1(L,D),
                                       apart-no_1(L,A), street_1(L,S),
                                       zip-code_1(L,Z), city_1(L,C),
                                       state_1(L,ST), country_1(L,T),
                                       located-in_1(M,L), Customer_1(M),
                                       customer-code_1(M,MC)
                                       | D::String, A::(String OR NULL),
                                         S::String, Z::String, C::String,
                                         ST::(State OR NULL), T::String,
                                         MC::String,
                                         identify([D],L), identify([MC],M)

Customer-table_1(Customer-code, Customer-description)
Customer-table_1(CC,D) :- Customer_1(C), customer-code_1(C,CC),
                          customer-description_(C,D)
                          | CC::String, D::String,
                            identify([CC],C)

Service-table_1(Code, Type)
Service-table_1(C,TC) :- Service_1(S), has-code_1(S,C), has-type_1(S,T),
                         Service-type_1(T), service-type-code_1(T,TC)
                         | C::String, T::String,
                           identify([C],S), identify([TC],T)

Data-Transmission-table_1(Code, Speed, Line)
Data-Transmission-table_1(C,E,L) :- Data-Transmission_1(S), has-code_1(S,C),
                                    has-type_1(S,"data"), speed_1(S,E),
                                    line-number_1(S,L)
                                    | C::String, E::Real, L::String,
```

```
                                     identify([C],S)

ISDN-table_1(Code, Line, Isdn-type)
ISDN-table_1(C,L,I) :- ISDN_1(S), has-code_1(S,C),
                       has-type_1(S,"isdn"), line-number_1(S,L), isdn-type(S,I)
                       | C::String, L::String, I::String,
                         identify([C],S)

Phone-table_1(Code, Line)
Phone-table_1(C,L) :- Phone_1(S), has-code_1(S,C), has-type_1(S,"phone"),
                      phone-number_1(S,L)
                      | C::String, L::String,
                        identify([C],S)

Mobile-Phone-table_1(Code, Line, Mob-type)
Mobile-Phone-table_1(C,L,T) :- Mobile-Phone_1(S), has-code_1(S,C),
                               has-type_1(S,"mob"), line-number_1(S,L),
                               mobile-ph-type(S,T)
                               | C::String, L::String, T::String,
                                 identify([C],S)

Other-table_1(Code, Subtype)
Other-table_1(C,T) :- Other_1(S), has-code_1(S,C), has-type_1(S,"other"),
                      sub-type_1(S,T)
                      | C::String, T::String,
                        identify([C],S)

Service-Type-table_1(Code, Description)
Service-Type-table_1(C,D) :- Service-type_1(S), service-type-code_1(S,C),
                             service-type-description_1(S,D)
                             | C::String, D:: String,
                               identify([C],S)

Includes-table_1(Service-type, Charge-type, Start-date, End-date)
Includes-table_1(ST,CT,D,E) :- Service-type_1(S), service-type-code_1(S,ST),
                               Charge-type_1(T), charge-type-code_1(T,CT),
                               start-date_1(S,T,D), end-date_1(S,T,E)
                               | ST:String, CT::String, D::DateValue,
                                 E::DateValue,
                                 identify([ST],S), identify([CT],T)

Agreement-table_1(Contract, Customer, Service)
Agreement-table_1(CC,TC,SC) :- contract-customer-service_1(C,T,S),
                               has-code_1(C,CC), customer-code_1(T,TC),
                               has-code_1(S,SC)
                               | CC::String, TC::String, SC::String,
                                 identify([CC],C), identify([TC],T),
                                 identify([SC],S)

Charge-Type-table_1(Code, Money, Frequency)
Charge-Type-table_1(C,M,F) :- Charge-type_1(T), charge-type-code_1(T,C),
                              amount-of-money_1(T,M), frequency_1(T,F)
                              | C::String, M::Real, F::String,
                                identify([C],T)

Charge-table_1(Code, Time, Date, Type, Contract)
Charge-table_1(C,T,D,Y,CTC) :- Charge_1(R), charge-code_1(R,C), time_1(R,T),
                               date-of-payment_1(R,D), charge-of-type_1(R,Y),
                               charged-with_1(R,CT), contract-code_1(CT,CTC)
```

```
                               | C::String, time::String,
                                 D::(DateValue OR NULL), Y::String,
                                 CTC::String,
                                 identify([C],R), identify([CTC],C)
```

3.2 Source 2: Logical Schema

The logical schema of Source 2 (Business contracts) is constituted by the following relational tables.

```
Call-table_2(Phone, Date, Start-time, End-time, Destination, Cost)
Call-table_2(SC,D,S,E,T,C) :- Call_2(L), service-call_2(L,V),
                              service-code_2(V,SC),
                              contract-customer-service_2(N,M,V),
                              contract-company(N,Y), name(Y,"TelecomItalia"),
                              date_2(L,D), start-time_2(L,S), end-time_2(L,E),
                              destination_2(L,T), cost_2(L,C)
                              | SC::String, D::DateValue, S::TimeValue,
                                E::TimeValue, T::String, C::Real,
                                identify([D,S,E,T],L),
                                identify([SC],V), identify(["TelecomItalia"],Y)

Service-table_2(Code, Type, Number)
Service-table_2(C,T,N) :- Service_2(S), has-code_2(S,C), has-type_2(S,T),
                          Number_2(S,N)
                          | C::String, T::String, N::(String OR NULL),
                            identify([C],S)

Contract-Customer-Service-table_2(Contract, Customer, Service)
Contract-Customer-Service-table_2(CC,TC,SC) :- contract-customer-service_2(C,T,S),
                                               contract-code_2(C,CC),
                                               tax-code_2(T,TC),
                                               service-code_2(S,SC)
                                               | CC::String, TC::String,
                                                 SC::String,
                                                 identify([CC],C),
                                                 identify([TC],T),
                                                 identify([SC],S)

Customer-table_2(Tax-code, Number-of-emp, Name, Type)
Customer-table_2(T,N,M,Y) :- Customer_2(C), tax-code_2(C,T),
                             company-name_2(C,N), company-type_2(C,Y)
                             | T::String, N::String, M::String, Y::String,
                               identify([T],C)

Location-table_2(Code, Apart-No, Street, No, Zip-code, City, State, Country, Cust)
Location-table_2(D,A,S,Z,C,ST,T,MC) :- Location_2(L), loc-code_2(L,D),
                                       apart-no_2(L,A), street_2(L,S),
                                       zip-code_2(L,Z), city_2(L,C),
                                       state_2(L,ST), country_2(L,T),
                                       located-in_2(M,L), Customer_2(M),
                                       customer-code_2(M,MC)
                                       | D::String, A::(String OR NULL),
                                         S::String, Z::String, C::String,
                                         ST::(State OR NULL), T::String,
                                         MC::String,
                                         identify([D],L), identify([MC],M)

Contract-table_2(Code, Date, Validity-period)
Contract-table_2(C,D,V) :- Contract_2(T), has-code_2(T,C), date_2(T,D),
```

```
                           validity_2(T,V)
                           | C::String, D::DateValue, V::Years,
                             identify([C],T)
```

3.3 Source 3: Logical Schema

The logical schema of Source 3 (Private contracts) is constituted by the following relational tables.

```
Call-table_3(Phone, Date, Start-time, End-time, Destination, Cost)
Call-table_3(SC,D,S,E,T,C) :- Call_3(L), phone-call_3(L,V),
                              service-code_3(V,SC),
                              agreement_3(N,M,V), contract-company(N,Y),
                              name(Y,"TelecomItalia"), date_3(L,D),
                              start-time_3(L,S), end-time_3(L,E),
                              destination_2(L,T), cost_2(L,C)
                              | SC::String, D::DateValue, S::TimeValue,
                                E::TimeValue, T::String, C::Real,
                                identify([D,S,E,T],L),
                                identify([SC],V),
                                identify(["TelecomItalia"],Y)

Customer-table_3(SSN, Family-name, First-name, Date-of-birth, Sex, Place-of-birth, Location)
Customer-table_3(S,F,N,D,S,P,LC) :- Customer_3(C), ssn_3(C,S),
                                    family-name_3(C,F), first-name_3(C,N),
                                    date-of-birth_3(C,D), sex_3(C,S),
                                    place-of-birth_3(C,P),
                                    customer-location_3(C,L), loc-code(L,LC)
                                    | S::String, F::String, N::String,
                                      D::DateValue, S::{f,m}, P::String
                                      LC::String,
                                      identify([S],C), identify([LC],L)

Location-table_3(Code, Apart-No, Street, No, Zip-code, City, State, Country)
Location-table_3(D,A,S,Z,C,ST,T) :- Location_3(L), loc-code_3(L,D),
                                    apart-no_3(L,A), street_3(L,S),
                                    zip-code_3(L,Z), city_3(L,C),
                                    state_3(L,ST), country_3(L,T),
                                    | D::String, A::(String OR NULL),
                                      S::String, Z::String, C::String,
                                      ST::(State OR NULL), T::String,
                                      identify([D],L)

Contract-table_3(Code, Date, Validity-period)
Contract-table_3(C,D,V) :- Contract_3(T), has-code_3(T,C), date_3(T,D),
                           validity_3(T,V)
                           | C::String, D::DateValue, V::Months,
                             identify([C],T)

Agreement-table_3(Contract, Customer, Service)
Agreement-table_3(CC,TC,SC) :- agreement_3(C,T,S), contract-code_2(C,CC),
                               ssn_3(T,TC), service-code_3(S,SC)
                               | CC::String, TC::String, SC::String,
                                 identify([CC],C), identify([TC],T),
                                 identify([SC],S)

Phone-Service-table_3(Code, Number)
Phone-Service-table_3(C,N) :- Service_3(S), service-code_3(S,C),
                              phone-number_3(S,N)
                              | C::String, N::String,
```

```
                                identify([C],S)

Special-Service-table_3(Code, Type)
Special-Service-table_3(C,T) :- Service_3(S), service-code_3(S,C),
                                sub-type_3(S,T)
                                | C::String, T::String,
                                  identify([C],S)

Maintenance-Task-table_3(Code, Description, Date-of-request, Date-of-execution, Service)
Maintenance-Task-table_3(C,D,R,E,SC) :- Maintenance-task_3(M), code_3(M,C),
                                        description_3(M,D),
                                        date-of-request_3(M,R),
                                        date-of-execution_3(M,E), order_3(M,S),
                                        service-code_3(S,SC)
                                        | C::String, D::String, R::DateValue,
                                          E::(DateValue OR NULL), SC::String,
                                          identify([C],M), identify([SC],S)
```

3.4 Source 4: Logical Schema

The logical schema of Source 4 (French calls) is constituted by the following relational tables.

```
Call-table_4(Phone, Date, Start-time, End-time, Destination, Cost)
Call-table_4(SC,D,S,E,T,C) :- Call_4(L), service-call_4(L,V),
                              service-code_4(V,SC),
                              agreement(N,M,V), contract-company(N,Y),
                              name(Y,"TelecomFrance"), date_4(L,D),
                              start-time_4(L,S), end-time_4(L,E),
                              destination_4(L,T), cost_4(L,C)
                              | SC::String, D::DateValue, S::TimeValue,
                                E::TimeValue, T::String, C::Real,
                                identify([D,S,E,T],L),
                                identify([SC],V), identify(["TelecomFrance"],Y)

Service-table_4(Code, Description, Number)
Service-table_4(C,D,N) :- Service_4(S), service-code_4(S,C),
                          service-description_4(S,D), number_4(S,N)
                          | C::String, D::String, N::String
                            identify([C],S)
```

3.5 Source 5: Logical Schema

The logical schema of Source 5 (German calls) is constituted by the following relational tables.

```
Call-table_5(Phone, Date, Start-time, End-time, Destination, Cost)
Call-table_5(SC,D,S,E,T,C) :- Call_5(L), service-call_5(L,V),
                              service-code_5(V,SC),
                              agreement(N,M,V), contract-company(N,Y),
                              name(Y,"DeutschTelecom"), date_5(C,D),
                              start-time_5(L,S), end-time_5(L,E),
                              destination_5(L,T), cost_5(L,C)
                              | SC::String, D::DateValue, S::TimeValue,
                                E::TimeValue, T::String, C::Real,
                                identify([D,S,E,T],L), identify([SC],V),
                                identify(["DeutschTelecom"],Y)

Service-table_5(Code, Description, Number)
Service-table_5(C,D,N) :- Service_5(S), service-code_5(S,C),
                          service-description_5(S,D), number_5(S,N)
```

```
| C::String, D::String, N::String,
  identify([C],S)
```

4 Data Integration and Reconciliation

In our methodology, the problem of data reconciliation is addressed in tow steps, namely, Interschema correspondences specification, and Mediator specification.

4.1 Interschema Correspondences

Interschema Correspondences are used to declaratively specify the correspondences between data in different schemas (either source schemas or data warehouse schema). Interschema Correspondences are defined in terms of relational tables, similarly to the case of the relations describing the sources at the logical level. The difference with source relations is that we conceive interschema correspondences as non-materialized relational tables, in the sense that their extension is computed by an associated program whenever it is needed. It follows that, to each interschema correspondence, we associate a *head*, a *body*, and an *adornment*. Differently from the case of a source relation, the adornment specifies which is the program that is able to compute the extension of the virtual table.

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

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

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

A Conversion Correspondence *convert* has the following form:

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

where *conj* is a conjunctive query, which specifies the conditions under which the conversion is applicable, and *program* is a predicate that we assume associated to a program that performs the conversion. In general, the program needs to take into account the additional parameters specified in the condition to actually perform the conversion. The conversion has a direction. In particular, it operates from a tuple of values satisfying the conditions specified for $\vec{\mathbf{x}}$ in *conj* to a tuple of values satisfying the conditions specified for $\vec{\mathbf{y}}$. This means that the conversion program receives as input a tuple $\vec{\mathbf{x}}$, and returns the corresponding tuple $\vec{\mathbf{y}}$, possibly using the additional parameter $\vec{\mathbf{z}}$ to perform the conversion.

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

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

where *conj* specifies the conditions under which the matching is applicable, and *program* is a predicate that we assume associated to a program that performs the matching. The program receives as input k tuples of values satisfying the conditions (and possibly the additional parameters in the condition) and returns whether they match or not.

Note that already specified Interschema Correspondences may be used to define new ones. As an example, the designer may want to define a Matching Correspondence between two tuples by using two already defined Conversion Correspondences, which convert to a common representation, and then by using equality. In this case, he could provide the following definition of the Matching Correspondence:

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

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

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

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

where *conj* specifies the conditions under which the reconciliation is applicable, and *program* is a predicate that we assume associated to a program that performs the reconciliation. Such correspondence specifies that the k tuples of values $\vec{\mathbf{x}}_1,\ldots,\vec{\mathbf{x}}_k$ coming from the sources are reconciled to the tuple $\vec{\mathbf{z}}$ in the Data Warehouse. Therefore, the associated program receives as input k tuples of values (and possibly the additional parameters in the condition) and returns a reconciled tuple.

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

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

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

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

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

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

Moreover, we assume that for each Conversion Correspondence $\mathit{convert}_i$ asserted by the designer, the system automatically asserts a new Matching Correspondence match_i as follows:

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

Moreover, for each Conversion Correspondence $\mathit{convert}_i$ asserted by the designer and for each Matching Correspondence match_j asserted by the designer or by default, the system automatically asserts a new

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

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

4.1.1 Example of Interschema Correspondences in the Case Study

The following Conversion Correspondence specifies that the cost of the service call in Source 4 which is expressed in French Franks can be converted in Italian Lire through the program `frank_to_lira`:

$$\begin{array}{rcl} convert_1([F],[L]) & \leftarrow & \texttt{Real}(F) \wedge \texttt{cost\_4}(C,F) \\ & & \mathsf{through}\ \texttt{frank\_to\_lira}(F,L) \end{array}$$

Similarly, the following Conversion Correspondence specifies that the cost of the service call in Source 5, which is expressed in Deutsch marks can be converted in Italian Lire through the program `mark_to_lira`:

$$\begin{array}{rcl} convert_2([M],[L]) & \leftarrow & \texttt{Real}(M) \wedge \texttt{cost\_5}(C,M) \\ & & \mathsf{through}\ \texttt{mark\_to\_lira}(M,L) \end{array}$$

Also dates in Source 4 and Source 5 follow the international format (YYYY/MM/DD), while in the tables of the other sources, as well in the the enterprise model, dates follow the Italian format (DD Mon YYYY), where Mon denote the first three letters of the month's name in Italian. In order to convert dates in Sources 4 and 5 to the Italian format, we use the following Conversion Correspondences:

$$\begin{array}{rcl} convert_3([D],[DI]) & \leftarrow & \texttt{DateValue}(D) \wedge \texttt{date\_4}(C,D) \\ & & \mathsf{through}\ \texttt{italian\_date\_format}(D,DI) \\ convert_4([D],[DI]) & \leftarrow & \texttt{DateValue}(D) \wedge \texttt{date\_5}(C,D) \\ & & \mathsf{through}\ \texttt{italian\_date\_format}(D,DI) \end{array}$$

Finally, the codes for customers, contracts and services in the Sources 1, 2, and 3, are converted to a common coding system which is adopted in the Data Warehouse.

$$\begin{array}{rcl} convert_5([CC],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{customer-code\_1}(C,CC) \\ & & \mathsf{through}\ \texttt{common\_customer\_code\_1}(CC,NC) \\ convert_6([CC],[CCnew]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{tax-code\_2}(C,CC) \\ & & \mathsf{through}\ \texttt{common\_customer\_code\_2}(CC,NC) \\ convert_7([CC],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{ssn\_3}(C,CC) \\ & & \mathsf{through}\ \texttt{common\_customer\_code\_3}(CC,NC) \\ convert_8([CC],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{has-code\_1}(C,CC) \wedge \texttt{Contract\_1}(C) \\ & & \mathsf{through}\ \texttt{common\_contract\_code\_1}(CC,NC) \\ convert_9([CC],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{has-code\_2}(C,CC) \wedge \texttt{Contract\_2}(C) \\ & & \mathsf{through}\ \texttt{common\_contract\_code\_2}(CC,NC) \\ convert_{10}([CC],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{has-code\_3}(C,CC) \wedge \texttt{Contract\_3}(C) \\ & & \mathsf{through}\ \texttt{common\_contract\_code\_3}(CC,NC) \\ convert_{11}([C],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{has-code\_1}(C,CC) \wedge \texttt{Service\_1}(C) \\ & & \mathsf{through}\ \texttt{common\_service\_code\_1}(CC,NC) \\ convert_{12}([C],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{has-code\_2}(C,CC) \wedge \texttt{Service\_2}(C) \\ & & \mathsf{through}\ \texttt{common\_service\_code\_2}(CC,NC) \\ convert_{13}([C],[NC]) & \leftarrow & \texttt{String}(CC) \wedge \texttt{service-code\_3}(C,CC) \\ & & \mathsf{through}\ \texttt{common\_service\_code\_3}(CC,NC) \end{array}$$

Observe that the system asserts automatically several others Conversion Correspondences, Matching Correspondence and the Reconciliation Correspondences according to the rules described above.

4.2 Specification of Mediators

As we said in the introduction, the problem of data integration and reconciliation is crucial for the task of designing the mediators that load the data in the Data Warehouse. Such a task aims at specifying, for every relation in the Data Warehouse Schema, how the tuples of the relation should be constructed from a suitable set of tuples extracted from the sources.

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

1. We specify T in terms of an adorned query

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

   Note that the adorned query associated to a table in a source is the result of a reverse engineering analysis of the source, whereas in this case the adorned query is a specification of what we want to materialize in the table of the Data Warehouse. Note also that we express the semantics of T again in terms of the conceptual model. Not only the sources, but also the relations in the Data Warehouse are seen as views of such a conceptual model.

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

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

   - a table in a source, or
   - a matching, conversion, or reconciliation predicate defined in the Interschema Correspondences.

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

   - Completeness of the rewriting. Obviously, the best situation is the one where the rewriting is complete, in the sense that the rewritten query is equivalent to the original query. Such a check can be done by exploiting the algorithm for query containment.
   - Accuracy, confidence, freshness, and availability of data in the source relations that the rewriting requires to access.

4.2.1 Example of Mediator in the Case Study

Suppose we want to store in the Data Warehouse a relation containing the information about customers that have a contract for a phone-line service, together with possible information on the maintenance tasks requested by the customer for that line.

For this purpose, we can make use of a relational table `TDW` with four components, defined by the following adorned query, where we have assumed that in the Data Warehouse we want to identify customers, services and maintenance tasks by their code, according to what specified in Section 4.1:

$$\begin{array}{rl}\texttt{TDW}(S,I,M) \leftarrow & \texttt{agreement}(X,R,C) \wedge \texttt{Customer}(X) \wedge \texttt{Phone}(R) \wedge \texttt{Contract}(C) \wedge \\ & \texttt{customer-code}(X,S) \wedge \texttt{service-code}(R,I) \wedge \texttt{order}(R,T) \wedge \\ & \texttt{task-code}(T,M) \\ & \vee \\ & \texttt{agreement}(X,R,C) \wedge \texttt{Customer}(X) \wedge \texttt{Phone} \sqcap \neg\exists\texttt{order}(R) \wedge \texttt{Contract}(C) \wedge \\ & \texttt{customer-code}(X,S) \wedge \texttt{service-code}(R,I) \wedge M = \texttt{NULL} \ \mid \\ & \mathit{identify}([S],X),\ S :: \texttt{String}, \mathit{identify}([I],R),\ I :: \texttt{String}, \\ & \mathit{identify}([M],T),\ M :: \texttt{String} \sqcup \texttt{NULL}\end{array}$$

Using the asserted and automatically derived Interschema Correspondences, the system is able to rewrite the above query in terms of the tables of the sources as follows:

$$\begin{array}{l}\texttt{TDW}(S_0,I_0,M_0) \leftarrow \\ \quad \texttt{Agreement-table\_1}(CC_1,TC_1,SC_1) \wedge \texttt{Maintainance-Task-table\_3}(M_0,D,R,E,SC_3) \wedge \\ \quad \mathit{convert}_{11}([SC_1],[SC]) \wedge \mathit{match}_{13}([SC_3],[SC]) \wedge \mathit{convert}_5([CC_1],[S_0]) \wedge \mathit{convert}_{11}([SC_1],[I_0]) \\ \quad \vee \\ \quad \texttt{Agreement-table\_3}(CC_3,TC_3,SC_3) \wedge \texttt{Maintainance-Task-table\_3}(M_0,D,R,E,SC_3) \wedge \\ \quad \mathit{convert}_7([CC_3],[S_0]) \wedge \mathit{convert}_{13}([SC_3],[I_0]) \\ \quad \vee \\ \quad \texttt{Agreement-table\_1}(CC_{11},TC_{11},SC_{11}) \wedge \\ \quad \mathit{convert}_5([CC_{11}],[S_0]) \wedge \mathit{convert}_{11}([SC_{11}],[I_0]) \wedge M_0 = \texttt{NULL} \\ \quad \vee \\ \quad \texttt{Agreement-table\_3}(CC_{31},TC_{31},SC_{31}) \wedge \\ \quad \mathit{convert}_7([CC_{31}],[S_0]) \wedge \mathit{convert}_{13}([SC_3],[I_0]) \wedge M_0 = \texttt{NULL} \\ \quad \textsf{merging} \quad (S_a,I_a,M_a) \textsf{ and } (S_c,I_c,\texttt{NULL}) \\ \quad \textsf{such that} \quad S_a = S_c \wedge I_a = I_c \\ \quad \textsf{into} \quad (S_a,I_a,M_a)\end{array}$$

where match_{13} is the matching corresponding to $\mathit{convert}_{13}$ automatically asserted by the system.

References

[CDGL+97] 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.

[CDGL+99a] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Experimental prototype supporting data reconciliation. Technical Report DWQ-UNIROMA-006, DWQ Consortium, October 1999.

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

[TLN99] Stefano M. Trisolini, Maurizio Lenzerini, and Daniele Nardi. Data integration and warehousing in Telecom Italia. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 538–539, 1999.

Appendix

We include as appendix the publication [TLN99].