Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia

Using Perspective Schema and a Reference Model to Design the ETL Process

Valéria Magalhães Pequeno

Dissertação apresentada para a obtenção do Grau de Doutor em Informática

João Carlos Gomes Moura Pires (Orientador)

Arguentes:
Bernadette Farias Lóscio (UFPE, Brasil) e
Gabriel de Sousa Torcato David (FE/UP, Portugal)
Vogais:
Henrique Madeira (UC, Portugal), Mário J. Gaspar da Silva (FC/UL, Portugal), Salvador Pinto de Abreu (UEVORA, Portugal), José Júlio Alferes (FCT/UNL, Portugal), e João Moura Pires (FCT/UNL, Portugal)

Março de 2011

Using Perspective Schema and a Reference Model to Design the ETL Process

Copyright © 2011, Valéria Magalhães Pequeno, FCT/UNL, UNL. All rights reserved.

"A Faculdade de Ciências e tecnologia e a Universidade Nova de Lisboa têm o direito perpétuo, e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objectivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor."

To *Antonio Junior, my lovely husband,*
Antonio Neto, my little prince, and
Helena, my litlle princess

Acknowledgments

I am deeply thankful to my supervisor, Prof. Dr. João Carlos Gomes Moura Pires, whose a patience and kindness, as well as guidance and academic experience, have been invaluable to me. Without his tutelage this work would never have existed. I would like to extend my thanks to my first supervisor, Prof. Dr. Joaquim Nunes Aparício (in memoriam), for his encouragement, and support in the initial phase of this work.

I would like to show my gratitude to all those who provided valuable comments on the work presented in this thesis. For their assistance through the last years, I would like to thank Profs. José Júlio Alferes and Salvador Abreu (mainly for his kind help in practical matters related to the work). I am grateful to my advisory committee for their suggestions, particularly to Prof. Dr. Bernadette Farias Lóscio. I am indebted to Sally Pethybridge who kindly accepted to read a preliminary version of this work and give me her remarks.

It is a pleasure to thank to all my teachers and professors for knowledge I have acquired, especially to Raimundo Alves Ferreira Filho, Jacinto and Assis Fernandes (in memoriam) who showed me how mathematics can be fun. I owe my deepest gratitude to my master supervisor Vânia Maria Pontes Vidal, who believed in me and gave me the confidence to explore my research interests.

Despite the geographical distance, my family was always nearby. Mom and Dad thank you very much for everything. I am indebted to my grandparents, especially Margela for her unconditional support and Haroldo for showing me the beauty of physics when all my teachers had failed. My most special thanks goes to my best partner and friend, my husband Junior, who gave me his unconditional support and love through all this long process. Thank you very much. I love you. My lovely son and daughter, thank you for your love and patience during the preparation of this thesis.

I would also like to acknowledge FCT (Fundação para a Ciência e a Tecnologia) - Lisboa, Portugal - for giving me a PhD grant for four years, and CENTRIA (Centro de Inteligência Artificial) - Departamento de Informática of Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa - for giving me a PhD grant during six months.

For giving me working conditions, I am very grateful to the Departamento de Informática of Faculdade de Ciências e Tecnologia of Universidade Nova de Lisboa, and its Centro de Inteligência Artificial (CENTRIA). I cannot forget to express my appreciation to the secretariat staff of Departamento de Informática and secretariat staff of Divisão Académica for their kind support and general help.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

Lisboa, Março de 2011
Valéria Magalhães Pequeno

Sumário

Desde há muitos anos que a comunidade de base de dados estuda o problema da integração de dados. Este problema consiste, resumidamente, em combinar os dados armazenados em diferentes fontes de informação, fornecendo uma vista unificada destes dados. O avanço das tecnologias de informação e de comunicação incrementou o interresse na área de integração de dados, uma vez que integrar e compartilhar informações tem surgido como uma exigência estratégica dos negócios actuais.

Ao lidar com o problema de integração de dados, o analista de sistemas geralmente depara-se com modelos de dados incompatíveis entre si, caracterizados por diferenças subtis em termos sintácticos e semânticos. Neste trabalho, nós propomos uma abordagem declarativa baseada na criação de um modelo de referência e de esquemas de perspectivas (no inglês *perspective schemata*) para tornar explícito o relacionamento entre os esquemas. O modelo de referência serve como um meta-modelo semântico comum à organização, enquanto os esquemas de perspectivas definem a correspondência entre esquemas usando asserções de correspondências. A nossa proposta oferece um meio formal e declarativo de definir os modelos existentes, bem como o relacionamento entre eles. O primeiro é conseguido através do uso da linguagem L_S (do inglês *schema language*), a qual é centrada no paradigma objecto-relacional; enquanto o segundo é realizado através da linguagem L_{PS} (do inglês *perspective schema language*).

A abordagem proposta torna claro o mapeamento que existe em um sistema de integração de dados, modulando-os de modo a facilitar a manutenção desses mapeamentos. Além disso, com base na arquitectura proposta, um mecanismo de inferência foi desenvolvido de modo a permitir a derivação semi-automática de novos mapeamentos a partir de outros já definidos. Nós também implementamos uma prova de conceito baseada em Prolog, para demonstrar a viabilidade da abordagem proposta.

Termos chave: integração de dados, mapeamento de esquemas, modelagem conceptual, processo de ETL

Abstract

For many years the database community has been wrestling with the problem of data integration. In short, this problem consists of combining data from different information sources, and providing the user with a unified view of these data. The advances of information and communications technologies have greatly increased the importance of data integration in recent years, since sharing and integrating information has emerged as a strategic requirement in modern business.

In dealing with the integration problem, the designer usually encounters incompatible data models, characterised by differences in structure and semantics. In this work, we propose a declarative approach based on the creation of a reference model and perspective schemata in order to make the relationship between schemata explicit. The reference model serves as a common semantic meta-model, while perspective schemata defines correspondence between schemata using correspondence assertions. Our proposal offers a way to express, in a formal and declarative way, the existing data models and the relationship between them. The former is done using the *schema language* focused on object-relational paradigm, while the latter is done using the language to describe perspective schemata (the *perspective schema language*).

The proposed approach makes the mappings that exist in data integration systems clear, and uncouples them in order to make their maintenance easier. Furthermore, based on the proposed architecture, which considers a reference model and correspondence assertions, an inference mechanism was developed to allow the (semi-) automatic derivation of new mappings from previous ones. A Prolog-based proof-of-concept was implemented in order to demonstrate the feasibility of the proposed approach.

Keywords: data integration, schema mapping, conceptual modelling, ETL process

Contents

Acknowledgments **v**

Sumário **vii**

Abstract **ix**

List of Figures **xvi**

List of Tables **xvii**

List of Acronyms **xix**

1 Introduction **1**

1.1 Context 1

1.2 Motivation 4

1.3 The Proposal 5

1.4 Objectives and Contributions 8

1.5 Thesis Structure 9

2 State of the Art **11**

2.1 Semantic Heterogeneity 13

2.2 Schema Mapping 16

2.3 Schema Matching 17

2.4 Instance Matching 19

2.5 ETL Tools . . . 21

2.6 Other Related Works . . . 24

2.6.1 Handling time in Data Warehouse (DW) systems . . . 24

2.6.2 The view maintenance problem . . . 25

3 Conceptual Modelling 29

3.1 Language L_S . . . 29

3.1.1 Structural aspects . . . 30

3.1.2 Adding behaviour . . . 45

3.1.3 Formal treatment of paths . . . 47

3.2 Comparison with Other Models . . . 49

3.2.1 Comparison with Relational Data Model (RDM) . . . 50

3.2.2 Comparison with Object-Oriented Data Model (OODM) . . . 50

3.2.3 Comparison with Object-Relational Data Model (ORDM) . . . 52

3.2.4 Comparison with commercial ORDMs . . . 55

3.3 Conclusions . . . 56

4 The Language $\mathbf{L}_{PS}$ 59

4.1 A Running Example . . . 59

4.2 Overview . . . 63

4.2.1 “Require” declarations . . . 63

4.2.2 Matching function signatures . . . 64

4.2.3 Correspondence assertions . . . 66

4.3 Correspondence Assertions (CAs) in detail . . . 69

4.3.1 Property Correspondence Assertion (PCA) . . . 69

4.3.2 Extension Correspondence Assertion (ECA) . . . 72

4.3.3 Summation Correspondence Assertion (SCA) 75

4.3.4 Aggregation Correspondence Assertion (ACA) 78

4.4 View relation declaration 80

4.5 Perspective schema 82

4.6 Pragmatic Examples 88

4.6.1 Examples involving semantic heterogeneity 89

4.6.2 Examples involving ECAs of union 93

4.6.3 Examples involving SCAs of group by 98

4.6.4 Examples involving methods and references 98

4.7 Conclusions 100

5 The Inference Mechanism 101

5.1 Rewriting Rules 102

5.2 Inference 110

5.3 Uses of the Inference Mechanism 116

5.4 Inference Mechanism Validations 120

5.5 Case Studies 132

5.5.1 Bank scenario 132

5.5.2 Insurance scenario 135

5.6 Conclusions 138

6 The REMA Proof-of-Concept 141

6.1 Contextual Logic Programming and ISCO 141

6.2 REMA Architecture 143

6.3 Implementation 145

6.4 Case Studies 153

6.4.1 Bank scenario 153
6.4.2 Insurance scenario 154
6.5 Conclusions 156
7 Conclusion and Future Work **157**
7.1 Conclusions 157
7.2 Open Issues and Future Work 159
Bibliography **161**
A Appendix **175**
A.1 Substitution-Rules 176
A.2 Rewritten-Rules to Rewrite CAs 180
A.2.1 Rewritten-rules to rewrite PCAs 181
A.2.2 Rewritten-rules to rewrite ECAs 184
A.2.3 Rewritten-rules to rewrite SCAs 186
A.2.4 Rewritten-rules to rewrite ACAs 188
B Appendix **193**
B.1 Correspondence between the Language L_S and the Prolog Notation 193
B.2 Correspondence between the Language L_{PS} and the Prolog Notation 195

List of Figures

1.1 Proposed architecture. 6

1.2 Examples of correspondence assertions. 7

2.1 Scope of the present investigation. 11

2.2 Areas related to our research. 13

4.1 Motivational example: source schemata $\mathbf{S}_1$ and $\mathbf{S}_2$. 60

4.2 Motivational example: the reference model. 61

4.3 Motivational example: the data warehouse schema. 62

4.4 Perspective schema. 63

4.5 Examples of matching function signatures. 66

4.6 Examples of correspondence assertions. 67

4.7 Predicates and non-predicates. 69

4.8 Examples of property correspondence assertions. 70

4.9 Examples of extension correspondence assertions. 73

5.1 Sketch of the inference mechanism. 102

5.2 Examples of use of the inference mechanism. 116

5.3 Examples of mapping strategies. 118

5.4 Example of inference in two-step. 119

5.5 Inference mechanism used in various Data Integration Systems (DISs). 120

5.6 Examples of Directed Acyclic Graphs (DAGs). 121

5.7 Example of an inference process. 122

5.8 Examples of subgraph of necessary conditions. 123

5.9 Example of a DAG schema graph with two different DAG entity graphs. 125

5.10 Examples of DAG structure graphs, from a same DAG schema graph, and DAG entity graphs. 127

5.11 Other examples of DAG structure graphs (also are presenting DAG schema graphs, and DAG entity graphs). 128

5.12 Part of the schemata present in the bank scenario. 132

5.13 The role of schema **V** when using the traditional approach and when using the proposed framework. 133

5.14 Relationships between some components of schemata **G**, **V**, and $\mathbf{S}_1$. 134

5.15 Mapping using traditional approaches. 136

5.16 Mapping using the proposed framework. 136

6.1 REMA architecture. 144

6.2 Example of a Information Systems COnstruction language (ISCO) class generated from a relation of a schema. 148

6.3 Example of a ISCO class generated from a relation of a perspective schema. . . . 149

6.4 Examples of correspondence assertions. 150

6.5 Another example of a ISCO class generated from a relation of a perspective schema.151

6.6 Example of a SQL view for the static class *sales_by_customerdw*. 151

6.7 Examples of correspondence assertions of relation SALES_BY_CUSTOMER$_{dw}$. 152

7.1 Example of a perspective schema of integration. 160

List of Tables

2.1 Time-varying classes, attributes and methods. 26

3.1 Comparison with RDM terminology. 50

3.2 Comparison with Object Data Management Group (ODMG-3) terminology. . . . 51

3.3 Comparison with the Data and Darwen model. 52

3.4 Comparison of some ORDMs. 55

4.1 Sketch of the pragmatic examples. 89

4.2 Mapping between grades and marks. 90

5.1 Sketch of the inference of CAs. 129

6.1 Examples of error messages and when they are displayed. 146

List of Acronyms

ACA Aggregation Correspondence Assertion

AI Artificial Intelligence

CA Correspondence Assertion

CDBS Component Database System

CxLP Contextual Logic Programming

DAG Directed Acyclic Graph

DBMS Data Base Management System

DIS Data Integration System

DM Data Mart

DW Data Warehouse

ECA Extension Correspondence Assertion

EER Enriched Entity-Relationship

ER Entity-Relationship

ETL Extract Transform Load

FDBS Federated Database System

ISCO Information Systems COnstruction language

KDD Knowledge Discovery in Databases

MAS Multi-Agent System

MF Matching Function

MQL Multidatabase Query Language

ODMG-3 Object Data Management Group

OID Object IDentity

OLAP On-Line Analytical Processing

OO Object-Oriented

OODB Object-Oriented Database

OODM Object-Oriented Data Model

OQL Object Query Language

OR Object-Relational

ORDM Object-Relational Data Model

ORDBMS Object-Relational Database Management System

OWL Web Ontology Language

PCA Property Correspondence Assertion

RDM Relational Data Model

RDBMS Relational Database Management System

SCA Summation Correspondence Assertion

SQL-3 Structured Query Language

TDB Temporal Database

UML Unified Modeling Language

UUID Universal Unique IDentifier

WWW World Wide Web

XML Extensible Markup Language

1

Introduction

Sharing and integrating information among multiple heterogeneous and autonomous databases has emerged as a strategic requirement in modern business. We deal with this problem by proposing a declarative approach based on the creation of a reference model and perspective schemata. The former serves as a common semantic meta-model, while the latter defines correspondence between schemata. Furthermore, using the proposed architecture, we develop an inference mechanism which allows the (semi-) automatic derivation of new mappings between schemata from previous ones.

This chapter introduces set of problems in dealing with data integration between various independent and heterogeneous systems in order to guarantee a unified and consistent view of data across the enterprise. We then describe our motivation for dealing with these problems and for using a declarative approach, followed by the objectives and main contributions of the thesis. Finally, we present the structure of the thesis.

1.1 Context

One of the leading issues in database research is to develop flexible mechanisms for providing integrated access to multiple, distributed, heterogeneous databases and other information systems (some within and some across enterprises). Businesses want to consolidate their databases and therefore need to transfer data from old databases to new ones; they require a multitude of information sources, which are normally produced independently, to share information between them; and sometimes, they also want to access integrated information sources on the World Wide Web.

The aim of Data Integration Systems (DISs) is, in general, to provide a unified and reconciled view of the data residing at the information sources, without affecting their autonomy (Ullman, 1997), in order to facilitate data access and to gain a more comprehensive and coherent basis to satisfy the information need. A wide range of approaches has been developed to address this problem, including approaches based on the creation of virtual integrated views (named *the virtual view approach*) (C. Batini, M. Lenzerini, S.B. Navathe, 1986) and on the creation of materialised integrated views (named *the materialised view approach*) (Zhou *et al.*, 1996). In the first case, the DIS acts as an interface between the user and the information sources: queries submitted to the DIS are divided at run time into queries on the information sources, with data access being transparent to the user. In the second case, information from each information source is extracted in advance, and then translated, filtered, merged and stored in a repository. Thus, when a user's query arrives, it can be evaluated directly at the repository, and no access to the information source is required (Zhou *et al.*, 1996). Therefore, the user's queries can be answered quickly and efficiently since the information is directly available. The problem, however, is that the materialised views must be updated to reflect changes made to information sources. In this work, we do not deal with the problem of maintaining materialised views.

Depending on the purpose of the data integration, different architectures can be used, among which we highlight:

- Federated Database Systems (FDBSs) are distributed systems "consisting of a collection of cooperating but (partly) autonomous and possibly heterogeneous Component Database Systems (CDBSs)" (Türker, 1996). A FDBS enables a unified view and transparent access to originally separated information sources. Basically there are two types of FDBSs: a tightly coupled FDBS, and a loosely coupled FDBS. Tightly coupled FDBSs provide a global schema expressed in a common, "canonical" data model, and global users do not need to know about the CDBSs. Integration in loosely coupled FDBSs is implicitly performed using a uniform Multidatabase Query Language (MQL), such as FraQL in (Sattler & Schallehn, 2001) and MQL in (Kim *et al.*, 2007). In this type of architecture, users have to know about the location of the requested data, but the query language hides technical and language heterogeneity. Usually, FDBSs use the virtual view approach.
- Mediation systems use the mediator architecture (Wiederhold, 1992) to provide integrated and transparent access to multiple information sources, which can be autonomous and

heterogeneous. Functionality of the DIS is divided into two kinds of subsystems: the *wrappers* and the *mediators*. The wrappers provide access to the data in the information sources using a common data model and a common language. The mediators are modules of software that support an integrated view of multiple information sources through a mediated schema (the global schema). Usually, mediation systems use the virtual view approach.

- Data Warehouse (DW) systems are highly specialised database systems that contain the unified history of an enterprise at a suitable level of detail for decision support. All data is normally integrated into a single repository (named *data warehouse*), with a generalised and global schema. The DW acts as a data source for a wide range of applications such as On-Line Analytical Processing (OLAP) and data mining. Although there are several ways to build a DW, we highlight the two major ones: the *top-down* and the *bottom-up* approaches. The top-down approach, also known as *Enterprise Data Warehouse*, was proposed by Inmon in 1996 (Inmon, 1996). In this approach, the DW must first be designed as a single corporate data model, which covers the whole enterprise, followed by data marts[1]. According to Inmon, the DW model must be atomic, slightly denormalised, and contain detailed historical information; since it will serve as data source to several types of applications (such as OLAP tools). The DMs are fed from the DW. Usually, data in the DMs is denormalised, and highly or slightly summarised. Traditional modelling is used in the DW, normally the Entity-Relationship (ER) model, while dimensional modelling is used in the DMs[2]. The bottleneck of this approach is that the design phase is very large, due to the very broad scope, with real benefits (i.e., the analytic queries over the data) taking a long time to be made. In the bottom-up approach, conceived by Kimball (Kimball, 1996; Kimball & Caserta, 2004), first a DM is developed using a dimensional modelling, then other DMs are incrementally developed. The data marts are guided by subject and built in such a way that they can connect with each other, therefore ensuring that different data marts can be generated in a safe and sound way (Kimball named it *data warehouse bus architecture*). Thus, the problem of the creation of the DW system is divided in chunks and incrementally and gradually resolved. This approach allows analytic queries to be returned as quickly as the first DM can be created.

[1] The Data Marts (DMs) are a departmentalised set of data, which are specifically suited to help in specific demands of a particular group of users.

[2] Dimensional modelling is an approach used in DW design, which is oriented around understandability and performance, rather than transaction-oriented (as occurs with, for example, an ER model). See (Kimball & Caserta, 2004) for more explanation about this subject.

1.2 Motivation

In DISs, the designer, independently of which data architecture (s)he is using (DW, mediator, or FDBS), usually deals with incompatible data models, characterised by subtle differences in structure and semantics, and should define mappings between the global and information source schemata. Normally, Extract Transform Load (ETL) tools are used in order to create these maps and build the DIS. Specifically, several tools can be used to carry out, for example, the following tasks:

1. to identify relevant information at the source systems;
2. to extract this information;
3. to enforce data quality and consistency standards;
4. to conform the data so that separate sources can be used together;
5. to deliver data prepared earlier into the target database.

These tools, although they have mostly graphical user-friendly interfaces, generate several codes. These codes hide knowledge of procedures and policies used to create the mappings, and are strongly dependent on experts and technicians. As the mapping between schemata is not explicit, it cannot be reused. This means, for instance, that if the designer has to define two maps: one from a class A to another class B, and the other from class B to class C and wants a mapping from A to C directly, (s)he has to create this map manually (i.e., the new mappings cannot be deduced based on previous ones).

In addition, ETL tools are very costly, people's environments are complex, the tools are not always prepared to deal with the nuances of the customer's legacy environments, there is no standard to draw models or to describe data, and each tool manages metadata differently. The other difficulty is that the data itself and the business rules[3] evolve requiring the code generated by the ETL tools to be modified and maintained properly.

[3]Business rules define aspects of the business (Business Rules Team, 2000). They can focus on, for example, access control issues (e.g., teachers only can modify student grades in courses in which they are instructors), and business calculations (e.g., convert between numeric grades and letter grades).

1.3 The Proposal

In order to deal with the problems stated in the previous Section, we propose to take a declarative approach based on the creation of a reference model and perspective schemata. A reference model (also known as *conceptual model*, *business data model*, or *enterprise data model*) is an abstract framework that represents the information used in an enterprise from a business viewpoint. It provides a common semantics that can be used to guide the development of other models and help with data consistency (Imhoff *et al.*, 2003). It also serves as a basis for multiple products such as application systems, DWs, and FDBSs (Geiger, 2009; Imhoff *et al.*, 2003), being a more stable basis for identifying information requirements to the DW systems than user query requirements, which are unpredictable and subject to frequent change (Moody & Kortink, 2000). The use of a reference model is not a new approach. It is advocated in DW systems (Imhoff *et al.*, 2003; Geiger, 2009; Moody & Kortink, 2000; Inmon *et al.*, 2001; Inmon *et al.*, 2008). Nevertheless, it is important to note that, when a reference model is used, it is essentially methodological and is not computable, serving only as a reference and guide proposal. The novelty of our proposal is that the reference model is formal and plays a more active role.

A perspective schema describes a data model, in part or completely (*the target*), in terms of other data models (*the base*). It represents the mapping from one or more schemata (e.g., the information sources) to another (e.g., the reference model). In the proposed approach, the relationship between the base and the target is made explicitly and declaratively through Correspondence Assertions (CAs). By using the perspective schemata the designer has a formal representation, with well defined semantics, which allows for: a) definition of diverse points of views of the (same or different) information source; b) dealing with semantic heterogeneity in a declarative way; and c) reuse (i.e, the same perspective schema can be used simultaneously in several systems: application, DW, FDBS, etc.).

We propose, as a general rule, that the relevant parts (those of interest to the DIS) of all schemata are only aligned to the reference model through perspective schemata, and that any other mapping (e.g., the direct mapping between the information sources and the global schema) is automatically (or semi-automatically) created using an inference mechanism. Figure 1.1 illustrates the basic components of the proposed architecture and their relationships. The schemata **RM**, **G**, $\mathbf{S}_1$,...,$\mathbf{S}_n$ represent, respectively, the reference model, the global schema, and the source schemata $\mathbf{S}_1$,...,$\mathbf{S}_n$. The relationship between the reference model and the other schemata is shown through the perspective schemata $\mathbf{P}_{s1|RM}$, $\mathbf{P}_{s2|RM}$, ...,

$\mathbf{P}_{sn|RM}$,$\mathbf{P}_{RM|G}$ (denoted by the solid arrows). Once the perspective schemata $\mathbf{P}_{s1|RM}$, $\mathbf{P}_{s2|RM}$, ..., $\mathbf{P}_{sn|RM}$,$\mathbf{P}_{RM|G}$ are declared, a new perspective schema ($\mathbf{P}_{s1,s2,...,sn|G}$) between the global schema and the source schemata (designed by a dotted arrow) can be deduced. This inferred perspective schema shows the direct relationship between the global schema and its information sources, and can be used to support the generation of code required by the ETL process in order to load and store data in the global schema. All schemata (including the perspective ones) are stored in a metadata repository.

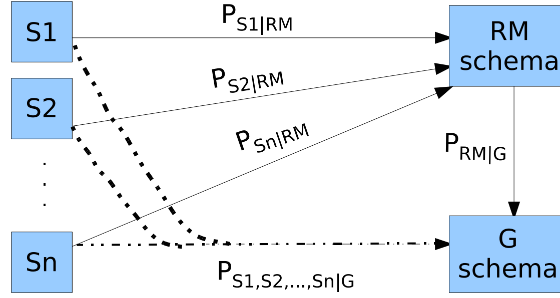


Figure 1.1: Proposed architecture.

Our proposal offers a way to express the existing data models (source schemata, reference model, and global schema) and the relationship between them. The approach is based on *Schema language* (L_S) and *Perspective Schema language* (L_{PS}).

The language L_S is used to describe the actual data models (source, reference model, and global schema). The formal framework focuses on an object-relational paradigm, which includes definitions adopted by the main concepts of object and relational models as they are widely accepted (Codd, 1970; Cattell *et al.*, 2000). In Figure 1.1, the schemata **RM**, $\mathbf{S}_1$,...,$\mathbf{S}_n$, and **G** are defined using the language L_S.

The language L_{PS} is used to describe *perspective schemata*. In Figure 1.1, for instance, $P_{s1|RM}$, $P_{s2|RM}$, ..., $\mathbf{P}_{sn|RM}$ are perspective schemata that map the reference model (**RM**) in terms of, respectively, the sources ($\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, ..., $\mathbf{S}_n$). L_{PS} mainly extends L_S with two components: Matching Function (MF) signatures and Correspondence Assertions (CAs). MF signatures indicate when two data entities represent the same instance of the real world. L_{PS} includes data transformations, such as name conversion and data type conversion. CAs formally specify the relationship between schema components.

CAs are classified in four groups: Property Correspondence Assertion (PCA), Extension Correspondence Assertion (ECA), Summation Correspondence Assertion (SCA), and

Aggregation Correspondence Assertion (ACA). Property correspondence assertions relate properties of a target to the properties of some schema in the base (called *base schema*). Extension correspondence assertions are used to describe which objects/tuples of a base schema should have a corresponding semantically equivalent object/tuple in the target. Summation correspondence assertions are used to describe the summary of a class/relation whose instances are related to the instances of another class/relation by breaking them down into logical groups that belong together. They are used to indicate that the relationship between the classes/relations involve some type of aggregate functions or a normalisation process. For example, a SCA is used when daily sales are mapped to monthly sales by region. Aggregation correspondence assertions link properties of the target to the properties of the base schema when a SCA is used. Two examples of CAs are shown in Figure 1.2.

ψ_1: $\mathsf{P_{RM|G}}$ [CUSTOMER] $\rightarrow$ RM [CONSUMER] (ECA)

ψ_2: $\mathsf{P_{RM|G}}$ [CUSTOMER] $\bullet$ **cust_id** $\rightarrow$ RM [CONSUMER] $\bullet$ **cons_id** (PCA)

Figure 1.2: Examples of correspondence assertions.

Correspondence assertion ψ_1 is an ECA and indicates that the relation CUSTOMER (in the target schema) is *mapped from* the relation CONSUMER (in the base schema). In this case, ψ_1 defines that the relation CUSTOMER is semantically equivalent to the relation CONSUMER. This means that for each instance $\mathbf{o}$ in CONSUMER, there is a correspondent instance $\mathbf{o}'$ in CUSTOMER such that $\mathbf{o}$ and $\mathbf{o}'$ represent the same entity in the real world. The CA ψ_2 defines the relationship between the property **cust_id** of the relation CUSTOMER and the property **cons_id** of the relation CONSUMER (i.e., it indicates that **cust_id** is mapped from **cons_id**).

Essentially, our proposal has the following positive features:

1. We propose a declarative language (the L_{LP}) to define mappings between schemata, instead of having just a code generated to move data from one schema to another. By using the language L_{PS} the mappings are explicit, reusable, and easy to maintain. Changes in source schemata affect only the mapping in which they are present, and changes in the target schema affect only the mapping in which it is present.

2. We propose that the reference model be formal, instead of just methodological. Even when the reference model is and is not computable, there are various benefits by using it. For example, it helps with the project scope definition once the designer can use the reference

model to identify the information that will be addressed by the systems. Moreover, the reference model is probably more stable and well documented than any other schema of the enterprise. Once the reference model is computable it is easier to maintain and the knowledge acquisition itself is facilitated.

3. We suggest that the mapping be defined to the reference model and from the reference model, rather than just between the information sources and the target schema. In this context, the use of the reference model simplifies the definition of mappings, since the designer just has to know about a model and part of the reference model, instead of two models (the source and the target) as is usually done to define the mappings. In addition, it is undoubtedly easier to define mappings between the global schema and the reference model, than to consider data integration aspects of the information sources again.

4. We propose an inference mechanism that almost automatically allows inferring new mappings based on previous ones. Specifically, it can generate the direct mapping between the information sources and the global schema, using the mappings defined to the reference model and from the reference model.

1.4 Objectives and Contributions

The objective of this thesis is to propose a way to facilitate the design of DISs using a declarative approach. In order to fulfil this main aim, the following objectives must be achieved:

- To make the relationship between information sources and the global schema clear, at a structural and instance level.
- To propose mechanisms that help the designer to maintain and re-use the mappings already defined.

In this thesis we propose a framework based on the creation of a reference model and perspective schemata, and develop an inference mechanism to (semi-) automatise the definition of direct mapping between the information sources and the global schema. The significance and innovation of this approach is as follows:

- Proposal of a formal and declarative language to define mappings between schemata. The proposed approach makes the mappings that there are in a DIS clear, and uncouples them in order to make their maintenance easier.

- Proposal of an architecture centred on the creation of a reference model, with a base in which an inference mechanism was developed to allow the (semi-) automatic derivation of new mappings.

In this thesis, we also propose a formal conceptual object-relational model for the modelling of the several existing schemata in the DISs, which intends to be general enough to allow the designing of relational, object-oriented and object-relational data models. A Prolog-based proof-of-concept was implemented in order to demonstrate the feasibility of the proposed approach.

This work does not deal with the loading phase of the ETL process. In this phase, the data from the information sources are loaded and stored in a global schema. We consider it out of the scope of this work.

1.5 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 is devoted to the state of art, where we focus on semantic mapping between schemata. Here, we discuss semantic conflicts, schema mapping, schema matching, and instance matching. We also outline some ETL tools, and then discuss other related works which are relevant in the context of a continuation of our work: temporal aspects in data models and data maintenance.

Chapter 3 shows the formal language to define schemata, which is rich enough to model schemata drawn in relational, object-relational, or object-oriented domains.

Chapter 4 presents the formal language to define perspective schemata, which allows the making of the mappings between schemata and deals with various usual types of semantic conflicts.

Chapter 5 details the process for inferring the new perspective schemata, and so creating new mappings based on the previous ones. Few case studies are analysed here.

Chapter 6 shows an implementation based on contextual logic programming, which serves as a proof-of-concept for demonstrating the feasibility of our proposal when: i) using perspective schemata to define the mappings between schemata; ii) using the inference mechanism to generate new mappings from previous ones. The implementation was used to construct information systems (as proof-of-concepts) that can transparently access data from various heterogeneous information sources in a uniform way, like a mediation system. The case studies introduced in Chapter 5 were implemented, and the more important results are explained here.

Chapter 7 provides the summary of the thesis, which points out the new features of the approach presented here and for future planned works in this area.

Also, in the thesis there are two appendices. Appendix A contains all the rules of the proposed inference mechanism, while Appendix B contains the correspondence between our formalism and the Prolog notation.

We have compared our research with other throughout the Thesis. Specifically, the reader can see some comparisons regarding the proposal in Chapters 4 and 7, and comparisons concerning the language L_S in Chapter 4.

2

State of the Art

This Chapter provides a brief overview of the scope of the current investigation and presents some related works.

For many years the database community has been wrestling with the problem of data integration (Stumptner *et al.*, 2004; Vidal *et al.*, 2001; T.Risch *et al.*, 2003; Risch & Josifovski, 2001; Rizzi & Saltarelli, 2003; Golfarelli *et al.*, 2004). Research on this area has developed in several important directions (see Figure 2.1), such as:

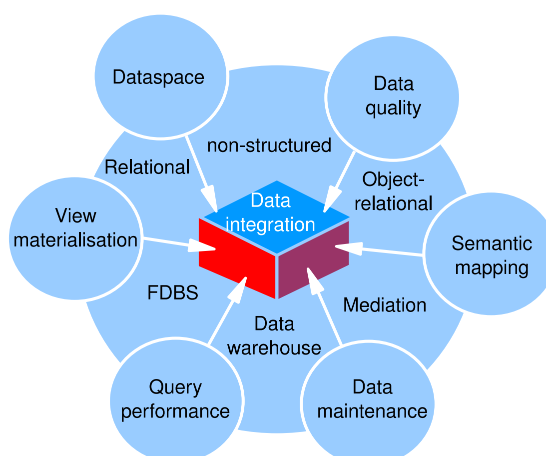


Figure 2.1: Scope of the present investigation.

- Semantic mapping (e.g., (Cali *et al.*, 2004; Fagin *et al.*, 2005; Peukert *et al.*, 2011; Shvaiko & Euzenat, 2008; Nottelmann & Straccia, 2007; Chiticariu *et al.*, 2007));
- Data maintenance (e.g., (Eder & Wiggisser, 2010; Agner, 2005));

- Query performance (e.g., (Duschka *et al.*, 2000; Petrini & Risch, 2007; Chen *et al.*, 2008));
- View materialisation (e.g., (Kumar *et al.*, 2010; Saeki *et al.*, 2007; Wrembel, 2000; Zhou *et al.*, 1996));
- Dataspaces (e.g., (Khalid *et al.*, 2010; Franklin *et al.*, 2008)); and
- Data quality (e.g., (Ganesh *et al.*, 1996; Bollacker *et al.*, 1998; Bilenko & Mooney, 2002; Gravano *et al.*, 2003; Sarawagi & Bhamidipaty, 2002)).

The investigations cover different architectures, such as FDBSs, DW systems and mediation systems, and different representation of data and associated data models (e.g., relational and non-structured). A survey can be found in (Halevy *et al.*, 2006). Specifically, recent research covering the different architectures has included:

- **FDBSs**: behaviour integration (Stumptner *et al.*, 2004), intelligence data (Yoakum-Stover & Malyuta, 2008), and interactive integration of data (Ives *et al.*, 2009; Mccann *et al.*, 2003);
- **mediation systems**: peer data management systems (Ives *et al.*, 2004; Gardarin *et al.*, 2008; T.Risch *et al.*, 2003), build wrappers (Schreiter, 2007; J.Wislicki *et al.*, 2008), and integration and/or sharing of non-conventional data (non-structured, multimedia, medical, biological, etc.) (Schreiter, 2007; Beneventano *et al.*, 2001; Sujansky, 2001; Haas *et al.*, 2001; Xu *et al.*, 2000);
- **DW systems**: design (Dias *et al.*, 2008; Malinowski & Zimányi, 2006; Haselmann *et al.*, 2007), data maintenance (Eder & Wiggisser, 2010; Agner, 2005), and federated data warehouse systems (Berger & Schrefl, 2008).

The present research is centred on semantic mapping, one of the major bottlenecks in building DISs. In this context, we devoted our attention to the following areas (see Figure 2.2): semantic heterogeneity (also known as semantic conflict), schema mapping, schema matching, instance matching, and ETL tools (only those that deal with data transformation and mapping), just because we think they are closest to our research. We consider works in DISs that define a global schema, such as in DW systems, mediation systems, and some types of FDBSs, specifically those covering relational, Object-Oriented (OO) and Object-Relational (OR) data. These are the subject of Sections 2.1 to 2.5 respectively.

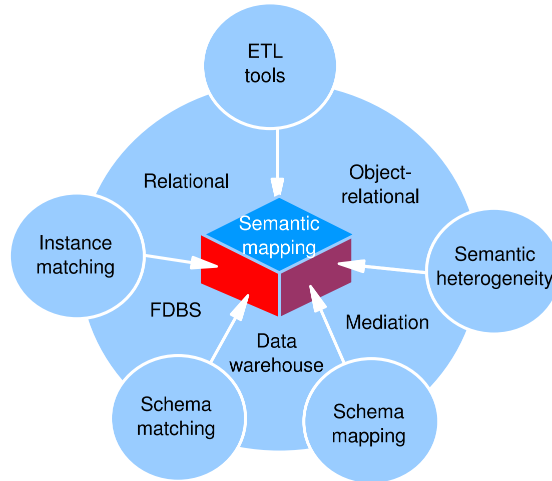


Figure 2.2: Areas related to our research.

Less important to the current work, but still relevant in the context of a continuation of our study, are the subjects of: handling time in DW systems and the view maintenance problem. Both are briefly discussed in the Section "Other related works".

2.1 Semantic Heterogeneity

With the growing request for access to integrated data from a multitude of heterogeneous information sources (sometimes across the World Wide Web), the treatment of difference between the structure and the semantic of the data plays an important role in DISs. Semantic heterogeneity "refers to the conflict caused by using different ways in heterogeneous systems to express the same entity in reality" (Han & Qing-zhong, 2004). They can be distinguished through data conflict and schematic conflict. Both classes of conflicts were enumerated and classified in works, such as (Han & Qing-zhong, 2004; Sheth & Kashyap, 1993). In the following text, we show some types of conflicts presented in (Sheth & Kashyap, 1993)(due to considering it more consistent), which will be useful to better understand the examples that will be shown in Chapter 4.

Data conflict

Data conflict occurs when there are different perceptions of the same concepts by different designers. In (Sheth & Kashyap, 1993), the authors present the following data conflicts (referred to as *domain incompatibility problems*):

- *Naming conflict*: concerns the meaning of the concepts (here referred to as properties). There are two types of naming conflicts: *synonyms* and *homonyms*. The former occurs when two properties (or attributes in the relational domain) with different names have the same domain, while the latter occurs when two properties with same names have different domains. As example of synonym, we could mention the name of a student, which can be presented by the property **name** in a relation, and by **student_name** in another. In the case of homonyms, we could have the property **book**, which in one relation can be a book title, and hotel booking in another.

- *Data representation conflict*: occurs when two semantically equivalent properties have different data types or representations. For example, an address can be represented by a string containing the street and the number in one relation, and by a tuple (using the relational notation) formed by two components: **street** and **number** in another.

- *Data scaling conflict* (also known as *data unit conflict* in (Han & Qing-zhong, 2004)): occurs when two semantically equivalent properties are represented using different units or measures. There is one-to-one mapping between the values of the domain of both properties. For example, the weight of a person can be in pounds in one system and in kilograms in another.

- *Data precision conflict*: occurs when two semantically equivalent properties are represented using different precision. In this case, there is no one-to-one mapping between the values of the domain of both properties. For example, the subject grades can be represented using values from 1 to 100 in one system, and values from A to F in another. In this case, the best thing we can do is a mapping of one-to-n between values from A to F and values from 1 to 100.

- *Default value conflict*: occurs when two semantically equivalent properties have different default values.

- *Attribute integrity constraint conflict*: occurs when two semantically equivalent properties are restricted by constraints, which are not consistent with each other. For example, the property **age** in one entity can have the constraint: **age** ≤ 18, and in another the constraint: **age** ≥ 21, which are inconsistent each other.

Schematic conflict

Schematic conflict occurs when there are different logical structures for the same concept. In (Sheth & Kashyap, 1993), the authors present the following schematic conflicts (referred to as *entity definition incompatibility problem*):

- *Database identifier conflict* (also known as *entity identifier conflict* in (Han & Qing-zhong, 2004)): caused when there are different identifiers assigned to the same concept in different systems. For example, the identifier of a student in one system is his/her identity card and in another is his/her name.

- *Naming conflict*: relates to the meaning of the concepts (here referred to relations/classes). We can have two type of naming conflicts: *synonyms* and *homonyms*. The former occurs when two classes or relations describing the same set of instances are named differently, while the latter occurs when two classes or relations with an unrelated set of instances have the same names. Employees stored in the relation EMPLOYEE in one system and STAFF in another is an example of synonyms. An example of a homonym is the relation PEOPLE, that can represent students in one system and can represent customers in another.

- *Union compatibility conflict*: occurs when two classes or relations with a set of semantically equivalent instances are not union compatible with each other. Two relations (or classes) are union incompatible when there is not one-to-one mapping between the set of their properties. For example, two systems could store different information about students: one could keep his/her name and grade, and the other could keep his/her name and address.

- *Schema isomorphism conflict*: occurs when two classes or relations with a set of semantically equivalent instances have different numbers of properties. For example, consider two systems that store information about students. One could keep his/her name and grade, and the other could keep his/her firstname, lastname, and grade.

- *Missing data item conflict*: occurs when two classes or relations with a set of semantically equivalent instances have "missing" information. For example, consider two systems that store information about students. One could keep the graduate students in a relation/class, and the under-graduate students in another; while the other system could keep all students together in a single relation/class with the property **student_type** to distinguish between

graduate and under-graduate students. In the first system, the only way to distinguish between graduate and under-graduate students is by them being in a different relation (the property **student_type** or something similar is missing).

The designer is often confronted with these and other types of conflicts when doing the mappings between schemata. This is the subject of the following Section.

2.2 Schema Mapping

Schema mapping is a high-level specification that describes the relationship between two or more schemata. "An extensive investigation of the foundations of schema mappings has been carried out in recent years" (Fagin *et al.*, 2008), such as (Atay *et al.*, 2010; Amano *et al.*, 2010; Fagin & Nash, 2010; Arenas & Libkin, 2008; Bernstein, 2003; Cali *et al.*, 2004; Fagin *et al.*, 2005). More closely related to ours are works that deal with mapping between the schema's components at a conceptual level, such as in (Ravat & Teste, 2000; Wrembel, 2000; Vidal *et al.*, 2001; Calvanese *et al.*, 2006). Ravat and Teste in (Ravat & Teste, 2000) focused on handling complex and temporal data using an OO data model for DW systems. They dealt with detailed and summarised data evolutions in a clear manner. However, they mentioned nothing about how they dealt with semantic heterogeneity. Also, they assumed that all information sources have object identifiers, which are used to connect the information sources (referred to as *origin*) to the global schema. This may not be a valid assumption since, even nowadays, most parts of the information sources are relational.

Wrembel in (Wrembel, 2000) also focused on DW systems. He considers the DW schema as an OO view schema, and focused on propagating modifications from objects (from information sources) to their materialised counterparts in the view schema. His approach consists of: 1) definition of an OO view schema; 2) development of data structures to establish the relationship between the schema's components (referred to as *CMS*) and the correspondence between instances (referred to as *OMS*). These structures are used for data movement, and not to make the correspondence between the schema's components explicit. The information about structures and how they deal with semantic heterogeneity is dispersed into Object Query Language (OQL) select command and conversion functions.

Calvanese and others in (Calvanese *et al.*, 2006) present a framework that was adopted in a *Data Warehouse Quality* project. Similar to ours, their work provides an active role for the

reference model (referred to as *enterprise model*). All schemata in their proposal, including the global one, are formed by relational structures, which are defined as views (queries) over the reference model. Using inference techniques on queries, they can discover when a set of tuples in a query is contained in, or disjointed from, a set of tuples of another query. Also, they can discover if, for a given query over the reference model, there is a database satisfying the reference model. This is not the role of a reference model in our work, as the Reader will see later. Their proposal also provides the user with various levels of abstraction: conceptual, logical, and physical. In their conceptual level, they introduce the notion of intermodel assertions that precisely capture the structure of an Enriched Entity-Relationship (EER) schema, or allow for specifying the relationship between diverse schemata. According to Calvanese and others in (Calvanese *et al.*, 1998), they dealt with the resolution of schematic and data conflicts such as synonym and homonym. However, different to our work, where everything is defined at a conceptual level, all data transformations are deferred to the logical level in their work.

Vidal and others in (Vidal *et al.*, 2001), similarly to us, define correspondence assertions. In this case, they are used to specify the semantics of mediators developed in XML. Their Correspondence Assertions (CAs) only deal with part of the semantic correspondence managed here. Furthermore, they assume that there is a universal key to determine when two distinct objects are the same entity in the real-world, which is often an unreal supposition. We have focused on making the relationship between schemata explicit (in terms of both structure and instance) pointing to situations that should be considered in a data integration context.

Writing the mappings (and managing them) is not an easy task. It requires expert knowledge (to express the mappings in a formal language) and business knowledge (in order to understand the meaning of the schemata being mapped). Hence, many researchers have focused on semi-automatically generating schema mappings, which is the subject of the following Section.

2.3 Schema Matching

Schema matching is the process of automatically identifying the semantic correspondence between schemas' components. In order to do this, the proposed techniques (such as in (Peukert *et al.*, 2011; Shvaiko & Euzenat, 2008; Nottelmann & Straccia, 2007; Chiticariu *et al.*, 2007)) exploit several kinds of information, including schema characteristics, background knowledge from dictionaries and thesauri, and characteristics of instances.

In current work the relationship between the schema's components is done manually, but there is a lot of research that proposes to undertake schema matching in a (semi-) automatic way. Salguero and others in (Salguero *et al.*, 2008), for example, used ontologies as a common data model to deal with the data integration problem. They extended Web Ontology Language (OWL) with temporal and spatial elements (they called STOWL), and used the annotation properties of OWL to store metadata on the temporal features of information sources, as well as information about data granularity. Their approach semi-automatically relates schema's components using STOWL features of reasoning, but they did not show in (Salguero *et al.*, 2008) how the correspondence is generated and whether the mappings are clear and can be reused. Skoutas and Simitsis in (Skoutas & Simitsis, 2006; Skoutas & Simitsis, 2007) also focused on an ontology-based approach to determine the mapping between attributes from the information sources to the global schema in a DW system, and to identify the ETL transformations required for correctly loading and storing data from information sources to the DW. Their ontology, based on a common vocabulary as well as on a set of data annotations, allows a formal and explicit description of the semantic of the sources and the global schemata. Based on this ontology and the annotated graphs, automated reasoning techniques were used to infer correspondences and conflicts between schemata.

Comprehensive surveys of automatic schema matching approaches are shown in (Rahm & Bernstein, 2001; Doan & Halevy, 2005; Kalfoglou & Schorlemmer, 2003; Choi *et al.*, 2006; Saleem & Bellahsene, 2007). We can group the approaches into two categories: those that typically exploit schema information, such as component names, data types, and schema structures (Do, 2006; Do & Rahm, 2002; Zhang *et al.*, 2006; Giunchiglia *et al.*, 2005); and those that exploit instance data (Bilke & Naumann, 2005; Nottelmann & Straccia, 2007; Chiticariu *et al.*, 2007; Gomes *et al.*, 2009). Few approaches try to combine both schema- and instance-based techniques (Drumm *et al.*, 2007; Dhamankar *et al.*, 2004; Doan *et al.*, 2001; Xu & Embley, 2003). Note that the suitability of instance-based approaches depends on the accessibility of representative instance data. In general, DW designers have no access to data in information sources, and so, instance-based schema matching has very restricted use in DW systems.

It is important to note that, although manual schema matching is usually a time-consuming and tedious process, in general full automatic schema matching cannot be done, primarily due to missing or diverging information (opaque names, unknown synonyms, etc.) about semantics of the involved schemas, which gives rise to many false positives, especially for large systems. Besides, most parts of tools for schema matching are application-specific. Various

tools have been developed to determine schema matches (semi-) automatically (Wang *et al.*, 2007; Doan *et al.*, 2003a; Bernstein *et al.*, 2006), some of which match two large schemas: COMA++(Do, 2006), PROTOPLASM (Bernstein *et al.*, 2004), CLIO (Chiticariu *et al.*, 2007), and OntoMatch (Bhattacharjee & Jamil, 2009). Only COMA++ and OntoMatch are generic matchers (i.e., they were not developed to a specific application). We believe that some of these approaches or tools can be used together with our proposal in order to help the designer to specify the correspondence assertions. However, it should be investigated in the next stage of the research.

Besides schema matching, the problem of instance matching is also becoming increasingly essential. Businesses want to share information from different information sources, and in order to do this, data that represents the same instance of the real world needs to be integrated. This is the focus of the instance matching problem, and is described in more detail in the following Section.

2.4 Instance Matching

In a data integration environment, there is a need to combine information from the same or different sources. This involves comparing the instances from the sources and attempts to determine when two instances of different schemata refer to the same real-world entity (the *instance matching* problem). Variants of this problem are known as *field matching* (Monge & Elkan, 1996), *merge/purge* problem (Hernandez & Stolfo, 1995), *object matching* (Doan *et al.*, 2003b; Zhou *et al.*, 1996), *record linkage* (Michalowski *et al.*, 2003), *data matching* (Doan & Halevy, 2004), *tuple matching* (Doan & Halevy, 2004), and *object fusion* (Papakonstantinou *et al.*, 1996), among others.

The management of instance matching is inevitable for making data integration possible. Thus, it plays a central role in many information contexts, including data warehousing. Unfortunately, instance matching is usually expensive to compute due to the complex structure of the schemata and the character of the data. So, the identification of instance matching can require complex tests to identify the equivalence between instances. "The inference that two data items represent the same domain entity may depend upon considerable statistical, logical, and empirical knowledge of the task domain" (Hernandez & Stolfo, 1995).

The instance matching problem has received much attention in the database, Artificial Intelligence (AI), Knowledge Discovery in Databases (KDD), and World Wide Web (WWW) communities, mainly covering complex scenarios, in order to improve the quality of data. Thus, they consider the use of techniques of identification of instances between multiple sources for removing duplicates and for correcting format inconsistencies in the data (e.g. (Ganesh *et al.*, 1996; Angluin, 1988; Arens *et al.*, 1993; Bickel, 1987; Bollacker *et al.*, 1998; Bilenko & Mooney, 2002; Gravano *et al.*, 2003; Sarawagi & Bhamidipaty, 2002)). It is not part of the current research to deal with the full instance matching problem. We assume, as DW designers usually do, that data quality tools were used and that, for example, duplicates were removed. Even then, in a data integration context, it is essential to provide a way to identify instances of different models that represent the same entity in the real-world in order to combine them appropriately.

There are many works in literature that address the instance matching problem in the desired context. Most systems assume that a universal key is available for performing the instance matching, or at least they suppose that the objects (or tuples) share the same set of properties. In this case, they match objects (or tuples) by comparing property similarity between the shared properties. For example, suppose that there are two relations PURCHASER and CUSTOMER each one having the property **idcard**, which are the primary keys of its respective relations. Also suppose that the properties have the same content (the identity card number of each customer), and the same data type. Thus, it is trivial to carry out the instance matching, simply by comparing the value of these properties. The works in (Hernandez & Stolfo, 1995; Tejada *et al.*, 2002; Monge & Elkan, 1996; Cohen & Richman, 2002), for example, use this approach. Even though this approach is largely used, it is not easy to implement due to, among other things, the semantic heterogeneity that exists between the several schemata and the heterogeneity of data types.

Another common way of dealing with instance matching uses look-up tables. In this technique the types of the classes (or relations) can be unconnected, but there is a look-up table that holds matching information that determines the matching between the objects (or tuples). For example, a look-up table can be provided to determine the equivalence of products from the two relations $\mathbf{S}_1$.PRODUCT and $\mathbf{S}_2$.PRODUCT_SALES. This approach requires that the equivalence between the instances be identified and managed, and is also subject to errors as the table is usually not automatically updated. Look-up tables are used, for example, in (Zhou *et al.*, 1996; Widjojo *et al.*, 1990; Kent *et al.*, 1993).

Works in (Zhou *et al.*, 1996; Widjojo *et al.*, 1990; Kent *et al.*, 1993) also consider the problem of instance matching in contexts where universal keys are not available. For example, (Zhou *et al.*, 1996) deals with user-defined Boolean functions to determine the equivalence between the objects. This function receives some input arguments and returns true or false. For example, two people, a customer and an employee, are the same person even if their name has been missspelt. A matching function can be specified to determine the "closeness" of the names, which takes two names as arguments and returns a Boolean value.

Works in (Doan *et al.*, 2003b; Michalowski *et al.*, 2003) exploit external information, for example: constraints, negative keys, and past matching, to maximise matching accuracy. It means that they focus on maximising the number of correct matches while minimising the number of false positives. For example, "negative keys" is information about objects (or tuples) that intuitively helps in deciding if two objects do not match. "They can be stated and implemented as constraints" (Widjojo *et al.*, 1990). For example, "no person has more than one social security number". It implies that if two persons with identical (or similar) names are found with different social security numbers, then they are two distinct persons.

Some solutions concerning instance matching have been developed based on rules (e.g., (Hernandez & Stolfo, 1995)), others in learning (e.g., (Tejada *et al.*, 2002; Cohen & Richman, 2002; Tejada, 2002)). Also there are solutions which focus on specific contexts, as to match strings (Monge & Elkan, 1996; Hylton, 1996), or focus on more complex domains, as in (Sivic & Zisserman, 2003; Torres, 2004).

When we work with data integration, we always have to deal with the instance matching problem. We identify the situations in which this problem appear in a data integration context, and propose matching function signatures in order to explicit them. However, we do not provide a global framework to implement the matching functions because this problem is better addressed when the application domain is observed.

The following Section focuses on state of the art for the ETL domain.

2.5 ETL Tools

Nowadays, there is a plethora of commercial ETL tools in the market place, with very few of them having been developed from academic research. Most of the tools suggest reduced support at the conceptual level.

In the academic area, ETL research for the DW environment is focused mainly on the process modelling concepts, data extraction and transformation, and cleaning frameworks (Luján-Mora *et al.*, 2004; Raman & Hellerstein, 2001; Albrecht & Naumann, 2008). So far, we are not aware of any research that precisely deals with both mappings (structural and instance) between the source schemata and the global schema, and with the problem of semantic heterogeneity at a whole conceptual level. There are a few prototypes, which are usually implemented to perform technical demonstration and validation of the developed research work (Vassiliadis *et al.*, 2001; Raman & Hellerstein, 2001). As an example of works that developed some implementation, we quote (Calvanese *et al.*, 2006; Skoutas & Simitsis, 2007; Boussaid *et al.*, 2008). We already talked about the two first works in the previous Section. However, from prototype viewpoint, reference in (Calvanese *et al.*, 2006) presents a methodology that was applied in the TELECOM ITALIA framework. The conceptual schemata are described using an external conceptual modelling tool, while their prototype focused on logical schemata and on data movement. Any transformation (e.g., restructuring of schema and values) or mapping of instances was deferred for the logical level. In our approach everything stays at the conceptual level. Skoutas and Simitsis in (Skoutas & Simitsis, 2007) used a graph-based representation to define the schemata (source and DW) and an ontology described in OWL-DL. Reference (Boussaid *et al.*, 2008) proposes a Multi-Agent System (MAS)-based prototype to integrate structured and unstructured data (named *complex* data) in a DW environment. They designated agents that communicate with each other, whose main function is to pilot the system, collect, structure, generate, and store data.

In the ETL market, some approaches focus on code generation from specifications of mapping and data movement, which are designed by Information Technology (IT) specialists, using graphical interfaces (Kimball & Caserta, 2004). It is the case, for example, of Pentaho Kettle, an open-source ETL tool, which has an easy-to-use graphical interface and a rich transformation library, but the designer only works with pieces of structures. Other ETL approaches focus on representation of ETL processes (Kimball & Caserta, 2004; Dessloch *et al.*, 2008). Orchid (Dessloch *et al.*, 2008), for instance, is a system part of IBM Information Server that facilitates the conversion from schema mappings to ETL processes and vice-versa. Some Database Management Systems (DBMS) vendors have embedded ETL capabilities in their products, using the database as "engine" and Structured Query Language (SQL) as supporting language. This is the case, for example, of Microsoft's Data Transformation Services, IBM's DB2 Data Warehouse Center, and Oracle's Oracle Warehouse Builder. Database-centric ETL solutions differ greatly in quality and functionality. In general, they are not so expensive

and support less resources when comparing them with stand-alone ETL tools. Moreover, the organisation stays dependent on a single proprietary vendor's engine.

Also, we can find tools that do not depend on any particular database technology, allowing easy integration with Business Intelligence (BI) projects deployment (e.g., Oracle Data Integration, IBM InfoSphere DataStage, Informatica Power Center, Ab Initio). In general, these tools have many functions covering all ETL process, and good performance, even for large data volume. The drawback of these tools, in regard to mappings, is that the data models are only used as way to obtain the data movement. Thus, only part of the schemata are analysed each time in order to identify who is the source, who is the target, and which are the transformations required, usually using a graphical interface. The data being mapped forms various blocks, sometimes named stages or phases. Each block is responsible for carrying out a single task (such as a data conversion, a merge, or a sort), and are grouped in a sequential flow, which starts in the source and ends in the target, forming an ETL process. In addition, these tools, although in the most part they are graphical user-friendly interfaces, generate diverse codes, which hide knowledge of procedures and policies used to create the maps, and are strongly dependent on experts and technicians. As the mapping between schemata is not explicit, it cannot be reused (only processes can be reused). Furthermore, people's environments are usually complex. An alternative to commercial packaged solutions are the Open Source data integration tools, which are less expensive than commercially licenced tools (some of them are even freeware), although they still have the same drawbacks. In this category are Pentaho Data Integration, Talend Open Studio, Clover.ETL, and KETL, to quote the most popular.

Some ETL tools are metadata-driven, which is becoming the current trend with ETL data processing. This approach addresses complexity, meets performance needs, and also enables re-use. Informatica PowerCenter was the pioneer. It presents an engine powered by open, interpreted metadata as the main driver for transformation processing.

Many of the ETL tools have integrated repositories that can synchronise metadata from source systems, target database, and other business intelligence tools. Most of these tools automatically generate metadata at every step of the process. Metadata are represented by proprietary scripting languages and each tool manages metadata differently. This makes the use of ETL tools in a data integration context difficult, where usually more than one tool is used in the ETL process.

2.6 Other Related Works

From the point of view of an organization, it is very important to keep information such as *when* and *how* data and structures have been changed over time (i.e., keep the history). Although we have the notion that this subject is out of the scope of the present work, we think it is important to discuss a little about it: a) how time is handling in a DIS like a DW system; b) which are the different types of changes for which a history is required; and c) how the history is managed. This is the subject of the following sections.

2.6.1 Handling time in DW systems

A data warehouse contains time-varying data for the purpose of keeping historic information in order to be used for decision-making users. Time is omnipresent in analysis tasks, and is a difficult element to handle. Although researches in the DW area have led to the maturing of data models, tools and methodologies (Sarda, 1999), there is little support to manage historical information, in spite of its importance. Some proposals to handle it are guided by personal experience, and so, the solutions offered, although practical (e.g. (Kimball, 1996)), lack a formal basis which would lead to better understanding of issues, alternatives and approaches.

Time is an important aspect of all real world phenomena, present in all DW applications, and it is the most dominant factor in data analysis for decision making processes. This situation has a parallel with the general Data Base Management System (DBMS) technologies. Realising the importance of modelling time and handling history data, an extensive research in Temporal Database (TDB) systems, covering models, query languages, and implementation techniques (Snodgrass & Jensen, 1999; Snodgrass, 1995; Tansel *et al.*, 1993) has been carried out since the mid 80s. The researches in TDBs deal with a large universe of perspectives including how the time line should be represented, whether the time is linear (there is a total order) or branch (there is a partial order), which granularity should be used (hour, day, month,...), if the time is explicit or implicit (in the last case the time is obtained through operators inside the language), if the time is absolute or relative, etc..

The major part of TDB study does not matter at all to the DW researchers. In DW, it is important keep the history of how the data as well as schema of a DW have evolved (i.e., all schema components - structures and instances - must have been dated). Thus, DW researchers can consider a smaller universe of study than one handled in TDB and take for granted that

time in DW is, usually, linear, discrete, multigranular, explicit and absolute. The time is linear because there is always a successor and/or a predecessor to some point in the time line. It is discrete because it is represented by points where periods of time can exist with no event or transaction. The time is multigranular since data in a DW does not always have the same behaviour, i.e. the data can be changed hourly, daily, and so on. The time is explicit because every data or event has a known time (either in the source systems or at the time when the data arrives in the DW or both). Finally, the time is relative when it is based on a calendar, although we can consider it to be absolute since the same calendar is used in the entire repository.

Research in the field of temporal support to DW is relatively recent. Some proposals include incorporating the temporal dimension in the data model (Malinowski & Zimányi, 2006; Ravat & Teste, 2000; Pedersen & Jensen, 1999), designing a suitable mechanism to maintain materialised views with temporal characteristics (Yang & Widom, 2000; Yang & Widom, 1998), and conducting correct aggregation in the presence of time-varying data (Yang & Widom, 2003; Mendelzon & Vaisman, 2003; Eder *et al.*, 2002; Yang & Widom, 2001; Hurtado *et al.*, 1999), etc.. Nevertheless, very little attention has been given by the researchers to drawing a conceptual model for DW (e.g., (Vassiliadis *et al.*, 2002; Husemann *et al.*, 2000; Moody & Kortink, 2000; Tryfona *et al.*, 1999; Bækgaard, 1999)), mainly with temporal support ((Malinowski & Zimányi, 2006; Bebel *et al.*, 2004; Koncilia, 2003; Mendelzon & Vaisman, 2003; Ravat & Teste, 2000)).

At present, we do not deal with time in our proposal. We only keep historical information using properties with data domain. However, we intend to add temporal aspects in the near future in order to deal with, for example, changes in DW (at a structural and instance level), temporal types (such as *valid time*, *transaction time*, and *lifespan*), and time structure (i.e., *instant*, *interval*, and *time element*). A more detailed study on this subject can be found in (Pequeno, 2006).

2.6.2 The view maintenance problem

In the materialised approach, when changes occur in information sources, the data of the global schema must be updated. It is referred to as the *view maintenance problem*, just as classes/relations of the global schema are considered as materialised views. Since information sources are usually autonomous, they can evolve over time independently. It means that the data and schemata of the information sources change without being checked at a global level

(e.g., DW system). There are different types of changes in the information sources that are important to the global schema, which can be grouped in five categories as follows:

1. **Content changes**: Data in the global schema can change instigated by source updates (insertions, deletions and updates). The Table 2.1 presents some types of content.[1]

   Table 2.1: Time-varying classes, attributes and methods.

Data Change	**Example**
For an instance of a class as a whole	inserting or deleting an employee
In an attribute value	an employee got married
In a method algorithm	the algorithm for tax calculations can change

   Data in the global schema also may change as time advances even when there is no update in the source systems. This is the case, for example, of a class/relation of the global schema that keeps the list of recent clients (where recent means that the record of new clients was done prior to one week.).

2. **Structure changes**: a component of the global schema can change instigated by changes in the structure of the information sources. For example, in the relational model this corresponds to add/rename/drop an attribute/column of a table.

3. **Schema changes**: the global schema can change when new elements are inserted in (or removed from) underlying schemata (of information sources). For example, in a relational model, it corresponds to creating or dropping a table.

4. **Constraint changes**: is when changes occur such as remove-key-constraint and add-containment-constraint.

5. **Metadata changes**: Statistics and metadata adjustments such as adding the fact that a relation A in a information source $\mathbf{S}_1$ is equivalent to another relation B in $\mathbf{S}_2$; or changing the period of updating of some relation/class of the global schema.

The majority of studies in DISs are only concerned with changes in data values. Approaches assume that only data changes and the schemata of both global and underlying information sources remain static throughout the maintenance process. A number of approaches have been devised in this area (cf. (Labio *et al.*, 2000; Agner, 2005; Chao, 2005) to cite a few); but only a

[1]The table was based on the proposal of Malinowski and Zimányi in (Malinowski & Zimányi, 2006).

few explicitly deal with data evolution, (i.e., keep the historic evolution of the data) (Malinowski & Zimányi, 2006; Ravat & Teste, 2000; Body *et al.*, 2002; Body *et al.*, 2003). However, in the real-world, changes are frequent in both content and structure (including schema changes); and they can happen in the information sources or in the global schema.

The problem of managing schema and structure changes have only partially been explored (Rizzi *et al.*, 2006). The approaches that deal with this subject can be divided into two categories, namely *evolution* and *versioning*: while both categories support schema and structure changes, only the latter keeps the historic evolution of the schema. We can cite (Blaschka *et al.*, 1999; Hurtado *et al.*, 1999; Fan & Poulovassilis, 2004; Kaas *et al.*, 2004) as examples of works in schema evolution; and cite (Morzy & Wrembel, 2003; Body *et al.*, 2002; Body *et al.*, 2003; Koncilia, 2003; Bebel *et al.*, 2004; Mendelzon & Vaisman, 2003; Grandi & Mandreoli, 2003; Solodovnikova, 2007; Solodovnikova, 2009; Golfarelli *et al.*, 2006; Shahzad *et al.*, 2005) as examples in versioning schema. We believe, as well as Rizzi et al. in (Rizzi *et al.*, 2006), that a versioning schema is more suitable in DW system, since it provides a better support to the complex analysis required in this type of environment.

Our proposal only deals with the early phase of the ETL process, when there is no data movement. Nevertheless, we intend to extend our framework with a mechanism to maintain the data of a global schema. Specifically, we intend to include approaches to correctly propagate the changes occurring in information sources in content, structure, schema, and metadata levels.

3

Conceptual Modelling

This chapter presents a conceptual OR model, its structures and integrity constraints. It has been developed for use in modelling the schemata of data integration systems (sources, reference model schema, global schema, etc.).

The proposed model allows for the representation of classes, as well as relations. Furthermore, it divides a schema into structural and behavioural aspects, and it deals with class hierarchy, as well as path expressions. The current work focuses on structure and semantics, not implementation.

The proposed framework intends to be a common, powerful, flexible and expressive language, general enough to allow the projection of traditional (based on relational models) and non-traditional (based on OO or OR models) applications, and so enable the manipulation of complex data. It is formal and supports integrated modelling of a data integration architecture: sources, mappings, ETLs, etc..

In this Chapter, we focus on presenting our language (named *schema language* L_S) to define schemata in a DIS. It is rich semantically, and has the flexibility to easily draw data models from diverse information sources without making deep changes in the original schemata. We briefly compare our model with others. Finally, some conclusions are drawn.

3.1 Language $\mathbf{L}_S$

One of the primary goals of our proposal is to provide a model general enough to be used in relational databases, in object-oriented databases, or in object-relational databases.

In our model we make a distinction between structural aspects, and behavioural aspects of a schema. The former defines particularly: the types, class and relation declarations, as well as the instances of a schema. The latter is mainly responsible for attaching methods to classes. We discuss each aspect of a schema in the following Sections.

3.1.1 Structural aspects

Firstly, we examine the main components of our model. Beginning with the definitions of type, class declaration, class hierarchy, relation declaration, keys and foreign keys declarations of a class/relation, and object-relational schema. Then we look at the instances of a schema, such as typed value, value, object, and tuple. Finally, we finish with definitions about validation of instances for a schema.

Definition 1 (*Type) Let us assume that $\mathcal{P}$ is a finite set of property names, and $\mathcal{C}$ is a finite set of class names. The set of all types is defined inductively as follows:*

- **Base types:** *integer, float, string, date, and boolean are types (called base types).*
- **Reference type:** *if* C_1, C_2, ..., $\mathrm{C}_n \in \mathcal{C}$ *are distinct class names, then every* $\natural\mathrm{C}_i$, $1 \leq i \leq n$, *is a type, and every expression of the form* $\natural\mathrm{C}_i/\natural\mathrm{C}_{i+1}/ \ldots/\natural\mathrm{C}_m$, $0 \leq i \leq n$ *and* $m \leq n$, *is a type (both called reference type).*
- **Structural type:** *If* $\boldsymbol{p}_1$, ..., $\boldsymbol{p}_n \in \mathcal{P}$ *are distinct property names, and* τ_1, ..., τ_n, $n > 0$, *are types, then every expression of the form* $\{\boldsymbol{p}_1{:}\tau_1, \ldots, \boldsymbol{p}_n{:}\tau_n\}$ *are types called structural type. Sub-expressions of the form* $\boldsymbol{p}_i{:}\tau_i$, $1 \leq i \leq n$ *are called components (of the structural type).*
- **Collections types:** *If* τ *is a type, then every expression which conforms to one of the following is also a type, called collection type:*
  1. $\{\tau\}$ *(also called set type).*
  2. $[\tau]$ *(also called list type).*
  3. $\langle\tau, \mathit{length}\rangle$ *(also called array type); where length ≥ 1, and it is the array's length.* □

Some types presented in Definition 1 deserve some explanation. The reference type can point to a single class ($\natural\mathrm{C}_i$) or to more than one class ($\natural\mathrm{C}_i/\natural\mathrm{C}_{i+1}/ \ldots/\natural\mathrm{C}_m$, with / meaning "or"). Informally, a set type represents a unordered collection of distinct values of the same

type. A list type represents an ordered collection of elements of the same type and an array type represents a static sized ordered collection of elements of the same type that requires an ordinal position information for each element. In SQL-3 language (Kriegel & Trukhnov, 2008), this position of an array is called "*index*". We say that a property is *multi-valued* when its type is a collection type, otherwise it is said to be *single-valued*. Example 1 present some examples concerning types.

Example 1 ———————————————

Observe the structural type:

{***identity***: *integer*, ***employeeName***: *string*, ***telephone***: *string*, ***salary***: *float*, ***dept***: $\natural$DEPARTMENT}

which has the properties: ***identity***, ***employeeName***, ***telephone***, *and* ***salary*** *all with base types; and* ***dept*** *with a reference type.*

Now observe the structural type:

{***deptName***: *string*, ***shortForm***: *string*, ***emps***: {$\natural$EMPLOYEE}, ***div***: $\natural$DIVISION}

which has the properties: ***deptName***, *and* ***shortForm*** *with base types;* ***emps*** *with a collection type, specifically a set of reference types; and* ***div*** *with a reference type.*

Note that the property ***emps*** *is multi-valued while, for instance, the property* ***salary*** *is single-valued.*

———————————————

Now we can define the class declaration as follows:

Definition 2 (*Class Declaration*) *Let $\mathcal{C}$ be a set of class names, $\mathcal{P}$ a set of property names, and $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$. A class declaration $\hat{\mathrm{C}}$ defined over $\mathcal{P}$, $\mathcal{C}$, and $\mathcal{T}$ is a 3-tuple of one of two forms: $(\mathrm{C}, \tau, \textbf{\textit{all}})$ or $(\mathrm{C}, \tau, \mathrm{C}')$ such that:*

- ***all*** *is a special name, and serves to indicate that* C *is a root class;*
- C *and* C′ *are class names in $\mathcal{C}$, with* C *and* C′ *being different from* ***all****;*

- τ *is a structural type in* $\mathcal{T}$*, called the base type of class* C*, that is the type explicitly defined for this class;*
- $\mathrm{C}' \neq \mathrm{C}$*. It indicates that* C *is not a root class (i.e., type of* C *is not single formed by its base type, but it involves the type of* C′ *too).* □

Example 2 ______________________

Here we give the classes corresponding to the structural types presented in Example 1.

(EMPLOYEE, {***identity***:*integer*, ***employeeName***:*string*, ***telephone***:*string*, ***salary***:*float*, ***dept***:♮DEPARTMENT}, ***all***)

(DEPARTMENT, {***deptName***:*string*, ***shortForm***:*string*, ***emps***:{♮EMPLOYEE}, ***div***:♮DIVISION}, ***all***)

____

The set of class declarations in our model defines a hierarchy, called *class hierarchy*. It implies that a class declaration can be a specialisation of one or several other classes (Lausen & Vossen, 1998). Specialisation defines a single inheritance relationship between classes that are subordinate to others in accordance with the definitions presented below:

Definition 3 (*Class Hierarchy) Let* $\mathcal{C}$ *be a set of class names, and* $\hat{\mathcal{C}}$ *be a set of class declarations defined over* $\mathcal{P}$*,* $\mathcal{C}$*, and* $\mathcal{T}$*. A set of class declarations* $\mathcal{H}$ *is a class hierarchy iff it conforms to the following properties:*

1. *For each* $\mathrm{C} \in \mathcal{C}$*, there is at the most only one class declaration* $\hat{\mathrm{C}}$ *whose class name is* C*.*
2. *For all* $\hat{\mathrm{C}} \in \hat{\mathcal{C}}$*, whose respective name is* C*, the ordered pair* (C, ***all***) *belongs to the transitive closure of* $\mathcal{H}$ *having a partial order. This transitive closure is called a* ***is-a*** *relationship.* □

For each pair (C,C′) in a **is-a** relationship we say:

- C is a sub-class of C′.
- C′ is a superclass of C.
- C **is-a** C′.

- In cases where there is a class declaration (C,τ,C$'$), we say that C is a "direct subclass" of C$'$, and vice-versa, C$'$ is a "direct superclass" of C.

Based on Definition 3, we can observe the following properties:

Let $\mathcal{C}$ be a set of class names in $\mathcal{C}$, and ***all*** be a special name.

- C **is-a** ***all***, for every class C $\in \mathcal{C}$;
- C **is-a** C, for every class C $\in \mathcal{C}$(reflexive property);
- if C **is-a** C$'$, and C$'$ **is-a** C$''$, then C **is-a** C$''$ (transitivity property);
- if C **is-a** C$'$, then C$'$ **is-a** C is not possible (anti-symmetry property).

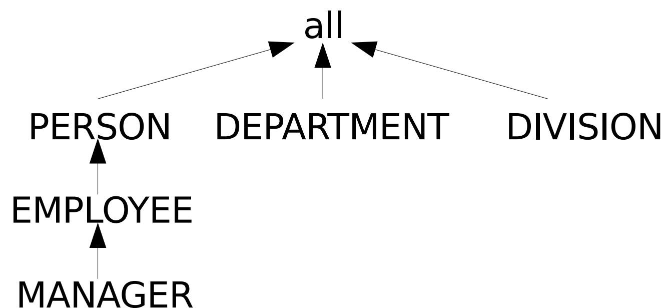
Note that a class hierarchy forms a tree rooted by the constant ***all***, where each class name corresponds to a vertex. Each edge represents the **is-a** relationship between one class name and its direct superclass. This can be better visualised graphically, as we can show in Example 3.

Example 3

Based on Example 1, let us consider now that an employee is a person with a salary and a department. In the same manner a manager is an employee that manages a department in a given period of time. In that case, the classes are as presented in the following text:

(PERSON, {***identity****:integer,* ***name****:string,* ***telephone****:string*}, ***all***)
(EMPLOYEE, {***salary****:float,* ***dept****:*$\natural$DEPARTMENT}, PERSON)
(MANAGER, {***startDate****: date,* ***endDate****:date,* ***bossOf****:*$\natural$DEPARTMENT}, EMPLOYEE)

The class hierarchy of this set of classes is shown as follows:



It is important to make a distinction between the type of a class declaration and the type of a class in accordance to its class hierarchy. The latter is defined as follows:

Definition 4 (*type* and *props* of a Class) *Let $\mathcal{C}$ be a set of class names, $\mathcal{P}$ a set of property names, and $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$. Also let* C *be a class name in $\mathcal{C}$. Thus:*

- ***type***(C)$=\emptyset$, *if there is not a class declaration whose class name is* C*;*
- ***type***(C)$=\tau$, *if the class declaration, whose class name is* C, *is* $(\mathrm{C}, \tau, \textbf{\textit{all}})$*; otherwise*
- *the class declaration is* $(\mathrm{C}, \tau, \mathrm{C}')$, *and then* ***type***(C)$=\{\boldsymbol{p}_1{:}\tau_1,\ \boldsymbol{p}_2{:}\tau_2,\ \ldots,\ \boldsymbol{p}_n{:}\tau_n\}$ *such that for all* $\boldsymbol{p}_i{:}\tau_i$, $1 \leq i \leq n$*:*
  - $\boldsymbol{p}_i{:}\tau_i$ *is a component of* τ*; otherwise*
  - $\boldsymbol{p}_i{:}\tau_i$ *is a component of* ***type***(C′), *and* $\nexists\ \boldsymbol{p}_j{:}\tau_j$ *in* τ *such that* $\boldsymbol{p}_i{=}\boldsymbol{p}_j$

We will refer to the set of property names $\{\boldsymbol{p}_1,\ \boldsymbol{p}_2,\ \ldots,\ \boldsymbol{p}_n\}$ *as defined in* ***type***(C) *as* ***props***(C)*.* □

Example 4

Based on the classes PERSON *and* EMPLOYEE *presented in Example 3, we have:*

- ***type***(PERSON) = {***identity***:*integer*, ***name***:*string*, ***telephone***:*string*}
- ***type***(EMPLOYEE) = {***identity***:*integer*, ***name***:*string*, ***telephone***:*string*, ***salary***:*float*, ***dept***:⫶DEPARTMENT}

- ***props***(PERSON) = {***identity***, ***name***, ***telephone***}
- ***props***(EMPLOYEE) = {***identity***, ***name***, ***telephone***, ***salary***, ***dept***}

In our model, we also deal with relation declarations. In the relational literature, the relation declaration is normally called *relation schema* (Elmasri & Navathe, 2006). It is formally defined in text below.

Definition 5 (*Relation Declaration*) *Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{P}$ a set of property names, and $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$. A relation declaration* $\hat{\mathrm{R}}$ *defined over $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{T}$ is a pair* (R, τ) *such that:*

- R *is a relation name in* $\mathcal{R}$*; and*
- τ *is a structural type in* $\mathcal{T}$*.*

We will use the notation ***type***(R) *to denote the type of a relation name* R*, which in this case is* τ *itself, and we will refer to the set of property names defined for a relation name* R *as* ***props***(R)*.* □

Example 5

Here we show three relations: MANUFACTURER*,* PRODUCT *and* CATEGORY*. The first contains information about suppliers of products sold in a business, the second keeps information about goods offered for sale in a business, and the third keeps information about the category of products.*

(MANUFACTURER,{***manufacturer***:*string*, ***region***:*string*, ***address***:*string*}),
(PRODUCT,{***number***:*integer*, ***productName***:*string*, ***category***:*integer*, ***supplier***:*string*, ***region***:*string*}),
(CATEGORY,{***code***:*integer*, ***name***:*string*, ***sort***:*string*})

In order to simplify reading, hereafter we will use the short forms *class* rather than *class declaration*, and *relation* instead of *relation declaration*, when there is no ambiguity.

Our model, like other relational models, includes the notion of keys and foreign keys of a relation. In addition, we extend these concepts to contemplate classes too[1]. Both are defined as follows:

Definition 6 (*Key Declaration)* *Let* $\mathcal{K}$ *be a set of key names,* $\mathcal{R}$ *a set of relation names,* $\mathcal{C}$ *a set of class names, and* $\mathcal{P}$ *a set of property names. Also let* R *be a relation name in* $\mathcal{R}$ *(or class name in* $\mathcal{C}$*). A key declaration of* R *is a 3-tuple* $(\mathrm{K}, \mathrm{R}, \boldsymbol{K}(\mathrm{R}))$ *defined over* $\mathcal{P}$*,* $\mathcal{K}$*,* $\mathcal{R}$*, and* $\mathcal{C}$*, such that:*

1. K *is a key name in* $\mathcal{K}$*;*

[1]Usually, there is the notion of Object IDentities (OIDs) in OO data models, which have a role similar to the key declarations in the relational data models. OIDs will be defined later.

2. R *is the owner of the key* K*;*

3. $\boldsymbol{K}(\mathrm{R})$ *is a non-empty set* $\boldsymbol{K}(\mathrm{R}) \subseteq \boldsymbol{props}(\mathrm{R})$*, such that each* $\boldsymbol{p} \in \boldsymbol{K}(\mathrm{R})$ *is a single-valued property with a non-structural type.* □

Some examples of key declarations are presented in the following text.

Example 6

Based on Example 5, some possible keys of MANUFACTURER*,* PRODUCT*, and* CATEGORY *are:*

(***K1***,PRODUCT,{***number***})	(***K3***,CATEGORY,{***code***})
(***K2***,PRODUCT,{***productName***})	(***K4***,MANUFACTURER,{***manufacturer,region***})

Based on Example 4, some possible keys of PERSON *and* EMPLOYEE *are:*

(**K*1***,PERSON,{***identity***})	(**K*3***,EMPLOYEE,{***identity***})
(**K*2***,PERSON,{***name,telephone***})	

A relation (or class) can have more than one key. In some relational models, it is usual for one of them to be designed as a *primary key*, and all others are designed as *alternate keys*.

Definition 7 (*Foreign Key Declaration)* *Let* $\mathcal{K}$ *be a set of key names,* $\mathcal{R}$ *a set of relation names and* $\mathcal{C}$ *a set of class names. Also let* R *be relation name in* $\mathcal{R}$ *(or class name in* $\mathcal{C}$*). A foreign key declaration of* R *is a 5-tuple* $\left(\mathrm{FK}, \mathrm{R}, \boldsymbol{FK}(\mathrm{R}), \mathrm{R}', \boldsymbol{K}(\mathrm{R}')\right)$*, such that:*

1. FK *is a key name in* $\mathcal{K}$*;*

2. R *is the owner of the key* FK*;*

3. $\boldsymbol{FK}(\mathrm{R})$ *is a non-empty set* $\boldsymbol{FK}(\mathrm{R}) \subseteq \boldsymbol{props}(\mathrm{R})$*, such that each* $\boldsymbol{p} \in \boldsymbol{FK}(\mathrm{R})$ *is a single-valued property with a non-structural type;*

4. R' *is a relation name in* $\mathcal{R}$ *(or class name in* $\mathcal{C}$*) that is referred by* R*;*

5. $\boldsymbol{K}(\mathrm{R}')$ *is a non-empty set* $\boldsymbol{K}(\mathrm{R}') \subseteq \boldsymbol{props}(\mathrm{R}')$ *such that there is a key declaration of* R' $(\mathrm{K}, \mathrm{R}', \boldsymbol{K'})$ *with* $\boldsymbol{K'} = \boldsymbol{K}(\mathrm{R}')$*.* □

Some examples of foreign key declarations are presented in the following text.

Example 7

Based on Examples 5 and 6, we have the following foreign key declarations:

(FK$_1$, PRODUCT, {***category***}, CATEGORY, {***code***})

(FK$_2$, PRODUCT, {***supplier, region***}, MANUFACTURER, {***manufacturer, region***})

Note that, in FK$_2$, ***FK***(PRODUCT) *is formed by two properties:* ***supplier*** *and* ***region****, in accordance to the key of the relation* MANUFACTURER *that* FK$_2$ *refers to.*

Having introduced these concepts, we can now define an OR schema.[2] An OR schema is a set of declarations that serve as templates to generate the instances of the application domain. A formal definition of an OR schema is as follows:

Definition 8 (*Object-Relational Schema) Let $\mathcal{P}$ be a set of property names, $\mathcal{C}$ a set of class names, $\mathcal{R}$ a set of relation names, $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$, $\mathcal{K}$ a set of key names, and $\mathcal{L}$ a set of schema names. An object-relational schema* **S** *defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{K}$ is a quadruplet $(\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ such that* **S** *is a schema name in $\mathcal{L}$, $\mathcal{H}$ is a class hierarchy, $\hat{\mathcal{R}}$ is a set of relation declarations defined over $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{T}$; and $\hat{\mathcal{K}}$ is a set of key declarations, and foreign key declarations, defined over $\mathcal{P}$, $\mathcal{K}$, $\mathcal{R}$, and $\mathcal{C}$, such that there exists at least one key declaration for every relation name in $\mathcal{R}$.* □

Note that, given an OR schema **S**=(**S**, $\mathcal{H}$, $\hat{\mathcal{R}}$, $\hat{\mathcal{K}}$), if $\mathcal{H}$ is empty then our schema is a relational one; otherwise, our schema is an object-relational one. Also, we want emphasise that we do not distinguish a key declaration as a primary key, but we reinforce the existence of a key declaration in the relations.

Now we define the instances of our model, which includes typed values, values, objects, and tuples.

[2]The behaviour of a schema will be addressed later.

Definition 9 (*Typed Value and Value*) *Let $\mathcal{P}$ be a set of property names, $\mathcal{C}$ a set of class names, $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$, and $\mathcal{O}$ be a set of object identities, where an object identity is composed from the symbol $\sharp$ followed by a positive integer number. Also let τ be a type in $\mathcal{T}$, and $\boldsymbol{v}$ a value. The set of all typed values is as follows:*

- **Atomic typed value or constant:** $\boldsymbol{w}=(\mathbf{v},\tau)$ *is an atomic typed value or a constant typed value when* $\mathbf{v}$ *is an integer number, a float, a string, a date, or a boolean value (or* $\mathbf{v}$ = ***nil***), *and τ is, respectively, an integer, float, string, date, or boolean type. In this case, we say that* $\mathbf{v}$ *is an atomic value or a constant.*

- **Reference typed value:** $\boldsymbol{w}=(\mathbf{v},\tau)$ *is a reference typed value when* $\mathbf{v}$ *is an object identity in $\mathcal{O}$ (or* $\mathbf{v}$ = ***nil***), *and τ is a reference type. In this case, we say that* $\mathbf{v}$ *is a reference value.*

- **Structural typed value**: *Let $n > 0$, $\boldsymbol{p}_i \in \mathcal{P}$ be distinct property names, and $\boldsymbol{w}_i=(\boldsymbol{v}_i,\tau_i)$ be typed values, for $1 \leq i \leq n$, then the pairs $(\{\boldsymbol{p}_1{:}\boldsymbol{v}_1,...,\boldsymbol{p}_n{:}\boldsymbol{v}_n\},\ \{\boldsymbol{p}_1{:}\tau_1,...,\boldsymbol{p}_n{:}\tau_n\})$ are structural typed values. In this case, we say that $\{\boldsymbol{p}_1{:}\boldsymbol{v}_1,\ ...,\ \boldsymbol{p}_n{:}\boldsymbol{v}_n\}$ is a structural value. Each $\boldsymbol{p}_i{:}\boldsymbol{v}_i$, $1 \leq i \leq n$, is called a component (of the structural value).*

- **Collections:** *Let $\boldsymbol{w}_i=(\boldsymbol{v}_i,\tau)$ be typed values, for $1 \leq i \leq n$, then the following couples also denote typed values called collections:*

  1. *$(\{\boldsymbol{v}_1,...,\boldsymbol{v}_n\},\{\tau\})$, with $\boldsymbol{v}_i$ distinct (also called set typed value). The value $\mathbf{v}=\{\boldsymbol{v}_1, ...,\boldsymbol{v}_n\}$ is called a set value. Each $\boldsymbol{v}_i$ in $\mathbf{v}$ is said to be a set element of $\mathbf{v}$.*
  2. *$([\boldsymbol{v}_1,...,\boldsymbol{v}_n],[\tau])$ (also called list typed value). The value $\mathbf{v}=[\boldsymbol{v}_1,...,\boldsymbol{v}_n]$ is called a list value. Each $\boldsymbol{v}_i$ in $\mathbf{v}$ is said to be a list element of $\mathbf{v}$.*
  3. *$(\langle\boldsymbol{v}_1,...,\boldsymbol{v}_n\rangle,\langle\tau,n\rangle)$ (also called array typed value). The value $\mathbf{v}=\langle\boldsymbol{v}_1,\ ...,\boldsymbol{v}_n\rangle$ is called an array value. Sub-expressions of the form $\boldsymbol{pos}[i]{:=}\ \boldsymbol{v}_i$, such that $\boldsymbol{pos}[i]$ points at the position $\boldsymbol{i}$ in an array, and $1 \leq i \leq n$, are called array elements.* □

In the remainder, to ease the reading, we will show the typed values without their types when there is no ambiguity. Some examples are shown in the following:

Example 8

Typed values can be simple or more complex, like the ones we see below:

Atomic

Typed Value	***Value***
(*10,integer*)	*10*
(*10.0,float*)	*10.0*
(*'Pat',string*)	*'Pat'*
(*True,boolean*)	*True*
(*'01/09/2008',date*)	*'01/09/2008'*

Collection

Typed Value	***Value***
({*nil*},{*float*})	({*nil*},{*float*})
(['*1*', '*f*'], [*string*])	['*1*', '*f*']
([{'*John*'},{'*p*'}], [{*string*}])	[{'*John*'},{'*p*'}]
(⟨'*b*', '*0.9*', '*e*'⟩, ⟨*string,3*⟩)	⟨'*b*', '*0.9*', '*e*'⟩
(⟨*1,2*⟩, ⟨*integer,3*⟩)	(⟨*1,2*⟩, ⟨*integer,3*⟩)

Reference

Typed Value	***Value***
(♯10, ♮C)	♯*10*

Structural

Typed Value	***Value***
({**p**$_1$:*1*, **p**$_2$:*'apple'*}, {**p**$_1$:*integer*, **p**$_2$:*string*})	{**p**$_1$:*1*, **p**$_2$:*'apple'*}
({**p**$_1$:{**city**:*'NY'*,**number**:*3*}},{**p**$_1$:{**city**:*string*,**number**:*integer*}})	{**p**$_1$:{**city**:*'NY'*,**number**:*3*}}

In our model, entities of the real world can be represented by objects. Each object has its own identity, which is system-generated, and remains unchanged throughout the whole lifetime of the object. We define the objects as follows.

Definition 10 (*Object*) *Let $\mathcal{O}$ be a set of object identities, $\mathcal{W}$ be a set of typed values, and $\mathcal{H}$ be a class hierarchy. Also let* $\hat{\mathrm{C}}$ *be a class declaration belonging to $\mathcal{H}$, whose class name is* C.

*An object is a pair $(\boldsymbol{o}, \boldsymbol{w})$, where $\boldsymbol{o}$ is an object identity in $\mathcal{O}$, and $\boldsymbol{w}$=$(\boldsymbol{v},\tau)$ is a typed value in $\mathcal{W}$ such that τ=**type**(C).* □

In order to simplify reading, we shall mostly use *object* instead of *object identity*; *structural value* instead of *structural typed value*; and *property* instead of *property name*, whenever no ambiguity exists. In the following text, we show some examples of objects.

Example 9 ---

Below are some objects that represent people (in general), managers, employees, and departments of a business in a given period of time, conforming to the types in Examples 2 and 3.

(♯13, {**identity**:*12298778*, **name**:*'Pimpinha Filha'*, **telephone**:*'214568729'*}),

(♯11, {**identity**:*23454458*, **name**:*'Victor Ramos'*, **telephone**:*'224509887'*, **salary**:*1800.0*, **dept**:♯29,
startDate:*'01/08/1977'*, ***endDate***:*nil*, ***bossOf***:♯29}),
(♯1, {**identity**:*12345678*, **name**:*'Tete Neto'*,**telephone**:*'212223455'*, **salary**:*1500.0*, **dept**:♯29}),
(♯29, {**deptName**:*'Training'*, **shortForm**:*'TN'*, **emps**:{♯11, ♯1}, **div**:♯15})

In our model, we may also represent entities of the real world through tuples. They are defined as follows.

Definition 11 (*Tuple) Let $\mathcal{W}$ be a set of typed values, and $\hat{\mathcal{R}}$ be a set of relation declarations. Also let* $\hat{\mathrm{R}}$*, whose relation name is* R*, be a relation declaration belonging to $\hat{\mathcal{R}}$. A tuple is a structural typed value $\boldsymbol{w}=(\boldsymbol{v},\tau) \in \mathcal{W}$ such that $\tau=\boldsymbol{type}(\mathrm{R})$.* □

Below are some tuples that represent products that a given business can sell in a given period of time, conforming to the type of the relation PRODUCT presented in Example 5.

{**number**:1, **productName**:'apple', **category**:1, **supplier**: 'SonsLtd', **region**: 'US'},
{**number**:2, **productName**:'orange', **category**:1, **supplier**: 'SonsLtd', **region**: 'US'},
{**number**:3, **productName**:'CSI', **category**:2, **supplier**: 'AllMidia', **region**: 'US'},
{**number**:4, **productName**:'friends', **category**:2, **supplier**: 'AllMidia', **region**: 'US'}

In the following, we define the instance or state of an OR schema,[3] which populates classes with object identities, assigns values to object identities, and assigns values to relations.

Definition 12 (*Instance of an OR Schema) Let* $\mathbf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be an object-relational schema defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{K}$; $\mathcal{W}$ be a set of typed values; and $\mathcal{O}$ be a set of object identities. An instance of a schema* $\mathbf{S}$ *is* $\boldsymbol{D}(\mathbf{S}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$*, where:*

- *$\hat{\mathcal{O}}$ is a set of objects such that:*
    1. *for all objects $(\boldsymbol{o}, \boldsymbol{w})$, $\boldsymbol{w}$ is a typed value $\boldsymbol{w}=(\boldsymbol{v},\tau)$ such that τ is the type of some class declaration in $\mathcal{H}$.*

[3] Also common are the expressions *database state*, and *database instance*.

2. *for any two different objects* $(\boldsymbol{o}_1, \boldsymbol{w}_1)$, *and* $(\boldsymbol{o}_2, \boldsymbol{w}_2)$ *in* $\hat{\mathcal{O}}$, $\boldsymbol{o}_1 \neq \boldsymbol{o}_2$, *i.e., no two objects in* $\hat{\mathcal{O}}$ *have the same identity.*

- $\mathcal{W}_t$ *is a set of tuples in* $\mathcal{W}$ *such that each tuple in* $\mathcal{W}_t$ *is of the type of some relation declaration in* $\hat{\mathcal{R}}$. □

In the following text we present some notation that is used through this work.

Let $\mathbf{S}$=$(\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ be an object-relational schema defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, and $\mathcal{K}$, and $\boldsymbol{D}(\mathbf{S}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$ an instance of $\mathbf{S}$. Let also $\mathcal{W}$ be a set of typed values, $\mathbf{v}$ a value, and $\mathcal{O}$ be a set of object identities.

- For every class name C in $\mathcal{C}$:
  1. $\boldsymbol{D}_b(\mathrm{C})$ (read base class extent of the class name C) is the set of object identities in $\mathcal{O}$ that are assigned to the class declaration $\hat{\mathrm{C}} \in \mathcal{H}$, whose respective name is C, in the instance $\boldsymbol{D}(\mathbf{S})$;
  2. $\boldsymbol{D}(\mathrm{C})$ (read class extent of the class name C) taking into account the class hierarchy, such that $\boldsymbol{D}(\mathrm{C}) = \boldsymbol{D}_b(\mathrm{C}) \cup \{\mathbf{o} \mid \mathbf{o} \in \boldsymbol{D}_b(\mathrm{C}'), \forall \mathrm{C}' \text{ in } \mathcal{C} \text{ such that } \mathrm{C}' \textbf{ is-a } \mathrm{C}\}$.
- For every relation name R in $\mathcal{R}$, $\boldsymbol{D}(\mathrm{R})$ (read relation extent of the relation name R) is the set of tuples $\mathbf{t}$ in $\mathcal{W}_t$ that are assigned to the relation declaration $\hat{\mathrm{R}} \in \hat{\mathcal{R}}$, whose respective name is R, in the instance $\boldsymbol{D}(\mathbf{S})$;
- If $\boldsymbol{p}{:}\tau \in \boldsymbol{type}(\mathrm{C})$, and $\mathbf{o} \in \boldsymbol{D}(\mathrm{C})$, then $\mathbf{o}\bullet\boldsymbol{p}$ assigns a typed value $\boldsymbol{w} \in \mathcal{W}$ to $\mathbf{o}$, such that $\boldsymbol{w}$=$(\mathbf{v}, \tau)$. The notation $\mathbf{o} \bullet \boldsymbol{p}|_{\boldsymbol{D}(\mathbf{S})}$ denotes the value of the property name $\boldsymbol{p}$ of object identity $\mathbf{o}$ in state $\boldsymbol{D}(\mathbf{S})$. In $\mathbf{o}\bullet\boldsymbol{p}$=$\boldsymbol{w}$ we say that $\mathbf{o}$ is related with the typed value $\boldsymbol{w}$ through property name $\boldsymbol{p}$.
- If $\boldsymbol{p}{:}\tau \in \boldsymbol{type}(\mathrm{R})$, and $\mathbf{t} \in \boldsymbol{D}(\mathrm{R})$, then $\mathbf{t}\bullet\boldsymbol{p}$ assigns a typed value $\boldsymbol{w} \in \mathcal{W}$ to $\mathbf{t}$, such that $\boldsymbol{w}$=$(\mathbf{v}, \tau)$. The notation $\mathbf{t}\bullet\boldsymbol{p}|_{\boldsymbol{D}(\mathbf{S})}$ denotes the value of the property name $\boldsymbol{p}$ of structural typed value $\mathbf{t}$ in state $\boldsymbol{D}(\mathbf{S})$. In $\mathbf{t}\bullet\boldsymbol{p}$=$\boldsymbol{w}$ we say that $\mathbf{t}$ is related with the typed value $\boldsymbol{w}$ through property name $\boldsymbol{p}$.
- For every type τ in $\mathcal{T}$, $\boldsymbol{dom}(\tau)$ is the set of all possible typed values of τ in instance $\boldsymbol{D}(\mathbf{S})$.
- For every foreign key declaration $(\mathrm{FK}, \mathrm{C}, \boldsymbol{FK}(\mathrm{C}), \mathrm{C}', \boldsymbol{K}(\mathrm{C}'))$ of C:

– The expression $\mathbf{o}\bullet\text{FK}$, with $\mathbf{o} \in \boldsymbol{D}(\text{C})$, returns the object identity $\mathbf{o}' \in \boldsymbol{D}(\text{C}')$ such that $\mathbf{o}\bullet\boldsymbol{p}_i = \mathbf{o}'\bullet\boldsymbol{p}'_i$, for all $\boldsymbol{p}_i \in \boldsymbol{FK}(\text{C})$, and respective $\boldsymbol{p}'_i \in \boldsymbol{K}(\text{C}')$. Thus, we have that C refers to C′ through to FK.

– The expression $\mathbf{t}\bullet\text{FK}$, with $\mathbf{t} \in \boldsymbol{D}(\text{R})$, returns the tuple $\mathbf{t}' \in \boldsymbol{D}(\text{R}')$ such that $\mathbf{t}\bullet\boldsymbol{p}_i = \mathbf{t}'\bullet\boldsymbol{p}'_i$, for all $\boldsymbol{p}_i \in \boldsymbol{FK}(\text{R})$, and respective $\boldsymbol{p}'_i \in \boldsymbol{K}(\text{R}')$. Thus, we have that R refers to R′ through to FK.

– The expression $\mathbf{o}'\bullet\text{FK}^{-1}$, with $\mathbf{o}' \in \boldsymbol{D}(\text{C}')$, returns an object identity (or a set of object identities) $\mathbf{o} \in \boldsymbol{D}(\text{C})$ such that $\mathbf{o}\bullet\boldsymbol{p}_i = \mathbf{o}'\bullet\boldsymbol{p}'_i$, for all $\boldsymbol{p}_i \in \boldsymbol{FK}(\text{C})$, and respective $\boldsymbol{p}'_i \in \boldsymbol{K}(\text{C}')$. Thus, we have that C′ refers to C through the inverse of FK (i.e., FK^{-1}).

– The expression $\mathbf{t}'\bullet\text{FK}^{-1}$, with $\mathbf{t}' \in \boldsymbol{D}(\text{R}')$, returns a tuple (or a set of tuples) $\mathbf{t} \in \boldsymbol{D}(\text{R})$ such that $\mathbf{t}\bullet\boldsymbol{p}_i = \mathbf{t}'\bullet\boldsymbol{p}'_i$, for all $\boldsymbol{p}_i \in \boldsymbol{FK}(\text{R})$, and respective $\boldsymbol{p}'_i \in \boldsymbol{K}(\text{R}')$. Thus, we have that R′ refers to R through the inverse of FK (i.e., FK^{-1}).

In order to simplify reading, the references to the current state $\boldsymbol{D}(\mathbf{S})$ will be omitted when possible. For instance, given the object identity $\mathbf{o}$, we will use $\mathbf{o}\bullet\boldsymbol{p}$ instead of $\mathbf{o}\bullet\boldsymbol{p}|_{\boldsymbol{D}(\mathbf{S})}$.

It is important to emphasise that each object identity is attached to a unique class (called its *base class*), although an object in general is an element of several class extents (in a class hierarchy). In the following text, we present some examples of class extents and of relation extents.

Example 10

Here we give the class extents corresponding to the classes presented in Examples 2 and 3, taking into account objects showed in Example 9.

$\boldsymbol{D}_b(\text{PERSON}) = \{\sharp 13\}$,	$\boldsymbol{D}(\text{PERSON}) = \{\sharp 13, \sharp 1, \sharp 11\}$,
$\boldsymbol{D}_b(\text{EMPLOYEE}) = \{\sharp 1\}$,	$\boldsymbol{D}(\text{EMPLOYEE}) = \{\sharp 1, \sharp 11\}$,
$\boldsymbol{D}_b(\text{MANAGER}) = \{\sharp 11\}$,	$\boldsymbol{D}(\text{MANAGER}) = \{\sharp 11\}$,
$\boldsymbol{D}_b(\text{DEPARTMENT}) = \{\sharp 29\}$,	$\boldsymbol{D}(\text{DEPARTMENT}) = \{\sharp 29\}$,

Example 11

Here we give the relation extent corresponding to PRODUCT *presented in Example 5.*

$\boldsymbol{D}($ PRODUCT$) = \{$
{***number***:*1*, ***productName***:*'apple'*, ***category***:*1*, ***supplier***: *'SonsLtd'*, ***region***: *'US'*},
{***number***:*2*, ***productName***:*'orange'*, ***category***:*1*, ***supplier***: *'SonsLtd'*, ***region***: *'US'*},
{***number***:*3*, ***productName***:*'CSI'*, ***category***:*2*, ***supplier***: *'AllMidia'*, ***region***: *'US'*},
{***number***:*4*, ***productName***:*'friends'*, ***category***:*2*, ***supplier***: *'AllMidia'*, ***region***: *'US'*}
}

Until now we have not restricted the values that we allow in the schemata defined in our model. However, an instance of a schema needs to follow some rules in order to be valid. Thus, in the following, we present some definitions to guarantee the validity of our instances.

Definition 13 (*Valid Reference Typed Value) Let* $\mathbf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be an object-relational schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, *and* $\mathcal{K}$, *and* $\boldsymbol{D}(\mathbf{S}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$ *an instance of* $\mathbf{S}$. *Also let* C, C_1, C_2, ..., C_n *be class names in* $\mathcal{C}$. *We say that:*

- *A given reference typed value* $\boldsymbol{w}=(\sharp n, \natural \mathrm{C})$ *is valid in* $\boldsymbol{D}(\mathbf{S})$ *iff there is an object* $(\sharp n, \boldsymbol{w})$ *in* $\hat{\mathcal{O}}$, *and* $\sharp n \in \boldsymbol{D}(\mathrm{C})$.
- *A given reference typed value* $\boldsymbol{w}=(\sharp n, \natural \mathrm{C}_1/\natural \mathrm{C}_2/.../\natural \mathrm{C}_n)$ *is valid in* $\boldsymbol{D}(\mathbf{S})$ *iff there is an object* $(\sharp n, \boldsymbol{w})$ *in* $\hat{\mathcal{O}}$, *and* $\sharp n \in \boldsymbol{D}(\mathrm{C}_1)$ *or* $\sharp n \in \boldsymbol{D}(\mathrm{C}_2)$ *or ... or* $\sharp n \in \boldsymbol{D}(\mathrm{C}_n)$. $\square$

Definition 14 (*Valid Key) Let* $\mathbf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be an object-relational schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, *and* $\mathcal{K}$, *and* $\boldsymbol{D}(\mathbf{S}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$ *be an instance of* $\mathbf{S}$. *Also let* R *be a relation name in* $\mathcal{R}$ *or* R *be a class name in* $\mathcal{C}$, *and* $(\mathrm{K}, \mathrm{R}, \boldsymbol{K}(\mathrm{R}))$ *be a key declaration of* R. $(\mathrm{K}, \mathrm{R}, \boldsymbol{K}(\mathrm{R}))$ *must satisfy the following entity integrity constraints:*

1. *for any tuple (or object identity)* $\boldsymbol{t} \in \boldsymbol{D}(\mathrm{R})$, $\boldsymbol{t}\bullet\boldsymbol{p}\neq nil$, *for all* $\boldsymbol{p} \in \boldsymbol{K}(\mathrm{R})$, *i.e., the values of the properties that form a key declaration of a relation (or class) name cannot be nil;*
2. *for any two different tuples (or object identities)* $\boldsymbol{t}_1$ *and* $\boldsymbol{t}_2$ *in the current instance* $\boldsymbol{D}(\mathrm{R})$, $\exists \boldsymbol{p} \in \boldsymbol{K}(\mathrm{R})$ *such that* $\boldsymbol{t}_1\bullet\boldsymbol{p} \neq \boldsymbol{t}_2\bullet\boldsymbol{p}$. *That is, no two tuples (or object identities) in* $\boldsymbol{D}(\mathrm{R})$ *have the same value on all properties in* $\boldsymbol{K}(\mathrm{R})$ *(Uniqueness property).* $\square$

At this point we want to emphasise that instances of a class can be identified through object identities or through keys, while instances of relations can only be identified through keys. Thus, in order to guarantee that we can always uniquely identify instances in our model, we reinforce the definition of at least a key declaration for every relation (see Definition 8).

Definition 15 (*Valid Foreign Key) Let* $\mathsf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be an object-relational schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, *and* $\mathcal{K}$, *and* $\boldsymbol{D}(\mathsf{S}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$ *be an instance of* S. *Also let* R *and* R′ *be relation names in* $\mathcal{R}$ *or* R *and* R′ *be class names in* $\mathcal{C}$, *and* $(\mathrm{FK}, \mathrm{R}, \boldsymbol{FK}(\mathrm{R}), \mathrm{R}', \boldsymbol{K}(\mathrm{R}'))$ *be a foreign key declaration of* R *that refers to* R′. $(\mathrm{FK}, \mathrm{R}, \boldsymbol{FK}(\mathrm{R}), \mathrm{R}', \boldsymbol{K}(\mathrm{R}'))$ *must satisfy the following referential integrity constraints:*

1. *Given* $\boldsymbol{p}_i{:}\tau_i \in \boldsymbol{type}(\mathrm{R})$, *for all* $\boldsymbol{p}_i \in \boldsymbol{FK}(\mathrm{R})$; *and* $\boldsymbol{p'}_i{:}\tau'_i \in \boldsymbol{type}(\mathrm{R}')$, *for all* $\boldsymbol{p'}_i \in \boldsymbol{K}(\mathrm{R}')$. *For all tuple (or object identity)* $\boldsymbol{t} \in \boldsymbol{D}(\mathrm{R})$, *and* $\boldsymbol{t'} \in \boldsymbol{D}(\mathrm{R}')$, $\boldsymbol{dom}(\tau_i) = \boldsymbol{dom}(\tau'_i)$.
2. *For every tuple (or object identity)* $\boldsymbol{t}$ *in* $\boldsymbol{D}(\mathrm{R})$, *if* $\boldsymbol{t} \bullet \boldsymbol{p}_i \neq$ *nil, for all* $\boldsymbol{p}_i \in \boldsymbol{FK}(\mathrm{R})$, *then there is a tuple (or object identity)* $\boldsymbol{t'}$ *in* $\boldsymbol{D}(\mathrm{R}')$ *such that* $\boldsymbol{t} \bullet \boldsymbol{p}_i = \boldsymbol{t'} \bullet \boldsymbol{p'}_i$, *for all* $\boldsymbol{p}_i \in \boldsymbol{FK}(\mathrm{R})$, *and respective* $\boldsymbol{p'}_i \in \boldsymbol{K}(\mathrm{R}')$. □

Note that, in the previous Definition, the relations/classes R and R′ do not need to be distinct.

Definition 16 (*Valid Instance of a Schema) Let* $\mathsf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be an object-relational schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, *and* $\mathcal{K}$, *and* $\boldsymbol{D}(\mathsf{S}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$ *be an instance of* S. *We say that an instance* $\boldsymbol{D}(\mathsf{S})$ *is valid iff:*

- *All reference typed values in* $\boldsymbol{D}(\mathsf{S})$ *are valid reference typed values;*
- *All key declarations in* $\boldsymbol{D}(\mathsf{S})$ *are valid key declarations;*
- *All foreign key declarations in* $\boldsymbol{D}(\mathsf{S})$ *are valid foreign key declarations.* □

Hereafter, we will only consider valid instances of schemata.

All Object-Relational Data Models (ORDMs) must deal with the behaviour part of the schemata. Accordingly, we present the definitions covering this subject in the next Section. However, here we do not explore behaviour aspects in a data integration context. This should be part of a future work.

3.1.2 Adding behaviour

The methods are the final ingredients of an OR data model. They are operations that can be applied to objects, and can be used in various ways. For instance, some of them can create or destroy an object, others can update the value of the object, and others can apply some calculation. A method is defined as follows:

Definition 17 (*Method) Let $\mathcal{C}$ be a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$. Let us assume that $\mathcal{M}$ is a finite set of method names. A Method defined over $\mathcal{M}$ is a 3-tuple $(\boldsymbol{M}, S, \mathcal{I}mp)$ such that:*

- *$\boldsymbol{M}$ is a method name in $\mathcal{M}$.*
- *S is a set of method signatures defined over $\mathcal{M}$, where, like in (Abiteboul* et al.*, 1995), each method signature is an expression of the form $\boldsymbol{M}$:$\mathrm{C}\times\tau_i\times\tau_{i+1}\times\ldots\times\tau_n\rightarrow\tau$, such that:*
  1. *C is a class name in $\mathcal{C}$;*
  2. *$\tau_i \in \mathcal{T}$, $0 \le i \le n$, are the expected types of the input argument, which can be empty; and*
  3. *$\tau \in \mathcal{T}$ states the type of the result, if applicable.*
- *$\mathcal{I}mp$ is a set of implementations, one for each signature in S, and which can be specified in a general-purpose programming language.*[4] □

A class specifies the set of properties and the set of methods that are common to all instances (the objects) of the class. Now we will extend our Definition 2 about class declaration in order to allow it to have methods.

Definition 18 (*Class Declaration - with Methods) Let $\mathcal{C}$ be a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$, and $\mathcal{M}$ a set of method names. A class declaration $\hat{\mathrm{C}}$ defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{T}$, and $\mathcal{M}$ is a quadruplet of one of two forms: $(\mathrm{C}, \tau, \boldsymbol{all}, \mathcal{S})$ or $(\mathrm{C}, \tau, \mathrm{C}', \mathcal{S})$ such that:*

- *C, and C' are class names, τ is a structural type, and* ***all*** *is a name, all just as defined in Definition 2; and*

[4] *Here we do not consider how methods are implemented.*

- *$\mathcal{S}$ is a set of method signatures defined over $\mathcal{M}$ and $\mathcal{C}$ such that for each $\boldsymbol{M}$:$\mathrm{C}\times\tau_i\times\tau_{i+1}\times\ldots\times\tau_n\rightarrow\tau$, and $\boldsymbol{M'}$:$\mathrm{C}\times\tau'_i\times\tau'_{i+1}\times\ldots\times\tau'_m\rightarrow\tau'$ in $\mathcal{S}$, $\boldsymbol{M}\neq\boldsymbol{M'}$. That means that there is only one method signature for each method name in the same class.* □

Example 12

Let us consider the class EMPLOYEE *shown in Example 2. In order to increase the salary, the method **addSalary** can be defined as follows:*

$$\boldsymbol{addSalary}\text{:EMPLOYEE}\times \textit{float}\rightarrow \textit{float}$$

We say that method **M** is defined for a class name C if there exists a signature of **M** at class $\hat{\mathrm{C}}$ (whose class name is C). The applicability of a method to objects in a class, additionally, follows from inheritance rules just as it occurs in the inheritance of properties. A formal definition is presented below.

Definition 19 (*Applicable methods at a class) Let $\mathcal{C}$ be a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$, and $\mathcal{M}$ a set of method names. Also let* C *be a class name in $\mathcal{C}$ to class declaration $\hat{\mathrm{C}}$. Thus, the set of applicable methods on a class declaration $\hat{\mathrm{C}}$, $\boldsymbol{meth}(\mathrm{C})$, is given by:*

1. *$\boldsymbol{meth}(\mathrm{C}) = \mathcal{S}$, if the class declaration $\hat{\mathrm{C}}=(\mathrm{C},\tau,\boldsymbol{all},\mathcal{S})$; otherwise*
2. *The class declaration is $\hat{\mathrm{C}}=(\mathrm{C},\tau,\mathrm{C}',\mathcal{S})$, and then $\boldsymbol{meth}(\mathrm{C}) = \{S_1, S_2, \ldots, S_n\}$ such that for all S_i, whose respective method name is $\boldsymbol{M}_i$, for $1 \leq i \leq n$:*
   - *$S_i \in \mathcal{S}$, i.e. S_i is defined at* C*; otherwise*
   - *$S_i \in \boldsymbol{meth}(\mathrm{C}')$, and $\nexists S_j$ in $\mathcal{S}$ such that $\boldsymbol{M}_i = \boldsymbol{M}_j$.* □

Hereafter, we only consider class declarations that may contain methods. Thus, our class hierarchy, and our OR schema, from now on will be defined using this new notion of class declaration.

Following this, let us define *path expressions* (or, short, *paths*). Paths are a distinctive feature of object-oriented languages that enable us to identify an object by expressing how to navigate to it. Here, we extend this to relate to objects and tuples.

3.1.3 Formal treatment of paths

Objects or tuples can be related to each other through paths connecting two or more properties. For instance, from Example 2 one can observe intuitively that an employee is related to his/her division through a path **dept•div**.

Before we define a path, we need to outline the concept of a link. Links represent the relationships between classes, relations, and each other. A link is formally defined as:

Definition 20 (*Link between Classes/Relations) Let* $\mathsf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be a schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{K}$, *and* $\mathcal{M}$. *Also let* C_1, C_2, ...,C_n *be class names in* $\mathcal{C}$ *or relation names in* $\mathcal{R}$. *We can identify three kinds of links between class names, relation names, and each other:*

- ***Reference Link:*** *Let* $\boldsymbol{p}{:}\tau \in \boldsymbol{type}(\mathrm{C}_1)$.
  - *If* $\tau = \natural\mathrm{C}_2$ *(or* τ *is a collection of the reference type* $\natural\mathrm{C}_2$*), then we say that there is a reference link between* C_1 *and* C_2. *The notation* $\boldsymbol{p}{:}\mathrm{C}_1 \twoheadrightarrow \mathrm{C}_2$ *denotes the link between* C_1 *and* C_2 *through property* $\boldsymbol{p}$.
  - *If* $\tau = \natural\mathrm{C}_2/\natural\mathrm{C}_3/.../\natural\mathrm{C}_n$ *(or* τ *is a collection of the reference type* $\natural\mathrm{C}_2/\natural\mathrm{C}_3\ /\ldots/\ \natural\mathrm{C}_n$*), then we say that there is a reference link between* C_1 *and* C_2, *or* C_1 *and* C_3, ..., *or* C_1 *and* C_n. *The notation* $\boldsymbol{p}{:}\mathrm{C}_1 \twoheadrightarrow \mathrm{C}_2/\mathrm{C}_3/.../\mathrm{C}_n$ *denotes the link between* C_1 *and* C_2, *or* C_1 *and* C_3, ..., *or* C_1 *and* C_n *through property* $\boldsymbol{p}$.
- ***Foreign Key Link:*** *If there is a foreign key declaration* $(\mathrm{FK}, \mathrm{C}_1, \boldsymbol{FK}(\mathrm{C}_1), \mathrm{C}_2, \boldsymbol{K}(\mathrm{C}_2))$ *in* C_1 *such that* C_1 *refers to* C_2 *through* FK, *then* $\mathrm{FK}{:}\mathrm{C}_1 \twoheadrightarrow \mathrm{C}_2$ *is a foreign key link between* C_1 *and* C_2.
- ***Inverse Link:*** *If* C_2 *has a link* $\ell{:}\mathrm{C}_2 \twoheadrightarrow \mathrm{C}_1$, *such that* ℓ *is a reference link or a foreign key link, then the inverse of* ℓ *is a link between* C_1 *and* C_2, *and it is represented by:* $\ell^{-1}{:}\mathrm{C}_1 \twoheadrightarrow \mathrm{C}_2$. □

Example 13 ______________________

Here we give some examples of links, based on Examples 2 and 7.

Reference link	***dept***:EMPLOYEE$\twoheadrightarrow$DEPARTMENT
	div:DEPARTMENT$\twoheadrightarrow$DIVISION
Foreign Key link	FK_1:PRODUCT$\twoheadrightarrow$CATEGORY
Inverse link	***dept***$^{-1}$:DEPARTMENT$\twoheadrightarrow$EMPLOYEE

We identify two kinds of paths: a *value path*, and a *reference path*. These are defined as:

Definition 21 (*Value Path) Let* $\mathbf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be a schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{K}$, *and* $\mathcal{M}$. *Given the links:* ℓ_1:$\mathrm{C}_1 \twoheadrightarrow \mathrm{C}_2$, ℓ_2:$\mathrm{C}_2 \twoheadrightarrow \mathrm{C}_3$, ..., ℓ_{n-1}:$\mathrm{C}_{n-1} \twoheadrightarrow \mathrm{C}_n$, *where* C_1, C_2, ..., C_n *are class names in* $\mathcal{C}$ *or relation names in* $\mathcal{R}$. *Thus,* $\varrho = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \boldsymbol{p}$ *is a value path of* C_1 *iff* $\boldsymbol{p}{:}\tau \in$ ***type***(C_n), *and* τ *is not a reference type. This means that the instances of class (or relation) name* C_1 *can be related to the typed value* $\boldsymbol{w}$ *(of type* τ*) of some property name from another class (or relation) name* C_n *through the value path* $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \boldsymbol{p}$. □

Definition 22 (*Reference Path) Let* $\mathbf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be a schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{K}$, *and* $\mathcal{M}$. *Given the links:* ℓ_1:$\mathrm{C}_1 \twoheadrightarrow \mathrm{C}_2$, ℓ_2:$\mathrm{C}_2 \twoheadrightarrow \mathrm{C}_3$, ..., ℓ_{n-1}:$\mathrm{C}_{n-1} \twoheadrightarrow \mathrm{C}_n$, *where* C_1, C_2, ..., C_n *are class names in* $\mathcal{C}$ *or relation names in* $\mathcal{R}$. *Thus,* $\varrho = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$ *is a reference path of* C_1. *This means that the instances of class (or relation) name* C_1 *can be related to the instances of another class (or relation) name through the reference path* $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$. □

In Example 2, we can see that **dept** and **dept • div** are reference paths (of class name EMPLOYEE), and **dept • div • divName** is a value path (of class name EMPLOYEE).

Definition 23 (*Valid Path) Let* $\mathbf{S} = (\mathbf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ *be a schema defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{K}$, *and* $\mathcal{M}$. *Given the links:* ℓ_1:$\mathrm{C}_1 \twoheadrightarrow \mathrm{C}_2$, ℓ_2:$\mathrm{C}_2 \twoheadrightarrow \mathrm{C}_3$, ..., ℓ_{n-1}:$\mathrm{C}_{n-1} \twoheadrightarrow \mathrm{C}_n$, *where* C_1, C_2, ..., C_n *are class names in* $\mathcal{C}$ *or relation names in* $\mathcal{R}$. *We say that a path is a valid path when it is a value path, or it is a reference path* $\varrho = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$ *such that* ℓ_{n-1}:$\mathrm{C}_{n-1} \twoheadrightarrow \mathrm{C}_n$ *is a reference link or an inverse link of a reference link.* □

Hereafter, we only consider valid paths. We can also identify *single-valued path* and *multi-valued path*. A path is said to be single-valued when every link of the path is only formed by single-valued properties, or by foreign key links, or by inverse links formed by single-valued

properties. In this case, each link in the path is a single-valued link. A path is denoted multi-valued when at least one of the links of the path is formed by a multi-valued property or by an inverse link formed by a multi-valued property. In this case, this link is a multi-valued link. In this work, for simplicity, we consider only single-valued paths, and a special kind of multi-valued paths, when all links of the path are single-valued except for the last one.

Here we adopt the following notation with respect to paths:

Let $\mathbf{S} = (\mathsf{S}, \mathcal{H}, \hat{\mathcal{R}}, \hat{\mathcal{K}})$ be a schema defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{T}$, $\mathcal{K}$, and $\mathcal{M}$, and $\boldsymbol{D}(\mathbf{S}) = (\hat{\mathcal{O}}, \mathcal{W}_t)$ an instance of $\mathbf{S}$. Given the links: ℓ_1:C$_1 \twoheadrightarrow$C$_2$, ℓ_2:C$_2 \twoheadrightarrow$C$_3$, ..., ℓ_{n-1}:C$_{n-1} \twoheadrightarrow$C$_n$, where C$_1$, C$_2$, ..., C$_n$ are class names in $\mathcal{C}$ or relation names in $\mathcal{R}$. Also let $\mathcal{W}$ be a set of typed values.

- Given the value path $\varrho = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \boldsymbol{p}$, with $\boldsymbol{p}{:}\tau \in$ $\boldsymbol{type}(\mathrm{C}_n)$, then $\mathbf{o}_1 \bullet \varrho$ assigns a constant, a structural typed value, a collection of constants, or a collection of structural typed values $\boldsymbol{w} \in \mathcal{W}$ to each $\mathbf{o}_1 \in \boldsymbol{D}(\mathrm{C}_1)$, such that there exist an object identity $\mathbf{o}_n \in \boldsymbol{D}(\mathrm{C}_n)$ where $\mathbf{o}_n \bullet \boldsymbol{p}_n = \boldsymbol{w}$, and there are the object identities (or tuples) $\mathbf{o}_2$, $\mathbf{o}_3$, ..., $\mathbf{o}_{n-1}$, with $\mathbf{o}_i \in \boldsymbol{D}(\mathrm{C}_i)$, $2 \leq i \leq n$, where $\mathbf{o}_i$ is related with $\mathbf{o}_{i+1}$ through the link ℓ_i, for $1 \leq i \leq n-1$. The notation $\mathbf{o} \bullet \varrho|_{\boldsymbol{D}(\mathbf{S})}$ denotes the value of path ϱ of the object identity (or tuple) $\mathbf{o}$ in state $\boldsymbol{D}(\mathbf{S})$.
- Given the reference path $\varrho = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$, then $\mathbf{o}_1 \bullet \varrho$ assigns an object identity (or a collection of object identities) $\mathbf{o}_n \in \boldsymbol{D}(\mathrm{C}_n)$ to each $\mathbf{o}_1 \in \boldsymbol{D}(\mathrm{C}_1)$, such that there are the object identities (or tuples) $\mathbf{o}_2$, $\mathbf{o}_3$, ..., $\mathbf{o}_{n-1}$, with $\mathbf{o}_i \in \boldsymbol{D}(\mathrm{C}_i)$, $2 \leq i \leq n$, where $\mathbf{o}_i$ is related with $\mathbf{o}_{i+1}$ through link ℓ_i, for $1 \leq i \leq n-1$. The notation $\mathbf{o} \bullet \varrho|_{\boldsymbol{D}(\mathbf{S})}$ denotes the value of path ϱ of the object identity (or tuple value) $\mathbf{o}$ in state $\boldsymbol{D}(\mathbf{S})$.

3.2 Comparison with Other Models

In the development of ORDMs, several approaches have been adopted, such as: a) create one from scratch; b) extend an Object-Oriented Data Model (OODM) with relational characteristics, such as integrity constraints, views, etc.; and c) incrementally add object features into a Relational Data Model (RDM). The main vendors of relational databases opted for the latter. We try to support the main characteristics of both the RDM standard and the Object Data Management Group (ODMG-3) standard. In the following text, we examine and note the similarities and differences between our model and other models.

3.2.1 Comparison with RDM

Our model can be used as a relational one, since we cover the main concepts of this model, as can be observed in Table 3.1.

Table 3.1: Comparison with RDM terminology.

Relational Model [5]	**Our model**
Relation schema	Relation declaration
Attribute	Property
Key (of a relation schema)	Key declaration (of class or relation)
Foreign key (of a relation schema)	Foreign key declaration (of class or relation)
Relation	Relation's extent
Tuple	Tuple

However, we augment the relational model as follows:

1. Here we allow more complex types than in a traditional relational model, for instance, collections and reference types (see Definition 1).

2. The value of a property can be constants as well as references to an object, collections (set, list or array) or structural values (tuples) (see Definition 9).

3. The properties of a relation can be related with properties of other relations or classes through paths (see Definitions 21 and 22).

Please note that general integrity constraints other than those of entity integrity constraint, and referential integrity constraint, such as check conditions are not discussed in our proposal.

3.2.2 Comparison with OODM

Regarding the object model, it is difficult to make a detailed comparison, because the terminology used in an OO environment differs from model to model, even though they are based on the same basic principles. We compare our model to the standard model described in ODMG-3 (Cattell *et al.*, 2000), though our terminology is somewhat different, as we can see in Table 3.2.

[5]The relational nomenclature is mainly that of (Elmasri & Navathe, 2006).

[6]Our reference path corresponds to one side of an ODMG-3's relationship, and there are not value paths in ODMG-3.

[7]ODMG-3 does not deal with foreign keys, only keys of a class, and like us, it does not require classes with one key. Only relations, in our model, are required to have at least one key.

[8]In ODMG-3, method is an implementation of an operation.

Table 3.2: Comparison with ODMG-3 terminology.

ODMG-3 Model	**Our model**
Class	Class declaration
Property	Property
Attribute	Property
Reference attribute	Property (reference path)
Relationship	Property (reference path)[6]
Object	Object
Literal	(Typed) value (except the reference typed value)
object/literal type	Type
Extent of a type/class	Class extent
Extends relationship	**is-a** relationship
Key (of a type/class)[7]	Key declaration (of a relation or class)
Operation	Method[8]
Operation signature	Method signature
Path expressions	Reference paths or Value paths

Below, we point out some differences:

1. The collection of types supported here is a subset of types described in ODMG-3. Namely we do not consider the types bag, dictionary, time, and timestamp.

2. In ODMG-3, Objects can have a name. Moreover, ODMG-3 deals with issues about creation, and lifetimes of objects. We do not.

3. In ODMG-3 an attribute, when defined in an interface, can mean a method, while in our model methods and properties are completely different.

4. Our **is-a** relationship is similar to the "*Extends*" relationship in ODMG-3. Both deal with single inheritance, but ODMG-3 also supports multiple inheritance of object behaviour through its "*type-subtype*" relationship (in interfaces).

5. In ODMG-3, more specifically in OQL, it is possible to navigate from one object to another using path expressions, which follow simple relationships. Our paths allow crossing from an object (or tuple) to another using reference types and/or foreign keys.

6. The operations in ODMG-3 have error handling. We did not define any form of exception handling.

7. In ODMG-3 literals are embedded in objects, and cannot be individually referenced. In our model tuples can be embedded in objects too, and in this case they cannot be individually referenced either. However, tuples can exist independently of objects, and thus they can be referenced through foreign keys.

8. ODMG-3 does not deal with foreign keys, only with keys of a class in an indirect way.

Note that our object representation, as occurs on most of the Object-Oriented Database (OODB) systems, supposes one most-specific type per object. Thus, it is not possible in our model for an object to be an instance of multiple classes (*multiple classification* (Kuno *et al.*, 1995)), which are not related to each other through a class hierarchy.

3.2.3 Comparison with ORDM

Regarding the ORDM, we only found one formal model proposed in (Darwen & Date, 1995; Date & Darwen, 2000). This model reinterprets the RDM in an OO vision. The authors matched relational domains to OO classes, and added two OO features to their model: user-defined types (and user-defined operators) and type inheritance.

Their formalism is innovative in various ways. For example, a relation is defined by two sets: one specifies the schema of the relation (their "*heading*") whereas another contains the tuples corresponding to "*heading*" (their "*body*"). They distinguish actual data representations from possible data representations, where more than one possible representation can be specified. Further, they also make distinctions between value and variable, where, basically, a value is an individual constant (i.e., immutable and without location in time or place), while variables are holders of values in a given time and place, like in programming languages.

This model is similar to ours in some aspects. For instance, it plainly identifies structure and content, but, there are some differences too. Table 3.3 outlines some features of our model comparing it with the model in (Date & Darwen, 2000).

Table 3.3: Comparison with the Data and Darwen model.

Features	**Our model**	**(Date & Darwen, 2000)**
Type	atomic, reference, structural, collection	scalar, tuple, relation, collection
Value	atomic, reference, structural, collection	scalar, tuple, relation, collection
Typed values	yes	yes
Variables	only relations and classes	scalar, tuple and relation (relvar) variables
Candidate keys	yes	yes
Paths	yes	only with Foreign Key links
Integrity constraints	partially	yes
Type hierarchy	single	multiple
Methods/Operations	yes	yes
Notion of schema	yes	yes[9]

[9]They call it database, such that an instance of a database is a set of relvars.

There are some entries in Table 3.3 that merit discussion. Firstly, the entry about some kinds of types in Date and Darwen's model. The *scalar type* (*boolean, integer, rational*, and *character*) is equal to our *atomic type*. The *tuple type* is equal to our *structural type*. The *relation type* is a set of *tuple types*, and is not in our model. The *collection types* are similar to ours, but the authors do not include them in the core of the model. The authors suggest *collection types* as something that should be supported, but only if they are absolutely necessary.

The entry concerning values: In Date and Darwen's model no nulls or object identifiers are allowed, since nulls and object identities are not values. Inasmuch as our model is to be used to design the models in a DW environment, we believe that both must be supported. Regarding nulls, in data warehousing huge data is acquired from multiple sources, possibly autonomous and heterogeneous. Thus, *null* value, representing unknown value, will be allowed, even though its widespread use must be advised against. Regarding object identities, it is generally known that they are efficient and fast. We need only navigate by way of path expressions instead of effecting a join of data; and they are independent of the property domains.

The entry concerning variables: We do not define them explicitly, because we believe that our classes and relations, which work like variables in some way, are enough for us. More specifically, a relvar is for a relation type just as our relation extent is for our relation declaration.

The entry concerning keys: We allow classes and relations to specify keys, but keys in a class are optional, because, in any case, the object identity determines the essential uniqueness of the object. Date and Darwen's model forces all relvars to have at least one key (named *candidate key*). There, keys are defined when relvars are defined, and have uniqueness and minimality properties. In our work, we do not force the minimality property, but we embolden it.

The entry concerning paths: We support navigation across links by reference and by foreign key. This is the novelty of our model, and an important characteristic. We can cross independently of whether the link is for a relation or for a class. Furthermore, in the same link we can refer to more than one class or more than one object in the same class. This is a valuable feature in our context, and a feature that is not present in Date and Darwen's model.

The reference to "partially" that concerns the entry about our integrity constraints: In Date and Darwen's model an *integrity constraint* is a well formed formula of the relational calculus. Their constraints are categorised into *type* (for scalar/atomic types), *attribute, relvar*, and *database* constraints. *Type constraints* specify, precisely, a definition of the set of values that constitute a given scalar type. For instance, it is possible to specify that a given property

identity, whose type is *string*, will have a length of 9 characters, and its value will start with the letters 'ID'. We do not deal with it. *Attribute, relvar*, and *database* constraints are constraints on the values that a given attribute, relvar, and database, respectively, is allowed to assume. In one sense, we deal with attribute constraints since all our values are typed. *Relvar constraints* concern individual relvars, and are expected to be expressed in terms of the relvar in question only. For instance, every employee must have a wage greater than 100. Our *entity integrity* constraint can be seen as a kind of relvar constraint. *Database constraints*, in the other hands, interrelates two or more distinct relvars. For instance, every employee of the department *training* must have a wage greater than 200. We only deal with a particular kind of database constraint, that is our *integrity referential* constraint.

The entry concerning type hierarchy: Like us, Data and Darwen's model does not adopt multiple classification, and each value only belongs to its more specific type. Unlike us, in their model only behaviour, and constraints are considered. They think that inheritance of structure is an implementation matter since that refers to inheritance of physical representation and, as such, not part of the model. In such a way, their types are treated like black boxes where the users (or programs) are accessed by operators only. We prefer to assume, in accordance with the object world, that inheritance means inheritance of structure and behaviour. Another distinction between our models in respect to type hierarchy is that we deal with classes only, excluding relations and predefined types. We think that type hierarchy between predefined types can be useful, but is outside the scope of our investigation.

The entry concerning methods: We explicitly define user-defined methods for classes, but, unlike Date and Darwen's model, we do not mention built-in methods for predefined types, but we assume that they exist.

Date and Darwen also proposed in (Date & Darwen, 2000) an algebra (named A-algebra) consisting of first order logic operators used to express several classes of queries in object-relational databases (Bahloul *et al.*, 2004). This algebra was later extended by Bahloul, Amghar and Sayah in (Bahloul *et al.*, 2004), named A^*-*algebra*, which "allows the manipulation of complex entities requiring symbolic representation in their definitions such as in the geometrical, and spatial databases" (Bahloul *et al.*, 2004). We do not propose any algebra for our model.

3.2.4 Comparison with commercial ORDMs

Table 3.4 summarises some of the differences between our model and commercial ORDMs: BD2 (DB2, n.d), Informix (Informix11, 2009; Informix11, 2008), Oracle (Belden & Greenberg, 2008), Ingres (Ingres92, 2008b; Ingres92, 2008a), PostgreSQL (PostgreSQL, n.d), Firebird (Firebird, n.d.), and SQL Server (documentation team, 2009). Each entry in this Table, with exception of the *Inheritance* column, is a “Y” (Yes), a “N”(No), or a “P” (Partially), indicating whether the model has the corresponding characteristic.

Table 3.4: Comparison of some ORDMs.

Model	Complex object	Inheritance	Triggers	Data types extension	Views
DB2 9.5	P	Single	Y	Y	Y
Informix IDS 11.5	P	Single	Y	Y	Y
Oracle 11g1	Y	Single	Y	Y	Y
Ingres 9.2	N	N	Y	Y	Y
PostgreSQL 8.4	Y	Multiple	Y	Y	Y
Firebird 2.1.2	N	N	Y	N	Y
SQL Server 2008	P	N	Y	Y	Y
Our model	Y	Single	N	Y	Y

In Table 3.4, we take into account the following elements: whether complex objects are supported; whether the model supports single or multiple inheritance; whether the model supports base data type extension; whether the model supports triggers, and whether the model supports views. These are, according to (Stonebraker & Brown, 1999), the fundamental characteristics of an ORDM.

There are some entries in Table 3.4 that merit discussion. Firstly, the entry, where “Partially” is recorded, concerns complex objects in DB2, Informix-IDS, and SQL Server. DB2 version 9.5 has a constructor to reference type (there it is named *rowid* type), to a structural type (there it is called *row type*), and offers a set of pre-defined extensions addressing video, audio, spatial data type, to mention a few. However, there is no collection type constructor, and it is not clear if it allows a structural type inside another one. Informix-IDS has constructors for complex data type such as structural type (there it is named *row type*), set type, and list type. However, it does not have a constructor to reference type, although it offers limited support to do references to the physical location of a *row* in a *table* (here the table is said be a relation declaration) through *rowids*. SQL Server deals with complex data types, such as: reference (there, it is named *ID*), xml, spatial (called *geography* and *geometry*), and data types to store

external files (named *filestream data type*) such as: audio, video, image, and so forth. However, there is no support for structural type or for collections.

Two entries concerning complex objects should still be commented on: "No" in Ingress, and "Yes" in PostgreSQL. In Ingress 9.2 it only allows Universal Unique IDentifiers (UUIDs) in *tables* like universal keys in the whole database. Hence, "No" is the appropriate entry in the table. In PostgreSQL 8.4, although we put a "Yes" in complex object entry, its current implementation of *object identifier type* (we call it reference type) is not large enough to provide uniqueness in large databases or even in a large individual table.

The entry concerning triggers in our model: In this case, we decided to simplify our language, and did not include triggers, since this functionality can be specified through behaviour (i.e., methods). Even if the source information is purely relational and implemented with triggers, we always can draw it like an object-relational.

The entry concerning data types extension in Firebird does not support user-defined types, but only deals with user-defined functions.

Finally, the entry regarding view in our model: in this Chapter we only present the language to describe schemata. In our framework, views are part of a distinct schema, which is defined using the language to describe perspective schemata that will be discussed in Chapter 4.

3.3 Conclusions

This Chapter described a formal language to define schemata, which is general enough to encompass the components of any OODM and RDM. The idea was to provide a rich language that serves as a common dialect to design the schemata of a DIS, without introducing great changes in the native schemata.

Also, this Chapter briefly compared our model with other academic and market models. This study served to identify similarities (and differences) between the concepts of our model and of the other models, as well as the flexibility of our language. Some data types (e.g., *time*, *timestamp*, *point*, and *polygon*), used in applications such as geographical and temporal, were not considered in this work. However, they can be easily added as base types. Also, our language does not include temporal aspects such as *valid-time* and *transaction-time*, which can be useful in the DW environment.

In the next Chapter, we will show the language that defines mappings between the schema's components in both instance and structure levels. This language can also be used to define different point of views of single or multiple information source.

4

The Language $\mathbf{L}_{PS}$

As well as providing rich constructors, having expressiveness or semantic power, from a conceptual viewpoint, models for data integration environment should:

- *allow the design of schemata*[1] *that represent different points of view for one or more schemata;*
- *support the mapping of structures between schemata; and*
- *relate instances of different schemata that represent the same entity in the real world.*

In our proposal, this is achieved by using the perspective schema language (L_{PS}), which is an extension of language L_S (shown in Chapter 3).

In this Chapter, we focus on showing the perspective schema language. We start by introducing a running example that will facilitate the explanation of the components of the language. Then, we present an overview of L_{PS}, followed by a more detailed explanation. Also, we show the usability of L_{PS} through some examples. Finally, we finish by pointing out the new features of the approach presented here and in on-going or planned future work on this topic.

4.1 A Running Example

Along the remainder of the text, consider a simple sales scenario comprising two data sources S_1 and S_2, a reference model **RM**, and a data warehouse **DW**. S_1 and S_2 include information about product sales in different shops of the same organisation. Part of their schemata is illustrated in Figure 4.1.

[1] *Known in literature as view schemata (Wrembel, 2000).*

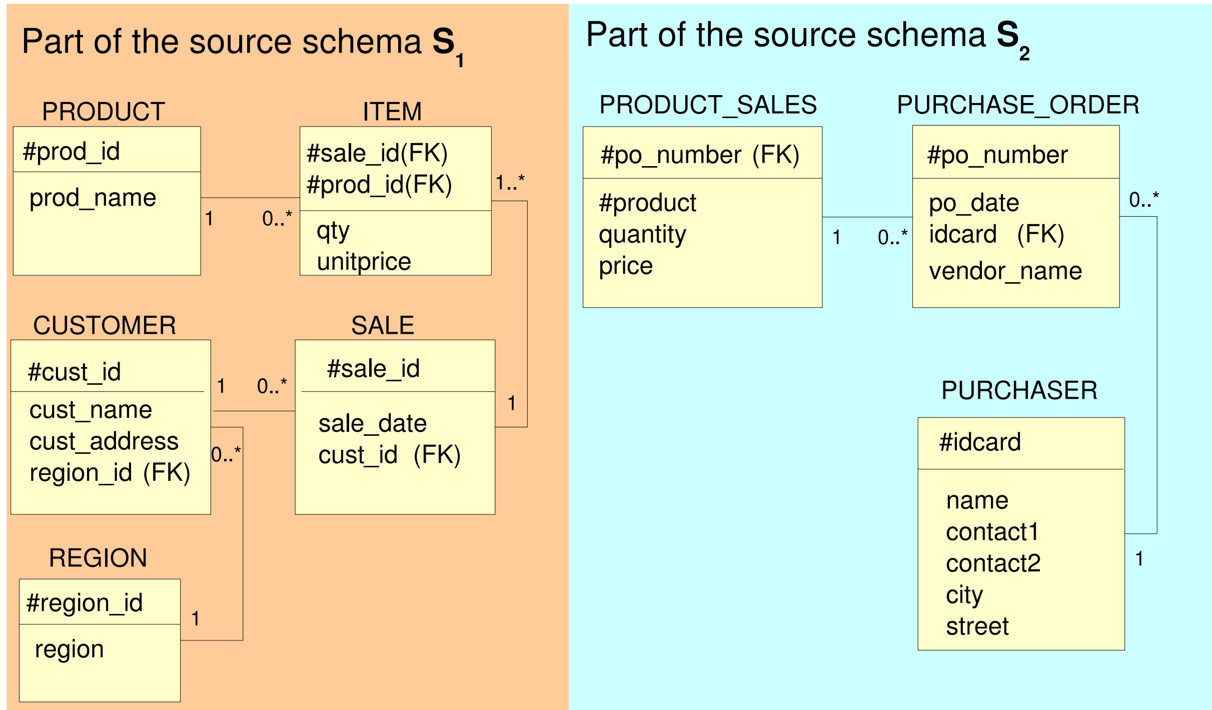


Figure 4.1: Motivational example: source schemata $\mathbf{S}_1$ and $\mathbf{S}_2$.

We use a notation based on the Unified Modeling Language (UML) class diagram. Classes/relations are rectangles, with their names at the top. All properties that are key to a relation are shown in Figures 4.1, 4.2, and 4.3 using "#" before their names, and are separated from the other properties by a horizontal line. Foreign keys are indicated by "(FK)" after the property names.

In $\mathbf{S}_1$ we have the relations: PRODUCT, ITEM, SALE, CUSTOMER, and REGION. PRODUCT contains information about the products: its identifier (**prod_id**), and its name (**prod_name**). ITEM stores the products sold (**prod_id**) in a given sale (**sale_id**), with the quantity sold (**qty**), and the unit price (**unitprice**) stored in dollars. SALE keeps the details of all orders: its identifier (**sale_id**), the date that the order was placed (**sale_date**), and the sales customer (**cust_id**). CUSTOMER contains information about sales customers: his/her identifier (**cust_id**), his/her name (**cust_name**), his/her address (**cust_address**), his/her region (**region_id**). REGION stores information of the region of the customers: the region identifier (**region_id**) and the region description (**region**). In $\mathbf{S}_2$ we have the relations PRODUCT_SALES, PURCHASE_ORDER, and PURCHASER. PRODUCT_SALES holds information about items sold (**product**, **quantity**, and **price**) in a purchase order (**po_number**), as well as information logically related to products themselves. PURCHASER_ORDER contains details of the orders: number of the order (**po_number**), sale date (**po_date**), and sales customers (**idcard**), including the salesperson's

name (**vendor_name**). PURCHASER stores all the main customer information: his/her identity card (**idcard**), his/her name (**name**), his/her telephone numbers (**contact1**, and **contact2**), his/her address (**city**, and **street**).

The reference model **RM** includes all information of interest to the organisation. Part of its schema is illustrated in Figure 4.2. Some properties of the reference model are in bold script because they will be used in the next Section.

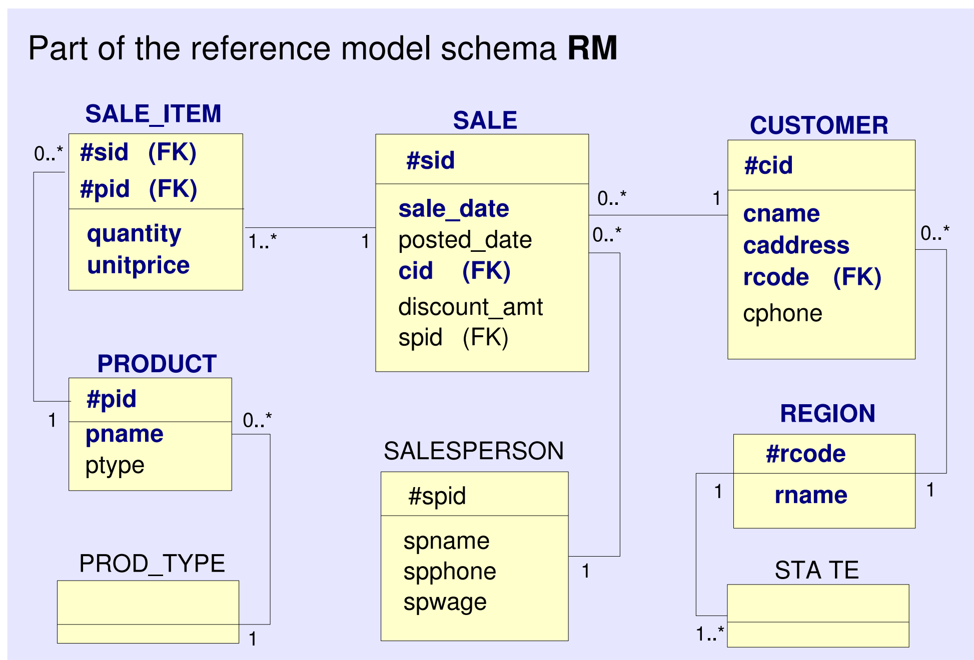


Figure 4.2: Motivational example: the reference model.

The part of the reference model that is of interest to the present work is formed by the relations: PRODUCT, SALE_ITEM, SALE, SALESPERSON, CUSTOMER, and REGION. PRODUCT contains information about the products: its identifier (**pid**), its name (**pname**), and its type (**ptype**). SALE_ITEM stores the products sold (**pid**) in a given sale (**sid**), with the quantity sold (**quantity**), and the unit price (**unitprice**) stored in euros. SALE keeps the details of all sales: its identifier (**sid**), the date that the sale was placed (**sale_date**), the date on which the sale was posted (**posted_date**), the sales customer (**cid**), the amount of sales discount (**discount_amt**), and the salesperson (**spid**). CUSTOMER contains information about sales customers: his/her identifier (**cid**), his/her name (**cname**), his/her address (**caddress**), his/her region (**rcode**), and his/her contact (**cphone**). REGION stores information of the region of the customers: the region identifier (**rcode**) and the region description (**rname**). SALESPERSON contains information about salespeople: his/her identifier (**spid**), his/her name (**spname**), his/her contact (**spphone**), and his/her salary (**spwage**).

The data warehouse schema **DW** contains, besides historical information about customers (CUSTOMER), products (PRODUCT) and vendors (VENDOR), summarised information regarding sales (SALES_BY_CUSTOMER and SALES_BY_VENDOR). Part of its schema is illustrated in Figure 4.3.

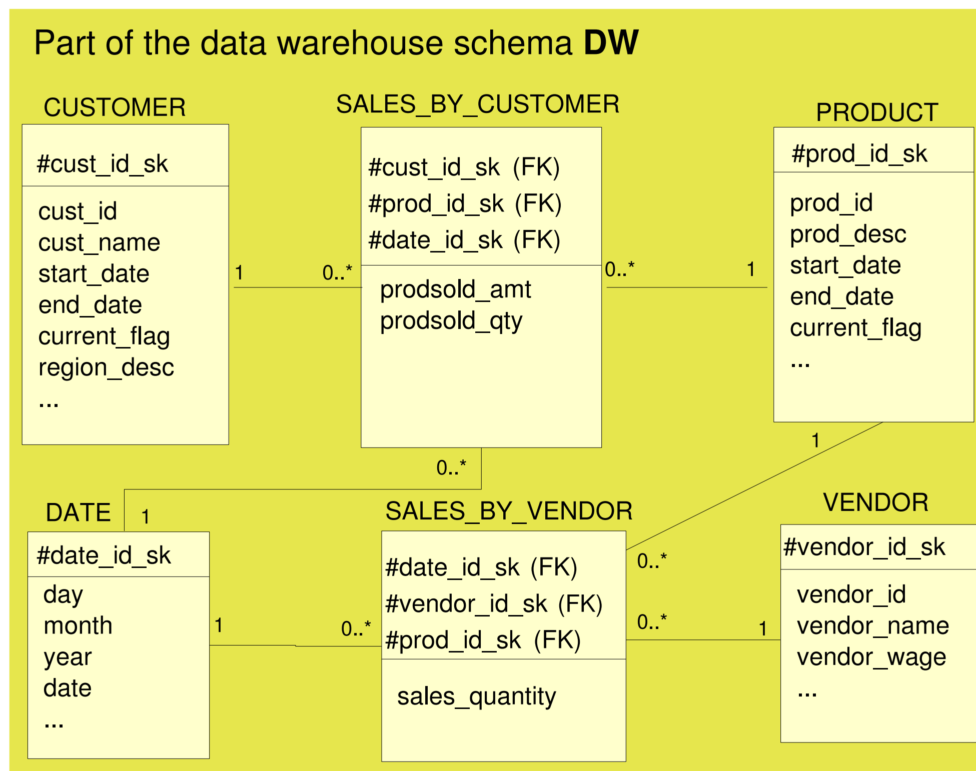


Figure 4.3: Motivational example: the data warehouse schema.

The relation **DW**.SALES_BY_CUSTOMER stores the amount (**prodsold_amt**$_{\text{DW}}$) and the total quantity (**prodsold_qty**) of each product (**prod_id_sk**) sold per customer (**cust_id_sk**) and per sale date (**date_id_sk**). The relation **DW**.SALES_BY_VENDOR stores the total (**sales_quantity**) of products sold (**prod_id_sk**) per employee (**vendor_id_sk**) and sale date (**date_id_sk**). The historical data, in relations PRODUCT and CUSTOMER, is kept using three properties: **start_date**, **end_date**, and **current_flag**. Both **Start_date** and **end_date** indicate the historical range of when each tuple was current. **Start_date** stores the data when the tuple was inserted into the relation. **End_date** stores the date when the tuple is no longer current, or is set to an arbitrary time far in the future when the tuple has been recently updated. **Current_flag** stores true or false indicating if the tuple is the current one or not. It is simply a convenient way to retrieve all the most current records in a relation/class (Kimball & Caserta, 2004, Ch. 5, p. 191). The properties **prod_id_sk**, **cust_id_sk**, **vendor_id_sk**, and **date_id_sk** are surrogate keys automatically generated by the system.

4.2 Overview

The language L_{PS} is used to define perspective schemata. A perspective schema describes a data model wholly or in part (*the target*) in terms of other data models (*the base*). A perspective schema is formed by the following components:

1. *Name* is a schema name with the notation: $\mathbf{P}_{\mathbf{B}|\mathbf{T}}$, being **B** the name of one or more base schemata and **T** the name of the target schema. For example, in Figure 4.4, $\mathbf{P}_{s1|RM}$ is the name of a perspective schema whose base is $\mathbf{S}_1$ and the target is **RM**;

2. *"Require" declarations* express the subset of the components of the target schema (classes, relations, keys, and foreign keys) that will be necessary in the perspective schema;

3. *Matching Function signatures* indicate which matching functions must be implemented to determine when two objects/tuples are distinct representations of the same object in the real-world;

4. *Correspondence Assertions* establish the semantic correspondence between schemata's components.

5. *View Relation declarations* explicitly define components in the perspective schema whose set of instances is formed by instances from relations, classes, or other view relation declarations.

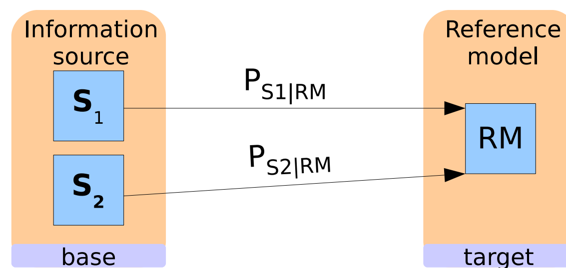


Figure 4.4: Perspective schema.

4.2.1 "Require" declarations

The target schema may have much more information than is required be represented in a perspective schema, namely when the target schema is the reference model. Hence, it is required

to clearly indicate which elements of the target schema are in the scope of the perspective schema.

$$\text{require}\,(<\text{name}>)\,,\text{or} \qquad (4.1)$$

$$\text{require}\,(<\text{name}>,<\text{properties}>)\,, \qquad (4.2)$$

being that <name> in (4.1) can be the name of a key or of a foreign key, while in (4.2) <name> can be the name of a class or relation. <properties> is a non-empty set of property names belonging to the class or relation indicated in <name> .

For example, consider the perspective schema $\mathbf{P_{S_1|RM}}$ between the schemata **RM** (the target) and $\mathbf{S}_1$ (the base), which are presented, respectively, in Figures 4.2 and 4.1. For this perspective schema, the following relations from **RM** are needed[2]:

- require(PRODUCT,{**pid**$_{\mathrm{RM}}$,**pname**$_{\mathrm{RM}}$})
- require(SALE_ITEM, {**sid**$_{\mathrm{RM}}$, **pid**$_{\mathrm{RM}}$, **quantity**$_{\mathrm{RM}}$, **unitprice**$_{\mathrm{RM}}$})
- require(SALE, {**sid**$_{\mathrm{RM}}$, **sale_date**$_{\mathrm{RM}}$, **cid**$_{\mathrm{RM}}$})
- require(CUSTOMER, {**cid**$_{\mathrm{RM}}$, **cname**$_{\mathrm{RM}}$, **caddress**$_{\mathrm{RM}}$, **rcode**$_{\mathrm{RM}}$})
- require(REGION, {**rcode**$_{\mathrm{RM}}$, **rname**$_{\mathrm{RM}}$})

Note that, for example, the properties: **ptype**$_{\mathrm{RM}}$ from PRODUCT, **posted_date**$_{\mathrm{RM}}$, **discount_amt**$_{\mathrm{RM}}$, and **spid**$_{\mathrm{RM}}$ from SALE, and **cphone**$_{\mathrm{RM}}$ from CUSTOMER are not considered as being required. The properties considered as required are presented in bold in Figure 4.2.

The "Require" declarations, besides allowing the selection of a set of structures of the target schema, have an important role in the verification of mapping between schemata. Once all classes, relations, properties, keys and foreign keys are acknowledged, it is easy to verify if all mappings were done or not. More details about this are shown later in this Chapter.

4.2.2 Matching function signatures

In a data integration environment, there is a need to combine information from the same or different sources. This involves comparing the instances from the sources and attempts to

[2]In order to clarify the reading of this Chapter, all the names of components of $\mathbf{S}_i$ will be followed by number "i", with i ∈ {1,2}, the names of components of **RM** are followed by letters "RM", and the names of components of **DW** are followed by letters "DW".

determine when two instances of different schemata refer to the same real-world entity (the *instance matching* problem).

We do not propose to resolve the instance matching problem. Here, we only use matching function signatures, which point to situations that should be considered in a data integration context. They are useful for a designer to know what matching function must be implemented when dealing with an ETL process. The matching function signatures define a 1:1 correspondence between the objects/tuples in families of corresponding classes/relations. In particular, this work is based on the following matching function signature:

$$\textbf{match} : ((\mathsf{S}_1\,[\mathrm{R}_1]\,,\tau_1) \times (\mathsf{S}_2\,[\mathrm{R}_2]\,,\{\tau_2\})) \rightarrow \text{Boolean}, \tag{4.3}$$

where S_i are schema names, R_i class/relation names in S_i, and τ_i the data type of the instances of R_i, for i $\in$ {1,2}. This signature indicates that the relationship between R_1 and R_2 should be checked at an instance level in order to verify if there are instance matching situations or not. When a call is done to **match**, it can be in one of two ways:

1. Both arguments are instantiated (i.e., an instance of $\mathsf{S}_1.\mathrm{R}_1$ and one instance of $\mathsf{S}_2.\mathrm{R}_2$ are given to be tested). In this case, **match** verifies whether two instances are semantically equivalent or not (i.e., if they represent the same instance of the real world or not); returning *true* when they are semantically equivalent and *false* otherwise.
2. Only the first argument is instantiated (i.e., $\mathsf{S}_1.\mathrm{R}_1$), then it obtains the semantically equivalent $\mathsf{S}_2.\mathrm{R}_2$ instance of the given $\mathsf{S}_1.\mathrm{R}_1$ instance; returning *true* when it is possible, and *false* when nothing is found or when there is more than one instance to match.

In some scenarios one-to-many correspondence between instances is common (e.g., when historical data is stored in the DW). In this case, a variant of **match** should be used to guarantee the one-to-one correspondence between instances. In this case **match** has the following syntax signature:

$$\textbf{match} : ((\mathsf{S}_1\,[\mathrm{R}_1]\,,\tau_1) \times (\mathsf{S}_2\,[\mathrm{R}_2\,(\textsf{\textbf{predicate}})]\,,\{\tau_2\})) \rightarrow \text{Boolean}. \tag{4.4}$$

In (4.4), **predicate** is a Boolean condition that determines the context in which instance matching must be applied in $\mathsf{S}_2.\mathrm{R}_2$. Some examples of matching function signatures involving schemata of Figures 4.1 and 4.2 are presented in Figure 4.5.

match:(($\mathbf{S}_1$ [PRODUCT],τ_1)×(**RM**[PRODUCT],{τ_2}))→Boolean

match:((**RM**[PRODUCT],τ_2)×(**DW**[PRODUCT (**current_flag** = true)],{τ_3}))→Boolean

Figure 4.5: Examples of matching function signatures.

The matching function signatures are locally declared in each perspective schema, even though many perspective schemata in the same environment probably exist, which need to declare matching function signatures involving the same classes/relations. This is done in order to guarantee the logical independence between the perspective schemata (i.e., each perspective schema must have all the declarations that are necessary to create it). The implementation of the matching functions shall be externally provided, since their specifications are very close to the application domain and to the application itself. Examples of matching function implementations can be found, for instance, in (Zhao & Ram, 2008; Davis *et al.*, 2003; Monge & Elkan, 1996; Winkler & Winkler, 2001; Castano *et al.*, 2008).

4.2.3 Correspondence assertions

Besides combining different instances that represent the same entity in a data integration environment, we must establish the semantic correspondence between schemata's components. This problem can be hard to deal with, as multiple, autonomous and heterogeneous information sources are usually involved. Thus, even after all source schemata are drawn to a common language (in this proposal using the language L_S), semantic heterogeneity (see Chapter 2) should still be treated.

Semantic correspondence between the schemata's components is formally declared in a declarative manner through the Correspondence Assertions (CAs). There are several kinds of CAs, which depend on the involved elements and the nature of the correspondence (i.e., of the imposed constraint). These assertions express the knowledge that *this element is related with this other element*, and therefore they impose constraints in the admissible states of the schema. The permissible states of the schema are those that satisfy the structure and the constraints of the schema. In this work, the relationship between the target and the base schemata can be specified by the following four kinds of correspondence assertions:

- Property Correspondence Assertion (PCA),

- Extension Correspondence Assertion (ECA),
- Summation Correspondence Assertion (SCA), and
- Aggregation Correspondence Assertion (ACA).

In general, the CAs have the form: $A^{\mathbf{T}} \rightarrow A^{\mathbf{B}}$, which means that a component of the target schema ($A^{\mathbf{T}}$) is mapped from one (or more) component(s) of the base schemata ($A^{\mathbf{B}}$). Some examples of CAs are shown in Figure 4.6.

ψ_1: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [PRODUCT] • $\mathbf{pid}_{\mathrm{RM}} \rightarrow \mathbf{S}_1$ [PRODUCT] • $\mathbf{prod_id}_1$ (PCA)

ψ_4: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [PRODUCT] $\rightarrow \mathbf{S}_1$ [PRODUCT] (ECA)

ψ_6: $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ [PRODUCT ($\mathbf{current_flag}_{\mathrm{DW}}$ = True)] $\rightarrow$ **RM** [PRODUCT] (ECA)

ψ_7: $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ [SALES_BY_CUSTOMER] ($\mathbf{prod_id_sk}_{\mathrm{DW}}, \mathbf{cust_id_sk}_{\mathrm{DW}}, \mathbf{date_id_sk}_{\mathrm{DW}}$) $\rightarrow$ (SCA)

$\rightarrow$ groupby (**RM** [SALE_ITEM] ($\mathbf{pid}_{\mathrm{RM}}$, $\boldsymbol{FK}_1$•$\mathbf{cid}_{\mathrm{RM}}$, $\boldsymbol{FK}_1$•$\mathbf{sale_date}_{\mathrm{RM}}$))

ψ_9: $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ [SALES_BY_CUSTOMER] • $\mathbf{prodsold_qty}_{\mathrm{DW}} \rightarrow \psi_7$, **sum** (**RM** [SALE_ITEM] • (ACA)

•$\mathbf{quantity}_{\mathrm{RM}}$)

Figure 4.6: Examples of correspondence assertions.

More details are shown later, but the key points of the notation used in Figure 4.6 are:

1. ψ_i ($1 \leq \mathrm{i} \leq 10$), is the CA name;
2. class/relation names are in "[ ]";
3. The expression **S**[C] means that C is a class/relation of the schema **S**;

Some CAs present Boolean conditions (or selection predicates), which are used to restrict the number of instances in a class/relation or to manage the value of a property (e.g., see ψ_6 in Figure 4.6). Predicates are defined using the grammar presented below.

Definition 24 (*Predicate*) *Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{W}$ a set of typed values, and $\mathcal{L}$ a set of schema names. Also let* $\mathbf{S} \in \mathcal{L}$, $\mathrm{R} \in \mathcal{C}$ *or* $\mathrm{R} \in \mathcal{R}$, *and* $\boldsymbol{w} \in \mathcal{W}$. *The predicate* **pred** *is a Boolean condition defined by the following grammar:*

$$\begin{aligned}\textbf{pred} ::= \; & A \;\; \boldsymbol{op} \;\; B \;\; | \\ & A \;\; \boldsymbol{op} \;\; B \;\; \textit{and} \;\; \textbf{pred} \;\; | \\ & A \;\; \boldsymbol{op} \;\; B \;\; \textit{or} \;\; \textbf{pred}\end{aligned}$$

$$\begin{aligned}A ::= \; & \mathbf{S}\,[\mathrm{R}] \bullet \boldsymbol{p}_i \;\; | \\ & \varphi(A_1, A_2, ..., A_n) \;\; | \\ & \mathbf{S}\,[\mathrm{R}] \bullet \boldsymbol{p}_i\{\boldsymbol{p}'_t\}\end{aligned} \qquad \begin{aligned}B ::= \; & \boldsymbol{w} \;\; | \\ & A\end{aligned} \qquad \begin{aligned}\boldsymbol{op} ::= \; & > \;\; | \;\; < \;\; | \\ & \geq \;\; | \;\; \leq \;\; | \\ & = \;\; | \;\; \neq \; .\end{aligned}$$

Where:

- A *and* B *are the parameters of the predicate* **pred**,
- $\boldsymbol{op}$ *is an operand,*
- φ *is a function with* $n \geq 1$ *arguments that returns a value,*
- $\boldsymbol{p}_i \in \boldsymbol{props}(\mathrm{R})$ *or* $\boldsymbol{p}_i$ *is a path of* R, *for* $i \geq 1$. *The notation* $\boldsymbol{p}_i\{\boldsymbol{p}'_t\}$ *means that* $\boldsymbol{p}_i$ *has a structural type with* $\boldsymbol{p}'_t$, *for* $t \geq 1$, *being one of the property names belonging to the type of* $\boldsymbol{p}_i$.[3] □

In Definition 24, A and B should refer to a same class/relation of a same schema. Furthermore, φ can be a function to perform: a) the mapping of domain types (e.g., changing the value from one type to another, or changing different formats of data or unit of measure), b) arithmetic calculations, c) operations on strings (as concatenation), to mention only a few.

Figure 4.7 gives examples of Boolean conditions that are predicates and others that are not, based on schemata presented in Figures 4.1 and 4.2. Note that the examples e) to h) are not legal Boolean conditions in language L_{PS} since: the properties being evaluated should belong to the same relation/class in a same schema, the first parameter of the predicate cannot be a value, and "not" is not a valid operand in L_{PS}.

The next Section will give detail concerning the specific types of CAs, followed by the Section describing view relation declarations.

---

[3] ***props***() *and path: see Chapter 3.*

Predicates:
a) RM[REGION]•$\mathbf{rcode}_{RM} \geq 100$ and RM[REGION]•$\mathbf{rcode}_{RM} \leq 500$
b) $\mathbf{S}_1$[CUSTOMER]•$\mathbf{region_id}_{S1} \neq 1000$
c) dollarTOeuro($\mathbf{S}_1$[ITEM]•$\mathbf{unitprice}_{S1}$) > 1000
d) $\mathbf{S}_1$[CUSTOMER]•$\mathbf{cust_address}_{S1}\{\mathbf{city}\}$ = "London"
Non-Predicates:
e) $\mathbf{S}_1$[CUSTOMER]•$\mathbf{cust_id}_{S1}$ = $\mathbf{S}_2$[PURCHASER]•$\mathbf{idcard}_{S2}$
f) $1000 \leq$ $\mathbf{S}_2$[PRODUCT_SALES]•$\mathbf{vendor_salary}_{S2}$
g) "John" $\neq$ "Mary"
h) not($\mathbf{S}_1$[LOCATION]•$\mathbf{region_id}_{S1}$) ≥ 100

Figure 4.7: Predicates and non-predicates.

4.3 CAs in detail

In this section we discuss the four types of CAs: Property Correspondence Assertion (PCA), Extension Correspondence Assertion (ECA), Summation Correspondence Assertion (SCA), and Aggregation Correspondence Assertion (ACA).

4.3.1 Property Correspondence Assertion (PCA)

Property Correspondence Assertions (PCAs) relate properties of a target schema to the properties of base schemata. They allow for dealing with several kinds of semantic heterogeneity such as: naming conflict, data representation conflict, and encoding conflict. Furthermore, the user can deal with: a) mappings involving a calculation of values in two or more properties of the same instance; or b) mapping involving one or several conditions that refer to properties of the same instance (the *case-base mapping*). A PCA is formally defined as follows:

Definition 25 (*Property Correspondence Assertion) Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}$ be a set of schema names. Also let* $\mathsf{S} \in \mathcal{L}$*;* $\mathrm{C} \in \mathcal{C}$*, or* $\mathrm{C} \in \mathcal{R}$*;* $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{C})$*;* $\psi \in \mathcal{A}$*; and* **pred***,* $\mathtt{A}$ *and* $\mathtt{B}$ *be, respectively, a predicate and the parameters of* **pred** *such as defined in Definition 24. A property correspondence assertion of* C *is a rule defined over* $\mathcal{C}$*,* $\mathcal{R}$*,* $\mathcal{A}$*, and* $\mathcal{L}$ *having one of the following forms:*

$$\psi : \quad \mathbf{S}[\mathrm{C}] \bullet \boldsymbol{p} \quad \rightarrow \quad A \quad | \quad (A_1, A_2, ..., A_n) \quad | \quad (B_1, \mathbf{pred}_1); (B_2, \mathbf{pred}_2); ...; (B_{m-1}, \mathbf{pred}_{m-1}); B_m$$

or

$$\psi : \quad \mathbf{S}[\mathrm{C}] \bullet \boldsymbol{p}\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\} \quad \rightarrow \quad (A_1, A_2, ..., A_n) \ . \qquad \square$$

Figure 4.8 shows some examples of PCAs relating schemata presented in Figures 4.1, 4.2, and 4.3.

ψ_1: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [PRODUCT] • $\mathbf{pid}_{\mathrm{RM}}$ → $\mathbf{S}_1$ [PRODUCT] • $\mathbf{prod_id}_1$

ψ_2: $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ [PRODUCT] • $\mathbf{prod_id}_{\mathrm{DW}}$ → **RM** [PRODUCT] • $\mathbf{pid}_{\mathrm{RM}}$

ψ_3: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [SALE_ITEM] • $\mathbf{unitprice}_{\mathrm{RM}}$ → **dollarTOeuro** ($\mathbf{S}_1$ [ITEM] • $\mathbf{unitprice}_1$)

ψ_{11}: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$[CUSTOMER]• $\mathbf{caddress}_{\mathrm{RM}}$ → **transformAddress**($\mathbf{S}_1$[CUSTOMER]•$\mathbf{cust_address}_1$\{**city**\}, $\mathbf{S}_1$ [CUSTOMER] • $\mathbf{cust_address}_1$\{**street**\})

ψ_{12}: $\mathbf{P}_{\mathbf{S}_2|\mathbf{RM}}$ [CUSTOMER] • $\mathbf{caddress}_{\mathrm{RM}}$\{**city**, **street**\} → ($\mathbf{S}_2$ [PURCHASER] • $\mathbf{city}_2$, $\mathbf{S}_2$ [PURCHASER] • •$\mathbf{street}_2$)

ψ_{13}: $\mathbf{P}_{\mathbf{S}_2|\mathbf{RM}}$ [CUSTOMER] • $\mathbf{cphone}_{\mathrm{RM}}$ → ($\mathbf{S}_2$ [PURCHASER] • $\mathbf{contact1}_2$, $\mathbf{S}_2$ [PURCHASER] • •$\mathbf{contact2}_2$)

ψ_{14}: $\mathbf{P}_{\mathbf{S}_1|\mathbf{RM}}$ [SALE]• $\mathbf{sale_date}_{\mathrm{RM}}$ → ($\mathbf{S}_1$ [SALE]• $\mathbf{sale_date}_1$, $\mathbf{S}_1$ [SALE]• $\mathbf{sale_date}_1$ ≠′ 31/12/2008′) ; ′01/01/2009′

ψ_{15}: $\mathbf{P}_{\mathbf{RM}|\mathbf{DW}}$ [CUSTOMER] • $\mathbf{region_desc}_{\mathrm{DW}}$ → **RM** [CUSTOMER] • FK_2 • $\mathbf{rname}_{\mathrm{RM}}$

Figure 4.8: Examples of property correspondence assertions.

PCAs ψ_1 and ψ_2 deal with naming conflict. ψ_1 links the property $\mathbf{pid}_{\mathrm{RM}}$ to the property $\mathbf{prod_id}_1$ and ψ_2 relates the property $\mathbf{prod_id}_{\mathrm{DW}}$ to the property $\mathbf{pid}_{\mathrm{RM}}$.

PCA ψ_3 deals with encoding conflict. It assigns $\mathbf{unitprice}_{\mathrm{RM}}$ to $\mathbf{unitprice}_1$ using the function **dollarTOeuro** to convert currencies from dollars (stored in $\mathbf{unitprice}_1$) to euros (stored in $\mathbf{unitprice}_{\mathrm{RM}}$).

PCAs ψ_{11}, ψ_{12}, and ψ_{13} deal with data representation conflict. We suppose that for ψ_{11}, the type of property $\mathbf{caddress}_{\mathrm{RM}}$ is a string and the type of property $\mathbf{cust_address}_1$ is the structural type: \{**city**:string, **street**:string\}. ψ_{11} links the property $\mathbf{caddress}_{\mathrm{RM}}$ to property

cust_address$_1$ changing the type of the latter for the type of the first using the function *transformAddress*. For ψ_{12} we suppose that the type of property **caddress**$_{\text{RM}}$ becomes the structural type: {**city**:string, **street**:string} and the type of the properties **city**$_2$ and **street**$_2$ is a string. ψ_{12} assigns the properties **city** and **street** of the property **caddress**$_{\text{RM}}$ to, respectively, the properties **city**$_2$ and **street**$_2$. Here it is not necessary to use a function, since each property in **caddress**$_{\text{RM}}$ has a corresponding property of the same type in the base schema. For ψ_{13} we suppose that the type of property **cphone**$_{\text{RM}}$ is a set of strings and the type of both properties **contact1**$_2$ and **contact2**$_2$ is a string. ψ_{13} links the property **cphone**$_{\text{RM}}$ to the properties **contact1**$_2$ and **contact2**$_2$, indicating that the values of the latter are the (set, list or array) elements of the former.

PCA ψ_{14} deals with case-base mapping. Here we suppose that all products sold in "31/12/2008" have their dates changed to "01/01/2009". It assigns the property **sale_date**$_{\text{RM}}$ to property **sale_date**$_1$, when **sale_date**$_1$ is different from "31/12/2008", otherwise the value of **sale_date**$_{\text{RM}}$ is the constant "01/01/2009".

PCA ψ_{15} deals with the denormalisation. It assigns the property **region_desc**$_{\text{DW}}$ to the path **RM** [CUSTOMER] •FK$_2$•**rname**$_{\text{RM}}$, where FK$_2$ is the name of a foreign key of **RM**.CUSTOMER that refers to **RM**.REGION.

The meaning of each rule in Definition 25 is as follows:

- If $\psi : \mathbf{S}\,[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow \mathtt{A}$ then at least three situations might exist, depending on the value of A:
  1. $\mathtt{A} = \mathbf{S}'\,[\mathrm{C}'] \bullet \boldsymbol{p}'$. In this case, $\mathbf{S}\,[\mathrm{C}] \bullet \boldsymbol{p} \equiv \mathbf{S}'\,[\mathrm{C}'] \bullet \boldsymbol{p}'$ (read $\boldsymbol{p}$ is semantically equivalent to $\boldsymbol{p}'$). $\boldsymbol{p}$ and $\boldsymbol{p}'$ have compatible domains. This is the case, for example, of the CAs ψ_1 and ψ_2 in Figure 4.8.
  2. $\mathtt{A} = \varphi(\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n)$, n ≥ 1. In this case, $\mathbf{S}[\mathrm{C}] \bullet \boldsymbol{p} \blacktriangleright \varphi(\mathtt{A}_1,\ \mathtt{A}_2,\ ...,\ \mathtt{A}_n)$ (read $\boldsymbol{p}$ is derived from $\varphi(\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n)$). $\boldsymbol{p}$ is a singlevalued property whose value is calculated, or, when n = 1, $\boldsymbol{p}$ has a different domain or diverse type from that presented in $\mathtt{A}_1$. This is the case, for example, of the CA ψ_{11} in Figure 4.8.
  3. $\mathtt{A} = \mathbf{S}'\,[\mathrm{C}'] \bullet \boldsymbol{p}'\{\boldsymbol{p}''_t\}$. In this case, $\mathbf{S}[\mathrm{C}] \bullet \boldsymbol{p} \equiv \mathbf{S}'\,[\mathrm{C}'] \bullet \boldsymbol{p}'\{\boldsymbol{p}''_t\}$ (read $\boldsymbol{p}$ is semantically equivalent to $\boldsymbol{p}''_t$ of $\boldsymbol{p}'$. $\boldsymbol{p}$ and $\boldsymbol{p}''_t$ have compatible domains.
- If $\psi : \mathbf{S}\,[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow (\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n)$ then $\mathbf{S}[\mathrm{C}] \bullet \boldsymbol{p} \equiv (\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n)$ (read $\boldsymbol{p}$ is set equivalent to $\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n$). In this case, $\boldsymbol{p}$ is a multivalued property (a set, a list, or a array of a given

type τ), and $\mathtt{A}_i$, for $1 \leq$ i $\leq$ n, is of the type τ. This is the case, for example, of the CA ψ_{13} in Figure 4.8.

- If ψ: $\mathsf{S}[\mathrm{C}] \bullet \boldsymbol{p}\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\} \rightarrow (\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n)$ then $\mathsf{S}[\mathrm{C}] \bullet \boldsymbol{p}\{\boldsymbol{p}_1, ..., \boldsymbol{p}_n\} \equiv (\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n)$ (read $\boldsymbol{p}$ is tuple equivalent to $\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n$). In this case, $\boldsymbol{p}$ is a singlevalued property of the structural type: $\{\boldsymbol{p}_1{:}\tau_1, \boldsymbol{p}_2{:}\tau_2, ..., \boldsymbol{p}_n{:}\tau_n\}$ and each $\mathtt{A}_i$ has the type τ_i, for $1 \leq$ i $\leq$ n. This is the case, for example, of the CA ψ_{12} in Figure 4.8.

- If $\psi : \mathsf{S}\,[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow (\mathtt{B}_1, \mathbf{pred}_1)\,; (\mathtt{B}_2, \mathbf{pred}_2)\,; ...; \left(\mathtt{B}_{m-1}, \mathbf{pred}_{m-1}\right); \mathtt{B}_m$ then $\mathsf{S}\,[\mathrm{C}] \bullet \boldsymbol{p}$ can have one of many values $\mathtt{B}_1$, $\mathtt{B}_2$, ..., $\mathtt{B}_m$ depending on conditions presented in each predicate $\mathbf{pred}_i$, for $1 \leq$ i $\leq$ m-1. The value in $\boldsymbol{p}$ will be the first $\mathtt{B}_i$ whose **pred** is true, or $\mathtt{B}_m$ if every **pred** is false. This is the case, for example, of the CA ψ_{14} in Figure 4.8.

It is important to note that it is possible to have more than one PCA specified for the same property of a class/relation of a same schema. For example, the relation **RM**.CUSTOMER, presented in Figure 4.2, can have two PCAs for the property $\mathbf{cname}_{\mathrm{RM}}$: one relating $\mathbf{cname}_{\mathrm{RM}}$ to property $\mathbf{cust_name}_1$ (ψ'_1: $\mathsf{P}_{\mathsf{S}_1,\mathsf{S}_2|\mathsf{RM}}$ [CUSTOMER] $\bullet$ $\mathbf{cname}_{\mathrm{RM}} \rightarrow$ S_1 [CUSTOMER] $\bullet$ $\mathbf{cust_name}_1$), and the other relating $\mathbf{cname}_{\mathrm{RM}}$ to property $\mathbf{name}_2$ (ψ'_2: $\mathsf{P}_{\mathsf{S}_1,\mathsf{S}_2|\mathsf{RM}}$ [CUSTOMER] $\bullet$ $\mathbf{cname}_{\mathrm{RM}} \rightarrow \mathsf{S}_2$ [PURCHASER] $\bullet$ $\mathbf{name}_2$). This means that the properties $\mathbf{cname}_{\mathrm{RM}}$, $\mathbf{cust_name}_1$, and $\mathbf{name}_2$ are *synonym properties* and thus, they must have the same value. Note that synonym properties can only exist in the context of a same perspective schema.

4.3.2 Extension Correspondence Assertion (ECA)

The Extension Correspondence Assertions (ECAs) are used to describe the relationship that exists between the instances of the target schema and the instances of the base schemata. For example, considering the schemata displayed in Figures 4.1, 4.2, and 4.3, the relation **RM**.PRODUCT is linked to relation S_1.PRODUCT through the ECA ψ_4 presented in Figure 4.9. ψ_4 determines that **RM**.PRODUCT and S_1.PRODUCT are equivalent (i.e., for each tuple in PRODUCT of the schema S_1 there is one semantically equivalent[4] tuple in PRODUCT of the schema **RM**, and vice-versa).

There are five different kinds of ECAs: *equivalence, selection, difference, union*, and *intersection.* The ECAs are used to define which objects/tuples of a base schema should have

[4]Remember that *semantically equivalent* means *represent the same instance of the real world.*

ψ_4: $\mathbf{P}_{\mathbf{S}_1\|\mathbf{RM}}$ [PRODUCT] $\rightarrow$ $\mathbf{S}_1$ [PRODUCT]	(ECA of equivalence)
ψ_5: $\mathbf{P}_{s1,s3\|v}$[PRODUCT] $\rightarrow$ $\mathbf{S}_1$ [PRODUCT] $\sqsupset\!\bowtie\!\sqsubset$ $\mathbf{S}_3$ [PROD]	(ECA of union)
ψ_6: $\mathbf{P}_{\mathbf{RM}\|\mathbf{DW}}$ [PRODUCT (**current_flag**$_{\text{DW}}$ = True)] $\rightarrow$ **RM** [PRODUCT]	(ECA of selection)
ψ_{16}: $\mathbf{P}_{s1,s2\|v'}$[CUSTOMER_NY] $\rightarrow$ $\mathbf{S}_1$ [CUSTOMER] $-$ $\mathbf{S}_2$ [PURCHASER]	(ECA of difference)
ψ_{17}: $\mathbf{P}_{s1,s2\|v'}$[SHARED_CUSTOMER] $\rightarrow$ $\mathbf{S}_1$ [CUSTOMER] $\cap$ $\mathbf{S}_2$ [PURCHASER]	(ECA of intersection)

Figure 4.9: Examples of extension correspondence assertions.

a corresponding semantically equivalent object/tuple in a target schema. An ECA is formally defined as follows:

Definition 26 (*Extension Correspondence Assertion) Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}$ a set of schema names. Also let $\mathrm{C}_i \in \mathcal{C}$, or $\mathrm{C}_i \in \mathcal{R}\}$; $\mathbf{S}_i \in \mathcal{L}$, for $1 \leq i \leq n$; ψ in $\mathcal{A}$;* **pred** *be a predicate as defined in Definition 24; and E be an expression defined by the following grammar:*

$$E \ ::= \ \mathbf{S}_i\,[\mathrm{C}_i] \quad | \quad \mathbf{S}_i\,[\mathrm{C}_i\,(\textbf{pred})]\ .$$

An extension correspondence assertion of C_1 is a rule defined over $\mathcal{C}$, $\mathcal{R}$, $\mathcal{A}$, and $\mathcal{L}$ having one of the following forms:

$$\begin{aligned} \psi :\ E \ \rightarrow\ & E_1 \quad | \\ & E_1 - E_2 \quad | \\ & E_1 \cap E_2 \cap \cdots \cap E_n \quad | \\ & E_1 \sqsupset\!\bowtie\!\sqsubset E_2 \sqsupset\!\bowtie\!\sqsubset \ldots \sqsupset\!\bowtie\!\sqsubset E_n\ . \end{aligned} \qquad \square$$

The meaning of each rule in Definition 26 is as follows:

1. If $\psi : \mathbf{S}_1\,[\mathrm{C}_1] \rightarrow \mathbf{S}_2\,[\mathrm{C}_2]$ then $\mathbf{S}_1[\mathrm{C}_1] \equiv \mathbf{S}_2[\mathrm{C}_2]$ (read C_1 is semantically equivalent to C_2) (**ECA of equivalence**). This means that for each tuple (or object) in C_2 of schema $\mathbf{S}_2$, there is one and only one tuple (or object) in C_1 of schema $\mathbf{S}_1$ that matches with it.

2. If $\psi : \mathbf{S}_1\,[\mathrm{C}_1] \rightarrow \mathbf{S}_2\,[\mathrm{C}_2\,(\textbf{pred})]$ then $\mathbf{S}_1[\mathrm{C}_1] \equiv \mathbf{S}_2[\mathrm{C}_2\,(\textbf{pred})]$ (read C_1 is semantically equivalent to $\mathrm{C}_2(\textbf{pred})$) (**ECA of selection**). This means that for each tuple (or object)

in C_2 of schema $\mathbf{S}_2$ that satisfies the condition in **pred**, there is one and only one tuple (or object) in C_1 of schema $\mathbf{S}_1$ that matches with it.

3. If $\psi : \mathbf{S}_1 [C_1] \rightarrow \mathbf{S}_2 [C_2] - \mathbf{S}_3 [C_3]$ then $\mathbf{S}_1[C_1] \equiv \mathbf{S}_2[C_2] - \mathbf{S}_3[C_3]$ (read C_1 is semantically equivalent to $C_2 - C_3$) (**ECA of difference**). This means that for each tuple (or object) in C_2 of schema $\mathbf{S}_2$, such that there is not any tuple (or object) in C_3 of schema $\mathbf{S}_3$ that matches with it, then there is one and only one tuple (or object) in C_1 of schema $\mathbf{S}_1$ that matches with it. C_2 and C_3 can be defined in a same or different schema.

4. If $\psi : \mathbf{S}_1 [C_1] \rightarrow \mathbf{S}_2 [C_2] ⟗ \mathbf{S}_3 [C_3] ⟗ \ldots ⟗ \mathbf{S}_n [C_n]$ then $\mathbf{S}_1[C_1] \equiv \underset{i=2}{\overset{n}{⟗}} \mathbf{S}_i[C_i]$ (read C_1 is semantically equivalent to union of C_i) (**ECA of union**). This means that for each tuple (or object) in C_2 of schema $\mathbf{S}_2$, there is one and only one tuple (or object) in C_1 of schema $\mathbf{S}_1$ that matches with it; or for each tuple (or object) in C_i of schema $\mathbf{S}_i$, $2 \leq$ i $\leq$ n-1, there is one and only one tuple (or object) in C_1 of schema $\mathbf{S}_1$ that matches with it; or for each tuple (or object) in C_n of schema $\mathbf{S}_n$, there is one and only one tuple (or object) in C_1 of schema $\mathbf{S}_1$ that matches with it; and vice-versa. C_i and C_j, $1 \leq$ i,j $\leq$ n, can be defined in a same or different schema.

5. If $\psi : \mathbf{S}_1 [C_1] \rightarrow \mathbf{S}_2 [C_2] \cap \mathbf{S}_3 [C_3] \cap \cdots \cap \mathbf{S}_n [C_n]$ then $\mathbf{S}_1[C_1] \equiv \bigcap_{i=2}^{n} \mathbf{S}_i[C_i]$ (read C_1 is semantically equivalent to the intersection of C_i) (**ECA of intersection**). This means that for each tuple (or object) in C_j of schema $\mathbf{S}_j$, $1 \leq$ i $\leq$ n, such that there is one tuple (or object) in C_i of schema $\mathbf{S}_i$, with i $\neq$ j and $1 \leq$ i $\leq$ n, that matches with it, then there is one and only one tuple (or object) in C_1 of schema $\mathbf{S}_1$ that matches with it, and vice-versa. C_i and C_j, $1 \leq$ i,j $\leq$ n, can be defined in a same or different schema.

6. Mutates mutandis when changing $\mathbf{S}_1[C_1]$ with $\mathbf{S}_1[C_1($**pred**$)]$ in items 1 to 5.

The ECA of union is not close to a union of sets, rather it indicates a relation similar to the natural outer-join of the usual relational models. For example, consider a perspective schema $\mathbf{P}_{s1,s3|v}$ with the relation PRODUCT(**code, description, category**) which is related to two relations: PRODUCT in schema $\mathbf{S}_1$ (see Figure 4.1), and PROD(**code, description,category**) in schema $\mathbf{S}_3$ (not presented in any Figure) through ECA ψ_5 shown in Figure 4.9. ψ_5 determines that PRODUCT in $\mathbf{P}_{s1,s3|v}$ is the union/join of PRODUCT in $\mathbf{S}_1$ and PROD in $\mathbf{S}_3$ (i.e., for each tuple of PRODUCT of the schema $\mathbf{S}_1$ there is one semantically equivalent tuple in PRODUCT of the perspective schema $\mathbf{P}_{s1,s3|v}$, or for each tuple of PROD of the schema $\mathbf{S}_3$ there is one semantically equivalent tuple in PRODUCT of the perspective schema $\mathbf{P}_{s1,s3|v}$).

In an ECA, any relation/class can appear with a condition of selection, which determines the subset of instances of the class/relation that is considered. This kind of ECA is especially important to the DW because through it the current instances of the DW can be selected and related to the instances of their sources (which usually do not have historical data). For example, consider the ECA ψ_6 presented in Figure 4.9. ψ_6 determines that a subset of instances of the relation PRODUCT of the perspective schema $\mathbf{P_{RM|DW}}$, whose value of the property **current_flag**$_{\text{DW}}$ is true, is the same as those instances of the relation PRODUCT of the schema **RM**. Examples of ECAs of difference and intersection can be seen in the following text.

Example 14

Suppose that the schemata $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *hold sales of shops located, respectively, in New York and in Spring Valley, and that there is a perspective schema* $\mathbf{P}_{s1,s2|v'}$ *that is the integration of both schemata.* $\mathbf{P}_{s1,s2|v'}$ *contains two relations:* CUSTOMER_NY *and* SHARED_CUSTOMER*.* CUSTOMER_NY *holds the customers that only go shopping in the New York shop, while* SHARED_CUSTOMER *stores information about customers that go shopping in both shops. The relationship between these relations with those of the base are described through the ECAs* ψ_{16} *and* ψ_{17} *presented in Figure 4.9.*

ψ_{16} *specifies that* CUSTOMER_NY *contains the instances of the relation* CUSTOMER *that are not instances of the relation* PURCHASER*.* ψ_{17} *specifies that the relation* SHARED_CUSTOMER *contains only the instances that are common to the relations* CUSTOMER *and* PURCHASER*.*

4.3.3 Summation Correspondence Assertion (SCA)

The Summation Correspondence Assertion (SCA) is used to describe the summary of a class/relation whose instances are related to the instances of another class/relation by breaking them down into logical groups that belong together. For example, a SCA is used when daily sales are mapped to monthly sales by region; or when a denormalised relation containing properties related to vendor, price quote and item is mapped to, for example, a normalised relation containing only vendor data. There are two kinds of SCAs: *groupby* and *normalise*. Both group instances are based on one or more properties, but *groupby* is used to indicate that some type of aggregate function will be used and *normalise* is used to indicate a normalisation process. An example is presented as follows:

Example 15

Consider that the instances of the relations **DW**.SALES_BY_CUSTOMER *and* **RM**.SALE_ITEM *are connected through the SCA* ψ_7 *shown as follows:*

ψ_7: $\mathbf{P}_{\mathbf{RM|DW}}$ [SALES_BY_CUSTOMER] ($\boldsymbol{prod_id_sk}_{DW}$, $\boldsymbol{cust_id_sk}_{DW}$, $\boldsymbol{date_id_sk}_{DW}$) $\rightarrow$
$\rightarrow$ *groupby* (**RM** [SALE_ITEM] ($\boldsymbol{pid}_{RM}$, $\boldsymbol{FK}_1 \bullet \boldsymbol{cid}_{RM}$, $\boldsymbol{FK}_1 \bullet \boldsymbol{sale_date}_{RM}$))

ψ_7 *determines that there is a tuple in* **DW**.SALES_BY_CUSTOMER *for each group of tuples in* **RM**.SALE_ITEM *that have the same value for product (*$\boldsymbol{pid}_{RM}$*), customer (***RM**.SALE_ITEM$\bullet\boldsymbol{FK}_1\bullet\boldsymbol{cid}_{RM}$*), and sale date (***RM**.SALE_ITEM$\bullet\boldsymbol{FK}_1\bullet\boldsymbol{sale_date}_{RM}$*). Note that* **RM**[SALE_ITEM] $\bullet$ $\boldsymbol{FK}_1$ $\bullet$ $\boldsymbol{cid}_{RM}$ *and* **RM**[SALE_ITEM] $\bullet$ $\boldsymbol{FK}_1$ $\bullet$ $\boldsymbol{sale_date}_{RM}$ *are path expressions, with* $\boldsymbol{FK}_1$ *being a foreign key of* **RM**.SALE_ITEM *that refers to* **RM**.SALE*. These paths mean that there is a link through* $\boldsymbol{FK}_1$ *from which are obtained, respectively, the customer identity (***RM**.SALE.$\boldsymbol{cid}_{RM}$*) and the value of sale date (***RM**.SALE.$\boldsymbol{sale_date}_{RM}$*).*

The formal definition of SCA is as follows:

Definition 27 (*Summation Correspondence Assertion*)
Let $\mathcal{R}$ *be a set of relation names,* $\mathcal{C}$ *a set of class names,* $\mathcal{P}$ *a set of property names,* $\mathcal{A}$ *a set of correspondence assertion names, and* $\mathcal{L}$ *a set of schema names. Also let* C_1 *and* $\mathrm{C}_2 \in \mathcal{C}$*, or* C_1 *and* $\mathrm{C}_2 \in \mathcal{R}$*,* $\boldsymbol{p}_i \in \boldsymbol{props}(\mathrm{C}_1)$ *(for* $1 \leq i \leq m$*, or* $i = k$*),* $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{C}_2)$ *or* $\boldsymbol{p}$ *is a path of* C_2*,* $\mathbf{S}_1$ *and* $\mathbf{S}_2 \in \mathcal{L}$*,* $\psi \in \mathcal{A}$*, and* $\mathtt{A}$ *and* $\mathtt{E}$ *are expressions as defined, respectively in Definitions 24 and 26:*

$$\begin{array}{lll} \mathtt{A} & ::= & \mathbf{S}_2\,[\mathrm{C}_2] \bullet \boldsymbol{p} \quad | \\ & & \varphi(\mathtt{A}_1, \mathtt{A}_2, ..., \mathtt{A}_n) \quad | \\ & & \mathbf{S}_2\,[\mathrm{C}_2] \bullet \boldsymbol{p}\{\boldsymbol{p'}_t\}\ . \end{array} \qquad \begin{array}{lll} \mathtt{E} & ::= & \mathbf{S}_2\,[\mathrm{C}_2] \quad | \\ & & \mathbf{S}_2\,[\mathrm{C}_2\,(\textbf{pred})]\ . \end{array}$$

A Summation correspondence assertion of C_1 *is a rule defined over* $\mathcal{C}$*,* $\mathcal{R}$*,* $\mathcal{P}$*,* $\mathcal{A}$*, and* $\mathcal{L}$ *having one of the following forms:*

$$\psi : \mathbf{S}_1 [\mathrm{C}_1] (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m) \rightarrow \mathsf{groupby}\, (E(A_1, A_2, \ldots, A_m)) \quad | \quad \mathsf{normalise}\, (E(A_1, A_2, \ldots, A_m)) \tag{4.5}$$

or

$$\psi : \mathbf{S}_1 [\mathrm{C}_1] (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m \mid \boldsymbol{p}_k) \rightarrow \mathsf{groupby}\, (E(A_1, A_2, \ldots, A_m)) \quad | \quad \mathsf{normalise}\, (E(A_1, A_2, \ldots, A_m)) \; . \tag{4.6}$$

□

In Definition 27:

- (4.6) is used when a key, named *surrogate key* and indicated here by $\boldsymbol{p}_k$, is automatically generated by the system as part of the process of normalisation (or aggregation).
- E indicates that the whole set of instances of a relation/class, or only part of them, are involved in the relationship.
- A indicates that the properties connected can have compatible or different domains, diverse types; or one of them can be a property in a structural type.
- $\mathtt{A}_i$, for $1 \leq \mathrm{i} \leq \mathrm{m}$, are the aggregate properties or aggregate expressions (i.e., are the properties or expressions that are the base for the grouping), such that $\boldsymbol{p}_i \blacktriangleright \mathtt{A}_i$, for $1 \leq \mathrm{i} \leq \mathrm{m}$.
- groupby indicates that some kind of aggregate function is involved in the relationship between C_1 and C_2.
- normalise indicates that a normalisation process is involved in the relationship between C_1 and C_2.

The following text is an example of a SCA of normalisation.

Example 16 ———————————

Consider the denormalised relation PURCHASE_ORDER *of source schema* S_2 *and the relation* SALESPERSON *of the schema* **RM***, presented, respectively, in Figure 4.1 and 4.2. The SCA*

ψ_8, displayed in the following text, determines the relationship between PURCHASE_ORDER *and* SALESPERSON *when a normalisation process is involved (i.e., it determines that* SALESPERSON *is a normalisation of* PURCHASE_ORDER *based on distinct values of property* $\boldsymbol{vendor_name}_{S2}$*. It also indicates that a surrogate key will be created,* $\boldsymbol{spid}_{RM}$*.*

ψ_8: $\mathbf{P}_{\mathbf{S}_2|\mathbf{RM}}$ [SALESPERSON] ($\boldsymbol{spname}_{RM}$ | $\boldsymbol{spid}_{RM}$) $\rightarrow$ *normalise* ($\mathbf{S}_2$ [PURCHASE_ORDER] ($\boldsymbol{vendor_name}_{S2}$))

This research also deals with denormalisations, which are defined using *path expressions* (component of the language L_S). Denormalisation was illustrated in Section 4.3.1.

4.3.4 Aggregation Correspondence Assertion (ACA)

Aggregation Correspondence Assertions (ACAs) link properties of the target schema to the properties of the base schema when a SCA is used. ACAs associated with SCAs of groupby contain aggregation functions. This proposal only deals with aggregate functions supported by most of the query languages, like SQL-3 (Elmasri & Navathe, 2006) (i.e., *summation, maximum, minimum, average* and *count*); although more complex aggregations are supported in some object-relational databases – cf. (IBM, 2008).

ACAs, similarly to PCAs, allow the description of several kinds of situations; therefore, the aggregate expressions can be more detailed than simple property references. Calculations performed can include, for example, ordinary functions (such as sum or concatenate two or more properties' values before applying the aggregate function), and Boolean conditions (e.g., count all male students whose grades are greater or equal to 10). A formal definition of ACA is as follows:

Definition 28 (*Aggregation Correspondence Assertion) Let $\mathcal{R}$ be a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{P}$ a set of property names, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}$ a set of schema names. Also let* $\mathbf{S} \in \mathcal{L}$, $\mathrm{C} \in \mathcal{C}$ *or* $\mathrm{C} \in \mathcal{R}$, $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{C})$, $\psi \in \mathcal{A}$, **pred**, *A*, *and B be, respectively, a predicate and the parameters of* **pred**, *such as defined in Definition 24. An aggregation correspondence assertion of* C *is a rule defined over $\mathcal{C}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{L}$ having one of the following forms:*

$$\psi : \mathsf{S}\,[\mathrm{C}] \bullet \boldsymbol{p} \rightarrow \psi', \gamma\,(A) \quad | \tag{4.7}$$

$$\psi', \gamma\,(A, \mathbf{pred}) \tag{4.8}$$

$$\psi', A \quad | \tag{4.9}$$

$$\psi', (A_1, A_2, ..., A_n) \quad | \tag{4.10}$$

$$\psi', (B_1, \mathbf{pred}_1)\,; (B_2, \mathbf{pred}_2)\,; ... \left(B_{m-1}, \mathbf{pred}_{m-1}\right); B_m \tag{4.11}$$

or

$$\psi : \mathsf{S}\,[\mathrm{C}] \bullet \boldsymbol{p}\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\} \rightarrow \psi', (A_1, A_2, ..., A_n)\ . \tag{4.12}$$

Where:

- *In* (4.7) *and* (4.8), ψ' *is a SCA of* groupby, *and it is a SCA of* normalise *in the remainder of cases.*
- γ *is one of the functions sum, max, min, avg, and count, which are used to, respectively, retrieve the total, the maximum value, the minimum value, the average of a set of values, and for counting the occurrence of a given value in a property.* □

Example 17

Returning to the motivating example, consider the ACAs ψ_9 *and* ψ_{10} *(displayed in the following text) that link the properties of* SALES_BY_CUSTOMER*:*

ψ_9: $\mathsf{P}_{\mathsf{RM|DW}}$ [SALES_BY_CUSTOMER] • $\boldsymbol{prodsold_qty}_{DW} \rightarrow \psi_7, \boldsymbol{sum}\,(\mathsf{RM}$ [SALE_ITEM] • $\boldsymbol{quantity}_{RM})$

ψ_{10}: $\mathsf{P}_{\mathsf{RM|DW}}$ [SALES_BY_CUSTOMER] • $\boldsymbol{prodsold_amt}_{DW} \rightarrow \psi_7, \boldsymbol{sum}\,(\mathsf{RM}$ [SALE_ITEM] • $\boldsymbol{quantity}_{RM} \times$
$\times \mathsf{RM}$ [SALE_ITEM] • $\boldsymbol{unitprice}_{RM})$

ψ_9 *determines that the value of the property* $\boldsymbol{prodsold_qty}_{DW}$ *is the summation of the quantity of product sold for each customer for each date, being that the grouping is obtained in* ψ_9 *by* ψ_7. ψ_{10} *determines that the value of the property* $\boldsymbol{prodsold_amt}_{DW}$ *is the amount of product sold for each customer for each date. In* ψ_{10} *the grouping is obtained by* ψ_7*, as occurs in* ψ_9*, and the amount of product sold is calculated using the formula: price* $\times$ *quantity.*

4.4 View relation declaration

In the relational world, a view is an important functionality that allows the designer to re-use a query frequently just as a relation schema, where its instances are derived of relation schemata or of previous views. Here, we have a similar concept, named *view relation* declarations, whose instances can be derived from classes, relations, or other view relations. A view relation declaration is the same as a relation declaration as presented in Chapter 3, Definition 5. We just use the abbreviation *VR* in the beginning of the view relation declarations to distinguish them from relation declarations. A view relation declaration must be assigned to classes, relations or other view relation declarations through correspondence assertions. This connection can involve or not, classes or relations of a same perspective schema. View relation declarations cannot have key declarations or foreign key declarations. An example is presented below.

Example 18

Consider the schemata presented in Figures 4.2 and 4.3. We want to store information about the quantity sold monthly by a salesperson. Thus, we create the view relation declaration MONTH_SALES *in* $\mathbf{P}_{RM|DW}$ *as follows:*

VR(MONTH_SALES,{***month***$_{DW}$*:integer,* ***year***$_{DW}$*:integer,* ***vendor_id_sk***$_{DW}$*:integer,* ***prod_id_sk***$_{DW}$*:integer,* ***amount***$_{DW}$*:float*})

This view relation declaration is assigned to the relation SALES_BY_VENDOR *of the perspective schema* $\mathbf{P}_{RM|DW}$ *through the following CAs:*

μ_1: **P**$_{\mathbf{RM|DW}}$ [MONTH_SALES] (***month***$_{DW}$, ***year***$_{DW}$, ***vendor_id_sk***$_{DW}$, ***prod_id_sk***$_{DW}$) → *groupby* (**DW** [SALES_BY_VENDOR] (***FK***$_2$•***month***$_{DW}$, ***FK***$_2$•***year***$_{DW}$, ***vendor_id_sk***$_{DW}$, ***prod_id_sk***$_{DW}$))

μ_2: **P**$_{\mathbf{RM|DW}}$ [MONTH_SALES]•***amount***$_{DW}$ → μ_1, ***sum*** (**DW** [SALES_BY_VENDOR]•***sales_quantity***$_{DW}$)

SCA μ_1 *determines that* MONTH_SALES *is an aggregation of* SALES_BY_VENDOR*.* **DW**[SALES_BY_VENDOR] •***FK***$_2$•***month***$_{DW}$ *and* **DW**[SALES_BY_VENDOR] •***FK***$_2$•***year***$_{DW}$ *are path expressions of* **DW**.SALES_BY_VENDOR*, with* ***FK***$_2$ *being a foreign key of* **DW**.SALES_BY_VENDOR *that refers to* **DW**.DATE*. The ACA* μ_2 *assigns the property* ***amount***$_{DW}$ *(of* MONTH_SALES*) to property* ***sales_quantity***$_{DW}$ *(of* SALES_BY_VENDOR*), indicating, by* μ_1*, that it is functionally dependent on properties (or paths)* ***FK***$_2$•***month***$_{DW}$*,* ***FK***$_2$•***year***$_{DW}$*,* ***vendor_id_sk***$_{DW}$*, and* ***prod_id_sk***$_{DW}$ *(all of relation* SALES_BY_VENDOR*).*

In order to simplify reading, hereafter we will use the short form *view relation* rather than *view relation declaration*, when there is no ambiguity.

The view relations must be created not only for reasons of convenience, but mainly for mapping necessities. We identify some cases, where a direct mapping from the base to the target cannot be realised using only the components defined in the base, which are as follows:

Consider C to be a class/relation of the target and C_i, $1 \leq i \leq n$, to be classes or relations of the base.

1. C is an aggregation (or normalisation) of the union of C_1, C_2, ..., C_n;
2. C is an aggregation (or normalisation) of the difference of C_1 and C_2;
3. C is an aggregation (or normalisation) of the intersection of C_1, C_2, ..., C_n;
4. C is a union or intersection or difference of C_1, ..., aggregation/normalisation of C_x, ..., C_n;
5. C is an aggregation (or normalisation) of an aggregation/normalisation of C_x;
6. C is a union or difference of an intersection or difference of C_1, C_2, ..., C_n;
7. C is an intersection or difference of an union or difference of C_1, C_2, ..., C_n.

In these situations, a view relation must be defined, which stores instances of the union/difference/intersection of C_1,...,C_i ($2 \leq i \leq n$), or of aggregation/normalisation of C_x, and it is this view relation that will be mapped to C. Note that, in this case, there are CAs from the classes or relations of the base to the view relation, and CAs from the view relation to the class or relation of the target. An example is presented in the following text.

Example 19

Based on the schemata presented in Figures 4.1, 4.2 and 4.3, consider a perspective schema $\mathbf{P}_{s1,s2|DW}$ *that maps the* **DW** *to the integration of the schemata* $\mathbf{S}_1$ *and* $\mathbf{S}_2$*. Suppose that in* $\mathbf{P}_{s1,s2|DW}$.SALES_BY_CUSTOMER *we want to store the quantity of the product sold per product, customer, and date; and that these values are derived from the union of* $\mathbf{S}_1$.ITEM *and* $\mathbf{S}_2$.PRODUCT_SALES*. We cannot map* $\mathbf{S}_1$.ITEM *and* $\mathbf{S}_2$.PRODUCT_SALES *to* SALES_BY_CUSTOMER *directly, since there is both an aggregation and a union involved. In this case, we first define a view relation, whose type must be formed by properties that will be mapped to properties of* SALES_BY_CUSTOMER*:*

VR(AUX_ITEM, {***product***:*integer*, ***customer***:*integer*, ***date***: *date*,
amount:*float*, ***quantity***:*integer*})

Then, we must define the CAs of AUX_ITEM*, as follows:*

μ_3: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] → $\mathbf{S}_1$ [ITEM] ⟗ $\mathbf{S}_2$ [PRODUCT_SALES]

μ_4: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***product*** → $\mathbf{S}_1$ [ITEM] • $\textit{prod_id}_{s1}$

μ_5: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***product*** → $\mathbf{S}_2$ [PRODUCT_SALES] • $\textit{product}_{s2}$

μ_6: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***customer*** → $\mathbf{S}_1$ [ITEM] • FK_3 • $\textit{cust_id}_{s1}$

μ_7: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***customer*** → $\mathbf{S}_2$ [PRODUCT_SALES] • FK_4 • $\textit{idcard}_{s2}$

μ_8: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***date*** → $\mathbf{S}_1$ [ITEM] • FK_3 • $\textit{sale_date}_{s1}$

μ_9: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***date*** → $\mathbf{S}_2$ [PRODUCT_SALES] • FK_4 • $\textit{po_date}_{s2}$

μ_{10}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***quantity*** → $\mathbf{S}_1$ [ITEM] • $\textit{qty}_{s1}$

μ_{11}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [AUX_ITEM] • ***quantity*** → $\mathbf{S}_2$ [PRODUCT_SALES] • $\textit{quantity}_{s2}$

The mapping between the view relation AUX_ITEM *and the base (*$\mathbf{S}_1$.ITEM *and* $\mathbf{S}_2$.PRODUCT_SALES*) is defined. The next step is to define the mapping from* AUX_ITEM *to* SALES_BY_CUSTOMER*. This is done using the following CAs:*

μ_{12}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [SALES_BY_CUSTOMER] ($\textit{prod_id_sk}_{DW}$, $\textit{cust_id_sk}_{DW}$, $\textit{date_id_sk}_{DW}$) →

→ *groupby* (**DW** [AUX_ITEM] (***product***, ***customer***, ***date***))

μ_{13}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{DW}}$ [SALES_BY_CUSTOMER]•$\textit{prodsold_qty}_{DW}$ → μ_{12}, ***sum*** (**DW** [AUX_ITEM] • ***quantity***)

Note that we use the notation **DW**[AUX_ITEM]*, although* AUX_ITEM *is only defined in the perspective schema. It is only a syntactic convention, since only schemata must appear on the right-side of a CA, and not perspective schemata.*

4.5 Perspective schema

Having introduced the concepts of "require" declarations, matching function signatures, correspondence assertions, and view relation declarations; we can define a perspective schema as follows:

Definition 29 (*Perspective schema) Let $\mathcal{P}$ be a set of property names, $\mathcal{K}$ a set of key names, $\mathcal{R}$ a set of relation names, $\mathcal{C}$ a set of class names, $\mathcal{V}$ a set of view relation names, $\mathcal{M}$ a set of method names, $\mathcal{T}$ a set of types defined over $\mathcal{P}$ and $\mathcal{C}$, $\mathcal{A}$ a set of correspondence assertion names, and $\mathcal{L}_p$ a set of perspective schema names. A perspective schema* **PS** *defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{V}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, and $\mathcal{K}$ has the form: $\left(\mathbf{S}, \hat{\mathcal{L}}, \left(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\right), \hat{\mathcal{V}}, \hat{\mathcal{A}}, \hat{\mathcal{F}}\right)$, such that:*

- **S** *is a schema name in $\mathcal{L}_p$ (the name of the perspective schema* **PS***);*
- *$\hat{\mathcal{L}}$ is a set of schemata, named **base**, whose schema names belongs to $\mathcal{L}$;*
- *$\hat{\mathcal{L}}_t$ is the **target** whose schema name belongs to $\mathcal{L}$;*
- *$\hat{\mathcal{V}}$ is a set, perhaps empty, of view relation declarations as defined in Section 4.4;*
- *$\hat{\mathcal{A}}$ is a set of correspondence assertions whose names belong to $\mathcal{A}$;*
- *$\hat{\mathcal{F}}$ is a set of matching function signatures;*
- *$\widehat{\mathcal{R}q}$ is a set of require declarations.* □

The correspondence assertions of a perspective schema relate the target schema's components to the base schemata's components, or relate the target schema's components to other target schemata's components, in the case of view relations. Perspective schemata can be used to:

- enforce access control (i.e., different users can see a same system in a diverse way);
- hide source data that is not required;
- combine data from different schemata of the same domain, and so provide an integrated access to this data.

As the structure of a perspective schema strongly depends on concepts of other schemata, it is necessary to verify if the perspective schema is well defined from a syntactic viewpoint (i.e., if it is a valid perspective schema). The following text shows some definitions to help in this task.

Definition 30 (*Valid "require" declarations) Let a perspective schema* **PS** $=\left(\mathbf{S}, \hat{\mathcal{L}}, \left(\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}\right), \hat{\mathcal{V}}, \hat{\mathcal{A}}, \hat{\mathcal{F}}\right)$ *defined over $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{V}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, and $\mathcal{K}$. A require declaration is a valid require declaration iff:*

- *The "require" declaration is an expression of the form:* require(C,{$\boldsymbol{p}_1$,$\boldsymbol{p}_2$,...,$\boldsymbol{p}_n$}) then:

  - C is a class name or a relation name of, respectively, a class declaration or a relation declaration declared in $\hat{\mathcal{L}}_t$;
  - $\boldsymbol{p}_i \in \boldsymbol{props}(\mathrm{C})$ and $\forall$i,j, if i $\neq$ j then $\boldsymbol{p}_i \neq \boldsymbol{p}_j$, $1 \leq \mathrm{i} \leq \mathrm{n}$;
  - If there is another "Require" declaration of the form: require(C$'$,$\{\boldsymbol{p'}_1, \boldsymbol{p'}_2, ..., \boldsymbol{p'}_m\}$) such that C = C$'$, then m=n and $\boldsymbol{p}_i = \boldsymbol{p'}_i$, for $1 \leq \mathrm{i} \leq \mathrm{n}$.

- The "require" declaration is an expression of the form: require($\boldsymbol{K}$) then:
  - $\boldsymbol{K}$ is a key name of a key declaration ($\boldsymbol{K}$,C,$\mathbf{K}$(C)) declared in $\hat{\mathcal{L}}_t$, such that there is a "Require" declaration require(C,$\{\boldsymbol{p}_1,\boldsymbol{p}_2,...,\boldsymbol{p}_n\}$) in $\widehat{\mathcal{R}q}$ and $\mathbf{K}(\mathrm{C}) \subseteq \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\}$;
  - $\boldsymbol{K}$ is a key name of a foreign key declaration ($\boldsymbol{FK}$,C,$\mathbf{FK}$(C),C$'$,$\mathbf{K}$(C$'$)) declared in $\hat{\mathcal{L}}_t$, such that there are the "Require" declarations require(C,$\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\}$) and require(C$'$,$\{\boldsymbol{p'}_1, \boldsymbol{p'}_2, ..., \boldsymbol{p'}_n\}$) in $\widehat{\mathcal{R}q}$, with $\mathbf{FK}(\mathrm{C}) \subseteq \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\}$ and $\mathbf{K}(\mathrm{C}') \subseteq \{\boldsymbol{p'}_1,\boldsymbol{p'}_2,...,\boldsymbol{p'}_n\}$;
  - If there is another "Require" declaration of the form: require($\boldsymbol{K'}$) then $\boldsymbol{K} \neq \boldsymbol{K'}$. □

Definition 31 (*Valid perspective schema) Let a perspective schema* **PS** $= \big(\mathbf{S}, \hat{\mathcal{L}}, (\hat{\mathcal{L}}_t, \widehat{\mathcal{R}q}), \hat{\mathcal{V}}, \hat{\mathcal{A}}, \hat{\mathcal{F}}\big)$ *defined over* $\mathcal{P}$, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{V}$, $\mathcal{M}$, $\mathcal{T}$, $\mathcal{A}$, $\mathcal{L}_p$, *and* $\mathcal{K}$. *Also let* C *be the name of a class/ relation in some "require" declaration in* $\widehat{\mathcal{R}q}$ *defined in* **PS**; V *be the name of some view relation declaration in* $\mathcal{V}$; *and* C_1, ..., C_n *be the name of a class/relation in some "require" declaration in* $\widehat{\mathcal{R}q}$. *A Perspective schema is a valid perspective schema iff:*

1. *All "require" declarations in* $\widehat{\mathcal{R}q}$ *are valid "require" declarations;*
2. $\hat{\mathcal{V}}$ *is a set, possibly empty, of view relation declarations;*
3. *For each* C *(or* V*):*
   (a) *There is at least one extension correspondence assertion of* C *(or* V*) or one summation correspondence assertion of* C *(or* V*) in* $\hat{\mathcal{A}}$;
   (b) *If there is more than one extension correspondence assertion of* C *(or* V*) or summation correspondence assertion of* C *(or* V*), then they connect* C *(or* V*) to distinct class names or relation names or view relation names.*
4. *For each* ψ *and* $\psi' \in \hat{\mathcal{A}}$ *such that* ψ *is an ACA and* ψ' *is the SCA assigned to* ψ, *then* ψ *and* ψ' *refer to the same class name or relation name of the same schema.*

5. *For each* $\boldsymbol{p}{:}\tau \in \boldsymbol{type}(\mathrm{C})$ *(or* $\boldsymbol{p}{:}\tau \in \boldsymbol{type}(\mathrm{V})$*):*

   (a) $\boldsymbol{p}$ *is referred to by a PCA* ψ *of* C *(or* V*), such that* ψ *relates* $\boldsymbol{p}$ *to one or more* $\boldsymbol{p}_1 \in \boldsymbol{props}(\mathrm{C}_1)$ *(or* $\boldsymbol{p}_1$ *is a path of* C_1*). Then, there is an ECA* ψ' *of* C *(or* V*) that relates* C *(or* V*) to* C_1*; or*

   (b) $\boldsymbol{p}$ *is referred to by an ACA* ψ *of* C *(or* V*), such that* ψ *relates* $\boldsymbol{p}$ *to one or more* $\boldsymbol{p}_1 \in \boldsymbol{props}(\mathrm{C}_1)$ *(or* $\boldsymbol{p}_1$ *is a path of* C_1*). Then, there is an SCA* ψ' *of* C *(or* V*), assigned to* ψ*, that relates* C *(or* V*) to* C_1*, and such that:*

      - *If* ψ' *is a SCA of* groupby*, then* ψ *is an ACA of* groupby *(Definition 28,* (4.7) *and* (4.8)*); otherwise,*
      - ψ *is an ACA of* normalise *(Definition 28,* (4.9) – (4.12)*)); or*

   (c) $\boldsymbol{p}$ *is referred to by a SCA* ψ *of* C *(or* V*), such that* ψ *relates* $\boldsymbol{p}$ *to one* $\boldsymbol{p}_1 \in \boldsymbol{props}(\mathrm{C}_1)$ *(or* $\boldsymbol{p}_1$ *is a path of* C_1*); or*

   (d) $\boldsymbol{p}$ *is a key of* C*. In this case, there is a key declaration* $(\boldsymbol{K}, \mathrm{C}, \{\boldsymbol{p}\}) \in \hat{\mathcal{K}}$*, or a "require" declaration require*$(\boldsymbol{K}) \in \widehat{\mathcal{R}q}$*, with* $(\boldsymbol{K}, \mathrm{C}, \boldsymbol{K}(\mathrm{C}))$ *being a key declaration in the target. The value of* $\boldsymbol{p}$ *is automatically generated by the system and* $\boldsymbol{p}$ *is named a surrogate key of* C*.*

   (e) *If there is more than one property correspondence assertion of* C *(or* V*) or aggregation correspondence assertion of* C *(or* V*) to the same* $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{C})$ *(or* $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{V})$*), then they connect* $\boldsymbol{p}$ *to properties or paths of distinct class names, relation names, or view relation names;*

6. $\forall\mathrm{C}$*, there is, at most, one surrogate key.*

7. $\hat{\mathcal{F}}$ *is a, possibly empty, set of matching function signatures built according to the following rules:*

   (a) *For each PCA or ACA* ψ *of* C *(or* V*); such that a)* ψ *relates* $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{C})$ *(or* $\boldsymbol{p} \in \boldsymbol{props}(\mathrm{V})$*) to* $\boldsymbol{p}_1 \in \boldsymbol{props}(\mathrm{C}_1)$ *(or* $\boldsymbol{p}_1$ *is a path of* C_1*), with* $\boldsymbol{p}$ *and* $\boldsymbol{p}_1$ *having, respectively, the reference types* $\natural\mathrm{C}_2$ *and* $\natural\mathrm{C}_3$*; and b)* C_2 *is semantically equivalent to* C_3*. Then there is a matching function signature in* $\hat{\mathcal{F}}$ *to match objects in* C_3 *to objects in* C_2*, in order to appropriately apply updates into property* $\boldsymbol{p}$*, when using a materialised view approach, or in order to put the correct value into* $\boldsymbol{p}$ *when using a virtual view approach. The same applies to the properties* $\boldsymbol{p}$ *and* $\boldsymbol{p}_1$ *being collections of reference types or structural types with at least one component having a reference type.*

(b) *For each ECA of equivalence or selection relating* C *(or* V*) to* C_1 *(e.g.,* $\mathbf{S}[C] \equiv \mathbf{S}_1[C_1]$ *or* $\mathbf{S}[C] \equiv \mathbf{S}_1[C_1\,(\textbf{pred})]$*), if a materialised view approach is used, then there is a matching function signature in* $\hat{\mathcal{F}}$ *to match instances of* C_1 *with instances of* C *(or* V*), in order to appropriately apply updates in* C *(or* V*) when insertions, changes (in property values), or deletions occur in* C_1*. Matching function signatures are not necessary when a virtual view approach is used.*

(c) *For each ECA of difference relating* C *(or* V*) to* C_1 *and* C_2 *(e.g.,* $\mathbf{S}[C] \equiv \mathbf{S}_1[C_1]$ - $\mathbf{S}_2[C_2]$*), if a materialised view approach is used then there are at least three matching function signatures in* $\hat{\mathcal{F}}$*: a) one to match instances of* C_1 *with instances of* C_2 *(in order to appropriately apply updates in* C *(or* V*) when insertions occur in* C_1*); b) another to match instances of* C_1 *with instances of* C *(or* V*) – in order to appropriately apply updates in* C *(or* V*) when deletions occur in* C_1*; c) and yet another to match instances of* C_2 *with instances of* C_1 *(in order to accordingly apply updates in* C *(or* V*) when insertions or deletions occur in* C_2*). Only one matching function signature is necessary to be defined when a virtual view approach is used: one to match instances of* C_2 *with instances of* C_1*.*

(d) *For each ECA of union relating* C *(or* V*) to* C_1*,* C_2*, ...,* C_n *(e.g.,* $\mathbf{S}[C] \equiv \mathbf{S}_1[C_1] ⟗ \mathbf{S}_2[C_2] ⟗ \ldots ⟗ \mathbf{S}_n[C_n]$*), if a materialised view approach is used then there are at least* $2n$ *matching function signatures in* $\hat{\mathcal{F}}$*: a)* n *to match instances of* C_i*, for* $1 \leq i \leq n$*, with instances of* C *(or* V*) – in order to properly apply updates in* C *(or* V*) when insertions or changes (in property values) occur in* C_i*; b) and* n *to match instances of* C *(or* V*) with instances of* C_i *(in order to appropriately apply updates in* C *when deletions occur in* C_i*). In this case, the functions previously defined to match instances of* C_j *with instances of* C *(or* V*) also are necessary, with* $j \neq i$*. In case a virtual view approach is used, there are at least n-2 matching function signatures in* $\hat{\mathcal{F}}$ *to match instances of* C_i *with instances of* C_{i-1}*,* $2 \leq i \leq n$*.*

(e) *For each ECA of intersection relating* C *(or* V*) to* C_1*,* C_2*, ...,* C_n *(e.g.,* $\mathbf{S}[C] \equiv \mathbf{S}_1[C_1] \cap \mathbf{S}_2[C_2] \cap \ldots \cap \mathbf{S}_n[C_n]$*), if the materialised view approach is used then there are at least* $2n$ *matching function signatures in* $\hat{\mathcal{F}}$*: a)* n *to match instances of* C_i*, for* $1 \leq i \leq n$*, with instances of* C *(or* V*) – in order to properly apply updates in* C *(or* V*) when deletions or changes (in property values) occur in* C_i*; b)* $n-1$ *to match instances of* C_i *with instances of* C_{i+1}*; c) and one to match instances of* C_n *with instances of* C_1 *(both are necessary in order to accordingly apply updates in* C *(or*

V*) when insertions occur in* C_i*). In case a virtual view approach is used, there are at least n matching function signatures in* $\hat{\mathcal{F}}$ *to match instances of* C_i *with instances of* C_{i+1}*,* $1 \leq i \leq n\text{-}1$. □

Hereafter we only work with valid perspective schemata.

Note that there are differences in the definitions of matching function signatures when they are used in a materialised view approach or a virtual one. This occurs mainly due to two reasons:

- when a virtual view approach is used, all data of the information sources is obtained once, when the target is accessed, and it is only in this situation that the matching function signatures are necessary;
- when using a materialised view approach, the matching function signatures are necessary when classes/relations of the target are firstly populated, and when these classes/relations must be updated. The matching function signatures in this case are defined with the aim of minimizing the access to the information sources.

In order to simplify the definition of perspective schemata, we propose two algorithms to identify missing matching function signatures: Procedure 1 and Procedure 2. Procedure 1 should be used when a virtual view approach is used, while Procedure 2 should be used when a materialised view approach is used.

Procedure 1 findMissingMF(`CAs`,`mfList`)

```
1:  for all ψ such that ψ ∈ CAs do
2:     if ψ is an ECA of difference relating C to C1 - C2 then
3:        findMF(C2,C1) or add MF(C2,C1) to mfList
4:     else if ψ is an ECA of union relating C to C1 ⊐⋈⊏ ...  ⊐⋈⊏ Cn then
5:        for i = 2 to n do
6:           findMF(Ci,Ci−1) or add MF(Ci,Ci−1) to mfList
7:        end for
8:     else if ψ is an ECA of intersection relating C to C1 ∩...∩ Cn then
9:        for i = 1 to n-1 do
10:          findMF(Ci,Ci+1) or add MF(Ci,Ci+1) to mfList
11:       end for
12:    end if
13: end for
```

In both Procedure 1 and Procedure 2, `CAs` is a list containing CA declarations of a perspective schema, mfList is a list containing MF signatures, C and C_1 are classes/relations,

findMF() is a procedure (predicate) to find a matching function signature between two classes/relations, and MF() is a procedure (predicate) to create a matching function signature between two classes/relations. In Procedure 1, lines 2 and 3 were based on Definition 31.7(c). Lines 4 to 7 were based on Definition 31.7(d). Lines 8 to 12 were based on Definition 31.7(e). In Procedure 2, lines 2 and 3 were based on Definition 31.7(a). Lines 4 and 5 were based on Definition 31.7(b). Lines 6 to 9 were based on Definition 31.7(c). Lines 10 to 14 were based on Definition 31.7(d). Lines 15 to 22 were based on Definition 31.7(e).

Procedure 2 findMissingMF(CAs,mfList)

```
1:  for all ψ such that ψ ∈ CAs do
2:     if ψ is a PCA or ψ is an ACA, such that ψ relates p:♮C to p1:♮C1 then[5]
3:        findMF(C1,C) or add MF(C1,C) to mfList
4:     else if ψ is an ECA of equivalence/selection relating C to C1 then
5:        findMF(C1,C) or add MF(C1,C) to mfList
6:     else if ψ is an ECA of difference relating C to C1 - C2 then
7:        findMF(C1,C2) or add MF(C1,C2) to mfList
8:        findMF(C1,C) or add MF(C1,C) to mfList
9:        findMF(C2,C1) or add MF(C2,C1) to mfList
10:    else if ψ is an ECA of union relating C to C1 ⟗ ... ⟗ Cn then
11:       for i = 1 to n do
12:          findMF(Ci,C) or add MF(Ci,C) to mfList
13:          findMF(C,Ci) or add MF(C,Ci) to mfList
14:       end for
15:    else if ψ is an ECA of intersection relating C to C1 ∩...∩ Cn then
16:       for i = 1 to n-1 do
17:          findMF(Ci,C) or add MF(Ci,C) to mfList
18:          findMF(Ci,Ci+1) or add MF(Ci,Ci+1) to mfList
19:       end for
20:       findMF(Cn,C) or add MF(Cn,C) to mfList
21:       findMF(Cn,C1) or add MF(Cn,C1) to mfList
22:    end if
23: end for
```

4.6 Pragmatic Examples

Up to this point we have shown, in our proposal, how to map between schemata and how to deal with usual ETL data transformations involving different classes of semantic heterogeneity. We illustrated the various types of mapping that we can do in our language. In this Section, we emphasise some situations that we deal with and point out some others that we cannot deal

[5] Also when type of $\boldsymbol{p}$ (and $\boldsymbol{p}_1$) is a collection of reference types or is a structural type with at least one component having a reference type.

with. Although it is non-exhaustive, the examples presented here give a good idea of the power of our language. Table 4.1 shows a summary of what is discussed in this Section.

Table 4.1: Sketch of the pragmatic examples.

Section	Emphasise	Scenario	Origin
4.6.1	Semantic heterogeneity	academic / market	(Sheth & Kashyap, 1993)
4.6.2	ECAs of union	market	(Calvanese, 2001) (Skoutas & Simitsis, 2006) (Vassiliadis *et al.*, 2002)
4.6.3	SCAs of groupby	academic	us
4.6.4	References and methods in perspective schemata	academic	(Santos *et al.*, 1994)

4.6.1 Examples involving semantic heterogeneity

We extracted some of the examples presented (Sheth & Kashyap, 1993) and defined them in our languages.

• **Data conflicts**

Naming conflicts:

Consider schema $\mathbf{S}_1$ having the relations:

(STUDENT, {**id**: integer, **name**: string, **address**: string})
(TEACHER, {**ss**: integer, **name**: string, **address**: string, **salary**: float})

with STUDENT.**id** and TEACHER.**ss** being synonyms. This may be indicated in our proposal by relating a same property of a relation (or class) of a perspective schema (e.g., **code** of relation PEOPLE) to STUDENT.**id** and TEACHER.**ss** as follows:

$$\omega_1\text{: } \mathbf{P}_{\mathbf{S}_1|\mathbf{S}}\,[\text{PEOPLE}] \bullet \mathbf{code} \rightarrow \mathbf{S}_1\,[\text{STUDENT}] \bullet \mathbf{id}$$

$$\omega_2\text{: } \mathbf{P}_{\mathbf{S}_1|\mathbf{S}}\,[\text{PEOPLE}] \bullet \mathbf{code} \rightarrow \mathbf{S}_1\,[\text{TEACHER}] \bullet \mathbf{ss}$$

Now consider the relation $\mathbf{S}_1$.BOOK defined as follows:

(BOOK, {**id**: integer, **name**: string, **author**: string})

STUDENT.**id** and BOOK.**id** are homonyms. We do not have a way of expressing this when two properties are homonyms, because we consider it unnecessary.

Data representation conflicts:

Consider, for example, that TEACHER.**ss** is a string, which is different from the data type of PEOPLE.**id**. In our proposal, we use functions to convert one data type into another. For this example, the mapping could be:

$$\omega_3\text{: } \mathbf{P}_{\mathbf{S}_1|\mathbf{S}}\ [\text{PEOPLE}] \bullet \mathbf{code} \rightarrow \textit{stringTOinteger}\ (\mathbf{S}_1\ [\text{TEACHER}] \bullet \mathbf{ss})$$

with *stringTOinteger* being a function to convert a numeric string to an integer.

Data unit conflicts:

Consider the example presented in (Hellerstein *et al.*, 1999) that maps a relation **S**.CUSTOMER from one data source (e.g., $\mathbf{S}_1$.CLIENT) using a simple conversion: changing names of states into a standard postal code. In our proposal, we deal with this type of conflict using functions. In this case, the mapping could be:

$$\omega_4\text{: } \mathbf{P}_{\mathbf{S}_1|\mathbf{S}}\ [\text{CUSTOMER}] \bullet \mathbf{state} \rightarrow \textit{twoLetter}\ (\mathbf{S}_1\ [\text{CLIENT}] \bullet \mathbf{state})$$

Data precision conflicts:

Consider the relations $\mathbf{S}_1$.SUBJECT with the property **marks** and **S**.SUBJECT with the property GRADES. **Marks** can have one integer value from 1 to 100, while **grades** can have one of the values in {A, B, C, D, E, F}. The mapping between the values of both properties is shown in Table 4.2

Table 4.2: Mapping between grades and marks.

Grades	A	B	C	D	E	F
Marks	88 – 100	73 – 87	58 – 72	42 – 57	21 – 41	1 – 20

In our proposal, we deal with this type of conflict using our PCA of case mapping, in which the value of a property depends on one or more conditions. For this example, the PCA could be as follows:

$$\begin{aligned}\omega_5\colon\quad \mathbf{P}_{\mathbf{S}_1|\mathbf{S}}\,[\textsc{subject}] \bullet \mathbf{grades} \rightarrow\quad & (\mathrm{A}, \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \geq 88)\,; \\ & (\mathrm{B}, \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \geq 73 \text{ and } \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \leq 87)\,; \\ & (\mathrm{C}, \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \geq 58 \text{ and } \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \leq 72)\,; \\ & (\mathrm{D}, \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \geq 42 \text{ and } \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \leq 57)\,; \\ & (\mathrm{E}, \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \geq 21 \text{ and } \mathbf{S}_1\,[\textsc{subject}] \bullet \mathbf{marks} \leq 41)\,; \\ & \mathrm{F}\end{aligned}$$

Default value conflicts:

There is no way of expressing default values in L_S or in L_{PS}, thus we cannot deal with this class of conflict in our proposal.

Attribute integrity constraint conflicts:

We do not deal with any type of integrity constraint other than *entity constraint* and *integrity referential constraint*. We cannot deal with this class of conflict.

• **Schematic conflicts**

Database identifier conflicts:

Consider the two key declarations presented below:

$(\mathbf{K}_1,\ \textsc{student1}, \{\mathbf{ss}\})$
$(\mathbf{K}_2,\ \textsc{student2}, \{\mathbf{name}\})$

$\mathbf{K}_1$ and $\mathbf{K}_2$ are semantically different. This class of conflict is important when dealing with the instance matching problem. In our proposal, the mapping at an instance level is done through MF signatures. We only identify the cases where the mapping is necessary and the implementation must be provided by the user. There is no way for the user to point out database identifier conflicts.

Naming conflicts:

Consider the schemata $\mathbf{S}_1$ and $\mathbf{S}$ having, respectively, the relations EMPLOYEE and WORKERS, which have the same set of instances. In our proposal, we can deal with this type of conflict by relating a relation (or class) to another using an ECA as follows:

$$\omega_6\colon\ \mathbf{P}_{\mathbf{S}_1|\mathbf{S}}\,[\textsc{workers}] \rightarrow \mathbf{S}_1\,[\textsc{employee}]$$

Now consider the relation $\mathbf{S}_2$.TICKETS and $\mathbf{S}_3$.TICKETS. The former models movie tickets, while the latter models traffic violation tickets. $\mathbf{S}_2$.TICKETS and $\mathbf{S}_3$.TICKETS are homonyms.

We do not have a way of expressing this when two classes or relations are homonyms. We consider it of no importance.

Union compatibility conflicts:

Consider schemata $\mathbf{S}_1$ and $\mathbf{S}_2$ having, respectively, the relations:

(STUDENT1, {**id**: integer, **name**: string, **grade**: integer})
(STUDENT2, {**id**: integer, **name**: string, **address**: string})

STUDENT1 and STUDENT2 are union incompatible. This problem does not occur in our proposal because our union is, indeed, the equivalent to the relational operator *natural outerjoin*. Thus, we do not need both relations (or classes) to have exactly the same type. In our proposal, we model the union using ECAs of union. In this example, the ECA of union could be as follows:

$$\omega_7\colon \mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}\,[\text{STUDENTS}] \rightarrow \mathbf{S}_1\,[\text{STUDENT1}] \;⟗\; \mathbf{S}_2\,[\text{STUDENT2}]$$

Supposing that ***type***(STUDENTS) = {**id**: integer, **name**: string, **grade**: integer, **address**: string}. The PCAs of STUDENTS could be as follows:

$$\omega_8\colon \mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}\,[\text{STUDENTS}] \bullet \mathbf{id} \rightarrow \mathbf{S}_1\,[\text{STUDENT1}] \bullet \mathbf{id}$$
$$\omega_9\colon \mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}\,[\text{STUDENTS}] \bullet \mathbf{id} \rightarrow \mathbf{S}_2\,[\text{STUDENT2}] \bullet \mathbf{id}$$
$$\omega_{10}\colon \mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}\,[\text{STUDENTS}] \bullet \mathbf{name} \rightarrow \mathbf{S}_1\,[\text{STUDENT1}] \bullet \mathbf{name}$$
$$\omega_{11}\colon \mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}\,[\text{STUDENTS}] \bullet \mathbf{name} \rightarrow \mathbf{S}_2\,[\text{STUDENT2}] \bullet \mathbf{name}$$
$$\omega_{12}\colon \mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}\,[\text{STUDENTS}] \bullet \mathbf{grade} \rightarrow \mathbf{S}_1\,[\text{STUDENT1}] \bullet \mathbf{grade}$$
$$\omega_{13}\colon \mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}\,[\text{STUDENTS}] \bullet \mathbf{address} \rightarrow \mathbf{S}_2\,[\text{STUDENT2}] \bullet \mathbf{address}$$

Schema isomorphism conflicts:

Consider schemata $\mathbf{S}_1$ and $\mathbf{S}_2$ having, respectively, the relations:

(INSTRUCTOR1, {**ss**: integer, **homePhone**: string, **OffPhone**: string})
(INSTRUCTOR2, {**ss**: integer, **phone**: string})

INSTRUCTOR1 and INSTRUCTOR2 have a schema isomorphism conflict.

In our proposal, we deal with this type of conflict using PCAs. In this example, we suppose that there is a relation (e.g., INSTRUCTOR) that is the union of INSTRUCTOR1 and INSTRUCTOR2, such that its property **phones** is a set of strings. In this case, we could have the following PCA for the property **phones**:

ω_{14}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}$[INSTRUCTOR] • **phones** → ($\mathbf{S}_1$[INSTRUCTOR1] • **homePhone**, $\mathbf{S}_1$[INSTRUCTOR1] • **OffPhone**)

ω_{15}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathbf{S}}$ [INSTRUCTOR] • **phones** → ($\mathbf{S}_2$ [INSTRUCTOR2] • **phone**)

We can also deal with schema isomorphism conflicts using PCAs with functions of transformations. Suppose, for example, schemata $\mathbf{S}_3$ and $\mathbf{S}_4$ having, respectively, the relations:

(CUSTOMER, {**firstname**: string, **lastname**: string, **address**: string})

(CLIENT, {**name**: string, **address**: string})

CUSTOMER and CLIENT have a schema isomorphism conflict. In this case, the value of **name** is constructed by concatenating the first name and last name. The PCA is as follows:

ω_{16}: $\mathbf{P}_{\mathbf{S}_3|\mathbf{S}_4}$ [CLIENT] • **name** → *concat* ($\mathbf{S}_3$ [CUSTOMER] • **firstname**, $\mathbf{S}_3$ [CUSTOMER] • **lastname**)

Missing data item conflicts:

Consider schema $\mathbf{S}_1$ with the relation STUDENT, and schema $\mathbf{S}_2$ with the relations GRAD_STUDENT and UNDERGRAD_STUDENT:

(STUDENT, {**ss**: integer, **name**: string, **type**: string})

(GRAD_STUDENT, {**ss**: integer, **name**: string})

(UNDERGRAD_STUDENT, {**ss**: integer, **name**: string})

STUDENT.**type** has values "UG" or "grad".

In our proposal, we deal with this type of conflict using PCAs of case mapping and the function *ground*, which determines if the instance came from GRAD_STUDENT or from UNDERGRAD_STUDENT. The PCA for this example could be as follows:

ω_{17}: $\mathbf{P}_{\mathbf{S}_2|\mathbf{S}_1}$ [STUDENT] • **type** → ("grad", *ground* ($\mathbf{S}_2$[GRAD_STUDENT]) = true) ; ' "UG"

4.6.2 Examples involving ECAs of union

In this Section we illustrate three situations involving ECAs of union. The examples were retrieved from (or based on) examples presented in (Calvanese, 2001; Skoutas & Simitsis, 2006; Vassiliadis *et al.*, 2002).

Example 1: specific "join conditions".

Based on the Example 4 presented in (Calvanese, 2001), consider two sources $\mathbf{S}_1$ and $\mathbf{S}_2$, both with the relation:

(PERSON, {**ssn**: integer, **name**: string, **dob**: date, **income**: float})

Suppose we want to store in the data warehouse pairs of people with a common income. In SQL-3, the correspondent query could be:

SELECT P1.**ssn**, P2.**ssn**, P1.**income**
FROM $\mathbf{S}_1$.PERSON as P1 and
$\mathbf{S}_2$.PERSON as P2
WHERE P1.**income** = P2.**income**

We cannot specify the join condition in our proposal and so we cannot map the previous example in our proposal. We assume that the matching ("join condition") between instances of the classes or relations in an ECA of union is done using only matching functions. This means that the "join condition" is only centred on identifying tuples/objects in order to match them (in the relational notation this would be equivalent to the "natural join"). We should create a new type of CA to deal with "join conditions" different to the "natural join". Even though this brings about some limitations to our proposal, it is not too critical. The main interest in a DIS design is to create *match classes* (Pequeno & Ponte Vidal, 2002; Zhou *et al.*, 1995b), which are classes/relations that are combined using the equivalent to the natural outerjoin in the relational notation.

Example 2: union with selection.

Consider the schemata $\mathbf{DS}_1$, $\mathbf{DS}_2$, and **DW** shown in (Skoutas & Simitsis, 2006):

$\mathbf{DS}_1$: (PRODUCTS, {**id**: integer, **name**: string, **amount**: float, **price**: float, **sid**: integer, **guarantee**: string, **type**: string})
(STORES, {**sid**: integer, **name**: string, **location**: string})

$\mathbf{DS}_2$: (SOFTWARE, {**id**: integer, **name**: string, **quantity**: integer, **price**: float, **sid**: integer})
(HARDWARE, {**id**: integer, **name**: string, **quantity**: integer, **price**: float, **sid**: integer})
(STORES, {**sid**: integer, **name**: string, **city**: string, **street**: string, **number**: string})

DW: (PRODUCTS, {**id**: integer, **name**: string, **price**: float, **quantity**: integer, **sid**: integer})
(STORES, {**sid**: integer, **name**: string, **city**: string, **street**: string})

The sources $\mathbf{DS}_1$ and $\mathbf{DS}_2$ contain information regarding products and stores. Products are distinguished between software and hardware. In $\mathbf{DS}_1$ this information is kept in

PRODUCTS.**type**. Prices are recorded in Euros in DS_1, and in Dollars in DS_2. The relation **DW**.PRODUCTS contain only software products, with prices ranging from 500 to 1500 Euros, a known quantity, and which are located in Rome or Athens. Each store has a name and an address, comprising city, street and number. In DS_1, this information is stored in property **location**. In **DW**, **street** contains both the street and number of the store. The properties **id** and **sid** in **DW** are surrogate keys, the primary keys from the sources are not kept in the data warehouse. Suppose that **sid** is also the name of the foreign key name of PRODUCTS (also SOFTWARE) that refers to STORES.

In our proposal the relations PRODUCTS and STORES of the **DW** are mapped using ECAs of union as follows:

$$\omega_{18}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{PRODUCTS}] \rightarrow \mathbf{DS}_1\,[\text{PRODUCTS}\,(\mathit{getcity}\,(\mathbf{sid} \bullet \mathbf{location}) = \text{“Rome”}$$

$$\text{or } \mathit{getcity}\,(\mathbf{sid} \bullet \mathbf{location}) = \text{“Athens” and } \mathbf{amount} \neq \text{null and } \mathbf{price} \geq 500 \text{ and } \mathbf{price} \leq 1500$$

$$\text{and } \mathbf{type} = \text{“software”})] \;\sqsupset\!\!\bowtie\!\!\sqsubset\; \mathbf{DS}_2\,[\text{SOFTWARE}\,(\mathbf{city} = \text{“Rome” or } \mathbf{city} = \text{“Athens” and}$$

$$\mathbf{amount} \neq \text{null and } \mathit{dollarTOeuro}\,(\mathbf{price}) \geq 500 \text{ and } \mathit{dollarTOeuro}\,(\mathbf{price}) \leq 1500)]$$

$$\omega_{19}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{STORES}] \rightarrow \mathbf{DS}_1\,[\text{STORES}] \;\sqsupset\!\!\bowtie\!\!\sqsubset\; \mathbf{DS}_2\,[\text{STORES}]$$

ω_{18} is an ECA of union involving a complex predicate (with functions, paths and various conditions). *getcity* is a function to retrieve the name of the city in location and *dollarTOeuro* is a function to convert Dollars to Euros. ω_{19} is trivial. Some mapping between properties is trivial and is not shown here, however others deserve our attention. They are presented in the following text.

$$\omega_{20}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{PRODUCTS}] \bullet \mathbf{sid} \rightarrow \mathbf{DS}_1\,[\text{PRODUCTS}] \bullet \mathbf{sid}$$

$$\omega_{21}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{PRODUCTS}] \bullet \mathbf{sid} \rightarrow \mathbf{DS}_2\,[\text{SOFTWARE}] \bullet \mathbf{sid}$$

$$\omega_{22}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{STORES}] \bullet \mathbf{city} \rightarrow \mathit{getcity}\,(\mathbf{DS}_1\,[\text{STORES}] \bullet \mathbf{location})$$

$$\omega_{23}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{STORES}] \bullet \mathbf{city} \rightarrow \mathbf{DS}_2\,[\text{STORES}] \bullet \mathbf{city}$$

$$\omega_{24}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{STORES}] \bullet \mathbf{street} \rightarrow \mathit{getstreet}\,(\mathbf{DS}_1\,[\text{STORES}] \bullet \mathbf{location})$$

$$\omega_{25}: \quad \mathbf{P}_{DS_1,DS_2|DW}\,[\text{STORES}] \bullet \mathbf{city} \rightarrow \mathit{concat}\,(\mathbf{DS}_2\,[\text{STORES}] \bullet \mathbf{street}, \mathbf{DS}_2\,[\text{STORES}] \bullet \mathbf{number})$$

ω_{20} and ω_{21} indicate the value in the source that must be used to find the correspondent surrogate key in **DW**. In this example, the primary key from the sources is not recorded in **DW**, however, we assume that it is stored somewhere else (e.g. the staging area), and so STORES.**sid** from the sources to STORES.**sid** of **DW** can be appropriately assigned. ω_{22} and ω_{23} map the properties **location** and **city**, using the function *getcity* when it is necessary. ω_{24} and ω_{25} map the properties **location**, **street** and **number**. ω_{24} uses the function *getstreet* to retrieve the street and the number of **location**, while ω_{25} uses the function *concat* to concatenate the street and the number.

Example 3: union in DW involving an aggregation in a source

Consider the schemata $\mathbf{S}_1$, $\mathbf{S}_2$, and **DW** shown in (Vassiliadis *et al.*, 2002):

$\mathbf{S}_1$: (PARTSUPP, {**pkey**: integer, **supkey**: integer, **qty**: integer, **cost**: float})

$\mathbf{S}_2$: (PARTSUPP, {**pkey**: integer, **supkey**: integer, **quantity**: integer, **cost**: float, **date**: date}
department: string)

DW: (PARTSUPP, {**pkey**: integer, **supkey**: integer, **quantity**: integer,**cost**: float, **date**: date})

"Relation **DW**.PARTSUPP stores daily (**date**) information for the available quantity (**qty**) and cost (**cost**) of parts (**pkey**) per supplier (**suppkey**)"(Vassiliadis *et al.*, 2002). The authors assume that data coming from $\mathbf{S}_2$ are in American values and formats and need be converted to European values and formats. Source $\mathbf{S}_2$ "captures information according to the particular department of the supplier organisation"(Vassiliadis *et al.*, 2002). In **DW**, this level of detail is not wanted, so the data from source $\mathbf{S}_2$ needs to be aggregated per **pkey**, **suppkey**, and **date** and a summation of cost and quantity must be done before the data is stored in **DW**.PARTSUPP.

In the example, **DW**.PARTSUPP must keep data from $\mathbf{S}_1$.PARTSUPP and the aggregation of $\mathbf{S}_2$.PARTSUPP. This type of mapping, as suggested in Section 4.4, is done with the help of view relations. In this case, we create a view relation, named AUX_PARTSUPP, as well as its CAs, to store aggregated data from source $\mathbf{S}_2$ and only afterwards can we define the ECA of union wanted. The view relation must keep all information wanted in **DW**.PARTSUPP, without links or path expressions to another class, relation, or view relation that is not in the perspective schema $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}$. AUX_PARTSUPP and its CAs are defined as follows:

(AUX_PARTSUPP, {**pkey**: integer, **supkey**: integer, **qty**: integer, **cost**: float, **date**: date})

ω_{26}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{AUX_PARTSUPP}]\,(\mathbf{pkey}, \mathbf{suppkey}, \mathbf{date}) \rightarrow \mathsf{groupby}\,(\mathbf{S}_2\,[\text{PARTSUPP}]\,(\mathbf{pkey}, \mathbf{suppkey}, \mathbf{date}))$

ω_{27}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{qty} \rightarrow \textit{sum}\,(\mathbf{S}_2\,[\text{PARTSUPP}] \bullet \mathbf{qty})$

ω_{28}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{cost} \rightarrow \textit{sum}\,(\mathbf{S}_2\,[\text{PARTSUPP}] \bullet \mathbf{cost})$

The ECA wanted, as well as the other CAs involving **DW**.PARTSUPP and AUX_PARTSUPP, are defined as follows:

ω_{29}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \rightarrow \mathbf{S}_1\,[\text{PARTSUPP}] ⟗ \mathbf{DW}\,[\text{AUX_PARTSUPP}]$

ω_{30}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{pkey} \rightarrow \mathbf{S}_1\,[\text{PARTSUPP}] \bullet \mathbf{pkey}$

ω_{31}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{pkey} \rightarrow \mathbf{DW}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{pkey}$

ω_{32}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{suppkey} \rightarrow \mathbf{S}_1\,[\text{PARTSUPP}] \bullet \mathbf{suppkey}$

ω_{33}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{suppkey} \rightarrow \mathbf{DW}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{suppkey}$

ω_{34}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{date} \rightarrow (\textit{aDateTOeDate}\,(\mathbf{DW}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{date}),$ $\textit{ground}\,(\mathbf{DW}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{date} = \text{true}))\,;\textit{today}$

ω_{35}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{qty} \rightarrow \mathbf{S}_1\,[\text{PARTSUPP}] \bullet \mathbf{qty}$

ω_{36}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{qty} \rightarrow \mathbf{DW}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{qty}$

ω_{37}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{cost} \rightarrow \mathbf{S}_1\,[\text{PARTSUPP}] \bullet \mathbf{cost}$

ω_{38}: $\mathbf{P}_{\mathbf{S}_1,\mathbf{S}_2|\mathrm{DW}}\,[\text{PARTSUPP}] \bullet \mathbf{cost} \rightarrow \textit{dollarTOeuro}\,(\mathbf{DW}\,[\text{AUX_PARTSUPP}] \bullet \mathbf{cost})$

Note that, although the view relation AUX_PARTSUPP is only defined in a perspective schema, it appears as **DW**[AUX_PARTSUPP] in CAs ω_{29} to ω_{38}. This is only a syntactic convention, since we only mention schemata, not perspective schemata, on the right side of our CAs. In ω_{34} *aDateTOeDate* is a function that transforms a date in American format to a date in European format; *today* is a function that retrieves the current date (remember that in $\mathbf{S}_1$.PARTSUPP there is no property corresponding to **DW**.PARTSUPP.**date**.).

4.6.3 Examples involving SCAs of group by

Here we illustrate an example created by us involving aggregated data in an academic scenario. Consider the source $\mathbf{S}_1$ and a data warehouse **DW** having the following relations:

$\mathbf{S}_1$: (GRADES, {**studentid**: integer, **courseid**: integer, **grade**: float})
(STUDENTS, {**studentid**: integer, **gender**: string})

DW: (EVALUATION, {**courseid**: integer, **grd_avg**: float, **male_grd_avg_abv10**: float, **count_abv10**: float, **count_bel10**: float})

Source $\mathbf{S}_1$ contains information about students, their courses and grades, while **DW** stores some statistics about students' grades: average grades by course (**grd_avg**), number of boys that passed in each course (**male_grd_avg_abv10**), number of students passed in each course (**count_abv10**), number of students failed in each course (**count_bel10**). Suppose that the foreign key name of GRADES that refers to STUDENT is **studentid**. In our proposal, this example is mapped using a SCA of group by and ACAs as follows:

ω_{39}: $\mathbf{P}_{\mathbf{S}_1|\mathrm{DW}}$ [EVALUATION] (**courseid**) $\rightarrow$ groupby ($\mathbf{S}_1$ [GRADES] (**courseid**))

ω_{40}: $\mathbf{P}_{\mathbf{S}_1|\mathrm{DW}}$ [EVALUATION] $\bullet$ **grd_avg** $\rightarrow \omega_{39}$, *avg* ($\mathbf{S}_1$ [GRADES] $\bullet$ **grade**)

ω_{41}: $\mathbf{P}_{\mathbf{S}_1|\mathrm{DW}}$ [EVALUATION] $\bullet$ **count_abv10** $\rightarrow \omega_{39}$, *count* ($\mathbf{S}_1$ [GRADES] $\bullet$ **grade**,
$\mathbf{S}_1$ [GRADES] $\bullet$ **grade** ≥ 10)

ω_{42}: $\mathbf{P}_{\mathbf{S}_1|\mathrm{DW}}$ [EVALUATION] $\bullet$ **count_bel10** $\rightarrow \omega_{39}$, *count* ($\mathbf{S}_1$ [GRADES] $\bullet$ **grade**,
$\mathbf{S}_1$ [GRADES] $\bullet$ **grade** < 10)

ω_{43}: $\mathbf{P}_{\mathbf{S}_1|\mathrm{DW}}$ [EVALUATION] $\bullet$ **male_grd_count_abv10** $\rightarrow \omega_{39}$, *count* ($\mathbf{S}_1$ [GRADES] $\bullet$ **grade**,
$\mathbf{S}_1$ [GRADES] $\bullet$ **grade** ≥ 10 and $\mathbf{S}_1$ [GRADES] $\bullet$ **studentid** $\bullet$ **gender** = "M")

4.6.4 Examples involving methods and references

In this Section, we present a simple scenario involving the creation of perspective schemata with methods and references (pointers). This example is based on another shown in (Santos *et al.*, 1994). Consider the source $\mathbf{S}_1$, named *company_schema* in (Santos *et al.*, 1994), and the schema **V**, which is the view schema *company_view* in (Santos *et al.*, 1994), having the following classes:

$\mathbf{S}_1$: (PERSON, {**id**: integer, **name**: string},all,∅)

(EMPLOYEE, {**salary**: float, **department**: ♮DEPARTMENT, **date_of_admission**: date}, PERSON, ∅)

(DEPARTMENT, {**name**: string, **boss**: ♮EMPLOYEE}, all, ∅)

V: (PERSON, {**id**: integer, **name**: string},all,∅)

(EMPLOYEE, {**salary**: float, **department**: ♮DEPARTMENT, **date_of_admission**: date}, PERSON, ∅)
(BOSS, {**subordinates**: {♮EMPLOYEE}}, EMPLOYEE, {*number_of_subordinates*: BOSS×♮EMPLOYEE →integer})

(DEPARTMENT, {**name**: string, **boss**: ♮BOSS, **employees**: {♮EMPLOYEE}}, all, {*number_of_employees*: BOSS×♮EMPLOYEE →integer})

A view schema in other languages corresponds to a schema and a perspective schema in our proposal. Thus, we must define the schema **V** and a perspective schema $\mathbf{P}_{s1|v}$ containing all components of **V** and the mapping between **V** and $\mathbf{S}_1$. However, the method signatures cannot be defined in the perspective schema, since there is nothing about them in language L_{PS}. We can define the relationship between the classes PERSON, EMPLOYEE, BOSS, and DEPARTMENT of $\mathbf{P}_{s1|v}$ and $\mathbf{S}_1$, using ECAs and a SCA of normalise as follows:

ω_{44}: **V** [PERSON] → $\mathbf{S}_1$ [PERSON]

ω_{45}: **V** [BOSS] (**name**) → normalise ($\mathbf{S}_1$ [EMPLOYEE] (**department • boss • name**))

ω_{46}: **V** [EMPLOYEE] → $\mathbf{S}_1$ [EMPLOYEE] − **V** [BOSS]

ω_{47}: **V** [DEPARTMENT] → $\mathbf{S}_1$ [DEPARTMENT]

However, some properties cannot be related: we cannot create a mapping to **V**.BOSS.**subordinates**, as well as to **V**.DEPARTMENT.**employee**.

4.7 Conclusions

This chapter covers a declarative approach to make explicit the relationship between information sources and the global schema in the DISs, not only at a structural level, but also at an instance level, independently of the ETL process involved. The language presented here is an extension of the language presented in the previous chapter, and is used to declaratively define perspective schemata.

A perspective schema describes a data model, part or whole (*the target*), in terms of other data models (*the base*). It represents the mapping from one or more schemata (e.g., the information sources) to another (e.g., the reference model). In the proposed approach, the relationship between the base and the target is made explicitly and declaratively through CAs. By using the perspective schemata the designer has a formal representation, with a well defined semantic, which allows for: a) the definition of diverse points of views of the (same or different) source information; b) dealing with semantic heterogeneity as well as the usual and non-trivial mapping between schemata in a declarative way; c) indicating the situations when the user must deal with the instance matching problem; and d) reusing (a same perspective schema can used simultaneously in several systems, such as application, DW, and FDBS).

This Chapter also illustrated some practical situations using short examples,. We try to demonstrate the feasibility and usefulness of our language and point out some cases which we cannot deal with yet. Parts of this Chapter were published in (Pequeno & Pires, 2009b).

In the current work, we do not define correspondence involving methods, nor how we should declare them in the perspective schemata. However, research about how we should deal with methods in a data integration environment is an interest subject.

In our proposal, all schemata must only be aligned to the reference model through perspective schemata, and any other map is automatically (or semi-automatically) created using an inference mechanism. The next chapter details the process used to infer a new perspective schema based on schemata and perspective schemata that have already been defined.

5

The Inference Mechanism

Up until now we have presented a way to express the existing data models (sources, global schema, and reference model) and the relationship between them. We propose that each schema must only be mapped with the reference model, rather than with each participating schema, being that the mapping between the global schema and its sources is (semi-) automatically generated by an inference mechanism.

This Chapter focuses on the deduction of new perspective schemata using a proposed inference mechanism.

We deal with the problem of integrating information among multiple heterogeneous and autonomous databases by proposing a declarative approach based on the creation of a reference model and perspective schemata. Using the proposed architecture, a new perspective schema can be inferred from schemata and perspective schemata previously defined. The inference mechanism parameters are:

1. a schema (named *destination* (see Fig. 5.1(a));
2. another schema (named *intermediary* (see Fig. 5.1(b));
3. a set of schemata (named *origin*) and perspective schemata (see Fig. 5.1(c)).

A subset of schemata in the *origin* will be the *base* of the new perspective schema, while the *destination* will be the *target*. The *intermediary* here works like a bridge between the *origin* and the *destination*. The perspective schemata are associated with the *destination*, the *intermediary* or the *origin* in the following way: they take some of the schemata of the origin as the *base* and the *intermediary* as the *target*; or they take the *intermediary* as the *base* and the *destination*

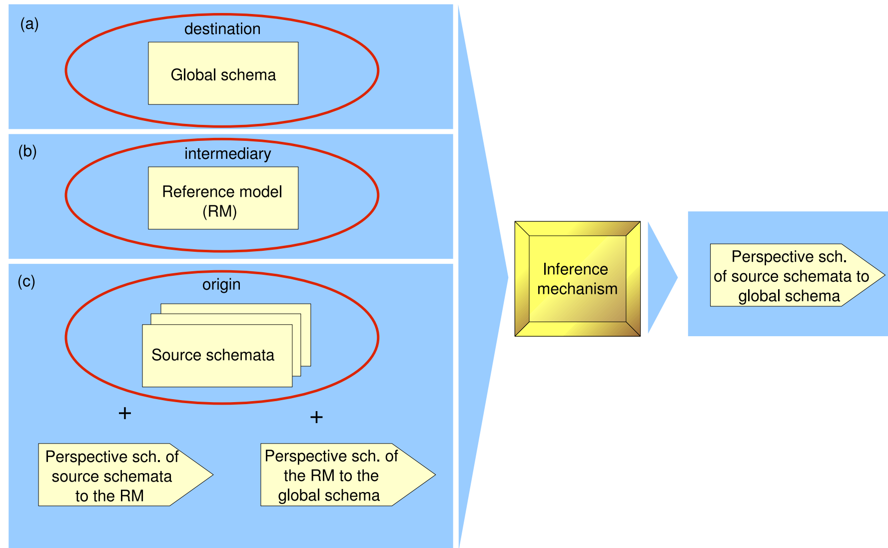


Figure 5.1: Sketch of the inference mechanism.

as the *target*. Normally, in DISs, there is only a perspective schema between the information sources and the intermediary. This perspective schema represents the data integration and is called *perspective schema of integration*, further details about this will be shown through the chapter.

The output of the inference mechanism is a new perspective schema from a subset of schemata in the origin to the destination. The new perspective schema will have the same classes, relations, keys and foreign keys as defined in the perspective schema associated with the *destination*. The CAs of the new perspective schema will be (semi-) automatically generated using a rule-based rewriting system, while the MF signatures will be automatically defined based on the CAs of the new perspective schema.

In the following text, we present the rewriting system. Then, we detail the inference process, followed by explanations of the inference mechanism. Then we present some inference mechanism validations. After that, we show some case studies. Finally, some conclusions are drawn.

5.1 Rewriting Rules

The inference mechanism uses a rule-based rewriting system to (semi-) automatically generate new CAs. The rule-based rewriting system is formed by a set of rules with the same general form:

$$\textbf{Rule :} \ \frac{X \Rightarrow Y}{Z} \quad \text{(read } X \text{ is rewritten in } Y \text{ if } Z \text{ is valid)} \ . \tag{5.1}$$

The notation used in (5.1) means:

- **Rule** is the name of the rule.
- X can be formed by any of the following expressions: a CA pattern expression; a component pattern expression; or an expression of form: (**S**[E],$\Diamond$). CA pattern expressions are expressions conforming to the L_{PS} syntax to declare CAs, being that some of their elements are variables to be used in a unification process. Component pattern expressions are expressions conforming to the L_S or the L_{PS} syntax to represent elements that can appear in CAs (properties, path expressions, functions with n-ary arguments, conditions of selection, values, or links of a path expression), being that some of their elements are variables to be used in a unification process. The expression (**S**[E],$\Diamond$) is an expression such that: E indicates a class/relation/view relation with or without a predicate of selection, **S** is the schema where E was defined, and $\Diamond$ is one of the operators used in an ECA (i.e., $\sqsupset\!\!\bowtie\!\!\sqsubset, -, \cap$).
- Y can be formed by any of the following expressions: a CA pattern expression, a component pattern expression; or the right-side of an ECA pattern expression. The latter is a particular type of CA pattern expression in which only ECAs are allowed.
- Z is a condition formed by a set of:
  - CA pattern expressions;
  - expressions of the forms:
    1. $A \Rightarrow B$, such that A and B are component pattern expressions; or A is a class/relation/view relation of a schema and B is the right-side of an ECA pattern expression;
    2. (***FK***,C,_,C′,_), a foreign key declaration, such that ***FK*** is a foreign key name of a class/relation C that refers to a class/relation C′;
    3. **p**:$\natural$C $\in$ ***type***(C′), such that **p** is a property name declared in a class/relation C′ that refers to a class C.[1]

[1] ***type***(): see Chapter 3, Definition 4.

– procedures, which are used to manipulate view relations: they verify if a view relation was created when an ECA or SCA was analysed; retrieve a view relation that was created when an ECA or SCA was analysed; create a view relation; and add properties to a view relation.

A condition Z is valid when all of its expressions are valid. More detail about this is shown later (Section 5.2). CA pattern expressions, and component pattern expressions, are formally defined in the following text. In order to simplify the definitions, some pattern expressions are broken down in simpler pattern expressions, which are defined as follows:

Definition 32 (*Basic pattern expression) Let $\mathcal{L}$ be a set of schema names. A basic pattern expression is an expression that conforms to the L_S or L_{PS} language having one of the following forms:*

$$\textit{Basic pattern expressions} \begin{cases} \underline{\mathbf{S}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} \\ \underline{\mathbf{S}}\,[\underline{\mathrm{C}}]\bullet\underline{\varrho} \\ \underline{\mathbf{S}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}}\{\underline{\boldsymbol{p}'}_t\} \\ \underline{\varphi}\left(\underline{A}_1, \underline{A}_2, \ldots, \underline{A}_n\right) \end{cases}$$

with $\underline{\mathbf{S}}$, $\underline{\mathrm{C}}$, $\underline{\boldsymbol{p}}$, $\underline{\varrho}$, and $\underline{A}_i$, for $1 \leq i \leq n$, being variables, which can be instantiated, respectively, with: a schema name in $\mathcal{L}$; a class/relation name of the schema $\mathbf{S}$*; a property name defined in the class/relation* C*; a path expression of class/relation* C*; and a basic pattern expression.* □

Hereafter, all variables are indicated in the text by an underline.

Definition 33 (*Predicate pattern expression) Let $\mathcal{L}$ be a set of schema names. A predicate pattern expression is an expression that conforms to the L_{PS} language to define predicates having one of the following forms:*

$$\textit{Predicate pattern expressions} \begin{cases} \underline{A}\ \underline{\boldsymbol{op}}\ \underline{B} \\ \underline{A}\ \underline{\boldsymbol{op}}\ \underline{B} \quad \textit{and}\ \underline{\textsf{pred}} \\ \underline{A}\ \underline{\boldsymbol{op}}\ \underline{B} \quad \textit{or}\ \underline{\textsf{pred}} \end{cases}$$

with $\underline{A}$, $\underline{B}$, $\underline{\textsf{pred}}$, and $\underline{\boldsymbol{op}}$ being variables that can be instantiated with, respectively: a basic pattern expression; a value or a basic pattern expression; a predicate pattern expression; and one of the predicate operands $(>, <, \geq, \leq, =, \neq)$ present in Chapter 4, Definition 24. □

Definition 34 (*PCA pattern expression*) *Let $\mathcal{L}$ be a set of schema names and $\mathcal{L}_p$ be a set of perspective schema names. A PCA pattern expression is an expression that conforms to the L_{PS} language to define PCAs having one of the following forms:*

$$\textit{PCA pattern expressions}\begin{cases}\mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \underline{A}\\ \mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \left(\underline{A_1},\underline{A_2},\ldots,\underline{A_n}\right)\\ \mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \left(\underline{B_1},\underline{\mathbf{pred}_1}\right);\ldots;\left(\underline{B_{n-1}},\underline{\mathbf{pred}_{n-1}}\right);\underline{B_n}\\ n\mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}}\{\underline{\boldsymbol{p}_1},\ldots,\underline{\boldsymbol{p}_n}\} & \rightarrow \left(\underline{A_1},\underline{A_2},\ldots,\underline{A_n}\right)\end{cases}$$

with $\mathbf{P}_{x|y}$ being a perspective schema name $\in \mathcal{L}_p$ (from schema $\mathbf{X}$ to schema $\mathbf{Y}$). $\underline{x}$ and $\underline{y}$, $\underline{\mathrm{C}}$, $\underline{\boldsymbol{p}}$, $\underline{A}$ (and A_i), $\underline{\mathbf{pred}}$, and $\underline{B_i}$ (for $1 \le i \le n$) are variables, which can be instantiated, respectively, with: a schema name in $\mathcal{L}$; a class/relation name of the schema $\mathbf{Y}$; a property name defined in class C; a basic pattern expression; a predicate pattern expression; and a value or a basic pattern expression. The variables in $\underline{A}$, $\underline{\mathbf{pred}}$, and $\underline{B}$ will be instantiated with elements of schema $\mathbf{X}$. □

Definition 35 (*Extension pattern expression*) *Let $\mathcal{L}$ be a set of schema names. An extension pattern expression is an expression that conforms to the L_{PS} language having one of the following forms:*

$$\textit{Extension pattern expressions}\begin{cases}\underline{\mathbf{S}}\,[\underline{\mathrm{C}}]\\ \underline{\mathbf{S}}\,[\underline{\mathrm{C}}\,(\underline{\mathbf{pred}})]\end{cases}$$

with $\underline{\mathbf{S}}$, $\underline{\mathrm{C}}$, and $\underline{\mathbf{pred}}$ being variables that can be instantiated with, respectively: a schema name $\in \mathcal{L}$; a class/relation name defined in $\mathbf{S}$; and a predicate pattern expression, whose variables will be instantiated with elements of schema $\mathbf{S}$. □

Definition 36 (*ECA pattern expression*) *Let $\mathcal{L}$ be a set of schema names, and $\mathcal{L}_p$ a set of perspective schema names. An ECA pattern expression is an expression that conforms to the L_{PS} language to define ECAs having one of the following forms:*

$$\textit{ECA pattern expressions}\begin{cases}\mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\right] & \rightarrow \underline{E_1}\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\left(\underline{\textbf{pred}}\right)\right] & \rightarrow \underline{E_1}\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\right] & \rightarrow \underline{E_1} - \underline{E_2}\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\left(\underline{\textbf{pred}}\right)\right] & \rightarrow \underline{E_1} - \underline{E_2}\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\right] & \rightarrow \underline{E_1} \cap \underline{E_2} \cap \cdots \cap \underline{E_n}\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\left(\underline{\textbf{pred}}\right)\right] & \rightarrow \underline{E_1} \cap \underline{E_2} \cap \cdots \cap \underline{E_n}\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\right] & \rightarrow \underline{E_1} ⟗ \underline{E_2} ⟗ \ldots ⟗ \underline{E_n}\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\left(\underline{\textbf{pred}}\right)\right] & \rightarrow \underline{E_1} ⟗ \underline{E_2} ⟗ \ldots ⟗ \underline{E_n}\end{cases}$$

with $\mathbf{P}_{x|y}$ *being a perspective schema name* $\in \mathcal{L}_p$ *(from schema* $\mathbf{X}$ *to schema* $\mathbf{Y}$*).* $\underline{x}$ *and* $\underline{y}$*,* $\underline{\mathrm{C}}$*,* $\underline{\textbf{pred}}$*, and* $\underline{E_i}$ *(for 1* $\leq i \leq$ *n) are variables that can be instantiated with, respectively: a schema name in* $\mathcal{L}$*, a class/relation name of schema* $\mathbf{Y}$*; a predicate pattern expression; and an extension pattern expression. The variables in* $\underline{\textbf{pred}}$*, and in* $\underline{E_i}$ *will be instantiated, respectively, with elements of schemata* $\mathbf{Y}$ *and* $\mathbf{X}$*.* □

Definition 37 (*SCA pattern expression) Let* $\mathcal{L}$ *be a set of schema names, and* $\mathcal{L}_p$ *a set of perspective schema names. A SCA pattern expression is an expression that conforms to the* L_{PS} *language to define SCAs having one of the following forms:*

$$\textit{SCA pattern expressions}\begin{cases}\mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\right]\left(\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\right) & \rightarrow \textsf{\textit{groupby}}\left(\underline{E}\left(\underline{A_1}, \underline{A_2}, \ldots, \underline{A_n}\right)\right)\\ \mathbf{P}_{\underline{x}|\underline{y}}\left[\underline{\mathrm{C}}\right]\left(\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\right) & \rightarrow \textsf{\textit{normalise}}\left(\underline{E}\left(\underline{A_1}, \underline{A_2}, \ldots, \underline{A_n}\right)\right)\end{cases}$$

with $\mathbf{P}_{x|y}$ *being a perspective schema name* $\in \mathcal{L}_p$ *(from schema* $\mathbf{X}$ *to schema* $\mathbf{Y}$*).* $\underline{x}$ *and* $\underline{y}$*,* $\underline{\mathrm{C}}$*,* $\underline{\boldsymbol{p}_i}$*,* $\underline{E}$*, and* $\underline{A_i}$*, for* $1 \leq i \leq n$*, are variables that can be instantiated with, respectively: a schema name in* $\mathcal{L}$*, a class/relation name of schema* $\mathbf{Y}$*; a property name defined in class/relation* C*; an extension pattern expression; and a basic pattern expression. The variables in* $\underline{E}$ *and in* $\underline{A_i}$ *will be instantiated with elements of schema* $\mathbf{X}$*.* □

Definition 38 (*ACA pattern expression) Let* $\mathcal{L}$ *be a set of schema names, and* $\mathcal{L}_p$ *a set of perspective schema names. An ACA pattern expression is an expression that conforms to the* L_{PS} *language to define ACAs having one of the following forms:*

$$\text{ACA pattern expressions}\begin{cases}\mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \underline{\psi}, \underline{\gamma}\,(\underline{\mathsf{A}})\\ \mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \underline{\psi}, \underline{\gamma}\,(\underline{\mathsf{A}}, \underline{\textbf{pred}})\\ \mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \underline{\psi}, \underline{\mathsf{A}}\\ \mathbf{P}_{\underline{x}|\underline{y}}\,[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \underline{\psi}, (\underline{\mathsf{A}_1}, \underline{\mathsf{A}_2}, \ldots, \underline{\mathsf{A}_n})\\ \mathbf{P}_{\underline{x}|\underline{y}}[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}} & \rightarrow \underline{\psi}, \left(\underline{\mathsf{B}_1}, \underline{\textbf{pred}_1}\right);\ldots;\left(\underline{\mathsf{B}_{n-1}}, \underline{\textbf{pred}_{n-1}}\right);\underline{\mathsf{B}_n}\\ \mathbf{P}_{\underline{x}|\underline{y}}[\underline{\mathrm{C}}]\bullet\underline{\boldsymbol{p}}\{\underline{\boldsymbol{p}_1},\ldots,\underline{\boldsymbol{p}_n}\} & \rightarrow \underline{\psi}, (\underline{\mathsf{A}_1}, \underline{\mathsf{A}_2},\ldots, \underline{\mathsf{A}_n})\end{cases}$$

with $\mathbf{P}_{x|y}$ *being a perspective schema name* $\in \mathcal{L}_p$ *(from schema* $\mathbf{X}$ *to schema* $\mathbf{Y}$*).* $\underline{x}$ *and* $\underline{y}$, $\underline{\mathrm{C}}$, $\underline{\boldsymbol{p}}$, $\underline{\psi}$, $\underline{\gamma}$, $\underline{\mathsf{A}}$ *(and* A_i*),* $\underline{\textbf{pred}}$ *(and* $\textbf{pred}_j$*, with 1* $\leq j \leq$ *n-1), and* $\underline{\mathsf{B}_i}$ *(1* $\leq i \leq n$*) are variables, which can be instantiated, respectively, with: a schema name in* $\mathcal{L}$*; a class/relation name of schema* $\mathbf{Y}$*; a property name defined in class* C*; the name of a SCA; an aggregation function (sum, max, min, count, and avg); a basic pattern expression; a predicate pattern expression; and a value or a basic pattern expression. The variables in* $\underline{\mathsf{A}}$, $\underline{\textbf{pred}}$*, and* $\underline{\mathsf{B}}$ *will be instantiated with elements of the schema* $\mathbf{X}$*.* □

Definition 39 **(***CA pattern expression) A CA pattern expression is any expression of form:*

$$\text{CA pattern expressions}\begin{cases}\text{PCA pattern expression}\\ \text{ECA pattern expression}\\ \text{SCA pattern expression}\\ \text{ACA pattern expression}\end{cases}$$

□

Definition 40 **(***Component pattern expression) Let* $\mathcal{L}$ *be a set of schema names, and* $\mathcal{W}$ *be a set of typed values. A Component pattern expression is an expression that conforms to the* L_{PS} *language having one of the following forms:*

$$\text{Component pattern expressions}\begin{cases}\underline{\mathsf{A}}\\ \underline{\textbf{pred}}\\ \underline{w}\\ \underline{\ell} : \underline{\mathrm{C}_1} \multimap \underline{\mathrm{C}_2}\\ \underline{\ell_1} : \underline{\mathrm{C}} \multimap \underline{\mathrm{C}_1} \bullet \underline{\ell_2} : \underline{\mathrm{C}_1} \multimap \underline{\mathrm{C}_2} \bullet \cdots \bullet \underline{\ell_n} : \underline{\mathrm{C}_{n-1}} \multimap \underline{\mathrm{C}_n}\end{cases}$$

With $\underline{A}$, $\underline{\textbf{pred}}$, $\underline{w}$, $\underline{\mathrm{C}_i}$, $\underline{\ell}$ *(and* $\underline{\ell_i}$*), being variables that can be instantiated with, respectively: a basic pattern expression; a predicate pattern expression; a value* $\in \mathcal{W}$*; a class/relation name; and a property name or a foreign key name of class/relation* C_{i-1} *that refers to class/relation name* C_i*.* □

When X and Y are CA pattern expressions, the rules are rewritten-rules that rewrite CAs in other CAs (RR-CAs). Otherwise, the rules are substitution-rules that rewrite components in other components or a set of instances in another set of instances (SR-Cs). The latter are used as an intermediary process by the RR-CAs.

An example of a RR-CA is as follows:

$$\textbf{RR-CA}: \frac{\mathbf{P}_{\mathbf{I}|\mathbf{D}}\left[\underline{\mathrm{C}^{\mathbf{D}}}\right] \rightarrow \mathbf{I}\left[\underline{\mathrm{C}^{\mathbf{I}}}\right] \Rightarrow \mathbf{P}_{\underline{\mathbf{S}}|\mathbf{D}}\left[\underline{\mathrm{C}^{\mathbf{D}}}\right] \rightarrow \underline{\mathbf{K}^{S}}}{\mathbf{P}_{\underline{\mathbf{S}}|\mathbf{I}}\left[\underline{\mathrm{C}^{\mathbf{I}}}\right] \rightarrow \underline{\mathbf{K}^{S}}}\ . \tag{5.2}$$

In (5.2), $\mathbf{D}$ is the *destination*, $\mathbf{I}$ is the *intermediary*, and $\underline{\mathbf{S}}$ is a variable that will be instantiated with some *origin* schemata. $\underline{\mathrm{C}^{\mathbf{D}}}$ is a variable that will be instantiated with a class/relation of $\mathbf{D}$; mutates mutandis to $\underline{\mathrm{C}^{\mathbf{I}}}$. $\underline{\mathbf{K}}$ is a variable that will be instantiated with the right side of an ECA pattern expression. The letter "S" in $\mathbf{K}^{\mathbf{S}}$ means that all elements in that expression belong to schema in $\underline{\mathbf{S}}$. The value of $\underline{\mathbf{S}}$ and $\underline{\mathbf{K}}$ will depend on which CA (declared in the perspective schema associated with some *origin* schemata) will unify with the condition of the rule. The rule **RR-CA** rewrites an ECA of equivalence in another ECA (maybe other than equivalence) by replacing a class/relation of the *intermediary* ($\underline{\mathrm{C}^{\mathbf{I}}}$) by one class/relation (or an expression containing classes/relations) of the *origin* schemata ($\underline{\mathbf{K}^{\mathbf{S}}}$); when an ECA is provided that connects the class/relation $\underline{\mathrm{C}^{\mathbf{I}}}$ to $\underline{\mathbf{K}^{\mathbf{S}}}$. A more realistic example is as follows:

Example 20

Consider the schemata presented in Figures 4.1 and 4.2, and its associated perspective schema (not presented in any figure), **DW** *being the destination,* **RM** *the intermediary, and* $\mathbf{S}_1$ *the origin schema. Also consider the following ECAs:*

ψ_{19}: $\mathsf{P}_{\mathsf{RM|DW}}$ [CUSTOMER] $\rightarrow$ **RM** [CUSTOMER]

ψ_{20}: $\mathsf{P}_{\mathsf{S1|RM}}$ [CUSTOMER] $\rightarrow$ **S1** [CUSTOMER]

Using CA ψ_{19} as a start point, we can use (5.2) *to create new CAs. Observe that ψ_{19} unifies with the CA pattern expression above left of the* ***RR-CA*** *in* (5.2)*, and ψ_{20} unifies with its condition. In this case, the new CA generated is:*

$$\alpha_1 : \mathsf{P}_{\mathsf{S1|DW}} [\text{CUSTOMER}] \rightarrow \mathsf{S1} [\text{CUSTOMER}] \quad .$$

An example of a SR-C is as follows:

$$\textbf{SR-C :} \frac{\mathbf{I} \left[\underline{\mathrm{C}^{\mathbf{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathbf{I}}} \Rightarrow \underline{\mathbf{A}}^{S}}{\mathbf{P}_{\underline{\mathbf{S}}|\mathbf{I}} \left[\underline{\mathrm{C}^{\mathbf{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathbf{I}}} \rightarrow \underline{\mathbf{A}}^{S}} \quad . \tag{5.3}$$

In (5.3), $\underline{\boldsymbol{p}^{\mathbf{I}}}$ is a variable that will be instantiated with a property of a class/relation of the *intermediary*, and $\underline{\mathbf{A}}$ is a variable that will be instantiated with a basic pattern expression. Similarly to $\underline{\mathbf{K}^{\mathbf{S}}}$ in (5.2), the letter “S” in $\underline{\mathbf{A}^{\mathbf{S}}}$ means that all elements in that expression belong to schema in $\underline{\mathbf{S}}$. The value of $\underline{\mathbf{S}}$ and $\underline{\mathbf{A}}$ will depend on which CA is declared in the perspective schema associated with some of the *origin* schemata that will unify with the condition of the rule. The rule **SR-C** rewrites a property $\underline{\boldsymbol{p}^{\mathbf{I}}}$ of a class/relation of the *intermediary* schema into a property, a path expression, or a function of a class/relation of some *origin* schema (here represented by the variable $\underline{\mathbf{A}^{\mathbf{S}}}$); when it is provided by a PCA that connects $\underline{\boldsymbol{p}^{\mathbf{I}}}$ to that property, path expression, or function. A more realistic example is as follows:

Example 21 ———————————

Continuing example 20, now consider the following PCAs:

ψ_{21}:	$\mathsf{P}_{\mathsf{RM|DW}} [\text{CUSTOMER}] \bullet \boldsymbol{cust_name}_{dw} \rightarrow \mathsf{RM} [\text{CUSTOMER}] \bullet \boldsymbol{cname}_{rm}$
ψ_{22}:	$\mathsf{P}_{\mathsf{S1|RM}} [\text{CUSTOMER}] \bullet \boldsymbol{cname}_{rm} \rightarrow \mathsf{S1} [\text{CUSTOMER}] \bullet \boldsymbol{cust_name}_{1}$

Using CA ψ_{21} as a start point, we can use the rule:

$$\textbf{\textit{RR-PCA :}} \frac{\mathbf{P}_{\mathbf{I}|\mathbf{D}} \left[\underline{\mathrm{C}^{\mathbf{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathbf{D}}} \rightarrow \underline{\mathbf{A}^{\mathbf{I}}} \Rightarrow \mathbf{P}_{\underline{\mathbf{S}}|\mathbf{D}} \left[\underline{\mathrm{C}^{\mathbf{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathbf{D}}} \rightarrow \underline{\mathbf{A}}^{S}}{\underline{\mathbf{A}^{\mathbf{I}}} \Rightarrow \underline{\mathbf{A}}^{S}} \quad . \tag{5.4}$$

to create new PCAs. Observe that ψ_{21} unifies with the CA pattern expression above left of the symbol $\Rightarrow$ of the ***RR-CA*** *in* (5.4)*, being $\underline{\mathbf{A}^{\mathbf{I}}} = \mathsf{RM} [\text{CUSTOMER}] \bullet \boldsymbol{cname}_{rm}$. $\underline{\mathbf{A}^{\mathbf{I}}}$ unifies with the*

component pattern expression above left of the symbol $\Rightarrow$ *of the* ***SR-C*** *in* (5.3)*, and* ψ_{22} *unifies with its condition. Thus,* $\underline{\mathbf{A}^S} = \mathbf{S1}\,[\text{CUSTOMER}] \bullet \boldsymbol{cust_name}_1$*, and the new CA generated is:*

$$\alpha_2 : \mathbf{P_{S1|DW}}\,[\text{CUSTOMER}] \bullet \boldsymbol{cust_name}_{dw} \rightarrow \mathbf{S1}\,[\text{CUSTOMER}] \bullet \boldsymbol{cust_name}_1 \quad .$$

The rules in (5.2), (5.3), and (5.4) are simplifications of, respectively, rules **RR-ECA1**, **SR-C1**, and **RR-PCA1**, all presented in Appendix A. In the case of rule (5.4), for example, there is a procedure *thereisNOTvRelation*(CA) in its condition that was not presented. This procedure has as input a CA that, in case of example 21, is ψ_{21}. It verifies if a view relation was created as result of the inference mechanism (in the example it verifies if a view relation was created when ψ_{19} was analysed). *thereisNOTvRelation*(CA) returns *true* when the view relation was not created, and false otherwise. This procedure is important in order for the inference mechanism to create new PCAs in accordance with what was previously generated (a class/relation or a view relation). Five other procedures can appear in the condition of a rule, which are explained later.

We define about 62 rules in our proposal, of which 39 are RR-CAs and 23 SR-Cs. For each type of ECA, PCA, SCA, and ACA, there is at least one RR-CA (above, in the left hand side of the symbol $\Rightarrow$, of the rule) that unifies with the CA. The rules use the transitivity principle and create new CAs in accordance with the L_{PS} syntax. The whole set of proposed rules can be found in Appendix A.

5.2 Inference

The inference mechanism will infer a new perspective schema in which the classes, relations, keys and foreign keys of the inferred perspective schema will be the same that appear in the *"require" declarations* of the perspective schema associated with the *destination*. Its CAs will be (semi-) automatically generated using a rule-based rewriting system. Its MF signatures will be obtained from CAs of the new perspective schema by using one of the procedures *findMissingMF* to find missing MF signatures (see Chapter 4, Procedures 1 and 2). A simplified pseudo-code is shown in Procedure 3, which shows the process for generating a new perspective schema by using the proposed inference mechanism.

Procedure 3 generateNewPerspective($\mathcal{D}$,$\mathcal{I}$,$\{P_D, P_I\}$, P_N)

```
1:  for each CA A^D → A^I in P_D.caList do
2:      infer_CAs(A^D → A^I,{A^D → A^Sj},{ViewRelations})
3:      add CAs A^D → A^Sj to P_N.caList
4:  end for
5:  findMissingMF(P_N.caList,P_N.mfList)
6:  for each E in classList/relationList/keyList do
7:      create a ``require'' declaration to P_N
8:      add it, appropriately, to P_N.classList/P_N.relationList/P_N.keyList
9:  end for
10: for each VR in {ViewRelations} do
11:     add VR to P_N.vrList
12: end for
```

In Procedure 3, **D** is the *destination*, **I** is the *intermediary*, $\mathbf{S}_j$ (with $1 \leq j \leq m$) are the *origin* schemata, $P_{\mathbf{D}}$ is the perspective schema from the *intermediary* to the *destination*; P_I is the perspective schema from the *origin* to the *intermediary*; and P_N is the inferred perspective schema from the (subset of) *origin* to *destination*. All elements of the perspective schemata are grouped in lists: *classList*, *relationList*, *keyList*, *caList*, *mfList* and *vrList*. The three initial lists hold "require" declarations of, respectively, classes, relations, and keys (including foreign keys). *caList* contains CA declarations, *mfList* has MF signatures, and *vrList* contains view relation declarations. Firstly, in Procedure 3, new CAs are inferred from CAs defined in $P_{\mathbf{D}}$ (lines 1 to 4). The CAs in $P_{\mathbf{D}}$.*caList* are firstly organised in accordance to the class/relation to which they belong, then by their types: ECAs, PCAs, SCAs, and ACAs (in this order). One by one the CAs in $P_{\mathbf{D}}$.*caList* are used by the procedure to rewrite CAs (**infer_CAs**) as a start point for creating new CAs for P_N. In some cases view relations can be generated, as well as their respective CAs, as a result of applying a rule for rewriting CAs, for the reasons pointed out in Chapter 4, Section 4.4. New MF signatures for P_N are created from the new CAs previously inferred, using the algorithm *findMissingMF*() (line 5). New "require" declarations are generated from the "require" declarations of $P_{\mathbf{D}}$ (lines 6 to 9), by adding all "require" declarations of $P_{\mathbf{D}}$ to P_N. Finally, the view relation declarations created by the *infer_CAs* are added to $P_{\mathbf{D}}$.*vrList* (lines 10 to 12); their CAs already having been inserted in $P_{\mathbf{D}}$.*caList* previously (lines 2 and 3).

A simplified pseudo-code description of the process for generating new CAs is shown in Procedure 4.

In Procedure 4, **D** is the *destination*, **I** the *intermediary*, and **S**, any of the *origin* schemata. The algorithm tries to find, for each CA of the general form $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$ (assigning the *destination*

Procedure 4 infer_CAs($A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$,*CAs*,*VRs*)

1: **repeat**
2: `find` $A^{D} \rightarrow A^{S}$ `applying the inference rule` R:
3: R: $\frac{A^{D} \rightarrow A^{I} \Rightarrow A^{D} \rightarrow A^{S}}{\texttt{conditions}}$
4: `add` $A^{D} \rightarrow A^{S}$ `to` CAs
5: **until** `all rules for rewriting CAs have been tested`

to the *intermediary*), one or more CAs $A^{\mathbf{D}} \rightarrow A^{\mathrm{S}}$ (assigning the *destination* to an *origin* schema) as a result of applying a rule for rewriting CAs to $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$. In some situations, as pointed out in Chapter 4, Section 4.4, is not possible to rewrite a new CA $A^{\mathbf{D}} \rightarrow A^{\mathrm{S}}$ directly from the CA $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$. When this is the case, the rule for rewriting CAs will create a new view relation (*VR*), as well as the CAs of this view relation that connect the view relation to the origin schemata ($A \rightarrow A^{\mathrm{S}}$), and CAs of the class/relation in $A^{\mathbf{D}}$ that connect this class/relation to the view relation ($A^{\mathbf{D}} \rightarrow A$). The view relations, as well as their CAs, are created using procedures which are part of the condition of some rules. In the inference mechanism, six procedures were developed in order to deal with these situations. These procedures are briefly presented in the following text:

1. *thereisNOTvRelation*(<<CA>>): verifies if a view relation was created when an ECA or a SCA was analysed. It has as input a PCA or an ACA $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$, and returns true when the view relation was created and false otherwise. It is used to guarantee that the rule used by the inference mechanism to assign new properties will infer the PCA or ACA in accordance to what was previously generated when an ECA or SCA was analysed.

2. *create_vRelationFromECA*(<<CA>>,VR): creates a view relation and an ECA that assigns the view relation to a class/relation of the origin. It has as input an ECA $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$ and returns the name of the view relation that will be used by the inference mechanism to generate the CA $A^{\mathbf{D}} \rightarrow A$, which relates a class/relation of the destination to the view relation.

3. *addProp_vRelationFromECA*(<<PCA>>,Exp): obtains the view relation that was created when the ECA assigned to the PCA was analysed; adds the property of the destination present in PCA to the view relation; and creates a new PCA $A \rightarrow A^{\mathrm{S}}$ (that assigns the property of the view relation to the property(ies) of the origin). It has as input a PCA $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$ and returns an expression that will be used by the inference mechanism to generate the PCA $A^{\mathbf{D}} \rightarrow A$.

4. *create_vRelationFromSCA*(<<CA1>>,<<CA2>>,VR,Exp): creates a view relation and a SCA that assigns the view relation to a class/relation of the origin. It differs from *create_vRelationFromECA*() because here the view relation will have an initial type formed by the properties of aggregations of the destination present in CA1, while the initial type in *create_vRelationFromECA*() is empty. It has as input two CAs: CA1, that is a SCA $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$, and CA2, that is an ECA $A^{\mathbf{I}} \rightarrow A^{\mathrm{S}}$; and returns the name of a view relation (VR) as well as an expression that will be used to generate the SCA $A^{\mathbf{D}} \rightarrow A$.

5. *addProp_vRelationFromSCA*(<<CA>>,<<VR>>,Exp): adds a property of the destination to the view relation VR given as input, and creates an ACA that assigns this property of the view relation to one or more property(ies) of a class/relation of the origin. It has as input, besides the view relation VR, an ACA $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$, and returns an expression that will be used by the inference mechanism to generate the ACA $A^{\mathbf{D}} \rightarrow A$.

6. *retrieve_vRelation*(<<CA>>,VR): retrieves a view relation that was created when a SCA was analysed. It is used together with the procedure *addProp_vRelationFromSCA*. It has as input the name of a SCA and returns the name of the view relation wanted.

The new CAs $A^{\mathbf{D}} \rightarrow A^{\mathrm{S}}$, or $A^{\mathbf{D}} \rightarrow A$, will be created only if the condition of the rule is valid. The condition of a rule is valid when all of its expressions are valid:

1. a CA pattern expression is valid if there is a CA which unifies with it;

2. a foreign key is valid if it is required in one of the perspective schemata associated with the *origin*;

3. an expression of the form $A \Rightarrow B$, such that A and B are component pattern expressions or A is a class/relation/view relation of a schema and B is the right-side of an ECA pattern expression, is valid if there is a rule which unifies with it and which is recursively applied;

4. an expression $\boldsymbol{p}{:}\natural\mathrm{C} \in \boldsymbol{type}(\mathrm{C}')$ is valid if $\boldsymbol{p}$ is required in one of the perspective schemata associated with the *origin*, such that $\boldsymbol{p}$ is defined in C' and $\boldsymbol{p}$ points to C;

5. a procedure is valid when it finishes and returns true, a name of a view relation, or an expression formed by properties of a view relation, being that the output depends on what procedure was used.

Consider, for instance, rules (5.2), (5.3), and (5.4) presented in Examples 20 and 21. The condition of rule **RR-CA** in (5.2) is valid if there is an ECA (declared in the perspective schema from some schema in the *origin* to the *intermediary*) that unifies with it. The condition of rule **RR-PCA** in (5.4) is valid if the expression $\mathbf{A^I} \Rightarrow \mathbf{A}^S$ is valid. It implies that a new rule should be called, for example **SR-C** when $\mathbf{A^I} = \mathbf{I}\left[\mathrm{C}^\mathbf{I}\right] \bullet \boldsymbol{p}^\mathbf{I}$. The condition of **SR-C** in (5.3) is valid if there is a PCA (declared in the perspective schema from some schema in the *origin* to the *intermediary*) that unifies with it.

A formal definition of a valid condition is as follows:

Definition 41 (*Valid condition – in an inference rule) Let $\hat{\mathcal{L}}_p$ be a set of perspective schemata, $\hat{\mathcal{A}}$ a set of correspondence assertions declared in some perspective schema belonging to $\hat{\mathcal{L}}_p$, $\hat{\mathcal{V}}$ a set of view relation declarations defined in some perspective schema belonging to $\hat{\mathcal{L}}_p$,* `A` *and* `B` *be expressions as defined for the parameters of a predicate in Definition 24, such that the properties in these expression are properties of some view relation belonging to $\hat{\mathcal{V}}$; and* **pred** *is a predicate as defined in Definition 24. Also let $\{X_1, X_2, ..., X_n\}$ be a condition Z in an inference rule. Z is a valid condition iff for each X_i, $1 \leq i \leq n$:*

- *X_i is a CA pattern expression, then there is a CA $\psi \in \hat{\mathcal{A}}$ that unifies with X_i;*
- *X_i is an expression of the form $A \Rightarrow B$, such that A and B are component pattern expressions or A is a class/relation/view relation of a schema and B is the right-side of an ECA pattern expression, then there is an inference rule that unifies with X_i whose condition is a valid condition;*
- *X_i is an expression of the form (**FK**,C,_,C′,_) then there is a "require" declaration require(**FK**) in some perspective schema belonging to $\hat{\mathcal{L}}_p$ such that the foreign key declaration unifies with X_i;*
- *X_i is an expression of form $\boldsymbol{p}$:♮C ∈ **type**(C′), then there is a "require" declaration require(C′,{..., $\boldsymbol{p}$, ...}) in some perspective schema belonging to $\hat{\mathcal{L}}_p$, such that $\boldsymbol{p}$ points to C;*
- *X_i is one of the following procedures:*
  - *thereisNOTvRelation(CA);*
  - *create_vRelationFromECA(CA,VR);*

– *addProp_vRelationFromECA(CA,Exp);*

– *create_vRelationFromSCA(CA_1, CA_2, VR, Exp);*

– *addProp_vRelationFromSCA(CA, VR, Exp);*

– *retrieve_vRelation(CA, VR).*

being that CA, CA_1, and CA_2 are CA pattern expressions for which there are CAs $\psi \in \hat{\mathcal{A}}$ that unify with them, VR is the name of a view relation belonging to $\hat{\mathcal{V}}$, and the procedures finish returning:

1. *true, if the procedure is thereisNOTvRelation;*
2. *the name of a view relation belonging to $\hat{\mathcal{V}}$, if the procedure is create_vRelationFromECA() or retrieve_vRelation();*
3. *an expression with one of the following forms:*
   – A
   – $(A_1, A_2, ..., A_n)$
   – $(B_1, \textbf{pred}_1)$; $(B_2, \textbf{pred}_2)$; ...; $\left(B_{n-1}, \textbf{pred}_{n-1}\right)$; B_n

   if the procedure is addProp_vRelationFromECA();
4. *the name of a view relation belonging to $\hat{\mathcal{V}}$, as well as an expression of form $\{A_1, A_2, ..., A_n\}$, if the procedure is create_vRelationFromECA(); and*
5. *an expression with one of the following forms, being ψ' a name of a SCA belonging to $\hat{\mathcal{A}}$:*
   – ψ', $\gamma\,(A)$
   – ψ', $\gamma\,(A, \textbf{pred})$
   – ψ', (A)
   – ψ', $(A_1, A_2, ..., A_n)$
   – ψ', $(B_1, \textbf{pred}_1)$; $(B_2, \textbf{pred}_2)$; ...; $\left(B_{n-1}, \textbf{pred}_{n-1}\right)$; B_n

   if the procedure is addProp_vRelationFromSCA. □

We need to give some explanation about the expressions that can be the output of the procedures called in some rules. The expressions that can be returned by the procedure *addProp_vRelationFromECA*() correspond to the right-side of a PCA (Definition 41, item 3). The expressions that can be returned by the procedure *create_vRelationFromSCA*() correspond

to the properties of aggregation (or expressions of aggregation) present in a SCA (Definition 41, item 4). The expressions that can be returned by the procedure *addProp_vRelationFromSCA*() correspond to the right-side of an ACA (Definition 41, item 5).

The rules are defined in an ordered way to reduce the search space. The choice of which rule to use is made by the Prolog unification mechanism. All rules, whose CA $A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$ unifies with the CA pattern expression above left of the rule, are used, being that each one can create a new CA. Notice that, within the condition of the rule, expressions of the form $A \Rightarrow B$ can exist. Only the Prolog backtracking mechanism is used.

In the following Section, we show some scenarios in which the inference mechanism can be used.

5.3 Uses of the Inference Mechanism

The simplest use of the inference mechanism is when there is only one information source, and only one perspective schema from the source to the reference model (as shown in Figure 5.2a). In this case, the inference mechanism generates a new perspective schema whose base contains only one schema. For instance, consider the schemata shown in Figures 4.1, 4.2, and 4.3. Suppose also that $\mathbf{S}_1$ is the unique information source for the DW. In this case, we can have the perspective schemata $\mathbf{P}_{S1|RM}$ and $\mathbf{P}_{RM|DW}$ and the inference mechanism creates the new perspective schema $\mathbf{P}_{S1|DW}$.

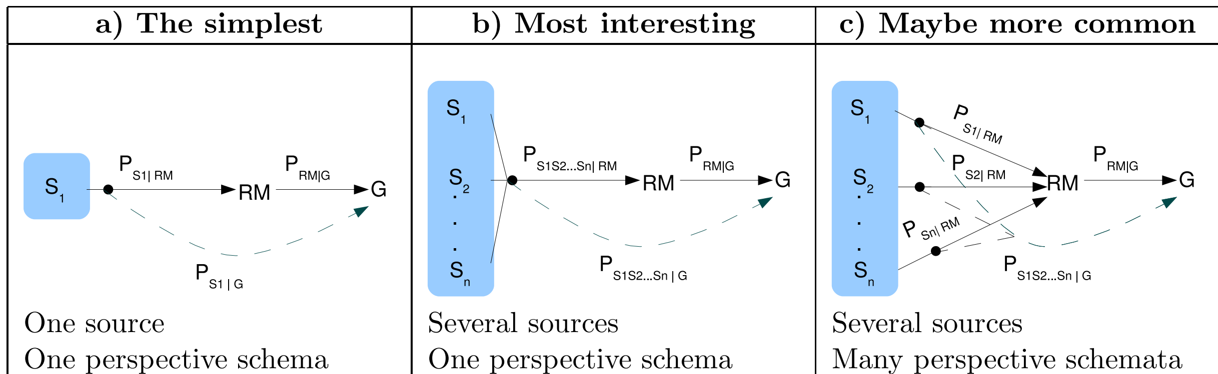


Figure 5.2: Examples of use of the inference mechanism.

A more interesting use of the inference mechanism is when there is more than one information source, which are mapped using a perspective schema of integration, as shown in Figure 5.2b). In this case, the inference mechanism creates a new perspective schema whose base can contain more

than one information source, being that this perspective schema is also a perspective schema of integration. Consider the schemata shown in Figures 4.1, 4.2, and 4.3 again. Now suppose that $\mathbf{S}_1$ and $\mathbf{S}_2$ are both the information sources for the DW and that there is a perspective schema of integration $\mathbf{P}_{S1,S2|RM}$. Using $\mathbf{P}_{S1,S2|RM}$ and $\mathbf{P}_{RM|DW}$ the inference mechanism creates the new perspective schema $\mathbf{P}_{S1,S2|DW}$ that is a perspective schema of integration.

Situations can also exist in which there is more than one information source, with each one having a distinct mapping (as shown in Figure 5.2c). In this case, the inference mechanism creates a new perspective schema, whose base can contain more than one information source, but, different to the previous case, the perspective schema here can be or not be a perspective of integration. It will be a perspective schema of integration if the information sources are disjointed (i.e, they are mapped to different parts of the reference model). When it is not the case, the inference mechanism can generate new CAs that relates the same component of the target to diverse components in different schemata of the base. These mappings do not represent an integration, since they were created from individual and not related mappings. For this type of perspective schema, the designer must decide what effective mapping will be used. An example is shown in the following text.

Example 22

Continuing Example 20 (page 108), consider the following ECA:

$$\psi_{23}\text{:}\ \mathbf{P_{S2|RM}}\,[\text{CUSTOMER}] \rightarrow \mathbf{S2}\,[\text{PURCHASER}]$$

Now, using CA ψ_{19} *as a start point, observe that, besides* ψ_{20}*,* ψ_{23} *also unifies with the condition of* ***RR-CA*** *in* (5.2)*. Thus, two new CAs are generated:*

$$\alpha_1 : \mathbf{P_{S1|G}}\,[\text{CUSTOMER}] \rightarrow \mathbf{S1}\,[\text{CUSTOMER}]$$
$$\alpha_2 : \mathbf{P_{S2|G}}\,[\text{CUSTOMER}] \rightarrow \mathbf{S2}\,[\text{PURCHASER}]\quad .$$[2]

This indicates that **G**.CUSTOMER *can have instances of* $\mathbf{S}_1$.CUSTOMER *or instances of* $\mathbf{S}_2$.PURCHASER*. Two interpretations are possible here: 1)* $\mathbf{S}_1$.CUSTOMER *and* $\mathbf{S}_2$.PURCHASER

[2] *Note that, the name of the inferred perspective schema must be* $\mathbf{P}_{s1,s2|G}$*. Here we have two names:* $\mathbf{P}_{s1|G}$ *and* $\mathbf{P}_{s2|G}$*. The correct name of the inferred perspective schema is only put in the CAs after all CAs of the perspective schema of the destination have been analysed, since it is a subset of the origin schemata (all those present in the new CAs).*

have the same set of instances, although it can be defined in a distinct way in each schema; or 2) **S**$_1$.CUSTOMER *and* **S**$_2$.PURCHASER *have different sets of instances and represent different contexts of the same domain. In the former both relations can be used indistinctly to form the set of instances of* **G**.CUSTOMER*, and the designer must indicate which mapping will be effectively used. In the latter, the designer must decide which context* **G**.CUSTOMER *will assume, or he/she made a mistake and it was necessary to do a data integration before he/she uses the inference mechanism.*

The scenario discussed in Example 22 is very useful in some types of DIS, such as FDBSs and mediation systems. In these scenarios, the designer can at any time choose the best source to get the information based on several criteria, such as network traffic, data confidentiality, etc.. Also, (s)he can use a source rather than another that is temporarily non-accessible. However, in most situations, namely in DW systems, it is necessary to do a data integration. The designer needs a perspective schema from the information sources to the global schema that represents a data integration of these sources. Using our proposal, the designer can obtain it in two ways:

1. If there is already a perspective schema of integration from information sources to the *intermediary* with all wanted information, then the designer just uses the inference mechanism to obtain, if it does not exist, the required perspective schema ; or

2. The designer defines an integrated perspective schema from information sources to the *intermediary*. Then, (s)he just uses the inference mechanism to obtain the required perspective schema.

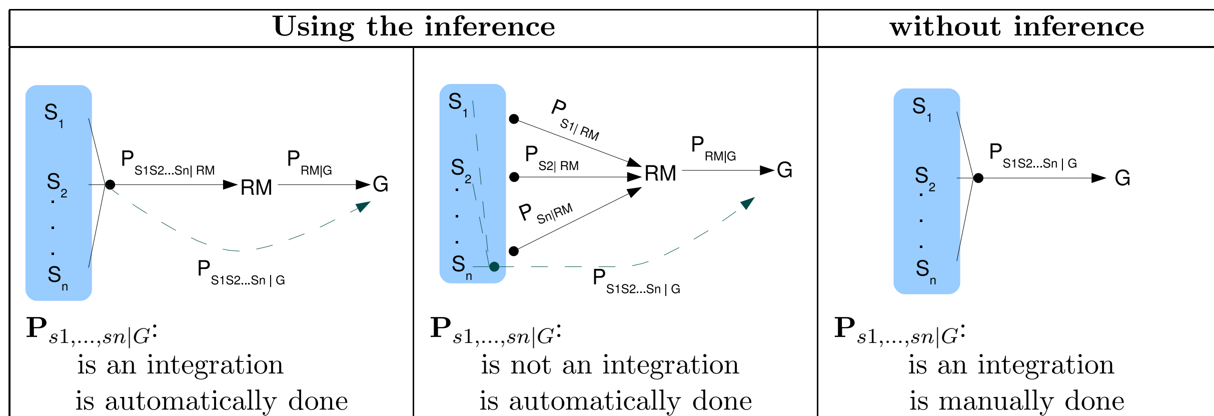


Figure 5.3: Examples of mapping strategies.

The designer, of course, would still prefer not to use the proposed framework (only to use the languages L_S and L_{PS}), and to define the direct mapping of the information sources to the global schema himself/herself (named the traditional way). A diagram with a resume of the mapping strategies is shown in Figure 5.3.

We still wanted to highlight that the inference mechanism is a one-step method. This means that it generates a new perspective schema relating to schemata, which are directly mapped to the reference model, to one schema target. There are situations in which, for example, one schema is mapped to another one, and only this latter one is mapped to the reference model, as shown in Figure 5.4.

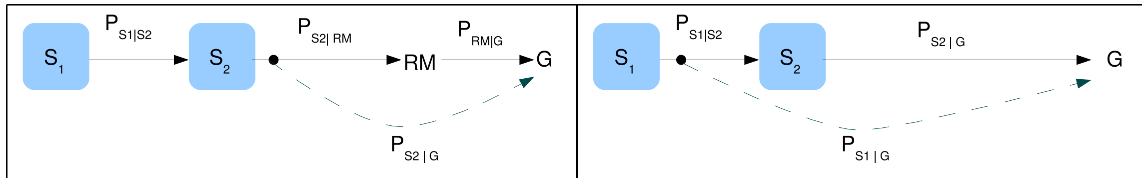


Figure 5.4: Example of inference in two-step.

In Figure 5.4, the schema $\mathbf{S}_1$ is mapped to the schema $\mathbf{S}_2$ that is mapped to the reference model that, finally, is mapped to the global schema. Using the reference model as the intermediary, the inference mechanism will generate the perspective schema $\mathbf{P}_{S2|G}$ (see the left-side of Figure 5.4). If we want to relate the global schema to a schema that is not derived from the other, then we should call for the inference mechanism again. At that time, $\mathbf{S}_2$ has the role of the intermediary, and the inference mechanism will create the perspective schema $\mathbf{P}_{S1|G}$ (see the right-side of Figure 5.4).

Finally, it is important to note that more than one DIS can exist in the same enterprise, which can share information sources, as shown in the left-side of Figure 5.5. In this case, the inference mechanism should be called for to generate a new perspective schema for each different target, possibly considering the same information sources and the same perspective schema of integration, as can be seen on the right-side of Figure 5.5.

Actually, the inference mechanism was implemented to allow as input, besides the perspective schema from the *intermediary* to the *destination* (e.g., $\mathbf{P}_{RM|G}$), various perspective schemata from only one information source to the *intermediary* (e.g., $\mathbf{P}_{S1|RM}$, $\mathbf{P}_{S2|RM}$, ..., $\mathbf{P}_{Sn|RM}$), and/or a perspective schema of integration from information sources to the *intermediary* (e.g., $\mathbf{P}_{S1,S2,...,Sn|RM}$). Nevertheless, it is not able to detect if the new perspective schema that was generated (e.g., $\mathbf{P}_{S1,S2,...,Sn|G}$) is or is not a perspective schema of integration.

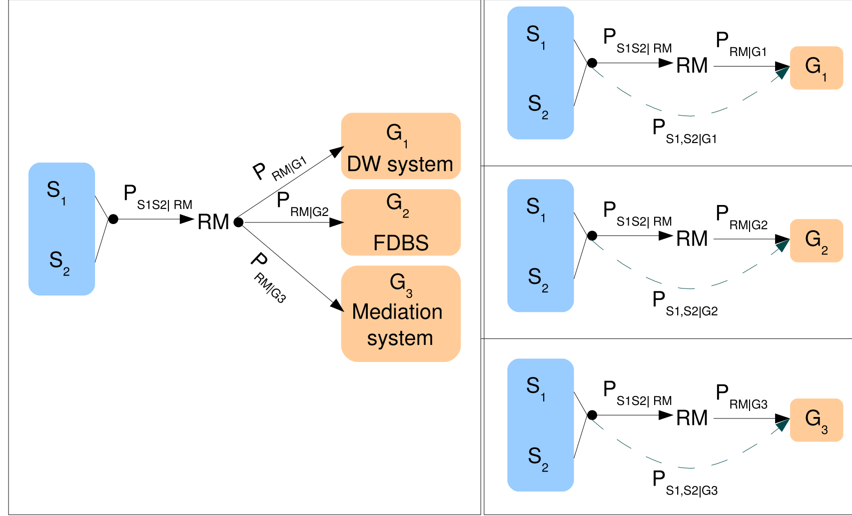


Figure 5.5: Inference mechanism used in various DISs.

It only indicates if there are perspective schemata and schemata that were not used in the inference, which is useful information enabling the designer to decide if (s)he should call for the inference mechanism again or not.

5.4 Inference Mechanism Validations

Up until now, we have presented the rules of the reference mechanism, on how the inference works, and on how the inference mechanism can be used. Here, we present some conditions that need to be satisfied before the inference mechanism is applied. It is important to note that the conditions do not guarantee that the new perspective schema is valid, because some structures can exist without the respective mapping (i.e., without CAs), even when the conditions are satisfied.

Given a set of schemata and the relationship between them (their associated perspective schemata), we can build a Directed Acyclic Graph (DAG) schema graph in the following way:

Definition 42 (*DAG schema Graph*) *Consider $\mathcal{L}$ to be a set of schema names and $\mathcal{L}_p$ a set of perspective schema names. A DAG schema Graph is a vertex-labelled DAG G_s, where:*

- $V(G_s) = \{\mathbf{S} \mid \mathbf{S} \in \mathcal{L}\}$;
- $E(G_s) = \{(\boldsymbol{v},\boldsymbol{w}) \mid \boldsymbol{v}, \boldsymbol{w} \in V(G_s), \mathbf{P}_{v|w}$ *or* $\mathbf{P}_{x1,x2,\ldots,v,\ldots|w} \in \mathcal{L}_p\}$; *and*

- *$L : V(G_s) \rightarrow \mathcal{L}$ is a vertex labelling function that defines the label $L(\boldsymbol{v}) \in \mathcal{L}$ for each vertex $\boldsymbol{v}$ in the graph.* □

Figure 5.6 shows some examples of DAG schema graphs.

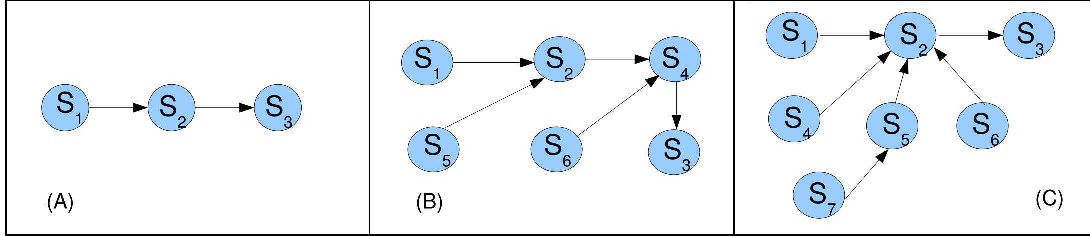


Figure 5.6: Examples of DAGs.

In Figure 5.6, the vertices of the graphs are schema names and the edges are perspective schema from one schema to another (the direction indicated by the arrow).

Note that, when the in-degree of a vertex[3] is more than 1, the meaning of these edges could be twofold: 1) there are different perspective schemata for the same target; or 2) there is a single perspective schema representing the integration of two or more schemata. For example, in graph (B) of Figure 5.6, the edges $\mathbf{S}_1\mathbf{S}_2$ and $\mathbf{S}_5\mathbf{S}_2$ can mean that there are two perspective schemata $\mathbf{P}_{\mathbf{S}1|\mathbf{S}2}$ and $\mathbf{P}_{\mathbf{S}5|\mathbf{S}2}$, or only a perspective schema of integration $\mathbf{P}_{\mathbf{S}1,\mathbf{S}5|\mathbf{S}2}$. Here the real meaning is not important.

A call for the inference mechanism, as mentioned at the beginning of this Chapter, contains three arguments and is formally defined as follows:

Definition 43 (*Call for the inference mechanism*) *Let $\hat{\mathcal{L}}$ be a set of schemata and $\hat{\mathcal{L}}_p$ a set of perspective schemata. A call for the inference mechanism is given by $\Im = (\mathcal{D},\mathcal{I},\hat{\delta})$, such that: $\mathcal{D}$ and $\mathcal{I} \in \hat{\mathcal{L}}$, with $\mathcal{D}$ being the destination and $\mathcal{I}$ the intermediary, and $\hat{\delta} = \{X \mid X \in \hat{\mathcal{L}}$ or $X \in \hat{\mathcal{L}}_p\}$.* □

Consider, for example, the call for the inference mechanism $\Im$ = (**DW**, **RM**, {$\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{P}_{RM|DW}$, $\mathbf{P}_{S1,S2|RM}$}). The DAG schema graph formed from schemata and perspective schemata in $\Im$ is shown on the left-side of Figure 5.7. The right-side of Figure 5.7 shows the DAG schema graph after using our inference mechanism in these schemata and perspective schemata. The green arrows represent the new perspective schema $\mathbf{P}_{S1,S2|DW}$ generated by the inference mechanism.

[3]The in-degree of vertex $\mathbf{v}$ in a graph G, $d_G^+(\mathbf{v})$, is the number of edges entering a vertex $\mathbf{v}$.

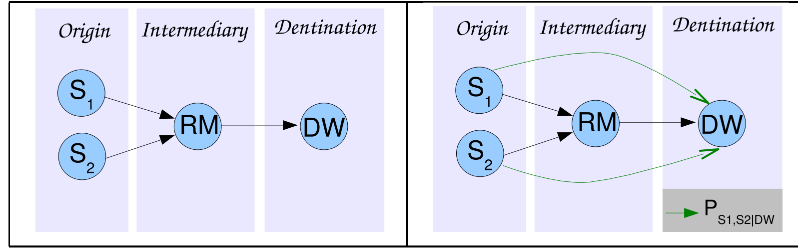


Figure 5.7: Example of an inference process.

The conditions required for the inference mechanism be applied can be grouped at three levels: 1) schema level, 2) class/relation level, and 3) property level. At a schema level, the inference mechanism can be run iff the following property is observed:

P1: There is at least one mapping from schemata in the *origin* to the *intermediary* and one mapping from the *intermediary* to the *destination*.

A call for the inference mechanism $\Im = (\mathcal{D}, \mathcal{I}, \hat{\delta})$ satisfies property **P1** iff $\mathcal{D}$, $\mathcal{I}$, and $\hat{\delta}$ form a DAG schema graph G_s such that:

- $\mathcal{D}$, named "destination vertex", has $d_G^+(\mathcal{D}) \geq 1$;
- $\mathcal{I}$, named "intermediary vertex", is a direct predecessor of $\mathcal{D}$ and has $d_G^+(\mathbf{v}) \geq 1$;
- ***origin***(G_s) is the non-empty set of all vertices $\mathbf{v} \in V(G_s)$, such that $\mathbf{v}$ are direct predecessors of $\mathcal{I}$.

If **P1** is not satisfied with a given call for the inference mechanism, it is not possible to carry out the inference process.

Consider, for instance, the Example shown in 5.7. The call $\Im$ satisfies property **P1**, since: 1) DW $= \mathcal{D}$ and $d_G^+(\mathcal{D}) = 1$; 2) RM $= \mathcal{I}$ and $d_G^+(\mathbf{v}) = 2$; and 3) ***origin***$(G_s) = \{$**S**1, **S**2$\}$ (a non-empty set).

Given a call for the inference mechanism and their correspondent DAG schema graph G_s, some vertices and edges in G_S can be unnecessary for the inference process. This occurs when:

1. There are successors to the destination vertex; or
2. There are predecessors to the destination vertex, other than the intermediary vertex; or
3. There are successors to the intermediary vertex, other than the destination vertex; or

4. There are predecessors to vertices belonging to $\boldsymbol{origin}(G_s)$.

Definition 44 (*subgraph of necessary conditions) Consider a call for the inference mechanism $\Im = (\mathcal{D}, \mathcal{I}, \hat{\delta})$ that satisfies property **P1** and its associated DAG schema graph G_s. A subgraph of necessary conditions is a subgraph G' of G_s such that $G' = G_s[V(G_s) \setminus \chi]$, with $\chi = \{\boldsymbol{v} \mid \boldsymbol{v}$ is a direct successor of $V(G_s) = \mathcal{D}$; or $\boldsymbol{v} \neq \mathcal{I}$ and $\boldsymbol{v}$ is a direct predecessor of $\mathcal{D}$; or $\boldsymbol{v} \neq \mathcal{D}$ and $\boldsymbol{v}$ is a direct successor of $\mathcal{I}$; or $\boldsymbol{v}$ is a direct predecessor of $\boldsymbol{origin}(G_s)\}$.* □

Consider the call for the inference mechanism $\Im_1 = (\mathbf{S}_3, \mathbf{S}_2, \{\mathbf{S}_1, \mathbf{P}_{S1|S2}, \mathbf{P}_{S2|S3}\})$. Its associated DAG schema graph G_s, shown in Figure 5.6(A), is equal to its subgraph of necessary conditions G' (i.e., χ is a empty set). Note that $d^+_{Gs}(\mathcal{D}) = 1$ (the intermediary vertex $\mathbf{S}_2$), $d^-_{Gs}(\mathcal{D}) = 0$,[4] $d^-_{Gs}(\mathcal{I}) = 1$ (the destination vertex), and $d^+_{Gs}(\mathbf{S}_1) = 0$, being $\mathbf{S}_1 \in \boldsymbol{origin}(G_s)$. Consider now the calls for the inference mechanism $\Im_2 = (\mathbf{S}_3, \mathbf{S}_4, \{\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_5, \mathbf{S}_6, \mathbf{P}_{S1,S5|S2}, \mathbf{P}_{S2|S4}, \mathbf{P}_{S4|S3}, \mathbf{P}_{S6|S4}\})$, and $\Im_3 = (\mathbf{S}_4, \mathbf{S}_2, \{\mathbf{S}_1, \mathbf{S}_3, \mathbf{S}_5, \mathbf{S}_6, \mathbf{P}_{S1,S5|S2}, \mathbf{P}_{S2|S4}, \mathbf{P}_{S4|S3}, \mathbf{P}_{S6|S4}\})$. Their associated DAG schema graph G_s is shown in Figure 5.6(B). Their associated subgraph of necessary conditions is shown, respectively, in the left-side and the right-side of Figure 5.8. Observe that, for $\Im_2$, $G' = G_s[V(G_s) \setminus \chi]$, with $\chi = \{\mathbf{S}_1, \mathbf{S}_5\}$, while that for $\Im_3$, $G' = G_s[V(G_s) \setminus \chi]$, with $\chi = \{\mathbf{S}_3, \mathbf{S}_6\}$.

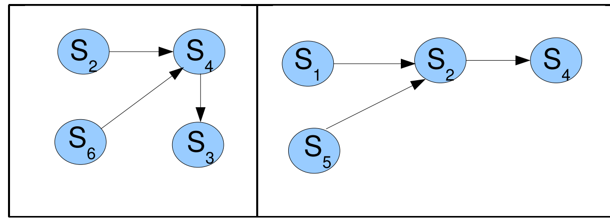


Figure 5.8: Examples of subgraph of necessary conditions.

Hereafter, for simplicity, we only consider calls for inference whose associate DAG schema graph is the same as its subgraph of necessary conditions.

We also must verify the connections at a class/relation level. Thus, given a set of schemata and the relationship between them (their associated perspective schemata), we can build a DAG, named *DAG entity graph*, considering only the relationship between classes, relations, and view relations (i.e., their ECAs and SCAs). This is done as follows:

[4] The out-degree of the vertex $\mathbf{v}$ in a graph G, $d^-_G(\mathbf{v})$, is the number of outward edges from a vertex $\mathbf{v}$.

Definition 45 (*DAG entity graph*) *Let G_s be a DAG schema graph. Also let $\mathcal{E}$ be a finite set of class/relation names of classes/relations declarations defined in schemata belonging to $V(G_s)$, and $\mathcal{V}$ be a finite set of view relation names of view relation declarations defined in the perspective schemata belonging to $E(G_s)$. A DAG entity graph is a vertex-labelled DAG G_e, where:*

- $V(G_e) = \{\mathrm{C}^x \mid \mathrm{C}^x \in \mathcal{E}$, *with x being the schema name where* C *is defined*$\}$,
- $E(G_e) = \{(\mathrm{C}^{\boldsymbol{v}}, \mathrm{C}^{\boldsymbol{w}}) \mid \mathrm{C}^{\boldsymbol{v}}, \mathrm{C}^{\boldsymbol{w}} \in V(G_e)$, *and:*
  - $A^{\boldsymbol{w}} \rightarrow A^{\boldsymbol{v}}$ *is an ECA or SCA defined in* $\mathbf{P}_{v|w}$ *(or* $\mathbf{P}_{x1,x2,\ldots,v,\ldots|w}$*) where* $\boldsymbol{vw} \in E(G_s)$ *with* $\mathrm{C}^{\boldsymbol{v}}$ *(respectively* $\mathrm{C}^{\boldsymbol{w}}$*) presenting in* $A^{\boldsymbol{v}}$ *(respectively* $A^{\boldsymbol{w}}$*); or*
  - *there is a reference link or a foreign key link*[5] $\ell : \mathrm{C}^v \multimap \mathrm{C}^w$*; or*
  - *there is a view relation* V*, whose name belonging to* $\mathcal{V}$*, and* $A^{\boldsymbol{w}} \rightarrow A$ *and* $A \rightarrow A^{\boldsymbol{v}}$ *are ECAs or SCAs defined in* $\mathbf{P}_{v|w}$ *(or* $\mathbf{P}_{x1,x2,\ldots,v,\ldots|w}$*) where* $\boldsymbol{vw} \in E(G_s)$ *with* V *is presenting in* A *and* $\mathrm{C}^{\boldsymbol{v}}$ *(respectively* $\mathrm{C}^{\boldsymbol{w}}$*) presenting in* $A^{\boldsymbol{v}}$ *(respectively* $A^{\boldsymbol{w}}$*)*$\}$,
- $L_e : V(G_e) \rightarrow \mathcal{E}$ *is a vertex labelling function that defines the label* $L_e(\boldsymbol{v}) \in \mathcal{E}$ *for each vertex* $\boldsymbol{v}$ *in the graph.* □

At a class/relation level, the inference mechanism can be run iff the following property is observed:

P2: (After **P1** has been satisfied). For each class/relation $\mathrm{C}^{\mathcal{D}}$ of the *destination*, there is a mapping from a class/relation $\mathrm{C}^{\mathcal{I}}$ of the *intermediary* to the $\mathrm{C}^{\mathcal{D}}$, being that there is also a mapping from a class/relation $\mathrm{C}^{\mathbf{s}}$ of a schema in the *origin* to the class/relation $\mathrm{C}^{\mathcal{I}}$.

A call for the inference mechanism $\Im = (\mathcal{D},\mathcal{I},\hat{\delta})$ satisfies property **P2** iff:

1. $\Im$ satisfies property **P1** (G_s being its DAG schema graph);
2. The classes/relations of schemata in $\mathcal{D}$, $\mathcal{I}$, and $\hat{\delta}$ form a DAG entity graph such that $\forall \mathrm{C}^{\mathcal{I}}\mathrm{C}^{\mathcal{D}} \in \mathrm{E(G}_e)$, there is at least one directed (graph) path ϑ that starts from $\mathrm{C}^{\mathbf{s}} \in V(G_e)$ and ends at $\mathrm{C}^{\mathcal{D}}$, such that $\mathrm{C}^{\mathcal{I}}\mathrm{C}^{\mathcal{D}} \in \vartheta$, and $\mathbf{s} \in \boldsymbol{origin}(\mathrm{G}_s)$.

Item 2 requires paths connecting classes/relations of the origin to classes/relations of the destination. Note that a class/relation of the destination can be linked to more than one

[5] Reference link and foreign key link: see Chapter 3, Section 3.1.3, Definition 20.

class/relation by a same ECA or SCA. Thus, we need to guarantee that there is a path for each pair (classes/relations of the intermediary, classes/relations of the destination), when there is a CA relating to each one. This enforces that all class/relation of the *destination* have a mapping.

If **P2** is not satisfied with a given call for the inference mechanism, it is not possible for the inference process to generate a valid perspective schema. Some examples are presented below:

Consider the call for the inference mechanism $\Im_4 = (\mathbf{S}_4, \mathbf{S}_3, \{\mathbf{S}_1,\ \mathbf{S}_2,\ \mathbf{P}_{S1,S2|S3},\ \mathbf{P}_{S3|S4}\})$. Figure 5.9 shows its associated DAG schema graph, and two examples of possible DAG entity graphs.

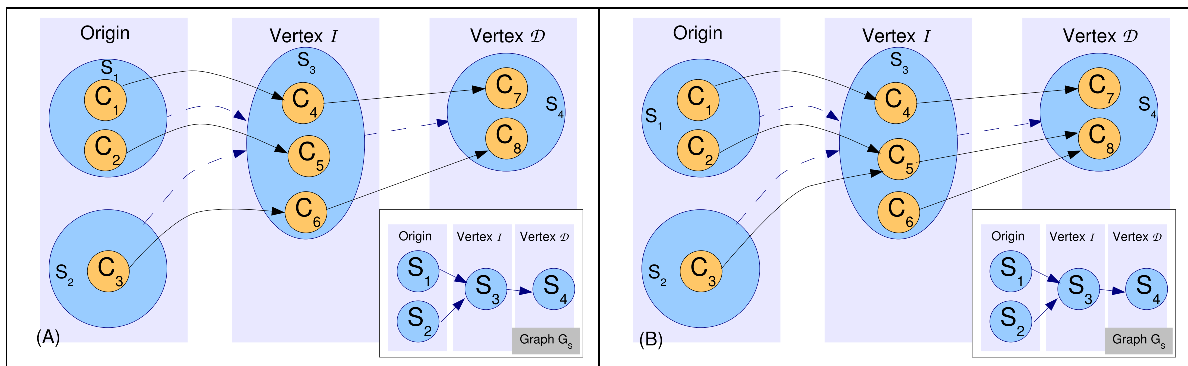


Figure 5.9: Example of a DAG schema graph with two different DAG entity graphs.

In Figure 5.9, the vertices of the DAG schema graph are blue circles and its edges are dotted arrows, while in the DAG entity graphs, the vertices are yellow circles and the edges are continuing arrows. Below on the right-side of Figure 5.9 (both A and B) a short diagram is shown containing only a DAG schema graph. The vertices of the DAG entity graphs are class/relation names and the edges indicate connections between them by mappings (ECAs or SCAs) or references (pointers to objects or referential constraints). Note that $\Im_4$ satisfies property **P1**, but it satisfies property **P2** only in graph (A). In graph (A) there is a path that starts from classes/relations declared in some schema belonging to origin(A) = $\{\mathbf{S}_1,\ \mathbf{S}_2\}$ and ends at classes/relations declared in the destination ($\mathbf{S}_4$), for each pair (classes/relations of $\mathbf{S}_3$, classes/relations of $\mathbf{S}_4$) connected by a CA. Note that, in graph (B) there is the pair (C_6, C_8) that does not have a path starting from the origin. This means that is not possible to infer a new CA of C_8 based on the CAs that generated the edges in Figure 5.9(B).

Once we verify the connections at a class/relation level, it is time to examine the connections at a property level. Thus, given a set of schemata and the relationship between them (their

associated perspective schemata), we can build a DAG, named *DAG structure graph*, considering only the relationship between properties (i.e., their PCAs, ACAs, and the aggregate properties in their SCAs). This is done as follows:

Definition 46 (*DAG structure graph) Let G_s be a DAG schema graph, and G_e be a DAG entity graph. Also let $\mathcal{P}$ be a set of property names, $\mathcal{K}$ a set of key (and foreign key) names, and $\mathcal{V}$ be a finite set of view relation names of view relation declarations defined in the perspective schemata belonging to $E(G_s)$. A DAG structure graph is a vertex-labelled DAG G_p such that:*

- $V(G_p) = \{\boldsymbol{p}^x \mid \boldsymbol{p}^x \in \boldsymbol{props}(\mathrm{C}^x)$ *or* $\boldsymbol{p}^x$ *is a foreign key name of* C^x*, with* $\mathrm{C}^x \in V(G_e)\}$,
- $E(G_p) = \{(\boldsymbol{p^v}, \boldsymbol{p^w}) \mid \boldsymbol{p^v}, \boldsymbol{p^w} \in V(G_p)$ *, and :*
  - $A^{\boldsymbol{w}} \rightarrow A^{\boldsymbol{v}}$ *is a SCA, PCA or ACA defined in* $\mathbf{P}_{v|w}$ *(or in* $\mathbf{P}_{x1,x2,...,v,...|w}$*) where* $\boldsymbol{vw} \in E(G_s)$ *with* $\boldsymbol{p^v}$ *(respectively* $\boldsymbol{p^w}$*) is presenting in* $A^{\boldsymbol{v}}$ *(respectively* $A^{\boldsymbol{w}}$*); or*
  - *there are reference links or foreign key links* $\boldsymbol{p^v}$*:* $\mathrm{C}^v \multimap \mathrm{C}_1^v$ *and* $\boldsymbol{p^w}$*:* $\mathrm{C}^w \multimap \mathrm{C}_1^w$ *as well as the undirected cycle in* G_e*:* $\mathrm{C}^v\mathrm{C}_1^v\mathrm{C}^w\mathrm{C}_1^w\mathrm{C}^v$*; or*
  - *there is a view relation* V*, whose name belonging to* $\mathcal{V}$*, and* $A^{\boldsymbol{w}} \rightarrow A$ *and* $A \rightarrow A^{\boldsymbol{v}}$ *are ECAs or SCAs defined in* $\mathbf{P}_{v|w}$ *(or* $\mathbf{P}_{x1,x2,...,v,...|w}$*) where* $\boldsymbol{vw} \in E(G_s)$ *with* $\mathbf{p} \in \boldsymbol{props}(\mathrm{V})$ *(or* $\mathbf{p}$ *is a path of* V*) is presenting in* A*, and* $\boldsymbol{p}^v$ *(respectively* $\boldsymbol{p}^w$*) belonging to* $\boldsymbol{props}(\mathrm{C}^{\boldsymbol{v}})$ *(respectively* $\boldsymbol{props}(\mathrm{C}^{\boldsymbol{w}})$*) presenting in* $A^{\boldsymbol{v}}$ *(respectively* $A^{\boldsymbol{w}}$*)*$\}$,
- L_p*:*$V(G_p) \rightarrow \{\mathcal{P}, \mathcal{K}\}$ *is a vertex labelling function that defines the label* $L_p(\boldsymbol{v}) \in \{\mathcal{P},\mathcal{K}\}$ *for each vertex* $\boldsymbol{v}$ *in the graph.* □

At a property level, the inference mechanism can be run iff the following property is observed:

P3: (After **P2** has been satisfied). For each property $\boldsymbol{p}^{\mathcal{D}}$ of classes/relations of the *destination*, there is a mapping from a property $\boldsymbol{p}^{\mathcal{I}}$ (or foreign key) of classes/relations of the *intermediary* to the $\boldsymbol{p}^{\mathcal{D}}$, being that there is also a mapping from a property $\boldsymbol{p}^{\mathbf{s}}$ (or foreign key) of classes/relations of a schema in the *origin* to the $\boldsymbol{p}^{\mathcal{I}}$.

A call for the inference mechanism $\Im = (\mathcal{D},\mathcal{I},\hat{\delta})$ satisfies property **P3** iff:

1. $\Im$ satisfies property **P2** (G_s being its DAG schema graph and G_e its DAG entity graph);
2. The properties and foreign keys of classes/relations in G_e form a DAG structure graph such that $\forall \boldsymbol{p}^{\mathcal{I}}\boldsymbol{p}^{\mathcal{D}} \in E(G_p)$, there is at least one directed (graph) path ϑ that starts from some $\boldsymbol{p}^{\mathbf{s}} \in \mathrm{V}(\mathrm{G}_p)$ and ends at $\boldsymbol{p}^{\mathcal{D}}$, such that $\boldsymbol{p}^{\mathcal{I}}\boldsymbol{p}^{\mathcal{D}} \in \vartheta$, and $\mathbf{s} \in \boldsymbol{origin}(\mathrm{G}_s)$.

Item 2 requires paths connecting properties/foreign keys of classes/relations of the origin to properties of classes/relations of the destination. Note that a same property of a class/relation of the destination can be linked to more than one property/foreign key of a class/relation by a same PCA, ACA, or SCA. Thus, we need to guarantee that there is a path for each pair (property/foreign key of classes/relations of the intermediary, property of classes/relations of the destination), when there is a CA relating each one. This enforces that all property of a class/relation of the *destination* have a mapping.

If **P3** is not satisfied with a given call for the inference mechanism, the inference process is unable to generate a valid perspective schema. Some examples are presented below:

Consider the call for the inference mechanism $\Im_5 = (\mathbf{S}_3, \mathbf{S}_2, \{\mathbf{S}_1, \mathbf{P}_{S1|S2}, \mathbf{P}_{S2|S3}\})$. Figure 5.10 shows its associated DAG schema graph, an example of a DAG entity graph, and two examples of possible DAG property graphs.

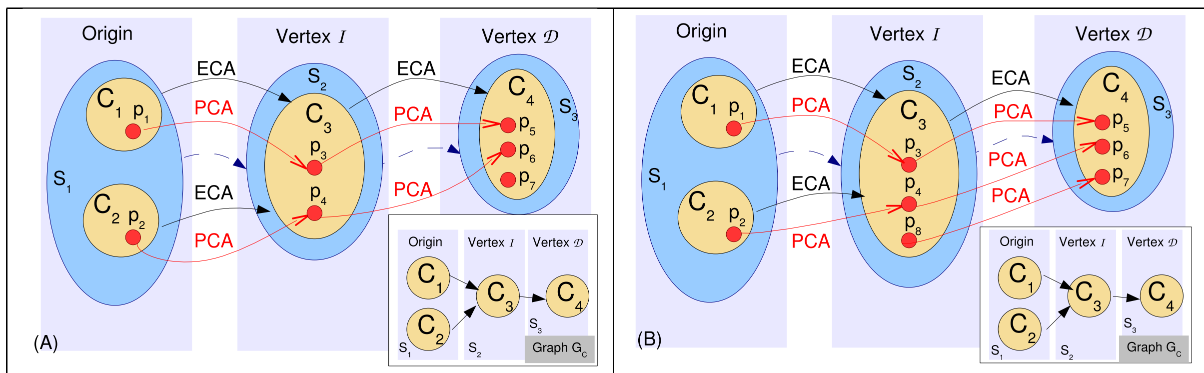


Figure 5.10: Examples of DAG structure graphs, from a same DAG schema graph, and DAG entity graphs.

In Figure 5.10, the notation for the DAG schema graph, and the DAG entity graph is the same as in Figure 5.9. Below on the right-side of Figure 5.10 (both A and B) a short diagram is shown containing only a DAG entity graph. The vertices of the DAG structure graphs, represented by red circles, are property names or foreign key names of classes/relations; and the edges, represented by red arrows, indicating connections between them by mappings (PCAs, ACAs, or SCAs) or references (pointers to objects or referential constraints). Note that, $\Im_5$ satisfies property **P2**, but property **P3** is only satisfied in graph (A). In graph (A) there is a path that starts from properties/foreign keys declared in some schema belonging to origin(A) = $\{\mathbf{S}_1\}$ and ends at properties declared in the destination ($\mathbf{S}_3$), for each pair (property/foreign key of $\mathbf{S}_2$, property of $\mathbf{S}_3$) that is connected by a CA. Observe that in graph (A) property $\boldsymbol{p}_7$ is connected to nothing. This means that it is a surrogate key and is not changed by the

inference mechanism. In graph (B) there is no path starting from the properties/foreign key of classes/relations declared in $\mathbf{S}_1$ to pair ($\boldsymbol{p}_8$, $\boldsymbol{p}_7$). This means that it is not possible to infer a new CA of C_4, referring to property $\boldsymbol{p}_7$ based on the CAs that generated the edges in Figure 5.10(B). In this case, the inference mechanism will generate a perspective schema, but it is not a valid perspective schema, because there is no mapping to property $\boldsymbol{p}_7$ (a warning is shown to the designer in order to alert him/her).

Now consider the call for the inference mechanism $\Im_6 = (\mathbf{S}_3,\mathbf{S}_2,\{\mathbf{S}_1, \mathbf{S}_4, \mathbf{P}_{S1,S4|S2}, \mathbf{P}_{S2|S3}\})$. Figure 5.11 shows its associated DAG schema graph, an example of a DAG entity graph, and two examples of possible DAG property graphs. Note that, $\Im_6$ satisfies property **P2**, but only in graph (B) does it satisfy property **P3**. In graph (B) there is a path that starts from properties/foreign keys declared in the origin ($\mathbf{S}_1$ and $\mathbf{S}_4$) and ends at properties declared in the destination ($\mathbf{S}_3$), for each pair (property/foreign key of $\mathbf{S}_2$, property of $\mathbf{S}_3$) that is connected by a CA. Observe that in graph (A), it is not possible to infer a new CA referring to property $\boldsymbol{p}_5$, since $\boldsymbol{p}_5$ is only linked to the foreign key $\mathbf{FK}_2$ (of schema $\mathbf{S}_2$), and this is linked to nothing.

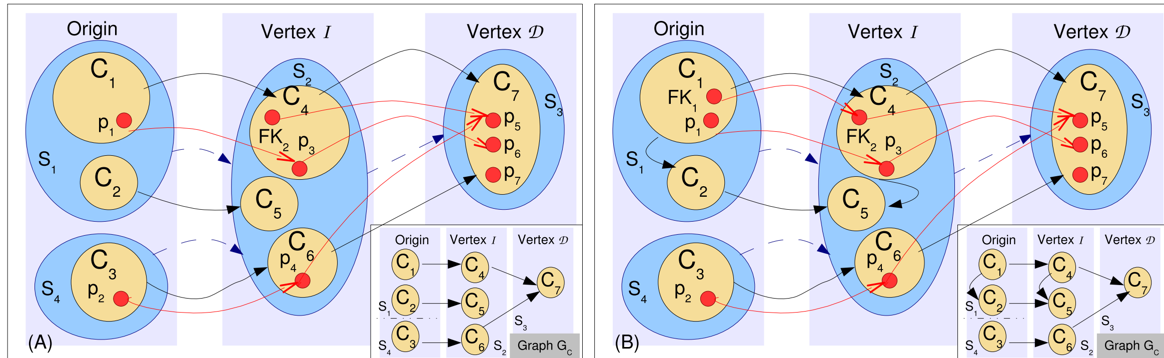


Figure 5.11: Other examples of DAG structure graphs (also are presenting DAG schema graphs, and DAG entity graphs).

Property **P3** determines the necessary conditions for the inference mechanism to create a valid perspective schema, but it is not enough to guarantee that a valid perspective schema is always generated. There are situations in which the inference mechanism cannot create new CAs, even when **P3** is satisfied. Table 5.1 shows some of these situations.

In Table 5.1, the first two columns present all possible types of CAs in perspective schemata from the *destination* to the *intermediary* and from the *intermediary* to the *origin*, respectively. The third column shows the inferred CA for each case, when it is possible create a new CA, or "–" when the inference mechanism can never generate a new CA. The fourth column indicates specific situations when it is not possible to generate new CAs, which are described as follows:

Table 5.1: Sketch of the inference of CAs.

$A^{\mathbf{D}} \rightarrow A^{\mathbf{I}}$	$A^{\mathbf{I}} \rightarrow A^{\underline{\mathbf{S}}}$	$A^{\mathbf{D}} \rightarrow A^{\underline{\mathbf{S}}}$	Unless
ECA	ECA	ECA	(I)
ECA	SCA	SCA	(I), (II)
SCA	ECA	SCA	(I), (III)
SCA	SCA	–	–
PCA	PCA	PCA	(I), (III)
PCA	ACA	ACA	(I), (III), (IV)
ACA	PCA	ACA	(I), (III), (IV)
ACA	ACA	–	–

(I) There is a predicate (a selection condition) in the *intermediary* that cannot be substituted by a predicate in the *origin* using some SR-C.

(II) There is some aggregation property (or path) in the *intermediary* that cannot be substituted by a property (or path) in the *origin* to form the new SCA, even with the help of view relations.

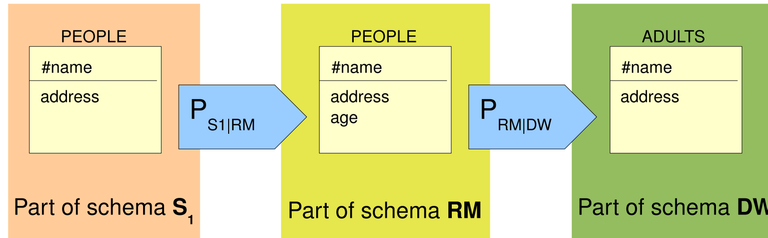
(III) There is some property (or path) in the *intermediary* that cannot be substituted by a property (or path) in the *origin.*

(IV) The SCA assigned to the ACA was not generated.

In order to get a better understanding, some examples are presented in the following text.

Example 23

This short example shows a situation in which a predicate cannot be substituted, and so the CA in which it appears cannot be inferred. Consider the following schemata, which store information about people:



Consider also the following CAs:

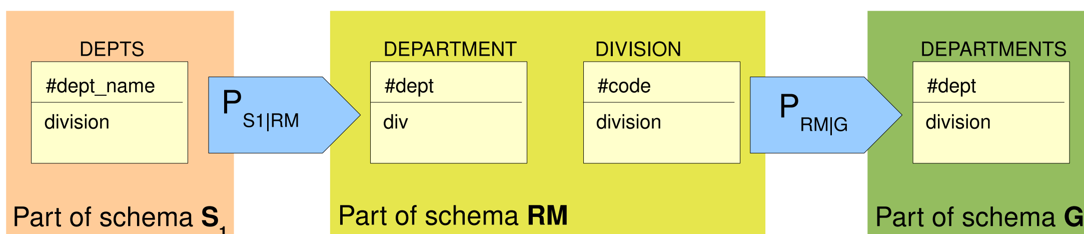
Extension Correspondence Assertion (ECA):
μ_1: $\mathbf{P}_{\mathbf{S}_1\|RM}$ [PEOPLE] $\rightarrow$ $\mathbf{S}_1$ [PEOPLE]
ω_1: $\mathbf{P}_{RM\|DW}$ [ADULTS] $\rightarrow$ **RM** [PEOPLE ($\boldsymbol{age} \geq 18$)]
Property Correspondence Assertion (PCA):
μ_2: $\mathbf{P}_{\mathbf{S}_1\|RM}$ [PEOPLE] • ***name*** $\rightarrow$ $\mathbf{S}_1$ [PEOPLE] • ***name***
μ_3: $\mathbf{P}_{\mathbf{S}_1\|RM}$ [PEOPLE] • ***address*** $\rightarrow$ $\mathbf{S}_1$ [PEOPLE] • ***address***
ω_2: $\mathbf{P}_{RM\|DW}$ [ADULTS] • ***name*** $\rightarrow$ **RM** [PEOPLE] • ***name***
ω_3: $\mathbf{P}_{RM\|DW}$ [ADULTS] • ***address*** $\rightarrow$ **RM** [PEOPLE] • ***address***

Let $\Im_7$= (**DW**,**RM**,{$\mathbf{S}_1$, $\mathbf{P}_{s1|RM}$, $\mathbf{P}_{RM|DW}$}) *be a call for the inference mechanism.* $\Im_7$ *satisfies* ***P3****, but the inference mechanism cannot infer a new CA, because there is not a CA mapping the property* ***age****.*

The following example presents a situation in which it is not possible to infer a new path.

Example 24

Consider the following schemata, which store information about units of an enterprise:



Consider also the following CAs:

Extension Correspondence Assertion (ECA):
μ_1: $\mathsf{P}_{\mathbf{S}_1\|RM}$ [DEPARTMENT] $\rightarrow$ $\mathbf{S}_1$ [DEPTS] ω_1: $\mathsf{P}_{RM\|G}$ [DEPARTMENTS] $\rightarrow$ **RM** [DEPARTMENT]
Property Correspondence Assertion (PCA):
μ_2: $\mathsf{P}_{\mathbf{S}_1\|RM}$ [DEPARTMENT] • ***dept*** $\rightarrow$ $\mathbf{S}_1$ [DEPTS] • ***detp_name*** μ_3: $\mathsf{P}_{\mathbf{S}_1\|RM}$ [DEPARTMENT] • ***div*** $\rightarrow$ $\mathbf{S}_1$ [DEPTS] • ***division*** ω_2: $\mathsf{P}_{RM\|G}$ [DEPARTMENTS] • ***dept*** $\rightarrow$ **RM** [DEPARTMENT] • ***dept*** ω_3: $\mathsf{P}_{RM\|G}$ [DEPARTMENTS] • ***division*** $\rightarrow$ **RM** [DEPARTMENT] • ***div*** • ***division***
Summation Correspondence Assertion (SCA):
μ_4: $\mathsf{P}_{\mathbf{S}_1\|RM}$ [DIVISION] (***division*** | ***code***) $\rightarrow$ *normalise* ($\mathbf{S}_1$ [DEPTS] (***division***))

Let $\Im_8$= (G,RM,$\{\mathbf{S}_1,\ \mathbf{P}_{s1|RM},\ \mathbf{P}_{RM|G}\}$) be a call for the inference mechanism, such that $\Im_8$ satisfies ***P3****. Note that RM.*DEPARTMENT*.****div*** *is a reference to RM.*DIVISION. *Note also that μ_3 identifies the relationship between RM.*DEPARTMENT*.****div****, (that is a surrogate key automatically generated in the normalisation process) and* $\mathbf{S}_1$.DEPTS*.****division****. Since the domain of* $\mathbf{S}_1$.DEPTS*.****division*** *is not a reference typed value and* $\mathbf{S}_1$.DEPTS*.****division*** *is not a foreign key, it is not possible to create a new path; and so a new PCA from ω_3 cannot be inferred.*

Our inference mechanism detects and informs the designer when a class/relation/property of the *destination* does not have a CA that refers to it because the inference mechanism could not infer a new one. Thus, (s)he can deal with the situation properly. In this case, the designer has two options:

1. To manually define in the inferred perspective schema the direct mapping between the destination and the origin schemata that could not be automatically created; or

2. To manually define additional mappings from the intermediary schema to the origin schemata in a way that the inference mechanism can generate the new missing CA.

The first option is not very convenient as it must be done every time that a call for the inference mechanism with the same parameters is made. However, sometimes it is the only solution. The second option is to avoid redefining the mappings when a recall for the inference mechanism is done.

5.5 Case Studies

We have developed two case studies in order to illustrate the practicality of our proposal in situations close to the real world. The first case study is a simple bank scenario extracted from (Dessloch *et al.*, 2008), while the second one is based on a real insurer. More details about each case study are presented in the following text. In this Chapter only a summary of the inference mechanism results will be shown, the whole description and analysis of the case studies can be seen in Annex 1 (Pequeno, 2010).

5.5.1 Bank scenario

The bank case study illustrates an ETL job that takes, as input, two relations: CUSTOMERS and ACCOUNTS; and gives, as output, two other relations: BIG_CUSTOMERS and OTHER_CUSTOMERS, depending on the total balance of each person's account: BIG_CUSTOMERS stores customers information when the total of its no-load accounts is more than $100,000, otherwise the customers are kept in OTHER_CUSTOMERS. Both relations store information about customer name, the total balance of its accounts, the age group to which he/she belongs, its country, etc..

Both relations CUSTOMERS and ACCOUNTS are defined in the same schema, which we named $\mathbf{S}_1$. The relations BIG_CUSTOMERS and OTHER_CUSTOMERS are defined in another schema, named here as **G**. There is not a reference model in (Dessloch *et al.*, 2008), but we created one to compare our approach with them. Part of the schemata $\mathbf{S}_1$ and **G** are presented in Figure 5.12. In order to clarify the reading, all the names of components of $\mathbf{S}_1$ are followed by number "1" and the names of components of **G** are followed by letter "g".

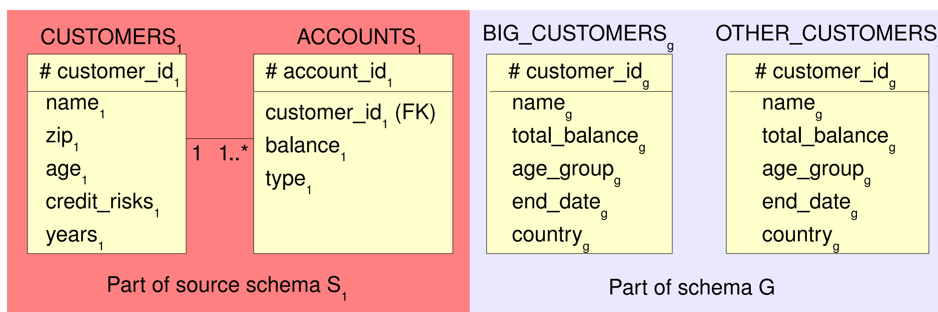


Figure 5.12: Part of the schemata present in the bank scenario.

Only no-load accounts must be analysed and some transformations must be done in CUSTOMERS$_1$ data before it is stored in BIG_CUSTOMERS$_g$ (or in OTHER_CUSTOMERS$_g$):

- age of the customers remains in groups: 1 if age $\leq$ 18, 2 if 18 $<$ age $\leq$ 35, 3 if 35 $<$ age $\leq$ 65, otherwise 4;
- zip-code is not required, but if there is a zip-code then the country value is "US", otherwise it is "unknown".

In order to deal with the transformations and aggregations required by the ETL job presented in (Dessloch *et al.*, 2008), we define a schema, named **V**, which stores data in the format required, and which is used as a source for the relations of **G**. Schema **V** contains two relations: CUSTOMERS$_v$ and BALANCE$_v$. CUSTOMERS$_v$ keeps information about customers, while BALANCE$_v$ keeps the summarisation of the account balance per customer. Note that, this solution presents two intermediary schemata (**RM** and **V**), instead of only one, as is originally proposed. Another solution would be to define CUSTOMER$_v$ and BALANCE$_v$ directly in the **RM**. However, we prefer the first approach because we think it is more in accordance with the role of a reference model, which should not have classes/relations due to ETL reasons.

The purpose of this case study is to show the advantages of using the proposed framework to model mappings and transformations that are normally hidden in ETL jobs. It also serves to illustrate how our proposal deals with non-trivial mapping involving data aggregation in which only part of the instances of a relation/class are used (i.e., aggregation with comparative operators).

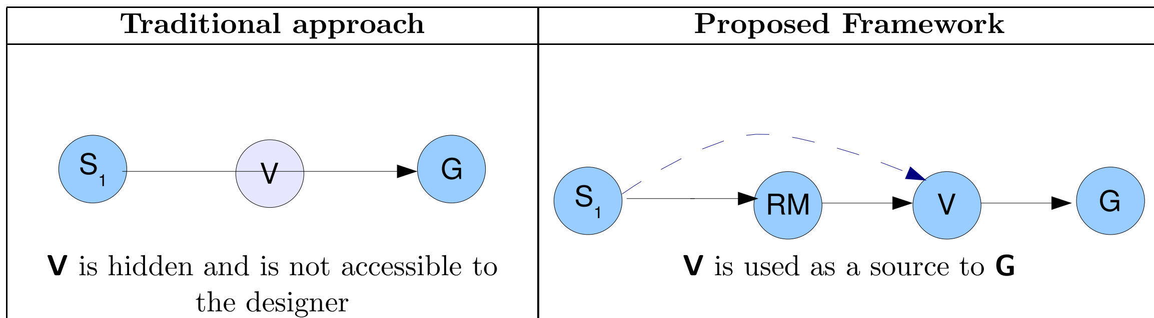


Figure 5.13: The role of schema **V** when using the traditional approach and when using the proposed framework.

We have defined $\mathbf{S}_1$, **G**, **V**, and the reference model **RM** in L$_S$. We also have defined the mappings between these schemata in L$_{PS}$, in accordance to our proposal: $\mathbf{P}_{s1|rm}$, $\mathbf{P}_{rm|v}$, $\mathbf{P}_{v|g}$, and $\mathbf{P}_{s1|v}$, being that the latter was automatically generated by the inference mechanism.

Figure 5.13 shows a graphical representation of the schemata and mappings used in a traditional approach and in ours. Manual mapping is represented by solid arrows, while automatic mapping is represented by a dashed arrow.

Figure 5.14 shows a graphical representation of the relationships between some components of schemata **G**, **V**, and $\mathbf{S}_1$. $\mathbf{customer_id}_1$ refers to property in ACCOUNTS$_1$, while $\mathbf{customer_id}_v$ refers to property in BALANCE$_v$. The expression $\mathbf{customer_id}_1 \bullet \mathbf{name}_1$, which is a simplification of a path expression, indicates that the relation involves a property in another relation, in this case CUSTOMERS$_1$, mutates mutandis for the other expressions in Figure 5.14.

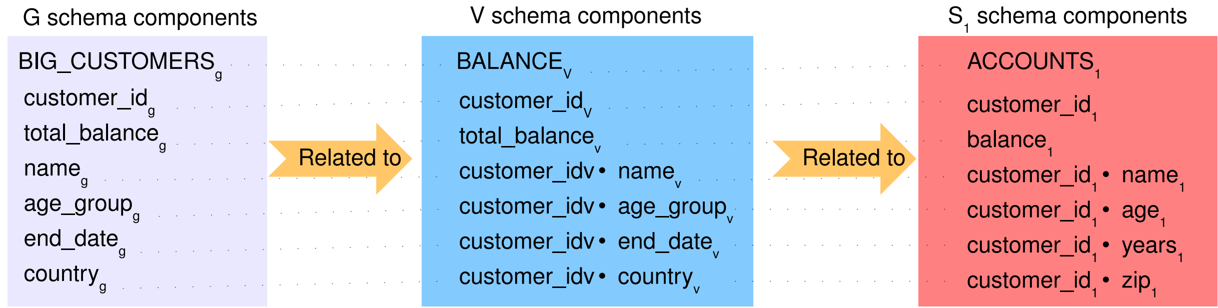


Figure 5.14: Relationships between some components of schemata **G**, **V**, and $\mathbf{S}_1$.

We used the inference mechanism to generate a new perspective schema ($\mathbf{P}_{s1|V}$) based on **V**, **RM**, $\mathbf{S}_1$, $\mathbf{P}_{s1|rm}$, and $\mathbf{P}_{rm|v}$. We were able to make very interesting observations on this part of the experiment. Specifically, we noticed the following non-trivial mappings, which were automatically inferred:

μ_4: $\mathbf{P}_{s1|v}$[CUSTOMERS$_v$]$\bullet\mathbf{age_group}_v \rightarrow$ (1, $\mathbf{S}_1$[CUSTOMERS$_1$]$\bullet$ $\mathbf{age}_1 \leq 18$); (2, $18 <$ $\mathbf{S}_1$[CUSTOMERS$_1$]$\bullet$ $\mathbf{age}_1 \leq 35$); (3, $35 <$ $\mathbf{S}_1$[CUSTOMERS$_1$]$\bullet$ $\mathbf{age}_1 \leq 65$); 4

μ_5: $\mathbf{P}_{s1|v}$[CUSTOMERS$_v$]$\bullet\mathbf{country}_v \rightarrow$ ("US", $\mathbf{S}_1$[CUSTOMERS$_1$]$\bullet$ $\mathbf{zip}_1 \neq$ null); "unknown"

μ_7: $\mathbf{P}_{s1|v}$[BALANCE$_v$]($\mathbf{customer_id}_v$) $\rightarrow$ groupby($\mathbf{S}_1$[ACCOUNTS$_1$ ($\mathbf{type}_1 \neq$ "L")] ($\mathbf{customer_id}_1$)) .

μ_4 and μ_5 are a case-based mapping (i.e., a property is assigned to one or more case conditions), and were generated using the rule RR-PCA5 and some substitution-rules. μ_7 is a mapping involving a set of instances, in which an aggregation and a selection condition are present. All these mappings, when using the traditional approaches, are hidden in ETL tools, and they are not available to the designer for his/her query and re-use.

5.5.2 Insurance scenario

The insurance case study is based on a real insurer, called Safeguard (a fictitious name). The business divides its insurances into two groups: life and non-life. The life insurance consists of health and injury. The non-life group consists of auto and property. There is a relational and distributed database for each type of insurance and another for keeping information about both people and organisations. The case study is based on the following three sources:

- Source 1: **Entities**

  This source contains data about people and organisations, which can be customers, suppliers, or agents (sellers).

- Source 2: **Health Insurance**

  This source contains data about health insurance.

- Source 3: **Auto Insurance**

  This source contains data about auto insurance.

The case study focuses on two applications related to information about policies: a user interface tool, named *salesTool*, and a Data Warehouse (DW) system. *salesTool* provides integrated access concerning sales in a given time, so agents can know about their commissions. In the DW system, Safeguard is mainly interested in statistical analysis. Safeguard uses a materialised and integrated schema, whose instances are derived from source 1, source 2 and source 3, as an information source for the DW system, named schema **V**. Safeguard has no reference model, but we will create one to compare our approach with theirs.

The main purpose of this case study is to show the advantages of using the proposed framework when the integrated data is used as a source for more than one system.

Traditionally, the designer should define the schemata $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, **S**, **V** and **W**. Also, (s)he must indicate, in some way, the several mappings: a) between the schema **S** and the information sources $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$; b) between **V**, and $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$; and c) between **V** and **W**. Using our framework, the designer must define in L_S the schemata $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, **S**, and **W**, as well as a reference model, here named **RM**, if it does not exist. Also, (s)he must define the perspective schemata: $\mathbf{P}_{s1,s2,s3|rm}$, $\mathbf{P}_{rm|s}$, and $\mathbf{P}_{rm|w}$. Here, the definition of schema **V** is not necessary since the perspective schema $\mathbf{P}_{s1,s2,s3|rm}$ represents the data integration that is required. The mapping between **S** and the information sources ($\mathbf{P}_{s1,s2,s3|s}$), as well as the mapping between

W and the information sources ($\mathbf{P}_{s1,s2,s3|w}$), were automatically created using the inference mechanism. Figures 5.15 and 5.16 show a graphical representation of the schemata used in the insurance case study, when using the traditional approach and our framework, with the manual mapping being represented by solid arrows, and the automatic mapping represented by dashed arrows.

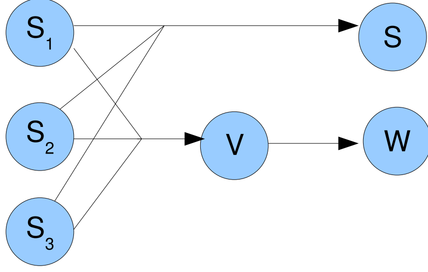


Figure 5.15: Mapping using traditional approaches.

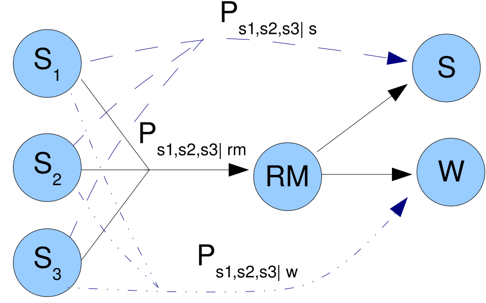


Figure 5.16: Mapping using the proposed framework.

Note that in the proposed framework, rather than defining various direct mappings, we define mappings using a intermediary schema and generate the required direct mappings. Also note that $\mathbf{P}_{s1,s2,s3|rm}$ was used twice: once to create $\mathbf{P}_{s1,s2,s3|s}$ and the other to generate $\mathbf{P}_{s1,s2,s3|w}$. It makes our proposal more robust, and easy to manage and maintain. Modifications occurring in $\mathbf{S}_1$, $\mathbf{S}_2$, or $\mathbf{S}_3$ should be reproduced only in $\mathbf{P}_{s1,s2,s3|rm}$, in contrast to the usual strategy, in which changes must be reflected in mappings: a) from $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$ to $\mathbf{S}$; and b) from $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$ to **V**.

We want to highlight some correspondence assertions that were used more than once in the process of inference in order to generate the new perspective schemata. Specifically, we noticed the following non-trivial mappings, which were automatically inferred:

μ_{11}: $\mathbf{P}_{s1,s2,s3|s}$[OPEN_RECEIPT$_s$] $\rightarrow$ $\mathbf{S}_2$[RECEIPT$_2$ (**pay_date**$_2$ = null)] ⟗ $\mathbf{S}_3$[RECEIPT$_3$ (**pay_date**$_3$ = null)]

μ_{18}: $\mathbf{P}_{s1,s2,s3|s}$[AGENT$_s$] • **agent_type**$_s$ $\rightarrow$ $\mathbf{S}_1$[AGENT$_1$]• **agent_fk**$_1$• **entity_fk**$_1$• **description**$_1$

μ_{41}: $\mathbf{P}_{s1,s2,s3|s}$[COMMISSION$_s$] • **commission_type**$_s$ $\rightarrow$ $\mathbf{S}_2$[COMMISSION$_2$] • **commission_type**$_2$

μ_{55}: $\mathbf{P}_{s1,s2,s3|w}$[AMOUNT_AGENT$_w$] (**agent_code_sk**$_w$, **date**$_w\langle$**date**$_w\rangle$) $\rightarrow$ groupby(**W**[AUX_COMMISSION$_w$] (**agent_code**, **pay_date**))

μ_{49}: $\mathbf{P}_{s1,s2,s3|w}$[AUX_COMMISSION] • **agent_code** $\rightarrow$ $\mathbf{S}_2$[COMMISSION$_2$] • **agent_code**$_2$

μ_{51}: $\mathbf{P}_{s1,s2,s3|w}$[AUX_COMMISSION] • **commission_amount** → $\mathbf{S}_2$[COMMISSION$_2$] • **commission_amount$_2$** .

μ_{11} is a mapping involving a selection of a set of instances and was generated using RR-ECA3 and some substitution-rules. μ_{18} and μ_{41} are mappings involving path expressions, in which the original paths had distinct sizes when compared to the new ones. The inferred CAs μ_{49}, μ_{51}, and μ_{55} were created from complex mappings involving a union and an aggregation:

ψ_{235}: $\mathbf{P}_{rm|w}$[AMOUNT_AGENT$_w$](**agent_code_sk$_w$**, **date$_w$**⟨**date$_w$**⟩) → groupby(**RM**[COMMISSION](**agent_code**, **pay_date**)

ψ_{153}: $\mathbf{P}_{s1,s2,s3|rm}$[COMMISSION] → $\mathbf{S}_2$[COMMISSION$_2$] ⊐⋈⊏ $\mathbf{S}_3$[COMMISSION$_3$]

The inference mechanism divided these complex mappings into simpler mappings: 1) it created a view relation (AUX_COMMISSION) in order to combine instances from the information sources, as well as to deal with transformations and denormalisations; 2) it used the view relation to do the aggregation. Note that each aggregation property generated a property in the view relation, which was properly mapped to the properties in the sources (e.g., see μ_{49}). The mapping between the view relation and the information sources was created using the procedure *create_vRelationFromSCA*, which was called for by rule RR-SCA3. μ_{51} involves simple structure mappings, was generated from any of the CAs:

ψ_{236}: $\mathbf{P}_{rm|w}$[AMOUNT_AGENT$_w$] • **max_commission$_w$** → ψ_{235}, *max*(**RM**[COMMISSION] • **commission_amount**)

ψ_{237}: $\mathbf{P}_{rm|w}$[AMOUNT_AGENT$_w$] • **total_commission$_w$** → ψ_{235}, *sum*(**RM**[COMMISSION] • **commission_amount**)

The focus here is that the inference mechanism considered the view relation AUX_COMMISSION created by other CA, instead of following the usual pattern.

There are few CAs (ψ_{223} and ψ_{224}) from which the inference mechanism could not infer new CAs:

ψ_{223}: $\mathbf{P}_{rm|s}$[OPEN_RECEIPT$_s$] • **seller_commission$_s$** → **RM**[RECEIPT] • **receipt_fk$_2$** • **commission_amount**

ψ_{224}: $\mathbf{P}_{rm|s}$[OPEN_RECEIPT$_s$] • **collector_commission**$_s$ → **RM**[RECEIPT] • **receipt_fk**$_3$ • **commission_amount**

This occurred because the property in the destination is mapped to a path in the intermediary, and it could not be substituted by another path in the origin. The only way to deal with this situation was to manually define the mapping between the property of the destination and the property in the origin. The new mappings are as follows:

μ_{27}: $\mathbf{P}_{s1,s2,s3|s}$[OPEN_RECEIPT$_s$] • **seller_commission**$_s$ → **S**$_2$[RECEIPT$_2$] • **seller_commission**$_2$

μ_{28}: $\mathbf{P}_{s1,s2,s3|s}$[OPEN_RECEIPT$_s$] • **seller_commission**$_s$ → **S**$_3$[RECEIPT$_3$] • **seller_commission**$_3$

μ_{29}: $\mathbf{P}_{s1,s2,s3|s}$[OPEN_RECEIPT$_s$] • **collector_commission**$_s$ → **S**$_2$[RECEIPT$_2$] • **collector_commission**$_2$

μ_{30}: $\mathbf{P}_{s1,s2,s3|s}$[OPEN_RECEIPT$_s$] • **collector_commission**$_s$ → **S**$_3$[RECEIPT$_3$] • **collector_commission**$_3$

5.6 Conclusions

We have presented a proposal to automatically connect a global schema to its sources by using an inference mechanism taking into account the explicit mappings between the sources and a reference model. The proposed approach makes clear the mappings that there are in a DIS, and uncouples them in order to make their maintenance easier. Besides, the relationship between the global schema and the source schemata is made explicitly and declaratively through correspondence assertions.

We have detailed the inference mechanism, its rules and operation. The feasibility of the proposed mechanism was tested via two case studies. We verified that it can deal with several types of mappings, including aggregations, selection conditions, and path expressions, which are very common in DW systems. Also, we identified the situations in which the inference mechanism cannot generate new mappings based on previous ones. Some examples were presented to clarify this subject, and the way to deal with it was suggested.

By using the proposed inference mechanism, the effort to describe the mappings between schemata is reduced, since mappings between the global schema and each source (and between sources themselves) is (semi-) automatically inferred. The inference mechanism also allows some changes in the actual source schemata, in the global schema, or in the mapping between schemata, which are common in the life cycle of any system, to be transparent to the DISs. The essence of this Chapter was published in (Pequeno & Pires, 2009a).

A proof-of-concept implementation, Prolog-based, was developed to allow for description of schemata and perspective schemata in the proposed language as well as to infer new perspective schemata based on others. This is the focus of the next Chapter.

6

The REMA Proof-of-Concept

The basis of this Chapter is to demonstrate the feasibility of our framework through a proof-of-concept implementation named REMA (REference Model-based Approach). It was developed using a standard Prolog programming language. It also uses a system, the Information Systems COnstruction language (ISCO) system, that provides a Logic-based programming layer and can generate applications with transparent access information sources in Relational Database Management Systems (RDBMSs).

We now briefly show the contextual logic programming and ISCO tool. We then deal with the issue of modelling and representation in data integration and in the ISCO framework. After that, we illustrate some details of implementation, as well as presenting some results of the case studies shown in the previous Chapter. The Chapter ends with some conclusions and future work planned on this topic.

6.1 Contextual Logic Programming and ISCO

Our choice for using a Prolog language as a base for our proof-of-concept was due to several factors, some of which are stated below:

- Declarative reading, which plays an important role in our work, as our formalism for describing the mappings between schemata can be achieved with some abstraction level on a declarative representation.
- The rapid prototyping ability and relative simplicity of program development.
- Natural choice due to our experienced usage with the Prolog language.

Logic Programming languages are akin to relational databases but provide a significantly higher expressive power, due to their two fundamental mechanisms of nondeterminism and unification, both of which form the basis of the Prolog language. However, it can be argued that standard Prolog is lacking in several areas, which include program structuring facilities and data persistence management. The ISCO programming system addresses both of these issues.

Program structuring is incorporated through the use of contexts (Abreu & Diaz, 2003). "Contextual Logic Programming (CxLP) is a simple yet powerful extension to the Prolog logic programming language, which provides a mechanism for modularity. In CxLP, a finite set of Horn clauses with a given name is designed by *unit*" (Lopes *et al.*, 2008). A unit is a parameterised named set of predicates, similar to a module, which can be dynamically combined to form an execution attribute called a *context*. Units represent a static definition block, while contexts can be thought of as a dynamic property of processing. "Goals are seen just as in regular Prolog, except for the fact that the matching predicates are to be located in all the units which make up the current context" (Abreu *et al.*, 2004). It is particularly relevant for our purposes to stress the *context-as-an-implicit-computation* aspect of CxLP, which views a context as a first-class Prolog entity – a term, behaving similarly to an object that holds a state (the unit argument terms) and responds to messages (goals evaluated in context). A more complete discussion about CxLP may be found in (Abreu & Diaz, 2003; Abreu & Nogueira, 2005).

A required feature to construct actual information systems using Logic Programming language is to provide persistence in this language. This could be provided by Prolog's internal database, but the ISCO authors believed that the best idea was to use a software program designed to handle large quantities of factual information efficiently, such as a relational database management system. "The semantic proximity between relational database query languages and logic programming makes the former a privileged candidate to provide Prolog with persistence" (Abreu & Nogueira, 2005).

ISCO (Abreu & Nogueira, 2006) is a proposal for Prolog persistence which includes support for multiple heterogeneous databases and which extends access to technologies other than relational databases, such as LDAP directory services or, more significantly, the semantic web in the form of SPARQL queries over OWL ontologies (Lopes *et al.*, 2008). "ISCO has been successfully used in a variety of real-world situations, ranging from the development of a university information system to text retrieval or business intelligence analysis tools" (Abreu & Nogueira, 2006).

"ISCO's approach for interfacing to DBMSs involves providing Prolog declarations for the database relations, which are equivalent to defining a corresponding predicate, which is then used as if it were originally defined as a set of Prolog facts. While this approach is convenient, its main weakness resides in its present inability to relate distinct database goals, effectively performing joins at the Prolog level. While this may be perceived as a performance-impairing feature, in practice it is not the show-stopper it would seem to be because the instantiations made by the early database goals turn out as restrictions on subsequent goals, thereby avoiding the filter-over-cartesian-product syndrome" (Abreu & Nogueira, 2006).

ISCO provides several useful features to our work, which are:

- high levels of performance, by virtue of being derived from GNU-prolog;
- expressiveness, due to the use of constraint logic programming;
- simplicity;
- persistence;
- structured code; and
- access to external and heterogeneous database in a uniform way.

6.2 REMA Architecture

The architecture comprises six cooperating modules, namely the *schema manager*, the *inference mechanism*, the *ISCO translator*, the *ISCO-generated applications*, the *schema repository*, and the *ISCO repository*. The REMA architecture is depicted in Figure 6.1 and its modules are briefly described as follows:

Schema manager:

The *schema manager* module was written using native-Prolog. It is used by the designer to manage the schemata (in language L_S) as well as the perspective schemata (in language L_{PS}). This module must be used to define all schemata of the information sources and the global schema, as well as the reference model, and the perspective schemata. It can also be used to carry out basic metadata queries. By using this module, the designer guarantees that the schemata and perspective schemata were defined in accordance to the proposed framework.

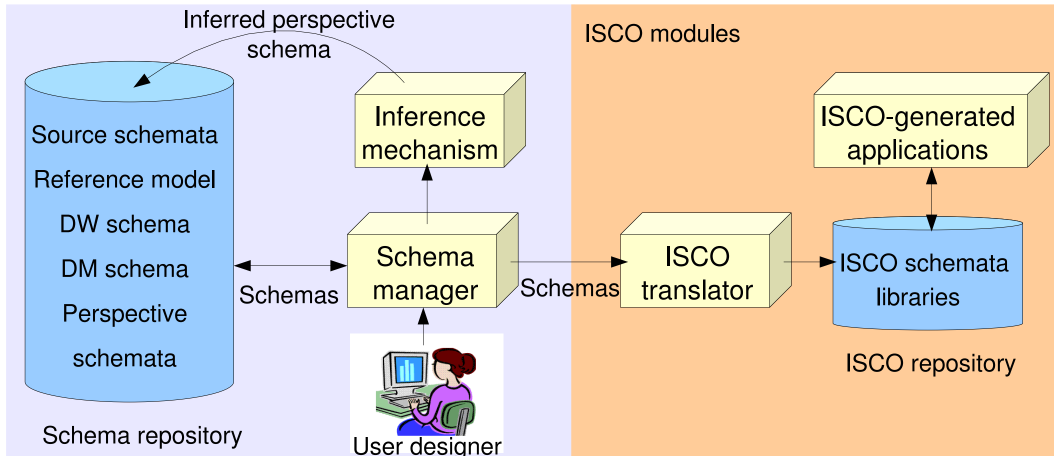


Figure 6.1: REMA architecture.

Inference mechanism:

The *inference mechanism* was written using native-Prolog. It is a rule-based rewriting system that (semi-) automatically generates new perspective schemata based on previous ones. It is formed by rules for rewriting CAs, and rules for rewriting components that are present in CAs, totalling of 62 rules. This module takes advantage of the Prolog unification and backtracking mechanisms.

ISCO translator:

The *ISCO translator* performs the mapping between schemata written in L_S or L_{PS} languages to ISCO schemata. Classes or relations of schemata written in language L_S are directly mapped to classes in ISCO, which are also called " classes" and which compile to regular Prolog access predicates. Classes, relations, and view relations of perspective schemata (written in language L_{PS}) are also mapped to ISCO classes, being that here their CAs and MF signatures also are used in the ISCO class definitions. *ISCO translator* has been written in native Prolog. All functions, whose names are used in some CAs of the original perspective schemata, must be defined manually in a library called *v_utils*, which is common to all ISCO schemata created. Note that, as the data of theses perspective schemata will be accessed using ISCO, all the MF signatures defined in the perspective schemata must have an implementation in *v_utils*.

ISCO-generated applications:

The *ISCO-generated applications* includes all files that are necessary to access data from information sources. So, data in any perspective schema mapped to ISCO, specifically any

inferred perspective schema between the global schema and its sources, can be queried in a transparent way, just as in mediation systems. This module is an integral part of the ISCO system, and it is only used here by us.

Repositories:

The *schema repository* stores both the schemata (in language L_S) and perspective schemata (in language L_{PS}), including any inferred perspective schema created by the inference mechanism. The *ISCO repository* is used to store ISCO schemata and ISCO files (libraries, units, etc.).

The next Section presents implementation details using the running example presented in Chapter 4 to describe the process of ISCO-generation applications in DISs.

6.3 Implementation

Schema manager:

In our proposal, the initial phase to model the ETL process starts by defining the schemata and perspective schemata in our language. Actually there is not a user-friendly interface to define schemata and perspective schemata, so any text editor (e.g., Emacs) can be used to do it. We define L_S (and L_{PS}) components as rules and facts Prolog. The designer must define the schemata and perspective schemata using this Prolog notation. The correspondence between L_S (and L_{PS}) components and the Prolog notation can be seen in Appendix B. Once the schema (or perspective schema) has been defined, it is necessary to guarantee that it is a valid schema (or a valid perspective schema). In order to do this, the designer should load the file program "schemaManager.pl" and ask one of the Prolog queries:

$$?-\ \mathit{checkSchema}(<<\mathit{Schema}>>).$$

$$?-\ \mathit{checkPerspectiveSchema}(<<\mathit{PerspectiveSchema}>>,<<\mathit{approach}>>).$$

checkSchema (and *checkPerspectiveSchema*) takes as input a schema (respectively a perspective schema) and gives as output a message indicating if it is valid or not. In the *checkPerspectiveSchema*, the designer should also indicate the type of approach that will be used in the perspective schema: "v", for a virtual view approach; "m", for a materialised view

approach, and "b" to consider both virtual and materialised view approaches. This information serves to generate the matching function signatures properly.

When the schema (or the perspective schema) is not valid, then a list of error messages is presented. Table 6.3 shows examples of possible error messages.

Table 6.1: Examples of error messages and when they are displayed.

error message	**displayed when...**
error05: invalid type. relation: ≪relation≫	there is not a property name or a type.
error06: key constraint. relation: ≪relation≫	a property name of a key is not defined in a class/relation, or a foreign key constraint occurred.
error12: undefined CA element. CA: ≪ca≫	some CA component was not defined in a schema, or in a class/relation.

In the context of the running example (see Figures 4.1, 4.2, and 4.3), the user can define, for example, the schema $\mathbf{S}_1$ in the file "s1.pl" and the perspective schema $\mathbf{P}_{S1|RM}$ in the file "p_s1_rm.pl". Algorithm 5 shows part of schema $\mathbf{S}_1$ in Prolog notation.

Algorithm 5 Schema $\mathbf{S}_1$ in Prolog notation.

```
1: schemanames(s1).
2: relationnames(s1,product).
3: relationnames(s1,item).
     ...
4: propertynames(cust_ids1).
5: propertynames(cust_addresss1).
     ...
6: keynames(s1,k1).
7: keynames(s1,k2).
     ...
8: relation(s1,product,struct(Components)):-
9: Components = [(prod_ids1,int),(prod_names1,text)].
     ...
10: key(s1,k1,product,[prod_ids1]).
11: key(s1,k2,item,[sale_ids1,prod_ids1]).
     ...
12: key(s1,fk1,item,[prod_ids1],product,[prod_ids1]).
     ...
```

After defining the schemata and perspective schemata, he/she should call *checkSchema*(*"s1.pl"*) and *checkPerspectiveSchema*(*"p_s1_rm.pl"*) in order to guarantee that they are valid schema and valid perspective schema, respectively.

Inference mechanism:

When the user wants to create a new perspective based on a previous one, he/she loads the file program "inference.pl", and asks the Prolog query:

?− *infer*(<<*Destination*>>,<<*Intermediary*>>,<<*SchemaList*>>).

infer takes as input a perspective schema of the *destination*, the name of the *intermediary* schema, and a set of origin schemata as well as their perspective schemata. It gives as output a new perspective schema, which is stored in a different file, as well as a text file containing a resume of which rules were used to create each new CA. This module verifies if the input obeys the necessary requisites presented in Chapter 5 (i.e., properties **P1**, **P2**, and **P3**); creates the "require" declarations of the new perspective schema based on the "require" declarations of the destination; creates new CAs that connect the origin directly to the destination; creates new MF signatures based on the new CAs; and, if something is wrong, presents a list of error messages. Errors can occur when is it not possible to establish a connection between some element (class, relation or property) of the destination and some element (class, relation or property) of the origin.

In the context of the running example, the user can call:

infer(*"p_rm_dw.pl"*, *"rm"*,[*"s1.pl"*, *"s2.pl"*, *"p_s1_rm.pl"*, *"p_s2_rm.pl"*]).

In this case, the output is the new perspective schema $\mathbf{P}_{S1,S2|DW}$, which is stored in file "p_s1s2_dw.pl".

ISCO translator:

Each class or relation in the (perspective) schemata is mapped to ISCO classes using the *ISCO translator*. The ISCO classes may reflect inheritance, keys, indexes, foreign keys and sequences to name a few. The process involved differs enough depending on whether the original schema was written in L_S language or if it is a perspective schema.

In the case of schemata defined in L_S language, a declaration is added to ISCO schema in order to provide ISCO with the necessary information to access an external data source, such as

an ODBC-accessed database. In the context of the running example, the ISCO schema mapped from schema $\mathbf{S}_1$ shall contain the clause:

$$\text{external}(\mathbf{S}_1, \text{postgres}(\mathbf{S}_1)).$$

This clause means that $\mathbf{S}_1$ is an outside database hosted in PostgreSQL.

All classes or relations in the original schema are simply mapped to ISCO classes, which should be declared as *external* and *mutable*. External means that the class has been created in an independent database, and mutable means that its instances can change. For instance, the relation $\mathbf{S}_1$.CUSTOMER is mapped to ISCO as illustrated in Figure 6.2.

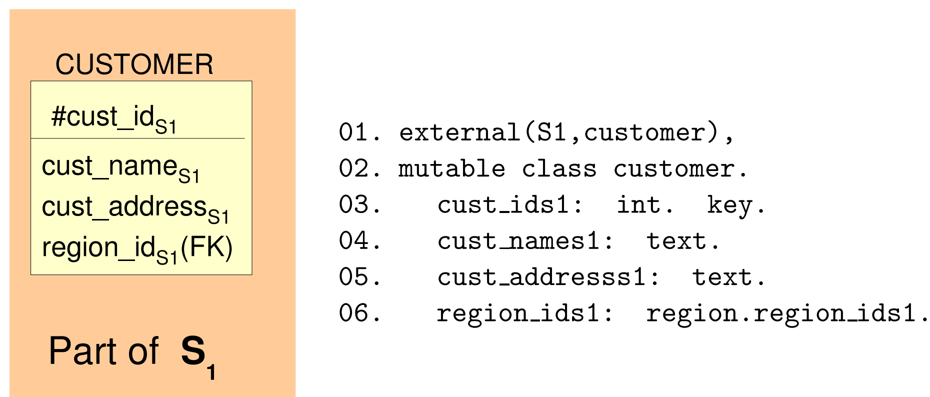


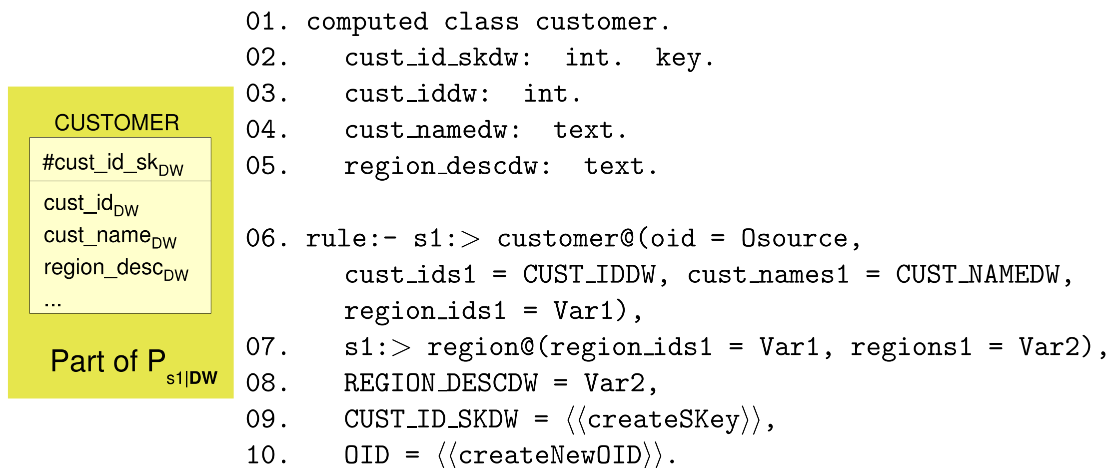
Figure 6.2: Example of a ISCO class generated from a relation of a schema.

In Figure 6.2, lines 1 and 2 mean that the instances of ISCO class CUSTOMER are in database $\mathbf{S}_1$, class CUSTOMER. Line 3 means that the property **cust_id**$_{S1}$ is an integer number and a primary key. Lines 4 and 5 mean that the properties **cust_name**$_{S1}$ and **cust_address**$_{S1}$ are of the type string. Line 6 means that the property **region_id**$_{S1}$ is a foreign key that refers to class REGION through property **region_id**$_{S1}$. Note that, in this example, the ISCO class and the class in database have the same name, but they could be different. The ISCO class CUSTOMER defines the predicate **customer/4**, which behaves as a database predicate but relies on an external system (in this case the PostgreSQL Object-Relational Database Management System (ORDBMS)) to provide the actual facts. In the current implementation it is assumed that all instances of ISCO classes have an OID, which is an integer number automatically generated by the system.

In the case of perspective schemata, the classes, relations, and view relations are usually mapped to *computed* classes. This means that the class instances will be generated each time

that a query is made to the class, similar to the concept of SQL-3 view. Computed classes are expected to contain one or more rules. These rules define how the computed class instances are obtained and always come after the computed class definition. In the body of each rule the variables, which represent the computed class arguments, must have the same name of the respective argument that they represent and they must all be in uppercase. This is necessary in order for Prolog to link each variable with its respective computed class argument correctly. View relations are always mapped to computed classes in ISCO, while classes and relations can be mapped to computed classes or to static classes in ISCO.

Classes or relations in the perspective schema are mapped to computed classes in ISCO when they are related to classes or relations through some ECA. An example is illustrated in Figure 6.3.




```
01. computed class customer.
02.    cust_id_skdw:  int.  key.
03.    cust_iddw:  int.
04.    cust_namedw:  text.
05.    region_descdw:  text.

06. rule:- s1:> customer@(oid = Osource,
       cust_ids1 = CUST_IDDW, cust_names1 = CUST_NAMEDW,
       region_ids1 = Var1),
07.    s1:> region@(region_ids1 = Var1, regions1 = Var2),
08.    REGION_DESCDW = Var2,
09.    CUST_ID_SKDW = ⟨⟨createSKey⟩⟩,
10.    OID = ⟨⟨createNewOID⟩⟩.
```

Figure 6.3: Example of a ISCO class generated from a relation of a perspective schema.

In Figure 6.3, lines 1 to 5 are the definition of the computed class CUSTOMER. It was defined based on the required declaration of relation CUSTOMER. Lines 6 to 10 are the rule of this computed class, which is based on the CAs of CUSTOMER. Specifically, line 6 defines a query to the ISCO class CUSTOMER in source schema $\mathbf{S}_1$. This query is obtained using the CAs $\psi_{19} - \psi_{22}$ (see Figure 6.4), being that the values of $\mathbf{cust_id}_{dw}$ and $\mathbf{cust_name}_{dw}$ are acquired directly from this query. The value of $\mathbf{region_desc}_{dw}$ requires an additional query to ISCO class REGION in schema $\mathbf{S}_1$ (lines 7 and 8). Lines 9 and 10 define the necessary steps to generate, respectively, surrogate keys and OIDs for the computed class.

Property Correspondence Assertions (PCAs)	
ψ_{19}:	$\mathbf{P}_{S1\|DW}$[CUSTOMER] • **cust_id**$_{\text{DW}}$ → $\mathbf{S}_1$ [CUSTOMER] • **cid**$_{S1}$
ψ_{20}:	$\mathbf{P}_{S1\|DW}$[CUSTOMER] • **cust_name**$_{\text{DW}}$ → $\mathbf{S}_1$ [CUSTOMER] • **name**$_{S1}$
ψ_{21}:	$\mathbf{P}_{S1\|DW}$[CUSTOMER] • **region_desc**$_{\text{DW}}$ → $\mathbf{S}_1$ [CUSTOMER] • FK_2 • region$_{S1}$
Extension Correspondence Assertion (ECA)	
ψ_{22}:	$\mathbf{P}_{S1\|DW}$[CUSTOMER] → $\mathbf{S}_1$ [CUSTOMER]

Figure 6.4: Examples of correspondence assertions.

The surrogate key is an integer number automatically generated based on the OID value in variable "Osource". The OID for the computed class is a compound OID with the following structure:

oid(schema,class,oids),

being that *class* is the name of the computed class to which the OID belongs, *schema* is the name of the schema to which *class* belongs, and *oids* is a list of compound OIDs of form ($\mathbf{S}'$,C',$\mathbf{o}'$), with C' being the name of a class or relation in a schema $\mathbf{S}'$ containing the OID **o'** from which the OID is derived. The OID $\mathbf{o}'$, in turn, can be a compound OID or a simple OID (an integer number). For example, OIDs for computed classes may look like:

oid($\mathbf{P}_{S1|DW}$,CUSTOMER,oid($\mathbf{S}_1$,CUSTOMER,667789))

which means the object in the computed class CUSTOMER is derived from the object in $\mathbf{S}_1$.CUSTOMER whose OID is 667789.

Classes/relations with aggregations in L_{PS} (i.e., that are connected to the base using a SCA) are mapped to a *static* access code in ISCO and SQL-3 view code. This occurs because ISCO does not support SQL's group-by and aggregation operators. When a class with aggregations is defined, ISCO simulates it by creating a static class in ISCO schema, whose instances cannot change, and a SQL-3 view in a database schema. The SQL-3 view does the processing and ISCO only takes the values. The ISCO class and the view SQL-3 must have the same name and structure in order for ISCO to do the correspondence with each other properly. An example is shown in Figures 6.5 and 6.6.

Figure 6.5 shows the static class *sales_by_customerdw*, which corresponds to the relation SALES_BY_CUSTOMER$_{dw}$ of the perspective schema $\mathbf{P}_{s1|dw}$. This relation is an aggregation of product sold per customer and date, and its relation to the source information $\mathbf{S}_1$ is defined by

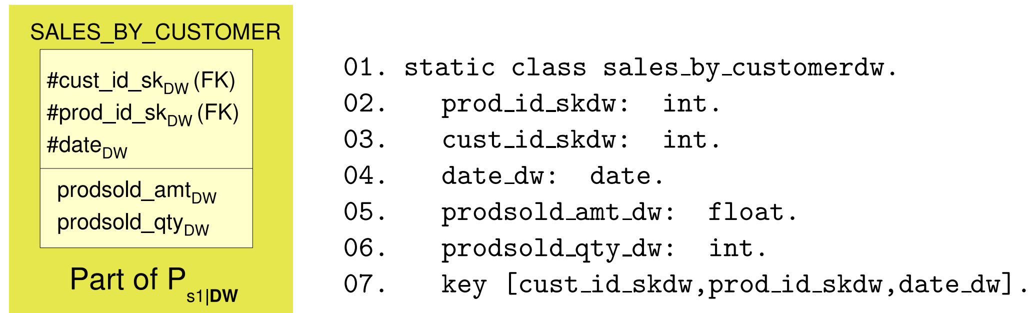


Figure 6.5: Another example of a ISCO class generated from a relation of a perspective schema.

the CAs presented in Figure 6.7. Line 7 of Figure 6.5 indicates the key of *sales_by_customerdw*, which is formed by the properties: $\mathbf{cust_id_sk}_{dw}$, $\mathbf{prod_id_sk}_{dw}$, and $\mathbf{date}_{dw}$.

```
01. CREATE VIEW sales_by_customerdw AS
02.   SELECT
03.     0 AS oid,
04.     MIN(customerdw.cust_id_skdw) AS cust_id_skdw,
05.     MIN(productdw.prod_id_skdw) AS prod_id_skdw,
06.     sale.sale_dates1 AS date_dw,
07.     SUM(item.qtys1) AS prodsold_qty_dw,
08.     SUM(item.qtys1*item.unitprices1) AS prodsold_amt_dw
09.   FROM item,sale,customerdw,productdw
10.   WHERE item.prod_ids1 = productdw.prod_id_dw AND
11.     item.sale_ids1 = sale.sale_ids1 AND
12.     sale.cust_ids1 = customerdw.cust_id_dw
13.   GROUP BY item.prod_ids1,sale.cust_ids1,sale.sale_dates1
```

Figure 6.6: Example of a SQL view for the static class *sales_by_customerdw*.

Figure 6.6 shows the SQL-3 view code corresponding to the static class *sales_by_customerdw*. This SQL-3 view was automatically created based on the CAs of $\text{SALES_BY_CUSTOMER}_{DW}$, presented in Figure 6.7. Line 3 assigns the value 0 (zero) to the OID, for simplicity of the presentation. Lines 4 to 8 assign values to the properties of the SQL-3 view. Note that *cust_id_skdw* and *prod_id_skdw* are surrogate keys and so their values do not exist in $\mathbf{S}_1$, but are obtained from other relations of the perspective schema (*customerdw* and *productdw*) using values of *prod_ids1* and *cust_ids1* (see CA ψ_{23}. Lines 7 and 8 were obtained from, respectively, CAs ψ_{24} and ψ_{25}. Lines 9 to 13 were created based on CA ψ_{23}). Lines 9 to 12 set the instances

Summation Correspondence Assertions (SCAs)
ψ_{23}: $\mathbf{P}_{S1\|DW}[\text{SALES_BY_CUSTOMER}_{DW}]\,(\mathbf{prod_id_sk}_{\text{DW}}, \mathbf{cust_id_sk}_{\text{DW}}, \mathbf{date}_{\text{DW}}) \rightarrow$ groupby$(\mathbf{S}_1\,[\text{ITEM}]\,(\mathbf{prod_id}_{S1}, \mathbf{FK4}_{S1} \bullet \mathbf{cust_id}_{S1}, \mathbf{FK4}_{S1} \bullet \mathbf{sale_date}_{S1})$
Aggregation Correspondence Assertion (ACA)
ψ_{24}: $\mathbf{P}_{S1\|DW}[\text{SALES_BY_CUSTOMER}_{DW}] \bullet \mathbf{prodsold_qty}_{DW} \rightarrow \psi_{23}, \text{sum}\,(\mathbf{S}_1\,[\text{ITEM}] \bullet \mathbf{qty}_{S1})$ ψ_{25}: $\mathbf{P}_{S1\|DW}[\text{SALES_BY_CUSTOMER}_{DW}] \bullet \mathbf{prodsold_amt}_{DW} \rightarrow \psi_{23},$ sum (times $(\mathbf{S}_1\,[\text{ITEM}] \bullet \mathbf{qty}_{S1}, \mathbf{S}_1\,[\text{ITEM}] \bullet \mathbf{unitprice}_{S1}))$

Figure 6.7: Examples of correspondence assertions of relation SALES_BY_CUSTOMER$_{dw}$.

of the relations that are used to get the values of the SQL-3 view, and line 13 defines the group by clause.

In our examples, for reasons of simplicity, we did not show translations to ISCO in which matching functions are present. However, as data should be accessed and integrated from information sources, the matching function signatures suggested in Definition 31 must be implemented. In order to make the implementation easier, we introduce a new variant of **match** in L_{PS}:

$$\mathbf{match} : \big(\big(\mathbf{S}_1\,[\text{R}_1]\,, \tau_1, \{\boldsymbol{p'}_1 : \tau'_1, ..., \boldsymbol{p'}_n : \tau'_n\}\big) \times \big(\mathbf{S}_2\,[\text{R}_2]\,, \{\tau_2\}, \{\boldsymbol{p''_1} : \tau''_1, ..., \boldsymbol{p''_n} : \tau''_n\}\big)\big) \rightarrow \rightarrow \text{Boolean}, \quad (6.1)$$

being that $\boldsymbol{p'}_i{:}\tau'_i \in \boldsymbol{type}(\text{R}_1)$; and $\boldsymbol{p''_i}{:}\tau''_i \in \boldsymbol{type}(\text{R}_2)$, $1 \leq \text{i} \leq \text{n}$.[1] This variant of the matching function signature indicates that the matching is done by a simple attribute comparison (i.e., each property $\boldsymbol{p'}_i$ of R_1 will be compared with the property $\boldsymbol{p''_i}$ of R_2, for $1 \leq \text{i} \leq \text{n}$). The matching functions can be implemented using Prolog itself or external functions. In the context of this proof-of-concept, only Prolog implementation was used.

ISCO-generated applications:

Once having the ISCO schemata, the following phase is to generate a GNU Prolog/CX executable containing the native-code executable version of all ISCO predicates. GNU Prolog/CX compiles Prolog (and ISCO) programs to native executables. Each schema and perspective schema described in ISCO, as well the library *v_utils*, will correspond to units whose

[1] ***type***(): see Chapter 3, Definition 4).

terms can be instantiated and collected into a list to form a context. A set of operations and operators are available in GNU Prolog/CX to construct contexts, and the *context extension* operation given by the operator `:>` is the most usual in our application. The goal **U** `:>` **G** extends the current context with the unit **U** and resolves **G** in the new context, as if it were regular Prolog. For instance, to make a query to the computed class CUSTOMER, we can use the following syntax:

```
v_utils :> p_s1_dw :> customer(A,B,C,D).
```

For this goal, we start by extending the initially empty context with unit v_utils. Afterwards, this new context is again extended with the unit $\mathbf{P}_{S1|DW}$, and it is in the latter context that goal `customer(A,B,C,D)` is derived.

6.4 Case Studies

In Chapter 5, we introduced two case studies. The schema manager module was used to define all schemata and perspective schemata presented there, and the inference mechanism module generated the new perspective schemata. Some results of the inference mechanism were shown and analysed. In the current Section we will continue showing the case studies, but here we focus on ISCO translation and ISCO application. Note that, because we developed only a proof-of-concept, we can only prove that our proposal works in scenarios close to the real-world. We did not do performance tests or compare our implementation with others.

6.4.1 Bank scenario

In this case study, only $\mathbf{S}_1$ is physically stored in a database, as **V** and **G** are derived from, respectively $\mathbf{S}_1$ and **V**. We suppose $\mathbf{S}_1$ is defined in PostgreSQL ORDBMS, named **S1_bank**. **V** and **G** are accessed through, respectively, the perspective schemata $\mathbf{P}_{s1|v}$ and $\mathbf{P}_{v|g}$. Although **V** and **G** are virtual, there is not a distinction between the form or the performance as data are accessed in $\mathbf{S}_1$, $\mathbf{P}_{s1|v}$ or $\mathbf{P}_{v|g}$.

The aim of this part of the experiment was simply to confirm the data access of a virtual source from another virtual source (in this case **G** is derived from **V** and this is derived from $\mathbf{S}_1$). Due to this, we only posted 6 registers in CUSTOMERS_1 and 10 registers in ACCOUNT_1.

Only $\mathbf{S}_1$, $\mathbf{P}_{s1|v}$ and $\mathbf{P}_{v|g}$ were necessary to be mapped to ISCO schemata (using the ISCO translator module). When $\mathbf{P}_{s1|v}$ was translated, a SQL-3 code was also created due to relation BALANCE$_v$, which involves aggregation functions. We also created the library *v_utils* in order to correctly define the new values of the OIDs.

We created the application *bank* using the ISCO-generated applications module. The following queries can be used to access data from, respectively, $\mathbf{S}_1$.CUSTOMERS$_1$, **V**.CUSTOMERS$_v$, and **G**.BIG_CUSTOMERS$_g$:

```
?- s1_bank :> customers1(A,B,C,D,E,F).
?- v_utils :> p_s1_v :> customersv(A,B,C,D,E).
?- v_utils :> p_v_g :> big_customersg(A,B,C,D,E,F).
```

A,B,C,D,E, and F are variables that will be unified to properties of the ISCO classes *customers1*, *customersv*, and *big_customersg*. As *customers1* is defined in schema $\mathbf{S}_1$, we extend the initial empty context with unit *s1* (remember that in ISCO each schema and perspective schema is a unit). In the case of the perspective schemata, the initial context must be the unit *v_utils*, since it contains Prolog predicates that will be used to process the data from the perspective schema.

6.4.2 Insurance scenario

In this case study we have two scenarios: a user interface tool (which provides an integrated access concerning sales in a given time), named salesTool, and a data warehouse system (which provides statistic analysis). Both scenarios have the same information sources $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$, which are physically stored in databases. The schemata **S** and **W**, since derived from $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$, are virtual, and are accessed through, respectively, the perspective schemata $\mathbf{P}_{s1,s2,s2|s}$ and $\mathbf{P}_{s1,s2,s2|w}$.

The aim of this part of the experiment was to confirm the integrated data access with or without the presence of statistical data.

Due to problems in how PostgreSQL accesses distinct schemata, and how ISCO deals with it, we had to assume that $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$ are in the same database, named *bd_insurer*. Note that the schemata are represented as separate units in ISCO, and so, they are logically separated.

Only $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, and $\mathbf{P}_{s1,s2,s2|s}$ were necessary to be mapped to ISCO schemata (using the ISCO translator module). The mapping of $\mathbf{P}_{s1,s2,s3|s}$ to ISCO only generated computed classes. We wanted to highlight that the classes OPEN_RECEIPT$_s$, COMMISSION$_s$, and POLICY$_s$ represent the union (data integration) of data from two information sources: $\mathbf{S}_2$ and $\mathbf{S}_3$. In this case, the union is mapped in ISCO by defining two rules for each computed class: one obtains all values of the relation of source $\mathbf{S}_2$, and the other takes all the instances of the relation of $\mathbf{S}_3$ that are distinct from the instances of the relation of $\mathbf{S}_2$. The match between instances is carried out using matching functions, which must be defined in the library *v_utils*.

In scenario salesToll, we created the application salesTool using the ISCO-generated applications module. The following queries can be used to access data from, respectively, $\mathbf{S}$.OPEN_RECEIPT$_s$, and $\mathbf{S}_3$.RECEIPT$_3$, when salesTool is executed:

```
?- s3 :> receipt3(A,B,C,D,E,F,G,H,I).
?- v_utils :> p_s1s2s3_s :> open_receipts(A,B,C,D,E,F,G).
```

In the data warehouse scenario, only $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, and $\mathbf{P}_{s1,s2,s2|w}$ were deemed necessary to be mapped to ISCO schemata (using the ISCO translator module). The mapping of $\mathbf{P}_{s1,s2,s3|w}$ to ISCO generates two computed classes (AGENT$_w$ and AUX_COMMISSION), one static class (AMOUNT_AGENT$_w$), and a view SQL-3 with aggregation functions AMOUNT_AGENT$_w$ (that correspond to the static class AMOUNT_AGENT$_w$). The problem is that that AMOUNT_AGENT$_w$ is derived from AUX_COMMISSION, and refers to the foreign key of AGENT$_w$, where both relations are only defined inside ISCO (Prolog code). Thus, in order for SQL-3 code to work, these two relations must be defined in SQL-3 too. The solution was to implement a library, named sql_materialisation, to materialise the computed classes in ISCO in the same database where AMOUNT_AGENT$_w$ is defined (i.e., bd_insurer). Here, we wanted to highlight that AMOUNT_AGENT$_w$ is derived from AUX_COMMISSION, which is a computed class defined only in Prolog. Also, AMOUNT_AGENT$_w$ refers to a foreign key of another computed class (AGENT$_w$). This implies that the instances of both classes AUX_COMMISSION and AGENT$_w$ must be available when the view SQL-3 is processed. Thus, in order for SQL-3 code to work, these two relations must be defined in SQL-3 too, with their data being materialised in *bd_insurer*. We have developed a library, named *sql_materialisation*, to materialise the computed classes in ISCO.

We created the application *warehouse* using the ISCO-generated applications module. The following queries can be used to access data from, respectively, **W**.AGENT$_w$, and **W**.AMOUNT_AGENT$_w$, when warehouse is executed:

```
?- v_utils :> p_s1s2s3_w :> agentw(A,B,C).
?- v_utils :> p_s1s2s3_w :> amount_agentw(A,B,C,D).
```

6.5 Conclusions

This Chapter has shown a proof-of-concept implementation. We detailed some aspects of its development and presented some practical examples of where it can be used. Parts of this Chapter were published in (Pequeno *et al.*, 2009).

This Chapter served to illustrate the feasibility of our proposal, but our intention is to extend it to a real prototype soon. We believe that it is possible, but it will be necessary to do more studies and benchmarking tests, and make some adjustments to the ISCO system. Another important direction for future work is the development of a graphical user-friendly interface to declare the schemata in the proposed language, and thus hide some syntax details.

7

Conclusion and Future Work

We emphasise the main aspects of this work, highlight some contributions, and pinpoint future directions.

7.1 Conclusions

This work focused on Data Integration Systems (DISs). First, we developed a formal language (L_{PS}) to define schemata; to make the compatibility between models clear in a declarative way, using correspondence assertions; and to identify the instances of different information sources that represent the same entity in the real-world, using matching function signatures. Then, we presented a proposal to automatically connect a global schema to its information sources by using an inference mechanism taking into account the explicit mappings between the information sources and a reference model. Finally, we showed a proof-of-concept implementation, which serves for demonstrating the feasibility of our proposal when: i) using perspective schemata to define the mappings between schemata; and ii) using the inference mechanism to generate new mappings from the previous ones. The implementation was Prolog-based and used the ISCO system, a tool developed using contextual logic programming with persistence; and allows the generation of applications which transparently access information sources in a uniform way, like a mediation system.

Our approach is innovative in several ways. It deals with mappings at the structural level and at the instance level. It also deals with various types of semantic heterogeneity conflicts (naming conflict, encoding conflict, etc.), and everything is completely defined in a conceptual model. Other works deal with mapping between schema's components and/or deal with the instance mapping problem. Wrembel in (Wrembel, 2000), for example, proposes structures to establish

the correspondence between schema's components (cited as *CMS*) and the correspondence between instances (cited as *OMS*). These structures are used for data movement, they are not used to make explicit the correspondence between schema's components. Thus, *CMS* only indicates that there is a relationship, but does not mention anything about how it occurs, nor about how semantic conflicts are resolved. In the same way, *OMS* only identifies which object identifiers of a class are related to object identifiers of another class, functioning more as "lookup tables" than as our matching function signatures. The work of Ravat and Teste in (Ravat & Teste, 2000) deal with mapping between schemata in a DW scenario, but they disregard the instance matching problem. They assume that all information sources have object identifiers, which are used to connect classes of the information sources to classes of the global schema.

Our approach, to declaratively define correspondence between schemata, makes clear the mappings that there are in a DIS. Moreover, the mappings are easy to maintain and can be reused.

The reference model here is formal and plays a more active role than the ones used in other approaches (Imhoff *et al.*, 2003; Geiger, 2009; Moody & Kortink, 2000; Inmon *et al.*, 2001; Inmon *et al.*, 2008), in which it serves only as a reference and guide proposals. The reference model is the core of an architecture based on which the proposed inference mechanism was developed. Furthermore, by using the reference model the designer does not need to map schemata with each other. This work is, in principle, reduced, since schemata (sources or global) must only align with the reference model, rather than with each participating schema. More similar to ours is the research of Calvanese and others in (Calvanese *et al.*, 2006), which also provides an active role to the reference model (called *enterprise model*). All schemata in their proposal, including the global one, are formed by relational structures, which are defined as views (queries) over the reference model. Using inference techniques on queries, they can discover when a set of tuples in a query is contained in, or disjoint of, a set of tuples of another query. Also, they can discover if, for a given query over the reference model, there is a database satisfying the reference model. This clearly is not the function of the reference model in our work. Our proposal also differs from theirs in the following aspects: a) they provide the user with various levels of abstraction: conceptual, logical, and physical; b) transformations in their work, such as restructuring of schema and values, conversion between diverse domains, data consolidation, and mapping of instances are deferred to the logical level; and c) they do not deal with complex data, integrity constraints, and path expressions.

Our inference mechanism allow us to (semi-) automatically generate new mappings based on previous ones. Specifically, it can create the direct mapping between the global schema and its information sources.

Our approach is useful for DISs that define a common or canonical schema (such as FDBSs and DW systems). Specifically, our implementation can be used by applications that hide the complexity involved in accessing multiple and heterogeneous data sources from their users, since these sources are in databases that can be queried via remote access in real-time. Here, we dealt with problems related to the early stage of the ETL process, specifically mappings and transformations. Researchers could use our proposal as a starting point for dealing with other tasks of the ETL process, such as data movement and management of historical data.

7.2 Open Issues and Future Work

There are several research issues left open on the basis of this work. Our proposal could be enriched with the improvement of the languages L_S and L_{PS}. In the context of our language L_S, it could be interesting to deal with other more complex data types, such as geo-spatial and multimedia, since the necessity to develop DISs involving these types of data is growing at a rapid rate. The insertion of geo-spatial, and maybe multimedia too, types in L_S can have an impact in the language L_{PS}, since that a specific type of CAs should be created to deal with spatial dependency between objects (Menezes, 2003). L_S also does not deal with general integrity constraints other than those of entity constraint and referential integrity constraint, such as check conditions and constraints that represent the business rules. The language L_{PS}, although it allows for expressing various usual types of mappings, does not deal with some situations that could be interesting to consider, such as those presented in Section 4.6. Specifically, we join a set of instances using only the criterion of representing the same entity in the real-world. It could be interesting to allow other criteria in the join condition beside this one, such as those shown in Example 4.6.2. Also, the behaviour part of schemata (the methods) were not much explored in this work. For example, nothing was mentioned about methods in the perspective schemata.

Another important direction for future work is to investigate how the perspective schemata can be used to automate the materialisation of the ETL process, which implies in dealing with the data movement. Data movement (or data maintenance) should take into account a multitude of issues, which includes: a) how long to wait until the data of the global schema

need to be updated (maintenance time) - *periodic maintenance, on-commit maintenance*, or *on-demand maintenance* (Gupta & Mumick, 1999); b) what maintenance strategy should be used - *incremental refreshing* or *full recomputing* (Fan, 2005); activeness of information sources - *sufficient activeness, restricted activeness*, or *no activeness* (Zhou *et al.*, 1995a); d) how historical data is managed; and e) materialisation - *full materialisation, partial materialisation*, or *full virtual* (Zhou *et al.*, 1995a). This information should be stored as metadata and approaches on how to manage this metadata and on how it can be used to load the global schema must be studied. We believe that the CAs and MF signatures, together with the metadata specific to deal with data movement (previously suggested), can be used to build algorithms, or triggers (similar to what occurred in (Pequeno, 2000)), to maintain the data of the global schema.

From a practical viewpoint, both L_S and L_{PS} could be easier to use if there was a graphical interface in order to hide the syntax details of the languages. Our implementation also needs to be extended to a real prototype in order for it to be able to be applied to various situations, some synthetic and others real, as means of providing a more complete experimental validation of the usefulness of the chosen approaches. We will need to do more studies, benchmarking tests, and probably some adjustments to the ISCO system.

In the future, we intend to develop a tool to help the designer in building new perspective schemata of integration (from the information sources to the reference model) based on already defined perspective schemata (see Figure 7.1). This tool would take as input, for example, more than one perspective schemata from the information source to the reference model, show all CAs of each class/relation of the reference model, and based on this information together with more semantic information, the designer could define the CAs for the new perspective schema of integration.

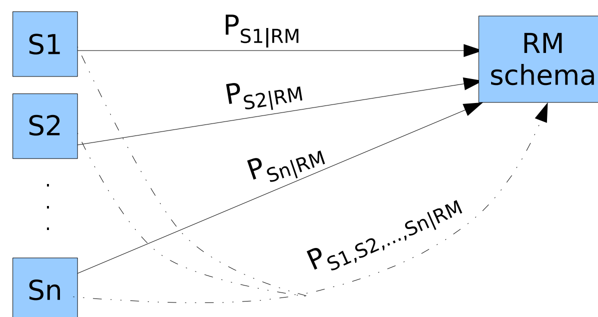


Figure 7.1: Example of a perspective schema of integration.

Bibliography

Abiteboul, Serge, Hull, Richard, & Vianu, Victor. 1995. *Fundations of Databases*. Addison-Wesley Publishing Company. Chap. 20-21, pages 508–573.

Abreu, Salvador, & Diaz, Daniel. 2003. Objective: in Minimum Context. *Pages 128–147 of:* Palamidessi, Catuscia (ed), *In Proc. of 19th Intl. Conf. in Logic Programming (ICLP 2003)*. Lecture Notes in Computer Science 2916, Springer-Verlag.

Abreu, Salvador, & Nogueira, Vitor. 2005 (October). Using a Logic Programming Language with Persistence and Contexts. *In:* Umeda, In Masanobu, & Wolf, Armin (eds), *Procs. of the 16th Intl. Conf. on Applications of Declarative Programming and Knowledge Management*. INAP'05. Waseda University.

Abreu, Salvador, & Nogueira, Vitor. 2006. Using a Logic Programming Language with Persistence and Contexts. *Pages 38–47 of:* Takata, Osamu, Umeda, Masanobu, Nagasawa, Isao, Tamura, Naoyuki, Wolf, Armin, & Schrader, Gunnar (eds), *16th Intl. Conf. on Applications of Declarative Programming and Knowledge Management*. INAP'05, vol. 4369. Springer. Revised Selected Papers. Lecture Notes in Computer Science.

Abreu, Salvador, Diaz, Daniel, & Nogueira, Vitor. 2004 (June). Organizational Information Systems Design and Implementation with Contextual Constraint Logic Programming. *In: IT Innovation in a Changing World – The 10th Intl. Conf. of European University Information Systems*.

Agner, Luciane Telinski Wiedermann. 2005. T-SVM: uma Abordagem para Manutenção de Visões Materializadas em Ambientes Data Warehouse. *RECEN - Revista Ciências Exatas e Naturais*, **7(1)**, 35–52.

Albrecht, Alexander, & Naumann, Felix. 2008. Managing ETL Processes. *Pages 12–15 of: Intl. Workshop on New Trends in Information Integration*. NTII'08.

Amano, Shun'ichi, David, Claire, Libkin, Leonid, & Murlak, Filip. 2010. On the Tradeoff between Mapping and Querying Power in XML Data Exchange. *Pages 155–164 of: Procs. of the 13th Intl. Conf. on Database Theory*. ICDT'10. USA: ACM.

Angluin, Dana. 1988. Queries and Concept Learning. *Machine Learning*, **2**(4), 319–342.

Arenas, Marcelo, & Libkin, Leonid. 2008. XML Data Exchange: Consistency and Query Answering. *J. ACM*, **55**(2), 1–72.

Arens, Yigal, Chee, Chin Y., Hsu, Chun-Nan, & Knoblock, Craig A. 1993. Retrieving and Integrating Data from Multiple Information Sources. *Intl. Journal on Intelligent and Cooperative Information Systems*, **2(2)**, 127–158.

Atay, Mustafa, Sun, Yezhou, Liu, Dapeng, Lu, Shiyong, & Fotouhi, Farshad. 2010. Mapping XML Data to Relational Data: A DOM-Based Approach. *CoRR*, **abs/1010.1746**.

Bækgaard, Lars. 1999. Event-Entity-Relationship Modeling in Data Warehouse Environments. *Pages 9–14 of: Procs. of the 2nd ACM Intl. Workshop on Data Warehousing and OLAP*. DOLAP'99. USA: ACM Press.

Bahloul, S. Nait, Amghar, Y., & Sayah, M. 2004. A* Algebra for an Extended Object/Relational Model. *Intl. Journal of Computer Science & Applications*, **1**(2), 76–95.

Bebel, Bartosz, Eder, Johann, Koncilia, Christian, Morzy, Tadeusz, & Wrembel, Robert. 2004 (March). Creation and Management of Versions in Multiversion Data Warehouse. *Pages 717–723 of: Procs. of the 2004 ACM Symposium on Applied Computing*. SAC'04.

Belden, Eric, & Greenberg, Janis. 2008 (August). *Oracle Database - Object-Relational Developer's Guide 11g Release 1 (11.1)*. Oracle Corporation.

Beneventano, Domenico, *et al.* 2001 (July). The MOMIS Approach to Information Integration. *Pages 194–198 of: Procs. of the 3rd Intl. Conf. on Enterprise Information Systems*, vol. 1.

Berger, Stefan, & Schrefl, Michael. 2008. From Federated Databases to a Federated Data Warehouse System. *Page 394 of: HICSS '08: 41st Annual Hawaii Intl. Conf. on System Sciences*. USA: IEEE Computer Society.

Bernstein, Philip A. 2003. Applying Model Management to Classical Meta Data Problems. *In: First Biennial Conf. on Innovative Data Systems Research, CIDR'03*.

Bernstein, Philip A., Melnik, Sergey, Petropoulos, Michalis, & Quix, Christoph. 2004. Industrial-Strength Schema Matching. *SIGMOD Rec.*, **33**(4), 38–43.

Bernstein, Philip A., Melnik, Sergey, & Churchill, John E. 2006. Incremental Schema Matching. *Pages 1167–1170 of: Procs. of the 32nd Intl. Conf. on Very Large Data Bases*. VLDB'06. VLDB Endowment.

Bhattacharjee, Anupam, & Jamil, Hasan. 2009. OntoMatch: a Monotonically Improving Schema Matching System for Autonomous Data Integration. *Pages 318–323 of: Procs. of the 10th IEEE Intl. Conf. on Information Reuse & Integration*. IRI'09. USA: IEEE Press.

Bickel, Michael Allen. 1987. Automatic Correction to Misspelled Names: A Fourth-Generation Language Approach. *Commun. ACM*, **30**(3), 224–228.

Bilenko, Mikhail, & Mooney, Raymond J. 2002 (February). *Learning to Combine Trained Distance Metrics for Duplicate Detection in Databases*. Tech. rept. AI 02-296. Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX.

Bilke, Alexander, & Naumann, Felix. 2005. Schema Matching Using Duplicates. *Pages 69–80 of: Procs. of the 21st Intl. Conf. on Data Engineering*. ICDE'05. USA: IEEE Computer Society.

Blaschka, Markus, Sapia, Carsten, & Hofling, Gabriele. 1999. On Schema Evolution in Multidimensional Databases. *Pages 153–164 of: Data Warehousing and Knowledge Discovery*.

Body, Mathurin, Miquel, Maryvonne, Bédard, Yvan, & Tchounikine, Anne. 2002. A Multidimensional and Multiversion Structure for OLAP Applications. *Pages 1–6 of: Procs. of the 5th ACM Intl. Workshop on Data Warehousing and OLAP*. DOLAP'02. USA: ACM.

Body, Mathurin, Miquel, Maryvonne, Bédard, Yvan, & Tchounikine, Anne. 2003. Handling Evolutions in Multidimensional Structures. *Procs. of the 19th Intl. Conf. on Data Engineering*, 581.

Bollacker, K.D., Lawrence, S., & Giles, C.L. 1998. Citeseer: An Autonomous Web Agent for Automatic Retrieval and Identication of Interesting Publications. *Pages 116–123 of: Proc. of 2nd Int. ACM Conf. on Autonomous Agents*.

Boussaid, Omar, Bentayeb, Fadila, & Darmont, Jérôme. 2008. An MAS-Based ETL Approach for Complex Data. *CoRR*, **abs/0809.2686**.

Business Rules Team. 2000. *Defining Business Rules - What Are They Really?* Tech. rept. the GUIDE Business Rules Project.

C. Batini, M. Lenzerini, S.B. Navathe. 1986. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, **18**(4), 323–364.

Cali, Andrea, Calvanese, Diego, De Giacomo, Giuseppe, & Lenzerini, Maurizio. 2004. Data Integration under Integrity Constraints. *Information Systems*, **29**(2), 147–163.

Calvanese, Diego. 2001. Data Integration in Data Warehousing. *Intl. Journal of Cooperative Information Systems*, **10(3)**, 237–271.

Calvanese, Diego, Giacomo, Giuseppe De, Lenzerini, Maurizio, Nardi, Daniele, & Rosati, Riccardo. 1998. Description Logic Framework for Information Integration. *Pages 2–13 of: Procs. of the 16th Intl. Conf. on Principles of Knowledge Representation and Reasoning*. KR'98.

Calvanese, Diego, Dragone, Luigi, Nardi, Daniele, Rosati, Riccardo, & Trisolini, Stefano M. 2006. Enterprise Modeling and Data Warehousing in TELECOM ITALIA. *Inf. Syst.*, **31**(1), 1–32.

Castano, S., Ferrara, A., Lorusso, D., & Montanelli, S. 2008. On the Ontology Instance Matching Problem. *Database and Expert Systems Applications, Intl. Workshop on*, **0**, 180–184.

Cattell, Rick G.G., Barry, Douglas K., Berler, Mark, Eastman, Jeff, Jordan, David, Russell, Conn, Schadow, Olaf, Stanienda, Torsten, & Velez, Fernando. 2000. *The Object Database Standard ODMG 3.0.* Morgan Kaufmann Publishers.

Chao, Ching-Ming. 2005 (May). Change Detection and Maintenance of an XML Web Warehouse. *Pages 52–59 of: Procs. of the 7th Intl. Conf. on Enterprise Information Systems*. ICEIS'05.

Chen, Xuedong, O'Neil, Patrick, & O'Neil, Elizabeth. 2008. Adjoined Dimension Column Clustering to Improve Data Warehouse Query Performance. *Pages 1409–1411 of: Procs. of the 2008 IEEE 24th Intl. Conf. on Data Engineering*. ICDE'08. USA: IEEE Computer Society.

Chiticariu, Laura, Hernández, Mauricio A., Kolaitis, Phokion G., & Popa, Lucian. 2007. Semi-Automatic Schema Integration in Clio. *Pages 1326–1329 of: Procs. of the 33rd Intl. Conf. on Very Large Data Bases*. VLDB'07. VLDB Endowment.

Choi, Namyoun, Song, Il-Yeol, & Han, Hyoil. 2006. A Survey on Ontology Mapping. *SIGMOD Rec.*, **35**(3), 34–41.

Codd, Edgar F. 1970. A Relational Model of Data for Large Shared Data Banks. *Pages 377–387 of: Communications of the ACM.*

Cohen, William W., & Richman, Jacob. 2002. Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration. *Pages 475–480 of: Procs. of the 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining.* USA: ACM.

Darwen, Hugh, & Date, C. J. 1995. The Third Manifesto. *ACM SIGMOD Record*, **24**(1), 39–49.

Date, C. J., & Darwen, Hugh. 2000. *Foundation for Future Database Systems: The Third Manifesto.* Addison-Wesley.

Davis, Peter T., Elson, David K., & Klavans, Judith L. 2003. Methods for Precise Named Entity Matching in Digital Collections. *Pages 125–127 of: Procs. of the 3rd ACM/IEEE-CS Joint Conf. on Digital Libraries.* JCDL'03. USA: IEEE Computer Society.

DB2. n.d. *DB2 Version 9 for Linux, UNIX, and Windows.* IBM Corporation. [Online, access date: 2009, August]. Retrieved from: http://public.boulder.ibm.com/infocenter/db2/luw/v9r5/topic.com.ibm.db2.luw.container.doc/docc0052964.html.

Dessloch, Stefan, Hernández, Mauricio A., Wisnesky, Ryan, Radwan, Ahmed, & Zhou, Jindan. 2008 (April). Orchid: Integrating Schema Mapping and ETL. *Pages 1307–1316 of: Procs. of the 24th Intl. Conf. on Data Engineering.*

Dhamankar, Robin, Lee, Yoonkyong, Doan, AnHai, Halevy, Alon Y., & Domingos, Pedro. 2004. IMAP: Discovering Complex Mappings between Database Schemas. *Pages 383–394 of: SIGMOD Conf.*

Dias, Maria M., Tait, Tania C., Menolli, André Luis A., & Pacheco, Roberto C. S. 2008. Data Warehouse Architecture through Viewpoint of Information System Architecture. *Pages 7–12 of: Procs. of the 2008 Intl. Conf. on Computational Intelligence for Modelling Control & Automation.* CIMCA'08. USA: IEEE Computer Society.

Do, Hong-Hai. 2006. *Schema Matching and Mapping-Based Data Integration: Architecture, Approaches and Evaluation.* Germany: VDM Verlag.

Do, Hong-Hai, & Rahm, Erhard. 2002. COMA - A System for Flexible Combination of Schema Matching Approaches. *Pages 610–621 of: VLDB.*

Doan, AnHai, & Halevy, Alon Y. 2004. *Semantic Integration Research in the Database Community: A Brief Survey.* American Association for Artificial Intelligence.

Doan, AnHai, & Halevy, Alon Y. 2005. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, **26**(1), 83–94.

Doan, AnHai, Domingos, Pedro, & Halevy, Alon Y. 2001. Reconciling Schemas of Disparate Data Sources: a Machine-Learning Approach. *SIGMOD Rec.*, **30**(2), 509–520.

Doan, AnHai, Madhavan, Jayant, Dhamankar, Robin, Domingos, Pedro, & Halevy, Alon. 2003a. Learning to Match Ontologies on the Semantic Web. *The VLDB Journal*, **12**(4), 303–319.

Doan, AnHai, *et al.* 2003b (Aug.). Object Matching for Information Integration: a Profiler-Based Approach. *In: Procs. of IJCAI-03 Workshop on Information Integration on the Web.*

documentation team, SQL Server. 2009. *SQL Server 2008 Books Online*. Microsoft Corporation. Books online. Retrieved from: http://msdn.microsoft.com/en-us/library/bb545450.aspx.

Drumm, Christian, Schmitt, Matthias, Do, Hong-Hai, & Rahm, Erhard. 2007. Quickmig: Automatic Schema Matching for Data Migration Projects. *Pages 107–116 of: Procs. of the 16th ACM Conf. on Information and Knowledge Management*. CIKM'07. USA: ACM.

Duschka, Oliver M., Genesereth, Michael R., & Levy, Alon Y. 2000. Recursive Query Plans for Data Integration. *The Journal of Logic Programming*, **43**(1), 49–73.

Eder, Johann, & Wiggisser, Karl. 2010. *Data Warehousing Design and Advanced Engineering Applications: Methods for Complex Construction*. Poitiers University, France. Chap. 10, pages 171–188.

Eder, Johann, Koncilia, Christian, & Morzy, Tadeusz. 2002 (May). The COMET Metamodel for Temporal Data Warehouses. *Pages 83–99 of: Advanced Information Systems Engineering, 14th Intl. Conf.* CAiSE'02.

Elmasri, Ramez, & Navathe, Shamkant B. 2006. *Fundamentals of Database Systems*. 5th edn. Addison Wesley.

Fagin, Ronald, & Nash, Alan. 2010. The Structure of Inverses in Schema Mappings. *J. ACM*, **57**(November), 31:1–31:57.

Fagin, Ronald, Kolaitis, Phokion G., Miller, Renée J., & Popa, Lucian. 2005. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, **336**(1), 89 – 124. Database Theory.

Fagin, Ronald, Kolaitis, Phokion G., Nash, Alan, & Popa, Lucian. 2008. Towards a Theory of Schema-Mapping Optimization. *Pages 33–42 of: Procs. of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS'08. USA: ACM.

Fan, Hao. 2005. *Investigating a Heterogeneous Data Integration Approach for Data Warehousing*. Ph.D. thesis, University of London.

Fan, Hao, & Poulovassilis, Alexandra. 2004. Schema Evolution in Data Warehousing Environments - A Schema Transformation-Based Approach. *Pages 639–653 of: ER*.

Firebird. n.d.. *Firebird DBMS*. Available in: http://firebird.sourceforge.net/. Last access in August,2009.

Franklin, Michael J., Halevy, Alon Y., & Maier, David. 2008. A First Tutorial on Dataspaces. *PVLDB*, **1**(2), 1516–1517.

Ganesh, M., Sirvastava, J., & Richardsons, T. 1996. Mining Entity-Identification Rules for Database Integration. *Pages 291–294 of: Procs. of the 2nd Intl. Conf. on Data Mining and Knowledge Discovery*.

Gardarin, Georges, Dragan, Florin, & Yeh, Laurent. 2008. P2P Semantic Mediation of Web Sources. *Pages 3–16 of:* Aalst, Will, Mylopoulos, John, Sadeh, Norman M., Shaw, Michael J., Szyperski, Clemens, Manolopoulos, Yannis, Filipe, Joaquim, Constantopoulos, Panos, & Cordeiro, José (eds), *Enterprise Information Systems*. Lecture Notes in Business Information Processing, vol. 3. Springer Berlin Heidelberg.

Geiger, Jonathan G. 2009. Why Build a Data Model? *Information Management Magazine*, June.

Giunchiglia, Fausto, Shvaiko, Pavel, Yatskevich, Mikalai, Giunchiglia, Fausto, Shvaiko, Pavel, & Yatskevich, Mikalai. 2005. Semantic Schema Matching. *Pages 347–365 of: In Procs. of CoopIS.*

Golfarelli, Matteo, Maniezzo, Vittorio, & Rizzi, Stefano. 2004. Materialization of Fragmented Views in Multidimensional Databases. *Data Knowl. Eng.*, **49**(3), 325–351.

Golfarelli, Matteo, Lechtenbörger, Jens, Rizzi, Stefano, & Vossen, Gottfried. 2006. Schema Versioning in Data Warehouses: Enabling Cross-Version Querying via Schema Augmentation. *Data Knowl. Eng.*, **59**(2), 435–459.

Gomes, Raphael do Vale, Leme, Luiz André P. Paes, & Casanova, Marco A. 2009. MatchMaking - A Tool to Match OWL Schemas. *Revista de Informática Teórica e Aplicada*, **16**(2), 71–76.

Grandi, Fabio, & Mandreoli, Federica. 2003. A Formal Model for Temporal Schema Versioning in Object-Oriented Databases. *Data Knowl. Eng.*, **46**(2), 123–167.

Gravano, L., Ipeirotis, P., Koudas, N., & Srivastava, D. 2003. Text Joins for Data Cleansing and Integration in a RDBMS. *In: Procs. of the 19th IEEE Intl. Conf. on Data Engineering.* ICDE'03. Poster paper.

Gupta, Ashish, & Mumick, Iderpal Singh (eds). 1999. *Materialized Views: Techniques, Implementations, and Applications.* USA: MIT Press. Chap. Maintenance policies, pages 9–11.

Haas, L., Schwarz, P., Kodali, P., Kotlar, E., Rice, J., & Swope., W. 2001. DiscoveryLink: A System for Integrated Access to Life Sciences Data Sources. *IBM Systems Journal*, **40**, 489–511.

Halevy, Alon Y., Rajaraman, Anand, & Ordille, Joann J. 2006. Data Integration: The Teenage Years. *Pages 9–16 of: VLDB.*

Han, Lu, & Qing-zhong, Li. 2004. Ontology Based Resolution of Semantic Conflicts in Information Integration. *Wuhan University Journal of Natural Sciences*, **9**, 606–610.

Haselmann, Till, Lechtenbörger, Jens, & Vossen, Gottfried. 2007. Data Warehouse Detective: Schema Design Made Easy. *Pages 606–608 of: Datenbanksysteme in Business, Technologie und Web.* BTW'07.

Hellerstein, J., Stonebraker, M., & Caccia, R. 1999. Independent, Open enterprise Data Integration. *IEEE Data Engineering Bulletin*, **22(1)**(March), 43–49.

Hernandez, Mauricio A., & Stolfo, Salvatore J. 1995. The Merge/Purge Problem for Large Databases. *Pages 127–138 of: SIGMOD Conf.*

Hurtado, Carlos A., Mendelzon, Alberto O., & Vaisman, Alejandro A. 1999. Maintaining Data Cubes under Dimension Updates. *Pages 346–355 of: Procs. of the 15th Intl. Conf. on Data Engineering.* ICDE'99. IEEE Computer Society.

Husemann, Bodo, Lechtenborger, Jens, & Vossen, Gottfried. 2000. Conceptual Data Warehouse Modeling. *Page 6 of: Design and Management of Data Warehouses.*

Hylton, Jemy A. 1996. *Identifying and Merging Related Bibliographic Records*. M.Phil. thesis, Massachusetts Institute of Technology - Department of Electrical Engineering and Comouter Science.

IBM. 2008. *DB2 Version 9.1 for z/OS - SQL reference*. 6th edn. IBM Corporation.

Imhoff, Claudia, Galemmo, Nicholas, & Geiger, Jonathan G. 2003. *Mastering Data Warehouse Design - Relational and Dimensional Techniques*. Wiley Publishing.

Informix11. 2008 (April). *IBM Informix - Version 11.5, Database Design and Implementation Guide*. IBM Corporation.

Informix11. 2009 (July). *IBM Informix - Version 11.5, Guide to SQL: Reference*. IBM Corporation.

Ingres92. 2008a. *Ingres 9.2 Object management Extension User Guide*. Ingres Corporation.

Ingres92. 2008b. *Ingres 9.2 SQL Reference Guide*. Ingres Corporation.

Inmon, W. H. 1996. *Building the Data Warehouse*. 2nd edn. Wiley Publishing.

Inmon, W. H., Imhoff, Claudia, & Sousa, Ryan. 2001. *Corporate Information Factory*. 2nd edn. John Wiley & Sons.

Inmon, William, Strauss, Derek, & Neushloss, Genia. 2008. *DW 2.0: the Architecture for the Next Generalization of Data Warehousing*. Morgan Kaufman.

Ives, Zachary G., Halevy, Alon Y., Mork, Peter, & Tatarinov, Igor. 2004. Web Semantics: Science, Services and Agents on the World Wide Web. *Pages 155–175 of: World Wide Web Conf.*, vol. 1.

Ives, Zachary G., Knoblock, Craig A., Minton, Steven, Jacob, Marie, Talukdar, Partha Pratim, Tuchinda, Rattapoom, Ambite, José Luis, Muslea, Maria, & Gazen, Cenk. 2009. Interactive Data Integration through Smart Copy & Paste. *In: 4th Biennial Conf. on Innovative Data Systems Research*. CIDR'09.

J.Wislicki, K.Kuliberda, T.Kowalski, R.Adamus, & K.Subieta. 2008. Implementation and Testing of SBQL Object-Relational Wrapper Supporting Query Optimisation. *Pages 39–56 of: Procs. of the First Intl. Conf. on Object Databases*. ICOODB'08.

Kaas, Christian, Pedersen, Torben Bach, & Rasmussen, Bjørn. 2004. Schema Evolution for Stars and Snowflakes. *Pages 425–433 of: ICEIS (1)*.

Kalfoglou, Yannis, & Schorlemmer, Marco. 2003. Ontology Mapping: the State of the Art. *Knowl. Eng. Rev.*, **18**(1), 1–31.

Kent, W., *et al.* 1993. Object Identification in Multi-Database Systems. *In:* Hsiao, D., Neuhold, E., & Sacks-Davis, R. (eds), *Interoperable Database Systems (DS-5) (A-25)*. North-Holland: Elsevier Science Publishers B. V.

Khalid, Belhajja, Paton, Norman W., Embury, Suzanne M., Fernandes, Alvaro A. A., & Hedeler, Cornelia. 2010. Feedback-Based Annotation, Selection and Refinement of Schema Mappings for Dataspaces. *Pages 573–584 of: Procs. of the 13th Intl. Conf. on Extending Database Technology*. EDBT'10. USA: ACM.

Kim, Mi-Yeon, Seo, Jung-Min, & Moon, Chang-Joo. 2007. SQL Extension for Multidatabase System. *Pages 283–289 of: Proc. of the The 2007 Intl. Conf. Computational Science and its Applications*. ICCSA'07. USA: IEEE Computer Society.

Kimball, R. 1996. *The Data Warehouse Toolkit*. John Wiley & Sons.

Kimball, Ralph, & Caserta, Joe. 2004. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons.

Koncilia, Christian. 2003. A Bi-Temporal Data Warehouse Model. *In: The 15th Conf. on Advanced Information Systems Engineering*. CAiSE'03.

Kriegel, Alex, & Trukhnov, Boris M. 2008. *SQL Bible*. 2nd edn. John Wiley and Sons.

Kumar, T.V. Vijay, Goel, Anurag, & Jain, Neeraj. 2010. Mining Information for Constructing Materialised Views. *Intl. Journal of Information and Communication Technology*, **2, Number 4 / 2010**, 386–405.

Kuno, H.A., Ra, Y.G., & Rudensteiner, E.A. 1995. *The Object-Slicing Technique: a Flexible Object Representation and its Evaluation*. Tech. rept. University of Michigan. No. CSE-TR-241-95.

Labio, Wilburt, Yang, Jun, Cui, Yingwei, Garcia-Molina, Hector, & Widom, Jennifer. 2000. Performance Issues in Incremental Warehouse Maintenance. *Pages 461–472 of: Procs. of the 26th Intl. Conf. on Very Large Data Bases*. VLDB'00. USA: Morgan Kaufmann Publishers Inc.

Lausen, Georg, & Vossen, Gottfried. 1998. *Models and Languages of Object-Oriented Databases*. Addison-Wesley Publishing Company. Chap. 3, pages 71–93.

Lopes, Nuno, Fernandes, Cláudio, & Abreu, Salvador. 2008. Representing and Querying Multiple Ontologies with Contextual Logic Programming. *In:* Pereira, M. J. Varanda, Henriques, P., & de Sousa, S. Melo (eds), *Procs. of the CoRTA'2008 - Compilers, Related Technologies and Applications*. Portugal: Universidade do Minho.

Luján-Mora, Sergio, Vassiliadis, Panos, & Trujillo, Juan. 2004. Data Mapping Diagrams for Data Warehouse Design with UML. *Pages 191–204 of: ER'04: 23rd Intl. Conf. on Conceptual Modeling*. Springer.

Malinowski, E., & Zimányi, E. 2006 (June). Object-Relational Representation of a Conceptual Model for Temporal Data Warehouses. *In: In Proc. of the 18th Int. Conf. on Advanced Information Systems Engineering*. CAiSE'06.

Malinowski, Elzbieta, & Zimányi, Esteban. 2006. A Conceptual Solution for Representating Time in Data Warehouse Dimensions. *In: 3th Asia-Pacific Conf. on conceptual modellling*. APCCM'06, vol. 53.

Mccann, Robert, Doan, Anhai, Varadarajan, Vanitha, & Kramnik, Er. 2003. Building Data Integration Systems via Mass Collaboration. *In: Intl. Workshop on the Web and Databases*. WebDB.

Mendelzon, Alberto O., & Vaisman, Alejandro A. 2003. Time in Multidimensional Databases. *Multidimensional Databases: Problems and Solutions*, 166–199.

Menezes, Antônio Márcio Adiodato de. 2003. *GeoBinder: Um Framework para Integração de Esquemas de Dados Geográficos*. Un-finished Masther Thesis.

Michalowski, Martin, Thakkar, Snehal, & Knoblock, Craig A. 2003. Exploiting Secondary Sources for Automatic Object Consolidation. *Pages 34–36 of: Procs. of the KDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation.*

Monge, Alvaro E., & Elkan, Charles. 1996. The Field Matching Problem: Algorithms and Applications. *Pages 267–270 of: Knowledge Discovery and Data Mining.*

Moody, Daniel L., & Kortink, Mark A. R. 2000. From Enterprise Models to Dimensional Models: a Methodology for Data Warehouse and Data Mart Design. *Page 5 of: Procs. of the 2nd Intl. Workshop on Design and Management of Data Warehouses*. DMDW'00.

Morzy, Tadeusz, & Wrembel, Robert. 2003 (Apr.). Modeling a Multiversion Data Warehouse: A Formal Approach. *Pages 120–127 of: Procs. of the 5th Intl. Conf. on Enterprise Information Systems.*

Nottelmann, Henrik, & Straccia, Umberto. 2007. Information Retrieval and Machine Learning for Probabilistic Schema Matching. *Inf. Process. Manage.*, **43**(3), 552–576.

Papakonstantinou, Yannis, Abiteboul, Serge, & Garcia-Molina, Hector. 1996. Object Fusion in Mediator Systems. *In: Intl. Conf. on Very Large Databases.*

Pedersen, Torben Bach, & Jensen, Christian S. 1999. Multidimensional Data Modeling for Complex Data. *Pages 336–345 of: Procs. of the 15th Intl. Conf. on Data Engineering.* ICDE'99. Australia: IEEE Computer Society.

Pequeno, Valéria M., & Pires, J. Carlos Gomes Moura. 2009a (November). Reference Model and Perspective Schemata Inference for Enterprise Data Integration. *In: 18th Intl. Conf. on Applications of Declarative Programming and Knowledge Management*. INAP'09. Revised selected papers Lecture Notes in Computer Science (LNAI'11), Springer (to appear).

Pequeno, Valéria M., & Pires, J. Carlos Gomes Moura. 2009b. Using Perspective Schemata to Model the ETL Process. *Pages 332–339 of: Intl. Conf. on Management Information Systems*. ICMIS'09. France: World Academy of Science, Engineering and Technology.

Pequeno, Valéria M., Abreu, Salvador, & Pires, J. Carlos Gomes Moura. 2009. Using a Contextual Logic Programming Language to Acess Data in Warehousing Systems. *In: 14th Portuguese Conf. on Artificial Intelligence*. EPIA'09.

Pequeno, Valéria Magalhães. 2000 (April). *Auto-Manutenção de Classes de Fusão em Visões de Integração de Dados*. M.Phil. thesis, Universidade Federal do Ceará, Brazil.

Pequeno, Valéria Magalhães, & Ponte Vidal, Vânia Maria. 2002 (April). Using Full Match Classes for Self-Maintenance of Mediated Views. *Pages 148–154 of: Procs. of the 4st Intl. Conf. on Enterprise Information Systems*. ICEIS'02. Enterprise Information Systems IV, 2003 (Selected paper).

Pequeno, Valéria Magalhães. 2006 (June). *Handling Time in Data Wahehouses*. Tech. rept. Universidade Nova de Lisboa.

Pequeno, Valéria Magalhães. 2010. *Using Perspective Schema and a Reference Model to Design the ETL Process, annexe 01.* Universidade Nova de Lisboa.

PETRINI, J., & RISCH, T. 2007. SWARD: Semantic Web Abridged Relational Databases. *Pages 455–459 of: 18th Intl. Conf. on Database and Expert Systems Applications*. DEXA'07.

PEUKERT, E., EBERIUS, J., & RAHM, E. 2011. AMC - A Framework for Modelling and Comparing Matching Systems as Matching Processes. *In: Proc. Int. Conf. on Data Engineering (Demo paper)*.

POSTGRESQL. n.d. *PostgreSQL 8.4 Documentation*. The PostgreSQL Global Development Group.

RAHM, ERHARD, & BERNSTEIN, PHILIP A. 2001. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, **10**(4), 334–350.

RAMAN, V., & HELLERSTEIN, J.M. 2001. Potter's Wheel: An Interactive Data Cleaning System. *In: 27th VLDB Conf.*

RAVAT, F., & TESTE, OLIVIER. 2000. Object-Oriented Decision Support System. *In: Proc. of the Int. Conf. on Enterprise Information Systems*.

RISCH, T., & JOSIFOVSKI, V. 2001. Distributed Data Integration by Object-Oriented Mediator Servers. *Concurrency and Computation: Practice and Experience*, **13**, 933–953.

RIZZI, STEFANO, & SALTARELLI, ETTORE. 2003. View Materialization vs. Indexing: Balancing Space Constraints in Data Warehouse Design. *Pages 502–519 of: Procs. of the 15th Intl. Conf. on Advanced Information Systems Engineering*. Berlin, Heidelberg: Springer-Verlag.

RIZZI, STEFANO, ABELLÓ, ALBERTO, LECHTENBÖRGER, JENS, & TRUJILLO, JUAN. 2006. Research in Data Warehouse Modeling and Design: Dead or Alive? *Pages 3–10 of: Procs. of the 9th ACM Intl. Workshop on Data Warehousing and OLAP*. DOLAP'06. USA: ACM.

SAEKI, SHIN'ICHIROU, BHALLA, SUBHASH, & HASEGAWA, MASAKI. 2007. Parallel Generation of Base Relation Snapshots for Materialised View Maintenance in Data Warehouse Environment. *Intl. Journal of Computational Science and Engineering*, **3, Number 2 / 2007**, 166–172.

SALEEM, KHALID, & BELLAHSENE, ZOHRA. 2007. *New Challenges in Data Integration: Large Scale Automatic Schema Matching*. Tech. rept. CCSd/HAL : e-articles server (Based on gBUS) [http://hal.ccsd.cnrs.fr/oai/oai.php] (France).

SALGUERO, ALBERTO, ARAQUE, FRANCISCO, & DELGADO, CECILIA. 2008. Ontology Based Framework for Data Integration. *WSEAS Trans. Info. Sci. and App.*, **5**(6), 953–962.

SANTOS, CASSIO, ABITEBOUL, SERGE, & DELOBEL, CLAUDE. 1994. Virtual Schemas and Bases. *Pages 81–94 of:* JAKE, M., BUBENKO, J., & JEFFERY, K. (eds), *Procs. of the 4th Intl. Conf. on Extending Databases Technology*.

SARAWAGI, S., & BHAMIDIPATY, A. 2002. Interactive Deduplication Using Active Learning. *In: In The 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD-2002)*.

SARDA, NANDLAL L. 1999. Temporal Issues in Data Warehouse Systems. *Pages 27–34 of: Proc. in Intl. Symposium on Database Applications in Non-Traditional Environments*. DANTE'99.

SATTLER, KAI-UWE, & SCHALLEHN, EIKE. 2001. A Data Preparation Framework Based on a Multidatabase Language. *Pages 219–228 of: Procs. of the Intl. Database Engineering & Applications Symposium*. IDEAS'01. USA: IEEE Computer Society.

SCHREITER, THOMAS. 2007. *Searching Contents of Wrapped MP3 Files from an Object-Relational Mediator System*. M.Phil. thesis, Information Technology Computing Science Department Uppsala University.

SHAHZAD, M.K., NASIR, J.A., & PASHA, M.A. 2005. CEV-DW: Creation and Evolution of Versions in Data Warehouse. *Asian Journal of Information Technology*, **4(10)**, 910–917.

SHETH, AMIT P., & KASHYAP, VIPUL. 1993. So Far (Schematically) yet So Near (Semantically). *Pages 283–312 of: Procs. of the IFIP WG 2.6 Database Semantics Conf. on Interoperable Database Systems*. DS-5. The Netherlands: North-Holland Publishing Co.

SHVAIKO, PAVEL, & EUZENAT, JÉRÔME. 2008. Ten Challenges for Ontology Matching. *Pages 1164–1182 of: Procs. of the OTM 2008 Confederated Intl. Confs., CoopIS, DOA, GADA, IS, and ODBASE 2008. Part II on On the Move to Meaningful Internet Systems*. OTM'08. Berlin, Heidelberg: Springer-Verlag.

SIVIC, J., & ZISSERMAN, A. 2003 (October). Video Google: A Text Retrieval Approach to Object Matching in Videos. *Pages 1470–1477 of: Procs. of the Intl. Conf. on Computer Vision*, vol. 2.

SKOUTAS, DIMITRIOS, & SIMITSIS, ALKIS. 2006. Designing ETL processes using semantic web technologies. *Pages 67–74 of: DOLAP'06: Procs. of the 9th ACM Intl. Workshop on Data Warehousing and OLAP*. USA: ACM.

SKOUTAS, DIMITRIOS, & SIMITSIS, ALKIS. 2007. Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. *Int. J. Semantic Web Inf. Syst.*, **3**(4), 1–24.

SNODGRASS, RICHARD T. (ed). 1995. *The TSQL2 Temporal Query Language*. Kluwer.

SNODGRASS, RICHARD T., & JENSEN, CHRISTIAN S. 1999. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers.

SOLODOVNIKOVA, DARJA. 2007. Data Warehouse Evolution Framework. *In: Colloquium on Databases and Information Systems*. SYRCODIS'07.

SOLODOVNIKOVA, DARJA. 2009. *Databases and Information Systems V: Selected Papers from the 8th Intl. Baltic Conf., DB&IS 2008*. IOS Prentice Hall. Chap. The Formal Model for Multiversion Data Warehouse Evolution, pages 93–103.

STONEBRAKER, MICHAEL, & BROWN, PAUL. 1999. *Object-Relational DBMSs, Tracking the Next Great Wave*. Morgan Kaufmann Publishers.

STUMPTNER, MARKUS, SCHREFL, MICHAEL, & GROSSMANN, GEORG. 2004. On the Road to Behavior-Based Integration. *Pages 15–22 of: First Asia-Pacific Conf. on Conceptual Modelling*. APCCM'04.

SUJANSKY, W. 2001. Heterogeneous Database Integration in Biomedecine. Methodological Review. *Journal of Biomedical Informatics*, **34**, 285–298.

Tansel, Abdullah Uz, Clifford, James, Gadia, Shashi K., Segev, Arie, & Snodgrass, Richard T. (eds). 1993. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings Publishing Company.

Tejada, Sheila. 2002 (August). *Learning Object Identification Rules for Information Integration*. Ph.D. thesis, Faculty of the graduate school. University of Southern California, USA.

Tejada, Sheila, Knoblock, Craig A., & Minton, Steven. 2002. Learning Domain-Independent String Transformation Weights for High Accuracy Object Identification. *Pages 350–359 of: Procs. of the 8th ACM SIGKDD Intl. Conf. on Knowledge discovery and Data mining*. USA: ACM.

Torres, Ricardo da Silva. 2004. *Ambiente de Gerenciamento de Imagens e Dados Espaciais para o Desenvolvimento de Aplicações em Biodiversidade*. Ph.D. thesis, Universidade Estadual de Campinas - Instituto de Computação.

T.Risch, V.Josifovski, & T.Katchaounov. 2003. *Functional Approach to Data Management - Modeling, Analyzing and Integrating Heterogeneous Data*. Springer. Chap. Functional Data Integration in a Distributed Mediator System.

Tryfona, Nectaria, Busborg, Frank, & Christiansen, Jens G. Borch. 1999. starER: a Conceptual Model for Data Warehouse Design. *Pages 3–8 of: Procs. of the 2nd ACM Intl. Workshop on Data Warehousing and OLAP*. DOLAP'99. USA: ACM.

Türker, Can. 1996 (November). An Approach to Supporting Semantic Integrity of Federated Databases. *In: 10th ERCIM Database Research Group Workshop on Heterogeneous Information Management*.

Ullman, Jeffrey D. 1997. Information Integration Using Logical Views. *Pages 19–40 of:* Afrati, Foto N., & Kolaitis, Phokion G. (eds), *Database Theory - ICDT '97, 6th Intl. Conf., Delphi, Greece, January 8-10, 1997, Procs.* Lecture Notes in Computer Science, vol. 1186. Springer.

Vassiliadis, Panos, Vagena, Zografoula, Skiadopoulos, Spiros, Karayannidis, Nikos, & Sellis, Timos K. 2001. ARKTOS: Towards the Modeling, Design, Control and Execution. *Information Systems*, **26(8)**, 537–561.

Vassiliadis, Panos, Simitsis, Alkis, & Skiadopoulos, Spiros. 2002. Conceptual Modeling for ETL Processes. *Pages 14–21 of: Procs. of the 5th ACM Intl. Workshop on Data Warehousing and OLAP*. DOLAP'02. USA: ACM.

Vidal, Vânia Maria Ponte, Lóscio, Bernadette Farias, & Salgado, Ana Carolina. 2001. Using Correspondence assertions for specifying the semantics of XML-Based mediators. *Pages 3–11 of: Workshop on Information Integration on the Web*.

Wang, G., Zavesov, V., Rifaieh, R., Rajasekar, A., Goguen, J., & Miller, M. 2007. Towards User Centric Schema Mapping Platform. *In: VLDB Workshop Semantic Data and Semantic Integration*.

Widjojo, S., *et al.* 1990. A Specificational Approach to Merging Persistent Object Bases: the 4th Intl. Workshop on Persistent Object Systems. *In:* Dearle, Al, Sahw, Gail, & Zdonik, Stanley (eds), *Implementing Persistent Object Bases*. Morgan Kaufmann.

Wiederhold, G. 1992. Mediators in the Architecture of Future Information Systems. *Pages 38–49 of: IEEE Computer*, vol. 25(3).

Winkler, William E., & Winkler, William E. 2001. Record Linkage Software and Methods for Merging Administrative Lists. *In: Statistical Research Report Series No. RR/2001/03, Washington DC, US Bureau of the Census 2001.*

Wrembel, R. 2000. On Materialising Object-Oriented Views. *Pages 15–28 of:* Barzdins J., Caplinskas A. (ed), *4th Intl. Baltic Workshop.* Selected papers. Kluwer Academic Publishers, March 2001.

Xu, Li, & Embley, David W. 2003. Discovering Direct and Indirect Matches for Schema Elements. *Pages 39–46 of: DASFAA.*

Xu, Yigang, Sauquet, Dominique, Zapletal, Eric, Lemaitre, David, & Degoulet, Patrice. 2000. Integration of Medical Applications: the 'Mediator Service' of the SynEx Platform. *Intl. Journal of Medical Informatics*, **58**, 157–166.

Yang, Jun, & Widom, Jennifer. 1998. Maintaining Temporal Views over Non-Temporal Information Sources for Data Warehousing. *Pages 389–403 of: Advances in Database Technology.* EDBT'98.

Yang, Jun, & Widom, Jennifer. 2000 (March). Temporal View Self-Maintenance. *Pages 395–412 of: Advances in Database Technology, 7th Intl. Conf. on Extending Database Technology.* EDBT'00.

Yang, Jun, & Widom, Jennifer. 2001 (April). Incremental Computation and Maintenance of Temporal Aggregates. *Pages 51–60 of: Procs. of the 17th Intl. Conf. on Data Engineering.* ICDE'01.

Yang, Jun, & Widom, Jennifer. 2003. Incremental Computation and Maintenance of Temporal Aggregates. *Very Large Database Journal*, **12**(3), 262–283.

Yoakum-Stover, S., & Malyuta, T. 2008. Unified Architecture for Integrating Intelligence Data. *In: DAMA: Europe Conf.*

Zhang, Zhi, Shi, Pengfei, Che, Haoyang, Sun, Yong, & Gu, Jun. 2006. Formulation Schema Matching Problem for Combinatorial Optimization Problem. *Intl. Journal of Interoperability in Business Information Systems*, **1**(1), 33–60.

Zhao, Huimin, & Ram, Sudha. 2008. Entity Matching across Heterogeneous Data Sources: An Approach Based on Constrained Cascade Generalization. *Data Knowl. Eng.*, **66**(September), 368–381.

Zhou, G., *et al.* 1996. Generating Data Integration Mediators That Use Materialization. *Journal of Intelligent Information Systems*, **6(2/3)**(May), 199–221.

Zhou, Gang, Hull, Richard, King, Roger, & Francitti, Jean-Claude. 1995a. Supporting Data Integration and Warehousing Using H2O. *IEEE Data Engineering*, **18**(2), 29–40.

Zhou, Gang, Hull, Richard, King, Roger, & Franchitti, Jean-Claude. 1995b (May). Using Object Matching and Materialization to Integrate Heterogeneous Databases. *In: Procs. of the Third Intl. Conf. on Cooperative Information Systems.* COOPIS'95.

A

Appendix

Here we present all the rules of our inference mechanism. The following notations will be used during all the appendices:

- The name of a perspective schema will be $\mathbf{P}_{\{S1,S2,\ldots,Sn\}|T}$, for i $\geq$ 1, rather than $\mathbf{P}_{S1,S2,\ldots,Sn|T}$ as usual.
- $\mathcal{D}$ is the *destination* schema.
- $\mathcal{I}$ is the intermediary schema.
- Z^T means that the component **Z** is defined in schema **T**.
- All variables are indicated by an underline:

Variable	Can be instantiated with
$\underline{\mathrm{C}}$	a class/relation/view relation of a schema or perspective schema
$\underline{\boldsymbol{p}}$	a property of a class/relation/view relation
$\underline{\mathtt{A}}$	a basic pattern expression
$\underline{\mathtt{B}}$	a value or a basic pattern expression
$\underline{\textbf{pred}}$	a predicate as defined in L_{PS}
$\underline{\textbf{op}}$	a operand of a predicate $(<, >, \leq, \geq, =, \neq)$
$\underline{\textbf{I}}$	a set of schemata, one of them being the intermediary schema
$\underline{\textbf{X}}$	a set of origin schemata
$\underline{\textbf{S}}$, $\underline{\textbf{S}}_i$	*origin* schemata belonging to **X**

A.1 Substitution-Rules

The following variables will be used in the substitution-rules:

Variable	Can be instantiated with
$\underline{\varrho}$	a (value or reference) path expression as defined in L_S[1]
$\underline{\ell}$	a link of a path expression[2]
$\underline{\boldsymbol{w}}$	a value
$\underline{\varphi}$	a function with n $\geq$ 1 arguments that returns a value
$\underline{\boldsymbol{FK}}$	a foreign key name
$\underline{\natural C}$	a class of a schema
$\underline{V}$	a view relation
$\underline{\Diamond}$	an operand appearing in an ECA ($-, \cap$, or $\bowtie$)
$\underline{\blacklozenge}$	an operand appearing in an ECA ($-, \cap$, or $\bowtie$), such that $\blacklozenge \neq \Diamond$
$\underline{\mathtt{E}}$	a class/relation/view relation (C) or
	a class/relation/view relation with a predicate (C(**pred**))
$\underline{\mathtt{E}}$(**pred**')	C(**pred**') or C(**pred** and **pred**')
$\underline{\theta}$	the keywords *groupby* or *normalise* (the two possible kinds of SCA)

The substitution-rules are formed by 23 rules as follows:

★ Substitution of $\mathtt{A}^{\mathcal{I}}$ by $\mathtt{A}^{S}$, or $\mathtt{B}^{\mathcal{I}}$ by $\mathtt{A}^{S}$ when $\mathtt{B}^{\mathcal{I}} = \mathtt{A}^{\mathcal{I}}$:

⋆ $\mathtt{A}^{\mathcal{I}}$ *is a property:*

$$\textbf{SR-C1}: \frac{\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\mathtt{A}^{S}}}$$

⋆ $\mathtt{A}^{\mathcal{I}}$ *is a path:*

$$\textbf{SR-C2}: \frac{\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\varrho^{\mathcal{I}}} \Rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{s}}\right] \bullet \underline{\varrho^{S}}}{\begin{array}{c}\underline{\ell^{\mathcal{I}}_{i+1}} \Rightarrow \underline{\varrho^{S}_{i+1}},\ \text{for } 0 \leq i \leq n-1, \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}_{n}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{S}_{n}}\right] \bullet \underline{\boldsymbol{p}^{S}}\end{array}}$$

[1] $\varrho = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \boldsymbol{p}$ or $\varrho = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$, see Definitions 21,22, Chapter 3. In Rules below, $\varrho^{\mathcal{I}} = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \boldsymbol{p}$ or $\varrho^{\mathcal{I}} = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$, while $\varrho^{S} = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{m-1} \bullet \boldsymbol{p}$ or $\varrho^{S} = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{m-1}$, m $\geq$ n.

[2] See Definition 20, Chapter 3.

$$\textbf{SR-C3}: \frac{\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\varrho^{\mathcal{I}}} \Rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{S}}\right] \bullet \underline{\varrho^{S}}}{\underline{\ell^{\mathcal{I}}_{i+1}} \Rightarrow \underline{\varrho^{S}_{i+1}},\ \text{for } 0 \leq i \leq n-1}$$

⋆ $\mathtt{A}^{\mathcal{I}}$ *is a function with n-ary arguments:*

$$\textbf{SR-C4}: \frac{\underline{\varphi}\left(\underline{\mathtt{A}^{\mathcal{I}}_1}, \underline{\mathtt{A}^{\mathcal{I}}_2}, \ldots, \underline{\mathtt{A}^{\mathcal{I}}_n}\right) \Rightarrow \underline{\varphi}\left(\underline{\mathtt{A}^{S}_1}, \underline{\mathtt{A}^{S}_2}, \ldots, \underline{\mathtt{A}^{S}_n}\right)}{\underline{\mathtt{A}^{\mathcal{I}}_i} \Rightarrow \underline{\mathtt{A}^{S}_i}, \text{for } 1 \leq i \leq n}$$

⋆ $\mathtt{A}^{\mathcal{I}}$ *is a property into a structural type:*

$$\textbf{SR-C5}: \frac{\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}}\{\underline{\boldsymbol{p}_i}\} \Rightarrow \underline{\mathtt{A}^{S}_i}}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}}\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_n\} \rightarrow \left(\underline{\mathtt{A}^{S}_1}, \underline{\mathtt{A}^{S}_2}, \ldots, \underline{\mathtt{A}^{S}_n}\right)}$$

★ **Substitution of $\mathtt{B}^{\mathcal{I}}$ by $\mathtt{B}^{S}$, when $\mathtt{B}^{\mathcal{I}}$ is a value:**

$$\textbf{SR-C6}: \underline{\boldsymbol{w}} \Rightarrow \underline{\boldsymbol{w}}$$

★ **Substitution of a predicate by another ($\mathbf{pred}^{\mathcal{I}} \Rightarrow \mathbf{pred}^{S}$):**

$$\textbf{SR-C7}: \frac{\underline{\mathtt{A}^{\mathcal{I}}}\quad \underline{\mathbf{op}}\quad \underline{\mathtt{B}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}\quad \underline{\mathbf{op}}\quad \underline{\mathtt{B}^{S}}}{\begin{array}{l}\underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}, \\ \underline{\mathtt{B}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{B}^{S}}\end{array}}$$

$$\textbf{SR-C8}: \frac{\underline{\mathtt{A}^{\mathcal{I}}}\quad \underline{\mathbf{op}}\quad \underline{\mathtt{B}^{\mathcal{I}}} \text{ and } \underline{\mathbf{pred}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}\quad \underline{\mathbf{op}}\quad \underline{\mathtt{B}^{S}} \text{ and } \underline{\mathbf{pred}^{S}}}{\begin{array}{l}\underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}, \\ \underline{\mathtt{B}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{B}^{S}}, \\ \underline{\mathbf{pred}^{\mathcal{I}}} \Rightarrow \underline{\mathbf{pred}^{S}}\end{array}}$$

$$\textbf{SR-C9}: \frac{\underline{\mathtt{A}^{\mathcal{I}}}\quad \underline{\mathbf{op}}\quad \underline{\mathtt{B}^{\mathcal{I}}} \text{ or } \underline{\mathbf{pred}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}\quad \underline{\mathbf{op}}\quad \underline{\mathtt{B}^{S}} \text{ or } \underline{\mathbf{pred}^{S}}}{\begin{array}{l}\underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}, \\ \underline{\mathtt{B}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{B}^{S}}, \\ \underline{\mathbf{pred}^{\mathcal{I}}} \Rightarrow \underline{\mathbf{pred}^{S}}\end{array}}$$

★ Substitution of a link by a path ($\ell^{\mathcal{I}} \Rightarrow \varrho^{S}$):

⋆ ϱ^{S} *has only a link:*

$$\textbf{SR-C10}: \frac{\underline{\boldsymbol{p}^{\mathcal{I}}} : \underline{\mathrm{C}^{\mathcal{I}}} \multimap \underline{\mathrm{C}_1^{\mathcal{I}}} \Rightarrow \underline{\boldsymbol{p}^{S}} : \underline{\mathrm{C}^{S}} \multimap \underline{\mathrm{C}_1^{S}}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{S}}\right] \bullet \underline{\boldsymbol{p}^{S}}, \\ \underline{\boldsymbol{p}^{S}} : \natural\underline{\mathrm{C}_1^{S}} \in \boldsymbol{type}(\mathrm{C}^{S})\end{array}}$$

$$\textbf{SR-C11}: \frac{\underline{\ell^{\mathcal{I}}} : \underline{\mathrm{C}^{\mathcal{I}}} \multimap \underline{\mathrm{C}_1^{\mathcal{I}}} \Rightarrow \underline{\boldsymbol{FK}^{S}} : \underline{\mathrm{C}^{S}} \multimap \underline{\underline{\mathrm{C}_1^{S}}}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{S}}\right], \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}_1^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}_1^{S}}\right], \\ (\boldsymbol{FK}^{S}, \mathrm{C}^{S}, _, \mathrm{C}_1^{S}, _)\end{array}}$$

$$\textbf{SR-C12}: \frac{\underline{\boldsymbol{FK}^{\mathcal{I}}} : \underline{\mathrm{C}^{\mathcal{I}}} \multimap \underline{\mathrm{C}_1^{\mathcal{I}}} \Rightarrow \underline{\boldsymbol{p}^{S}} : \underline{\mathrm{C}^{S}} \multimap \underline{\underline{\mathrm{C}_1^{S}}}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{S}}\right], \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}_1^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}_1^{S}}\right], \\ \underline{\boldsymbol{p}^{S}} : \natural\underline{\mathrm{C}_1^{S}} \in \boldsymbol{type}(\underline{\mathrm{C}^{S}})\end{array}}$$

⋆ ϱ^{S} *has more than one link:*

$$\textbf{SR-C13}: \frac{\underline{\ell^{\mathcal{I}}} : \underline{\mathrm{C}^{\mathcal{I}}} \multimap \underline{\mathrm{C}_n^{\mathcal{I}}} \Rightarrow \underline{\ell_1^{S}} : \underline{\mathrm{C}^{S}} \multimap \underline{\mathrm{C}_1^{S}} \bullet \underline{\ell_2^{S}} : \underline{\mathrm{C}_1^{S}} \multimap \underline{\mathrm{C}_2^{S}} \bullet \cdots \bullet \underline{\ell_n^{S}} : \underline{\mathrm{C}_{n-1}^{S}} \multimap \underline{\underline{\mathrm{C}_n^{S}}}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{S}}\right], \\ \underline{\ell_1^{S}} : \underline{\mathrm{C}^{S}} \multimap \underline{\mathrm{C}_1^{S}}, \underline{\ell_2^{S}} : \underline{\mathrm{C}_1^{S}} \multimap \underline{\mathrm{C}_2^{S}}, \ldots, \underline{\ell_n^{S}} : \underline{\mathrm{C}_{n-1}^{S}}, \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}_n^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}_n^{S}}\right]\end{array}}$$

★ Substitution of a path by a link ($\varrho^{\mathcal{I}} \Rightarrow \ell^{S}$):

$$\textbf{SR-C14}: \frac{\begin{array}{l}\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\ell^{\mathcal{I}}} : \underline{\mathrm{C}^{\mathcal{I}}} \multimap \underline{\mathrm{C}_1^{\mathcal{I}}} \bullet \underline{\ell_2^{\mathcal{I}}} : \underline{\mathrm{C}_1^{\mathcal{I}}} \multimap \underline{\mathrm{C}_2^{\mathcal{I}}} \bullet \cdots \bullet \underline{\ell_n^{\mathcal{I}}} : \underline{\mathrm{C}_{n-1}^{\mathcal{I}}} \multimap \underline{\mathrm{C}_n^{\mathcal{I}}} \Rightarrow \\ \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{s}}\right] \bullet \underline{\ell^{S}} : \underline{\mathrm{C}^{S}} \multimap \underline{\mathrm{C}_n^{S}}\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}^{S}}\right], \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}_n^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}_n^{S}}\right], \\ \underline{\ell^{S}} : \underline{\mathrm{C}^{S}} \multimap \underline{\mathrm{C}_n^{S}}\end{array}}$$

$$\textbf{SR-C15}: \frac{\begin{array}{l}\mathcal{I}\left[\underline{C^{\mathcal{I}}}\right] \bullet \underline{\ell^{\mathcal{I}}} : \underline{C^{\mathcal{I}}} \rightarrowtriangle \underline{C_1^{\mathcal{I}}} \bullet \underline{\ell_2^{\mathcal{I}}} : \underline{C_1^{\mathcal{I}}} \rightarrowtriangle \underline{C_2^{\mathcal{I}}} \bullet \cdots \bullet \underline{\ell_n^{\mathcal{I}}} : \underline{C_{n-1}^{\mathcal{I}}} \rightarrowtriangle \underline{C_n^{\mathcal{I}}} \bullet \boldsymbol{p}^{\mathcal{I}} \Rightarrow \\ \underline{\mathbf{S}}\left[\underline{C^{s}}\right] \bullet \underline{\ell^{S}} : \underline{C^{S}} \rightarrowtriangle \underline{C_n^{S}} \bullet \boldsymbol{p}^{S}\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{C^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{C^{S}}\right], \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{C_n^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{C_n^{S}}\right], \\ \underline{\ell^{S}} : \underline{C^{S}} \rightarrowtriangle \underline{C_n^{S}}, \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{C_1^{\mathcal{I}}}\right] \bullet \boldsymbol{p}^{\mathcal{I}} \rightarrow \underline{\mathbf{S}}\left[\underline{C_1^{S}}\right] \bullet \boldsymbol{p}^{S}\end{array}}$$

★ Substitution involving extensions:

These type of substitution-rules can run the following procedure:

Procedure	*create_vRelationFromECA*(CA: $A^{\mathcal{I}} \rightarrow A^{S}$, ViewRelation).
Input	an ECA or a SCA.
Description	Creates a view relation that stores data from classes or relations in A^{S}; Returns the name of this view relation (ViewRelation); If the input is an ECA then it creates an ECA that relates ViewRelation to classes or relations in A^{S} based on $A^{\mathcal{I}} \rightarrow A^{S}$ If the input is a SCA then the ViewRelation will be a initial type formed by properties of aggregation in $A^{\mathcal{I}} \rightarrow A^{S}$ and a SCA is created such that it relates ViewRelation to classes or relations in A^{S} based on $A^{\mathcal{I}} \rightarrow A^{S}$.

⋆ *View relations are not necessary:*

$$\textbf{SR-C16}: \frac{\left(\mathcal{I}\left[\underline{C^{\mathcal{I}}}\right], \underline{\Diamond}\right) \Rightarrow \underline{\mathbf{S}}\left[\underline{E^{S}}\right]}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{C^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{E^{S}}\right]}$$

$$\textbf{SR-C17}: \frac{\left(\mathcal{I}\left[\underline{C^{\mathcal{I}}}\left(\underline{\textsf{pred}^{\mathcal{I}}}\right)\right], \underline{\Diamond}\right) \Rightarrow \underline{\mathbf{S}}\left[\underline{E^{S}}\left(\underline{\textsf{pred}^{\mathcal{I}}}\right)\right]}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{C^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{E^{S}}\right], \\ \underline{\textsf{pred}^{\mathcal{I}}} \Rightarrow \underline{\textsf{pred}^{S}}\end{array}}$$

$$\textbf{SR-C18}: \frac{\left(\mathcal{I}\left[\underline{C^{\mathcal{I}}}\right], \underline{\Diamond}\right) \Rightarrow \underline{\mathbf{S}}\left[\underline{E_1^{S}}\right] \Diamond \underline{\mathbf{S}}\left[\underline{E_2^{S}}\right] \Diamond \ldots \Diamond \underline{\mathbf{S}}\left[\underline{E_n^{S}}\right]}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{C^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{E_1^{S}}\right] \Diamond \underline{\mathbf{S}}\left[\underline{E_2^{S}}\right] \Diamond \ldots \Diamond \underline{\mathbf{S}}\left[\underline{E_n^{S}}\right]}$$

$$\textbf{SR-C19}: \frac{\begin{array}{l}(\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\left(\underline{\textbf{pred}^{\mathcal{I}}}\right)\right], \underline{\Diamond}) \Rightarrow \\ \underline{\mathbf{S}}\left[\underline{\mathrm{E}_1^S}\left(\underline{\textbf{pred}_1^S}\right)\right] \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_2^S}\left(\underline{\textbf{pred}_2^S}\right)\right] \Diamond \ldots \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_n^S}\left(\underline{\textbf{pred}_n^S}\right)\right]\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}_1^S}\right] \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_2^S}\right] \Diamond \ldots \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_n^S}\right] \\ \underline{\textbf{pred}^{\mathcal{I}}} \Rightarrow \underline{\textbf{pred}_i^S}, 1 \leq i \leq n\end{array}}$$

⋆ *View relations are involved:*

$$\textbf{SR-C20}: \frac{(\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right], \underline{\Diamond}) \Rightarrow \mathcal{D}\left[\underline{\mathrm{V}}\right]}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}_1^S}\right] \blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_2^S}\right] \blacklozenge \ldots \blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_n^S}\right], \\ \textit{create_vRelationFromECA}\left(\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}_1^S}\right] \blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_2^S}\right] \blacklozenge \ldots \blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_n^S}\right], \mathcal{D}\left[\underline{\mathrm{V}}\right]\right)\end{array}}$$

$$\textbf{SR-C21}: \frac{(\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\left(\underline{\textbf{pred}^{\mathcal{I}}}\right)\right], \underline{\Diamond}) \Rightarrow \mathcal{D}\left[\underline{\mathrm{V}}\right]}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}_1^S}\right] \blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_2^S}\right] \blacklozenge \ldots \blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_n^S}\right], \\ \left(\underline{\textbf{pred}^{\mathcal{I}}}\right) \Rightarrow \left(\underline{\textbf{pred}^S}_i\right), \\ \textit{create_vRelationFromECA}\left(\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\left(\underline{\textbf{pred}^{\mathcal{I}}}\right)\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}_1^S}\left(\underline{\textbf{pred}^S}_1\right)\right] \blacklozenge\right. \\ \left.\blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_2^S}\left(\underline{\textbf{pred}^S}_2\right)\right] \blacklozenge \ldots \blacklozenge \underline{\mathbf{S}}\left[\underline{\mathrm{E}_n^S}\left(\underline{\textbf{pred}^S}_n\right)\right], \mathcal{D}\left[\underline{\mathrm{V}}\right]\right)\end{array}}$$

$$\textbf{SR-C22}: \frac{(\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right], \underline{\Diamond}) \Rightarrow \underline{\mathcal{D}}\left[\underline{\mathrm{V}}\right]}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right]\left(\boldsymbol{p}_1{}^{\mathcal{I}}, \ldots, \boldsymbol{p}_n{}^{\mathcal{I}}\right) \rightarrow \theta\left(\underline{\mathbf{S}}\left[\underline{\mathrm{E}^S}\right]\left(\boldsymbol{p}_1{}^S, \ldots, \boldsymbol{p}_n{}^S\right)\right), \\ \textit{create_vRelationFromECA}\left(\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right]\left(\boldsymbol{p}_1{}^{\mathcal{I}}, \ldots, \boldsymbol{p}_n{}^{\mathcal{I}}\right) \rightarrow\right. \\ \left.\theta\left(\underline{\mathbf{S}}\left[\underline{\mathrm{E}^S}\right]\left(\boldsymbol{p}_1{}^S, \ldots, \boldsymbol{p}_n{}^S\right)\right), \mathcal{D}\left[\underline{\mathrm{V}}\right]\right)\end{array}}$$

$$\textbf{SR-C23}: \frac{(\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\left(\underline{\textbf{pred}^{\mathcal{I}}}\right)\right], \underline{\Diamond}) \Rightarrow \underline{\mathcal{D}}\left[\underline{\mathrm{V}}\right]}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right]\left(\boldsymbol{p}_1{}^{\mathcal{I}}, \ldots, \boldsymbol{p}_n{}^{\mathcal{I}}\right) \rightarrow \theta\left(\underline{\mathbf{S}}\left[\underline{\mathrm{E}^S}\right]\left(\boldsymbol{p}_1{}^S, \ldots, \boldsymbol{p}_n{}^S\right)\right), \\ \left(\underline{\textbf{pred}^{\mathcal{I}}}\right) \Rightarrow \left(\underline{\textbf{pred}^S}\right), \\ \textit{create_vRelationFromECA}\left(\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\left(\underline{\textbf{pred}^{\mathcal{I}}}\right)\right]\left(\boldsymbol{p}_1{}^{\mathcal{I}}, \ldots, \boldsymbol{p}_n{}^{\mathcal{I}}\right) \rightarrow\right. \\ \left.\theta\left(\underline{\mathbf{S}}\left[\underline{\mathrm{E}^S}\left(\underline{\textbf{pred}^S}\right)\right]\left(\boldsymbol{p}_1{}^S, \ldots, \boldsymbol{p}_n{}^S\right)\right), \mathcal{D}\left[\underline{\mathrm{V}}\right]\right)\end{array}}$$

A.2 Rewritten-Rules to Rewrite CAs

The rules to rewrite CAs are subdivided in four groups in accordance to the type of CA involved. Thus, there are rules for rewriting PCAs, ECAs, SCAs and ACAs, which are presented in following text.

A.2.1 Rewritten-rules to rewrite PCAs

The following variables will be used in RR-PCAs:

Variable	Can be instantiated with
$\underline{\mathtt{G}}^S$	the right side of a CA pattern expression of property consisting of one of two forms: ($\mathtt{A}_1^S$,$\mathtt{A}_2^S$, ...,$\mathtt{A}_n^S$) or ($\mathtt{B}_1^S$,$\textbf{pred}_1^S$),($\mathtt{B}_2^S$,$\textbf{pred}_2^S$),...,($\mathtt{B}_{n-1}^S$,$\textbf{pred}_{n-1}^S$),$\mathtt{B}_n^S$.
$\underline{\mathtt{K}}$	right side of an ECA pattern expression.

These type of rule can run one of the following procedures:

Procedure	*thereisNOTvRelation*(PCA).
Input	a PCA.
Description	returns true when a view relation was created when the ECA assigned to PCA was analysed, otherwise it returns false.

Procedure	*addProp_vRelationFromECA*(PCA,ViewExp).
Input	a PCA.
Description	from the PCA, tries to find the view relation that was created when the ECA assigned to the PCA was analysed; adds as properties in the view relation, the properties and the paths of the intermediary; creates PCAs/ACAs relating properties of the view relation to properties of classes or relations of the origin; returns ViewExp, which is an expression that represents the right-side of a PCA containing only elements of the view. This expression is built based on the PCA of the input.

The RR-PCAs are formed by 15 rules as follows:

★ PCAs are rewritten in other PCAs and no view relation is involved:

⋆ *The property, path, property into a structural type, or function with n-ary arguments is rewritten in another property, path, property into a structural type, or function with n-ary arguments:*

$$\textbf{RR-PCA1}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathtt{A}^{S}}}{\begin{array}{l} \textit{thereisNOTvRelation}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathtt{A}^{\mathcal{I}}}\right), \\ \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}} \end{array}}$$

⋆ *The property is rewritten in a conditional expression or an expression of form* ($\mathtt{A}_1$,...,$\mathtt{A}_n$)*:*

$$\textbf{RR-PCA2}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathtt{G}^{S}}}{\begin{array}{l} \mathit{thereisNOTvRelation}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}}\right), \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\mathtt{G}^{S}} \end{array}}$$

⋆ *The property is tuple equivalent to* $\mathtt{A}_1$,...,$\mathtt{A}_n$, *and each* $\mathtt{A}_i$ *is rewritten in other* $\mathtt{A}_i$*:*

$$\textbf{RR-PCA3}: \frac{\begin{array}{c} \mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, ..., \underline{\boldsymbol{p}_n}\} \rightarrow \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, ..., \underline{\boldsymbol{p}_n}\} \rightarrow \left(\underline{\mathtt{A}_1^{S}}, \underline{\mathtt{A}_2^{S}}, \ldots, \underline{\mathtt{A}_n^{S}}\right) \end{array}}{\begin{array}{l} \mathit{thereisNOTvRelation}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, ..., \underline{\boldsymbol{p}_n}\} \rightarrow \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right)\right), \\ \underline{\mathtt{A}_i^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}_i^{S}}, \text{ for } 1 \leq i \leq n \end{array}}$$

⋆ *The property is set equivalent to* $\mathtt{A}_1$,...,$\mathtt{A}_n$, *and each* $\mathtt{A}_i$ *is rewritten in other* $\mathtt{A}_i$*:*

$$\textbf{RR-PCA4}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{A}_1^{S}}, \underline{\mathtt{A}_2^{S}}, \ldots, \underline{\mathtt{A}_n^{S}}\right)}{\begin{array}{l} \mathit{thereisNOTvRelation}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right)\right), \\ \underline{\mathtt{A}_i^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}_i^{S}}, \text{ for } 1 \leq i \leq n \end{array}}$$

⋆ *The conditional expression is rewritten in another conditional expression:*

$$\textbf{RR-PCA5}: \frac{\begin{array}{c} \mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{B}_1^{\mathcal{I}}}, \underline{\textsf{pred}_1^{\mathcal{I}}}\right); \left(\underline{\mathtt{B}_2^{\mathcal{I}}}, \underline{\textsf{pred}_2^{\mathcal{I}}}\right); ...; \left(\underline{\mathtt{B}_{n-1}^{\mathcal{I}}}, \underline{\textsf{pred}_{n-1}^{\mathcal{I}}}\right); \underline{\mathtt{B}_n^{\mathcal{I}}} \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{B}_1^{S}}, \underline{\textsf{pred}_1^{S}}\right); \left(\underline{\mathtt{B}_2^{S}}, \underline{\textsf{pred}_2^{S}}\right); ...; \left(\underline{\mathtt{B}_{n-1}^{S}}, \underline{\textsf{pred}_{n-1}^{S}}\right); \underline{\mathtt{B}_n^{S}} \end{array}}{\begin{array}{l} \mathit{thereisNOTvRelation}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{B}_1^{\mathcal{I}}}, \underline{\textsf{pred}_1^{\mathcal{I}}}\right); ...; \left(\underline{\mathtt{B}_{n-1}^{\mathcal{I}}}, \underline{\textsf{pred}_{n-1}^{\mathcal{I}}}\right); \underline{\mathtt{B}_n^{\mathcal{I}}}\right), \\ \underline{\mathtt{B}_i^{\mathcal{I}}} \Rightarrow \underline{\mathtt{B}_i^{S}}, \\ \underline{\textsf{pred}_i^{\mathcal{I}}} \Rightarrow \underline{\textsf{pred}_i^{S}}, \text{ for } 1 \leq i \leq n \end{array}}$$

★ PCAs are rewritten in ACAs and no view relation is involved:

⋆ *The property, path, property into a structural type, or function with n-ary arguments is rewritten in another property, path, property into a structural type, or function with n-ary arguments present in an ACA:*

$$\textbf{RR-PCA6}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{A}^{S}}}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{A}^{S}}}$$

⋆ *The property is rewritten in a conditional expression or an expression of form* ($\mathtt{A}_1$,...,$\mathtt{A}_n$) *present in an ACA:*

$$\textbf{RR-PCA7}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\textbf{sca}}, \underline{\mathtt{G}^{S}}}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\textbf{sca}}, \underline{\mathtt{G}^{S}}}$$

⋆ *Each argument in a function n-ary in a PCA is rewritten in another argument in the same function n-ary in an ACA:*

$$\textbf{RR-PCA8}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \varphi\left(\mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}_1^{\mathcal{I}}}, \ldots, \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\textbf{sca}}, \varphi\left(\underline{\mathtt{A}_1^{S}, \ldots, \mathtt{A}_n^{S}}\right)\end{array}}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\textbf{sca}}, \underline{\mathtt{A}_i^{S}}, \text{ for } 1 \leq i \leq n}$$

⋆ *The property into a structural type is rewritten in another property, path, property into a structural type, or function with n-ary arguments present in an ACA:*

$$\textbf{RR-PCA9}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}}\{\underline{\boldsymbol{p}_i}\} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\textbf{sca}}, \underline{\mathtt{A}_i^{S}}}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\} \rightarrow \underline{\textbf{sca}}, \left(\underline{\mathtt{A}_1^{S}}, \underline{\mathtt{A}_2^{S}}, \ldots, \underline{\mathtt{A}_n^{S}}\right)}$$

⋆ *The property is rewritten in another property, path, property into a structural type, or function with n-ary arguments present in an ACA of group by:*

$$\textbf{RR-PCA10}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\textbf{sca}}, \gamma\left(\underline{\mathtt{A}^{S}}\right)}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\textbf{sca}}, \gamma\left(\underline{\mathtt{A}^{S}}\right)}$$

$$\textbf{RR-PCA11}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\textbf{sca}}, \gamma\left(\underline{\mathtt{A}^{S}}, \textsf{\textbf{pred}}^{S}\right)}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\textbf{sca}}, \gamma\left(\underline{\mathtt{A}^{S}}, \textsf{\textbf{pred}}^{S}\right)}$$

★ **Rewritten rules of PCA that consider view relations:**

⋆ *The property is rewritten in another property or function with n-ary arguments of a view relation:*

$$\textbf{RR-PCA12}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathtt{A}}}{\textit{addProp_vRelationFromECA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathtt{A}^{\mathcal{I}}}, \mathtt{A}\right)}$$

⋆ *The property is tuple equivalent to* $\mathtt{A}_1,...,\mathtt{A}_n$*, and each* $\mathtt{A}_i$ *is rewritten in other* $\mathtt{A}_i$*, whose elements belong to a view relation:*

$$\textbf{RR-PCA13}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\,[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, ..., \underline{\boldsymbol{p}_n}\} \rightarrow \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, ..., \underline{\boldsymbol{p}_n}\} \rightarrow (\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n})\end{array}}{\begin{array}{l}\mathit{addProp_vRelationFromECA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, ..., \underline{\boldsymbol{p}_n}\} \rightarrow \right. \\ \left.\left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right), (\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n})\right)\end{array}}$$

⋆ *The property is set equivalent to* $\mathtt{A}_1,...,\mathtt{A}_n$*, and each* $\mathtt{A}_i$ *is rewritten in other* $\mathtt{A}_i$*, whose elements belong to a view relation:*

$$\textbf{RR-PCA14}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\,[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow (\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n})\end{array}}{\begin{array}{l}\mathit{addProp_vRelationFromECA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\,[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \right. \\ \left.\left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right), (\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n})\right)\end{array}}$$

⋆ *The conditional expression is rewritten in other conditional expression, whose elements belong to a view relation:*

$$\textbf{RR-PCA15}: \frac{\begin{array}{l}\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{B}_1^{\mathcal{I}}}, \underline{\mathbf{pred}_1^{\mathcal{I}}}\right); \left(\underline{\mathtt{B}_2^{\mathcal{I}}}, \underline{\mathbf{pred}_2^{\mathcal{I}}}\right); ...; \left(\underline{\mathtt{B}_{n-1}^{\mathcal{I}}}, \underline{\mathbf{pred}_{n-1}^{\mathcal{I}}}\right); \underline{\mathtt{B}_n^{\mathcal{I}}} \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{B}_1}, \underline{\mathbf{pred}_1}\right); \left(\underline{\mathtt{B}_2}, \underline{\mathbf{pred}_2}\right); ...; \left(\underline{\mathtt{B}_{n-1}}, \underline{\mathbf{pred}_{n-1}}\right); \\ ; \underline{\mathtt{B}_n}\end{array}}{\begin{array}{l}\mathit{addProp_vRelationFromECA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{C}^{\mathcal{D}}}] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \left(\underline{\mathtt{B}_1^{\mathcal{I}}}, \underline{\mathbf{pred}_1^{\mathcal{I}}}\right); ...\right. \\ \ldots; \left(\underline{\mathtt{B}_{n-1}^{\mathcal{I}}}, \underline{\mathbf{pred}_{n-1}^{\mathcal{I}}}\right); \underline{\mathtt{B}_n^{\mathcal{I}}}, \\ \left.\left(\left(\underline{\mathtt{B}_1}, \underline{\mathbf{pred}_1}\right); \left(\underline{\mathtt{B}_2}, \underline{\mathbf{pred}_2}\right); ...; \left(\underline{\mathtt{B}_{n-1}}, \underline{\mathbf{pred}_{n-1}}\right); \underline{\mathtt{B}_n}\right)\right)\end{array}}$$

A.2.2 Rewritten-rules to rewrite ECAs

The following variables will be used in some RR-ECAs:

Variable	**Can be instantiated with**
$\underline{\mathtt{a}}$	a property, a path expression, or a property in a structural type
$\underline{\mathtt{Q}}$	the right side of a CA pattern expression of summation.

The RR-ECAs are formed by 6 rules as follows:

★ **ECA of equivalence is rewritten in other ECA, maybe of equivalence too:**

$$\textbf{RR-ECA1}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \underline{\mathtt{K}^{S}}}{\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathtt{K}^{S}}}$$

★ **ECA of selection is rewritten in other ECA selection:**

$$\textbf{RR-ECA2}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\left(\underline{\textsf{pred}^{\mathcal{I}}}\right)\right] \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}^{S}}\left(\underline{\textsf{pred}^{S}}\right)\right]}{\begin{array}{c}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}^{S}}\right], \\ \underline{\textsf{pred}^{\mathcal{I}}} \Rightarrow \underline{\textsf{pred}^{S}}\end{array}}$$

★ **ECA of selection is rewritten in other ECA (difference, union or intersection) with selection:**

$$\textbf{RR-ECA3}: \frac{\begin{array}{l}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\left(\underline{\textsf{pred}^{\mathcal{I}}}\right)\right] \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}_{1}^{S}}\left(\underline{\textsf{pred}_{1}^{S}}\right)\right] \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_{2}^{S}}\left(\underline{\textsf{pred}_{2}^{S}}\right)\right] \Diamond \ldots \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_{n}^{S}}\left(\underline{\textsf{pred}_{n}^{S}}\right)\right]\end{array}}{\begin{array}{c}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{E}_{1}^{S}}\right] \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_{2}^{S}}\right] \Diamond \ldots \Diamond \underline{\mathbf{S}}\left[\underline{\mathrm{E}_{n}^{S}}\right], \\ \underline{\textsf{pred}^{\mathcal{I}}} \Rightarrow \underline{\textsf{pred}_{i}^{S}}, \text{ for } 1 \leq i \leq n\end{array}}$$

★ **ECA of union/difference/intersection is rewritten in other ECA (difference, union or intersection), with or without selection; and with or without creation of view relations:**

$$\textbf{RR-ECA4}: \frac{\begin{array}{l}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \mathcal{I}\left[\underline{\mathrm{C}_{1}^{\mathcal{I}}}\right] \Diamond \ldots \Diamond \mathcal{I}\left[\underline{\mathrm{C}_{j}^{\mathcal{I}}}\left(\underline{\textsf{pred}_{j}^{\mathcal{I}}}\right)\right] \Diamond \ldots \Diamond \mathcal{I}\left[\underline{\mathrm{C}_{n}^{\mathcal{I}}}\right] \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \underline{\mathtt{K}_{1}} \Diamond \underline{\mathtt{K}_{2}} \Diamond \ldots \Diamond \underline{\mathtt{K}_{j}} \Diamond \ldots \Diamond \underline{\mathtt{K}_{n}}\end{array}}{\begin{array}{l}\left(\mathcal{I}\left[\underline{\mathrm{C}_{i}^{\mathcal{I}}}\right], \Diamond\right) \Rightarrow \underline{\mathtt{K}_{i}^{S}}, \text{ for } 1 \leq i \leq j-1, j+1 \leq i \leq n, \\ \left(\mathcal{I}\left[\underline{\mathrm{C}_{j}^{\mathcal{I}}}\left(\underline{\textsf{pred}_{j}^{\mathcal{I}}}\right)\right], \Diamond\right) \Rightarrow \underline{\mathtt{K}_{j}}\end{array}}$$

★ **ECA of equivalence is rewritten in a SCA (with or without selection):**

$$\textbf{RR-ECA5}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{E}}\right] \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{E}}\right]\left(\underline{\boldsymbol{p}_{1}^{\mathcal{D}}}, \ldots, \underline{\boldsymbol{p}_{n}^{S}}\right) \rightarrow \underline{\mathtt{Q}^{S}}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right]\left(\underline{\boldsymbol{p}_{1}^{\mathcal{I}}}, \ldots, \underline{\boldsymbol{p}_{n}^{\mathcal{I}}}\right) \rightarrow \underline{\mathtt{Q}^{S}}, \\ \mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}_{i}^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}_{i}^{\mathcal{I}}}, \text{ for } 1 \leq i \leq n\end{array}}$$

★ ECA of selection is rewritten in a SCA (with or without selection):

$$\textbf{RR-ECA6}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\,[\underline{\mathrm{E}}] \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}\left(\textbf{pred}^{\mathcal{I}}\right)}\right] \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{E}}]\left(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\right) \rightarrow \\ \underline{\theta}\left(\underline{\mathbf{S}}\left[\underline{\mathrm{E}^S\left(\textbf{pred}^S\right)}\right]\left(\underline{\mathtt{a}_1^S},\ldots,\underline{\varphi}\left(\underline{\mathtt{A}_1^S},\ldots,\underline{\mathtt{A}_n^S}\right),\ldots,\underline{\mathtt{a}_n^S}\right)\right)\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right]\left(\underline{\boldsymbol{p}_1^{\mathcal{I}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{I}}}\right) \rightarrow \underline{\theta}(\underline{\mathbf{S}}\,[\underline{\mathrm{E}^S}]\,(\underline{\mathtt{a}_1^S},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1^S},\ldots,\underline{\mathtt{A}_n^S}),\ldots,\underline{\mathtt{a}_n^S})) \\ \underline{\textbf{pred}^{\mathcal{I}}} \Rightarrow \underline{\textbf{pred}^S}, \\ \mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}_i^{\mathcal{D}}} \rightarrow \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}_i^{\mathcal{I}}},\ \text{for } 1 \leq i \leq n\end{array}}$$

A.2.3 Rewritten-rules to rewrite SCAs

The following procedure will be used in some RR-SCAs:

Procedure	*create_vRelationFromSCA*(SCA: $\mathrm{A}^{\mathcal{D}} \rightarrow \mathrm{A}^{\mathcal{I}}$, ECA: $\mathrm{A}^{\mathcal{I}} \rightarrow \mathrm{A}^S$, ViewRelation, $\{\mathtt{A}_1, ..., \mathtt{A}_n\}$).
Input	an SCA from the destination to the intermediary and an ECA from the intermediary to the origin.
Description	Creates a view relation (ViewRelation) that stores data from classes or relations in A^S; ViewRelation will have a initial type formed by properties (or paths) of aggregation in $\mathrm{A}^{\mathcal{D}} \rightarrow \mathrm{A}^{\mathcal{I}}$; an ECA is created such that it relates ViewRelation to classes or relations in A^S based on $\mathrm{A}^{\mathcal{I}} \rightarrow \mathrm{A}^S$, and one or more PCAs are created relating properties of ViewRelation to properties or paths of classes or relations in A^S. Returns the name of the view relation (ViewRelation), as well as the set of expressions $\{\mathtt{A}_1, ..., \mathtt{A}_n\}$ formed by properties of ViewRelations. These expressions are built based on $\mathrm{A}^{\mathcal{D}} \rightarrow \mathrm{A}^{\mathcal{I}}$ and will be used in the rewritten of the new SCA.

The RR-SCAs are formed by 4 rules as follows:

★ View relations are not involved:

⋆ *SCA (without predicate in the intermediary) is rewritten in other SCA:*

$$\textbf{RR-SCA1}: \frac{\begin{array}{l}\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{E}}]\Big(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\Big) \rightarrow \underline{\theta}(\mathcal{I}[\underline{\mathrm{C}^{\mathcal{I}}}](\underline{\mathtt{a}_1^{\mathcal{I}}},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_m^{\mathcal{I}}}),\ldots,\underline{\mathtt{a}_n^{\mathcal{I}}}))\Rightarrow\\ \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{E}}]\,\Big(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\Big) \rightarrow \underline{\theta}(\underline{\mathbf{S}}\,[\underline{\mathrm{C}^{S}}]\,(\underline{\mathtt{a}_1^{S}},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1^{S}},\ldots,\underline{\mathtt{A}_m^{S}}),\ldots,\underline{\mathtt{a}_n^{S}}))\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\,[\underline{\mathrm{C}^{\mathcal{I}}}] \rightarrow \underline{\mathbf{S}}\,[\underline{\mathrm{C}^{S}}]\,,\\ \mathcal{I}\,[\underline{\mathrm{C}^{\mathcal{I}}}]\bullet\underline{\mathtt{a}_i^{\mathcal{I}}} \Rightarrow \underline{\mathbf{S}}\,[\underline{\mathrm{C}^{S}}]\bullet\underline{\mathtt{a}_i^{S}},\ \text{for } 1\le i\le n,\\ \underline{\varphi}(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_m^{\mathcal{I}}})\Rightarrow \underline{\varphi}(\underline{\mathtt{A}_1^{S}},\ldots,\underline{\mathtt{A}_m^{S}})\end{array}}$$

⋆ *SCA (with predicate in the intermediary) is rewritten in other SCA:*

$$\textbf{RR-SCA2}: \frac{\begin{array}{l}\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{E}}]\Big(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\Big) \rightarrow \underline{\theta}(\mathcal{I}[\underline{\mathrm{C}^{\mathcal{I}}}\big(\underline{\textsf{pred}^{\mathcal{I}}}\big)](\underline{\mathtt{a}_1^{\mathcal{I}}},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_n^{\mathcal{I}}}),\ldots,\\ \ldots,\underline{\mathtt{a}_n^{\mathcal{I}}})) \quad \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{E}}]\,\Big(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\Big) \rightarrow\\ \underline{\theta}(\underline{\mathbf{S}}\,[\underline{\mathrm{C}^{S}}\,\big(\underline{\textsf{pred}^{S}}\big)]\,(\underline{\mathtt{a}_1^{S}},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1^{S}},\ldots,\underline{\mathtt{A}_n^{S}}),\ldots,\underline{\mathtt{a}_n^{S}}))\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\,[\underline{\mathrm{C}^{\mathcal{I}}}] \rightarrow \underline{\mathbf{S}}\,[\underline{\mathrm{C}^{S}}]\,,\\ \underline{\textsf{pred}^{\mathcal{I}}} \Rightarrow \underline{\textsf{pred}^{S}},\\ \mathcal{I}\,[\underline{\mathrm{C}^{\mathcal{I}}}]\bullet\underline{\mathtt{a}_i^{\mathcal{I}}} \Rightarrow \underline{\mathbf{S}}\,[\underline{\mathrm{C}^{S}}]\bullet\underline{\mathtt{a}_i^{S}},\ \text{for } 1\le i\le n,\\ \underline{\varphi}(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_n^{\mathcal{I}}})\Rightarrow \underline{\varphi}(\underline{\mathtt{A}_1^{S}},\ldots,\underline{\mathtt{A}_n^{S}})\end{array}}$$

★ View relations are involved:

⋆ *SCA (without predicate in the intermediary) is rewritten in other SCA:*

$$\textbf{RR-SCA3}: \frac{\begin{array}{l}\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{E}}]\Big(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\Big) \rightarrow \underline{\theta}(\mathcal{I}[\underline{\mathrm{C}^{\mathcal{I}}}](\underline{\mathtt{a}_1^{\mathcal{I}}},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_m^{\mathcal{I}}}),\ldots,\underline{\mathtt{a}_n^{\mathcal{I}}}))\Rightarrow\\ \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{E}}]\,\Big(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\Big) \rightarrow \underline{\theta}(\mathcal{D}\,[\underline{\mathrm{V}}]\,(\underline{\mathtt{a}_1},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1},\ldots,\underline{\mathtt{A}_m}),\ldots,\underline{\mathtt{a}_n}))\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\,[\underline{\mathrm{C}^{\mathcal{I}}}] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}_1^{S}}\right]\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_2^{S}}\right]\Diamond\ldots\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_q^{S}}\right],\\ \textit{create_vRelationFromSCA}\,\Big(\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{E}}]\Big(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\Big)\rightarrow\\ \underline{\theta}\,\Big(\mathcal{I}[\underline{\mathrm{C}^{\mathcal{I}}}]\Big(\underline{\mathtt{a}_1^{\mathcal{I}}},\ldots,\underline{\varphi}\,\Big(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_m^{\mathcal{I}}}\Big),\ldots,\underline{\mathtt{a}_n^{\mathcal{I}}}\Big)\Big),\\ \mathbf{P}_{\underline{X}|\mathcal{I}}\,[\underline{\mathrm{C}^{\mathcal{I}}}] \rightarrow \underline{\mathbf{S}}\left[\underline{\mathrm{C}_1^{S}}\right]\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_2^{S}}\right]\Diamond\ldots\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_q^{S}}\right],\\ \mathcal{D}\,[\underline{\mathrm{V}}]\,,\{\underline{\mathtt{a}_1},\ldots,\underline{\varphi}\,(\underline{\mathtt{A}_1},\ldots,\underline{\mathtt{A}_m})\,,\ldots,\underline{\mathtt{a}_n}\})\end{array}}$$

⋆ *SCA (with predicate in the intermediary) is rewritten in other SCA:*

$$\textbf{RR-SCA4}: \frac{\begin{array}{l}\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{E}}]\left(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\right)\rightarrow\underline{\theta}(\mathcal{I}[\underline{\mathrm{C}^{\mathcal{I}}}\,(\mathbf{pred}^{\mathcal{I}})](\underline{\mathtt{a}_1^{\mathcal{I}}},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_m^{\mathcal{I}}}),\ldots,\underline{\mathtt{a}_n^{\mathcal{I}}}))\Rightarrow\\ \mathbf{P}_{\underline{X}|\mathcal{D}}\,[\underline{\mathrm{E}}]\left(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\right)\rightarrow\underline{\theta}(\mathcal{D}\,[\underline{\mathrm{V}}]\,(\underline{\mathtt{a}_1},\ldots,\underline{\varphi}(\underline{\mathtt{A}_1},\ldots,\underline{\mathtt{A}_m}),\ldots,\underline{\mathtt{a}_n}))\end{array}}{\begin{array}{l}\mathbf{P}_{\underline{X}|\mathcal{I}}\,[\underline{\mathrm{C}^{\mathcal{I}}}]\rightarrow\underline{\mathbf{S}}\left[\underline{\mathrm{C}_1^S}\right]\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_2^S}\right]\Diamond\ldots\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_q^S}\right],\\ \textit{create_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}[\underline{\mathrm{E}}]\left(\underline{\boldsymbol{p}_1^{\mathcal{D}}},\ldots,\underline{\boldsymbol{p}_n^{\mathcal{D}}}\right)\rightarrow\right.\\ \underline{\theta}\left(\mathcal{I}[\underline{\mathrm{C}^{\mathcal{I}}}\,(\mathbf{pred}^{\mathcal{I}})]\left(\underline{\mathtt{a}_1^{\mathcal{I}}},\ldots,\underline{\varphi}\left(\underline{\mathtt{A}_1^{\mathcal{I}}},\ldots,\underline{\mathtt{A}_m^{\mathcal{I}}}\right),\ldots,\underline{\mathtt{a}_n^{\mathcal{I}}}\right)\right),\\ \mathbf{P}_{\underline{X}|\mathcal{I}}\,[\underline{\mathrm{C}^{\mathcal{I}}}]\rightarrow\underline{\mathbf{S}}\left[\underline{\mathrm{C}_1^S}\right]\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_2^S}\right]\Diamond\ldots\Diamond\underline{\mathbf{S}}\left[\underline{\mathrm{C}_q^S}\right],\\ \left.\mathcal{D}\,[\underline{\mathrm{V}}]\,,\{\underline{\mathtt{a}_1},\ldots,\underline{\varphi}\left(\underline{\mathtt{A}_1},\ldots,\underline{\mathtt{A}_m}\right),\ldots,\underline{\mathtt{a}_n}\}\right)\end{array}}$$

A.2.4 Rewritten-rules to rewrite ACAs

The following variables will be used in RR-ACAs:

Variable	**Can be instantiated with**
$\underline{\gamma}$	an aggregation function (sum, count, min, max, avg).
sca	the name of the respective SCA assigned to an ACA.

The following procedures can be used in some of rules RR-ACAs:

Procedure	*thereisNOTvRelation*(scaName)
Input	the name of the sca assigned to an ACA
Description	returns true when a view relation was created when the SCA assigned to ACA was analysed, otherwise returns false.

Procedure	*retrieve_vRelation*(scaName,ViewRelation)
Input	the name of the sca assigned to an ACA and the view relation created when this SCA was analysed.
Description	Gets the view Relation (ViewRelation) that was created when the SCA (scaName) was analysed.

Procedure	*addProp_vRelationFromSCA*(ACA,ViewRelation,ViewExp)
Input	an ACA and the name of the view relation created when SCA assigned to ACA was analysed
Description	adds as properties in the view relation, the properties and the paths of the classes or relations of the intermediary present in the ACA; creates ACAs relating properties of the view relation to properties of classes or relations of the origin; returns ViewExp, which is an expression that represent the right-side of an ACA containing only elements of the view relation. This expression is built based on ACA of the input.

The RR-ACAs are formed by 14 rules as follows:

★ **ACA of normalisation is rewritten in other ACA of normalisation when view relations are not involved:**

⋆ *The property, path, property into a structural type, or function with n-ary arguments is rewritten in another property, path, property into a structural type, or function with n-ary arguments:*

$$\textbf{RR-ACA1}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{A}^{S}}}{\begin{array}{l} thereisNOTvRelation\left(\underline{\mathbf{sca}}\right), \\ \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}} \end{array}}$$

⋆ *The property is rewritten in a conditional expression or an expression of form ($\mathtt{A}_1$,...,$\mathtt{A}_n$):*

$$\textbf{RR-ACA2}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \mathcal{I}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{G}^{S}}}{\begin{array}{l} thereisNOTvRelation\left(\underline{\mathbf{sca}}\right), \\ \mathbf{P}_{\underline{X}|\mathcal{I}}\left[\underline{\mathrm{C}^{\mathcal{I}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{I}}} \rightarrow \underline{\mathtt{G}^{S}} \end{array}}$$

⋆ *The property is tuple equivalent to $\mathtt{A}_1$,...,$\mathtt{A}_n$, and each $\mathtt{A}_i$ is rewritten in other $\mathtt{A}_i$:*

$$\textbf{RR-ACA3}: \frac{\begin{array}{c} \mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{S}}, \underline{\mathtt{A}_2^{S}}, \ldots, \underline{\mathtt{A}_n^{S}}\right) \end{array}}{\begin{array}{l} thereisNOTvRelation\left(\underline{\mathbf{sca}}\right), \\ \underline{\mathtt{A}_i^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}_i^{S}}, \text{ for } 1 \leq i \leq n \end{array}}$$

⋆ *The property is set equivalent to* $\mathtt{A}_1,...,\mathtt{A}_n$*, and each* $\mathtt{A}_i$ *is rewritten in other* $\mathtt{A}_i$*:*

$$\mathbf{RR\text{-}ACA4}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{S}}, \underline{\mathtt{A}_2^{S}}, \ldots, \underline{\mathtt{A}_n^{S}}\right)\end{array}}{\begin{array}{l} thereisNOTvRelation\left(\underline{\mathbf{sca}}\right), \\ \underline{\mathtt{A}_i^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}_i^{S}},\ \text{for } 1 \leq i \leq n \end{array}}$$

⋆ *The conditional expression is rewritten in another conditional expression:*

$$\mathbf{RR\text{-}ACA5}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{B}_1^{\mathcal{I}}}, \underline{\mathsf{pred}_1^{\mathcal{I}}}\right); \ldots; \left(\underline{\mathtt{B}_{n-1}^{\mathcal{I}}}, \underline{\mathsf{pred}_{n-1}^{\mathcal{I}}}\right); \underline{\mathtt{B}_n^{\mathcal{I}}} \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{B}_1^{S}}, \underline{\mathsf{pred}_1^{S}}\right); \ldots; \left(\underline{\mathtt{B}_{n-1}^{S}}, \underline{\mathsf{pred}_{n-1}^{S}}\right); \underline{\mathtt{B}_n^{S}}\end{array}}{\begin{array}{l} thereisNOTvRelation\left(\underline{\mathbf{sca}}\right), \\ \underline{\mathtt{B}_i^{\mathcal{I}}} \Rightarrow \underline{\mathtt{B}_i^{S}}, \\ \underline{\mathsf{pred}_i^{\mathcal{I}}} \Rightarrow \underline{\mathsf{pred}_i^{S}},\ \text{for } 1 \leq i \leq n \end{array}}$$

★ ACA of groupby is rewritten in other ACA of groupby when view relations are not involved:

⋆ *There is no predicate in the intermediary*

$$\mathbf{RR\text{-}ACA6}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{\mathcal{I}}}\right) \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{S}}\right)}{\begin{array}{l} thereisNOTvRelation\left(\underline{\mathbf{sca}}\right), \\ \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}} \end{array}}$$

⋆ *There is a predicate in the intermediary*

$$\mathbf{RR\text{-}ACA7}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{\mathcal{I}}}, \underline{\mathsf{pred}^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{S}}, \underline{\mathsf{pred}^{S}}\right)\end{array}}{\begin{array}{l} thereisNOTvRelation\left(\underline{\mathbf{sca}}\right), \\ \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \underline{\mathtt{A}^{S}}, \\ \underline{\mathsf{pred}^{\mathcal{I}}} \Rightarrow \underline{\mathsf{pred}^{S}} \end{array}}$$

★ ACA of normalisation is rewritten in another ACA of normalisation when view relations are involved:

⋆ *The property, path, property into a structural type, or function with n-ary arguments is rewritten in another property, path, property into a structural type, or function with n-ary arguments:*

$$\textbf{RR-ACA8}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{A}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{A}}}{\begin{array}{l} \textit{retrieve_vRelation}\left(\underline{\mathbf{sca}}, \mathcal{D}\left[\underline{\mathrm{V}}\right]\right), \\ \textit{addProp_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{A}^{\mathcal{I}}}, \mathcal{D}\left[\underline{\mathrm{V}}\right], \{\underline{\mathtt{A}}\}\right) \end{array}}$$

⋆ *The property is rewritten in a conditional expression or an expression of form ($\mathtt{A}_1$,...,$\mathtt{A}_n$):*

$$\textbf{RR-ACA9}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{G}^{\mathcal{I}}} \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{G}}}{\begin{array}{l} \textit{retrieve_vRelation}\left(\underline{\mathbf{sca}}, \mathcal{D}\left[\underline{\mathrm{V}}\right]\right), \\ \textit{addProp_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\mathtt{G}^{\mathcal{I}}}, \mathcal{D}\left[\underline{\mathrm{V}}\right], \{\underline{\mathtt{G}}\}\right) \end{array}}$$

⋆ *The property is tuple equivalent to $\mathtt{A}_1$,...,$\mathtt{A}_n$, and each $\mathtt{A}_i$ is rewritten in other $\mathtt{A}_i$:*

$$\textbf{RR-ACA10}: \frac{\begin{array}{c} \mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n}\right) \end{array}}{\begin{array}{l} \textit{retrieve_vRelation}\left(\underline{\mathbf{sca}}, \mathcal{D}\left[\underline{\mathrm{V}}\right]\right), \\ \textit{addProp_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}}\{\underline{\boldsymbol{p}_1}, \underline{\boldsymbol{p}_2}, \ldots, \underline{\boldsymbol{p}_n}\} \rightarrow \right. \\ \left. \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right), \mathcal{D}\left[\underline{\mathrm{V}}\right], \{\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n}\}\right) \end{array}}$$

⋆ *The property is set equivalent to $\mathtt{A}_1$,...,$\mathtt{A}_n$, and each $\mathtt{A}_i$ is rewritten in other $\mathtt{A}_i$:*

$$\textbf{RR-ACA11}: \frac{\begin{array}{c} \mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n}\right) \end{array}}{\begin{array}{l} \textit{retrieve_vRelation}\left(\underline{\mathbf{sca}}, \mathcal{D}\left[\underline{\mathrm{V}}\right]\right), \\ \textit{addProp_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{A}_1^{\mathcal{I}}}, \underline{\mathtt{A}_2^{\mathcal{I}}}, \ldots, \underline{\mathtt{A}_n^{\mathcal{I}}}\right), \right. \\ \left. \mathcal{D}\left[\underline{\mathrm{V}}\right], \{\underline{\mathtt{A}_1}, \underline{\mathtt{A}_2}, \ldots, \underline{\mathtt{A}_n}\}\right) \end{array}}$$

⋆ *The conditional expression is rewritten in another conditional expression:*

$$\textbf{RR-ACA12}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{B}_1^{\mathcal{I}}}, \underline{\mathsf{\mathbf{pred}}_1^{\mathcal{I}}}\right); \ldots; \left(\underline{\mathtt{B}_{n-1}^{\mathcal{I}}}, \underline{\mathsf{\mathbf{pred}}_{n-1}^{\mathcal{I}}}\right); \underline{\mathtt{B}_n^{\mathcal{I}}} \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \left(\underline{\mathtt{B}_1}, \underline{\mathsf{\mathbf{pred}}_1}\right); \ldots; \left(\underline{\mathtt{B}_{n-1}}, \underline{\mathsf{\mathbf{pred}}_{n-1}}\right); \underline{\mathtt{B}_n}\end{array}}{\begin{array}{l} \textit{retrieve_vRelation}\left(\underline{\mathbf{sca}}, \mathcal{D}\left[\underline{\mathrm{V}}\right]\right), \\ \textit{addProp_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}},\right. \\ \left(\underline{\mathtt{B}_1^{\mathcal{I}}}, \underline{\mathsf{\mathbf{pred}}_1^{\mathcal{I}}}\right); \ldots; \left(\underline{\mathtt{B}_{n-1}^{\mathcal{I}}}, \underline{\mathsf{\mathbf{pred}}_{n-1}^{\mathcal{I}}}\right); \underline{\mathtt{B}_n^{\mathcal{I}}}, \mathcal{D}\left[\underline{\mathrm{V}}\right], \\ \left.\left\{\left(\underline{\mathtt{B}_1}, \underline{\mathsf{\mathbf{pred}}_1}\right); \ldots; \left(\underline{\mathtt{B}_{n-1}}, \underline{\mathsf{\mathbf{pred}}_{n-1}}\right); \underline{\mathtt{B}_n}\right\}\right) \end{array}}$$

★ ACA of groupby is rewritten in other ACA of groupby when view relations are involved:

⋆ *There is no predicate in the intermediary*

$$\textbf{RR-ACA13}: \frac{\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{\mathcal{I}}}\right) \Rightarrow \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}}\right)}{\begin{array}{l} \textit{retrieve_vRelation}\left(\underline{\mathbf{sca}}, \mathcal{D}\left[\underline{\mathrm{V}}\right]\right), \\ \textit{addProp_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{\mathcal{I}}}\right), \mathcal{D}\left[\underline{\mathrm{V}}\right],\right. \\ \left.\left\{\underline{\mathtt{A}}\right\}\right) \end{array}}$$

⋆ *There is a predicate in the intermediary*

$$\textbf{RR-ACA14}: \frac{\begin{array}{c}\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{\mathcal{I}}}, \underline{\mathsf{\mathbf{pred}}^{\mathcal{I}}}\right) \Rightarrow \\ \mathbf{P}_{\underline{X}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}}, \underline{\mathsf{\mathbf{pred}}}\right)\end{array}}{\begin{array}{l} \textit{retrieve_vRelation}\left(\underline{\mathbf{sca}}, \mathcal{D}\left[\underline{\mathrm{V}}\right]\right), \\ \textit{addProp_vRelationFromSCA}\left(\mathbf{P}_{\mathsf{I}|\mathcal{D}}\left[\underline{\mathrm{C}^{\mathcal{D}}}\right] \bullet \underline{\boldsymbol{p}^{\mathcal{D}}} \rightarrow \underline{\mathbf{sca}}, \underline{\gamma}\left(\underline{\mathtt{A}^{\mathcal{I}}}, \underline{\mathsf{\mathbf{pred}}^{\mathcal{I}}}\right),\right. \\ \left.\mathcal{D}\left[\underline{\mathrm{V}}\right], \left\{\underline{\mathtt{A}}, \underline{\mathsf{\mathbf{pred}}}\right\}\right) \end{array}}$$

B

Appendix

Here we present the basic correspondence between our formalism and the Prolog notation used in our proof-of-concept.

B.1 Correspondence between the Language $\mathbf{L}_S$ and the Prolog Notation

Namespace

Formalism	**Prolog notation**
$\mathcal{P}$ (set of property names)	propertynames($\boldsymbol{P}$)
$\mathcal{R}$ (set of relation names)	relationnames(R)
$\mathcal{C}$ (set of class names)	classnames(C)
$\mathcal{K}$ (set of key names)	keynames($\boldsymbol{K}$)
$\mathcal{L}$ (set of schema names)	schemanames(**S**)

Legend: $\boldsymbol{P}$ is the name of a property, R is the name of a relation, C is the name of a class, $\boldsymbol{K}$ is the name of a key or of a foreign key, and **S** is the name of a schema.

Data types

Formalism	**Prolog notation**
integer	int
float	float
string	text
date	date
boolean	bool
reference	ref(**S**,C)

Continued on Next Page...

continuation ...

Formalism	Prolog notation
structural	struct$([(\boldsymbol{P}_(1),\tau_(1)),\ldots,(\boldsymbol{P}_(n),\tau_(n))])$
set	set(τ)
list	list(τ)
array	array(τ,Length)

Legend: $\boldsymbol{P}$ and $\boldsymbol{P}_($i) are names of properties, C is the name of a class, **S** is the name of a schema, τ and $\tau_($n) are data types, and Length is a integer number.

Other components

Formalism	Prolog notation
class declaration	(**S**,C,struct([($\boldsymbol{P}_1$,τ_1),...,($\boldsymbol{P}_n$,τ_n)],all) or
	(**S**,C_1,struct([($\boldsymbol{P}_1$,τ_1),...,($\boldsymbol{P}_n$,τ_n)],C_2)
relation declaration	(**S**,R,struct([($\boldsymbol{P}_1$,τ_1),...,($\boldsymbol{P}_n$,τ_n)])
key declaration	(**S**,$\boldsymbol{K}$,C,[$\boldsymbol{P}_1$,...,$\boldsymbol{P}_n$]) or
	(**S**,$\boldsymbol{K}$,R,[$\boldsymbol{P}_1$,...,$\boldsymbol{P}_n$])
foreing key declaration	(**S**,$\boldsymbol{K}$,C,[$\boldsymbol{P}_1$,...,$\boldsymbol{P}_n$],C′,[$\boldsymbol{P'}_1$,...,$\boldsymbol{P'}_n$]) or
	(**S**,$\boldsymbol{K}$,R,[$\boldsymbol{P}_1$,...,$\boldsymbol{P}_n$],C′,[$\boldsymbol{P'}_1$,...,$\boldsymbol{P'}_n$]))
method	not defined
paths	see L_{PS} Prolog notation

Legend: C, and C_i are names of a classes (with C_2 being a superclass), $\boldsymbol{P}_i$, and $\boldsymbol{P'}_i$ are names of properties, R is the name of a relation, $\boldsymbol{K}$ is the name of a key or of a foreign key, C′ is the name of a class or relation refered by other class or relation.

B.2 Correspondence between the Language L_{PS} and the Prolog Notation

Namespace

Formalism	**Prolog notation**
B (base schema)	sourceSchema(**S**)
T (target schema)	referenceSchema(**S**)
$\mathcal{L}_p$ (set of perspective schemas)	perspective_schemanames(p([$\mathbf{B}_1$,...,$\mathbf{B}_n$],**T**))
$\mathcal{A}$ (set of names of CAs)	perspective_canames(**PS**, **CA**)

Legend: **S** is the name of a schema, $\mathbf{B}_i$ are names of schemas that are used as base schemas, **T** is the name of a schema that is used as target, **PS** is the name of a perspective schema with this form: p([$\mathbf{B}_1$,...,$\mathbf{B}_n$],**T**), **CA** is the name (identifier) of a CA.

L_{PS} components

Formalism	**Prolog notation**
require declarations:	
class	require_class(**PS**, C, [$\boldsymbol{P}_1$, ... $\boldsymbol{P}_n$])
relation	require_relation(**PS**, R, [$\boldsymbol{P}_1$, ... $\boldsymbol{P}_n$])
key	require_key(**PS**,$\boldsymbol{K}$)
matching function	mFunction(**PS**,**M**, ($\mathbf{S}_1$,C_1), ($\mathbf{S}_1$,C_2)) or
signature	mFunction(**PS**,**M**,($\mathbf{S}_1$,C_1,[$\boldsymbol{P}_1$,...,$\boldsymbol{P}_n$]),($\mathbf{S}_1$,C_2,[$\boldsymbol{P'}_1$,.., $\boldsymbol{P'}_n$]))
correspondence assertions	see next table
view relation declaration	tempRelation(**PS**, V, struct([($\boldsymbol{P}_1$,τ_1),...,($\boldsymbol{P}_n$,τ_n)]))

Legend: **PS** is the name of a perspective schema with this form: p([$\mathbf{B}_1$,...,$\mathbf{B}_n$],**T**), C is the name of a class, $\boldsymbol{P}_i$ and $\boldsymbol{P'}_i$ are names of properties, R is the name of a relation, $\boldsymbol{K}$ is the name of a key or foreing key, **M** is the name of a matching function signature, $\mathbf{S}_i$ is the name of a schema or of a perspective schema, C_i is the name of a class or of a relation, τ_i are data types.

Correspondence assertions

Formalism	Prolog notation
PCA	ca(**CA**,pca,p(**B**,**T**,C,***P***),**SourcePropId**)
	ca(**CA**,pca,p(**B**,**T**,C,***P***),(Function,[**SourcePropId**,...])
	ca(**CA**,pca,p(**B**,**T**,C,***P***),if([(**SourcePropId**,**PredId**),...],Value))
ECA	
equivalence	ca(**CA**,eca,p(**B**,**T**,C),**SourceId**)
selection	ca(**CA**,eca,p(**B**,**T**,(C,**PredId**)),**SourceId**) or
	ca(**CA**,eca,p(**B**,**T**,C),(**SourceId**,pred(**PredId**))
union	ca(**CA**,eca,p(**B**,**T**,C),(union,[**SourceId**,...))
intersection	ca(**CA**,eca,p(**B**,**T**,C),(inter,[**SourceId**,...))
difference	ca(**CA**,eca,p(**B**,**T**,C),(dif,[**SourceId**,...))
SCA	
groupby	ca(**CA**,sca,p(**B**, **T**, C, ([$\boldsymbol{P}_1$, ..., $\boldsymbol{P}_n$], 0)), groupby, e(**S**, C′,
	[**SourcePropId**, ...], isa))
	ca(**CA**, sca, p(**B**, **T**, C, ([$\boldsymbol{P}_1$, ..., $\boldsymbol{P}_n$], **Skey**)), groupby, e(**S**, C′,
	[**SourcePropId**, ...], isa))
normalise	ca(**CA**, sca, p(**B**, **T**, C, ([$\boldsymbol{P}_1$, ..., $\boldsymbol{P}_n$], **Skey**)), groupby, e(**S**, C′,
	[**SourcePropId**, ...], isa))
ACA	ca(**CA**,aca,**SCA**,p(**B**,**T**,C,***P***),(**AggFuntion**,[**SourcePropId**,...]))
	ca(**CA**,aca,**SCA**,p(**B**,**T**,C,***P***),**SourcePropId**)
	ca(**CA**,aca,**SCA**,p(**B**,**T**,C,***P***),(Function,[**SourcePropId**,...])
	ca(**CA**,aca,**SCA**,p(**B**,**T**,C,***P***),if([(**SourcePropId**,**PredId**),...],Value))

Legend: **CA** is the name (identifier) of a CA, $\mathbf{B}_i$ are names of schemas that are used as base schemas, **T** is the name of a schema that is used as target, C is the name of a class or relation, $\boldsymbol{P}$, $\boldsymbol{P}_i$ are names of properties, **SourceId** and **SourcePropId** are identifiers of Prolog predicates that represent a class/relation of a base schema or a property of a base schema (more details later), Function is the name of a function, **PredId** is the name of a predicate (a condition of selection), **S** is the name of a schema, **Skey** is the name of a property that is a surrogate key in the target schema, **SCA** is the name (identifier) of a SCA, **AggFuntion** is the name of one of the aggregation functions: sum, max, min, count, avg.

Other components

Prolog identifier	**Prolog predicate**
PredId	perspective_pred(**PS**,**PredId** ,pred(**Op**,[**Argument1**,**Argument2**]))
SourceId	perspective_source(**PS**,**SourceId**,e(**S**,C,isa))
SourcePropId	perspective_source(**PS**,**SourcePropId**,e(**S**,C,$\boldsymbol{P}$))
	perspective_source(**PS**,**SourcePropId**,e(**S**,C,($\boldsymbol{P}$,[$\boldsymbol{P}_1$,...,$\boldsymbol{P}_n$])))
	perspective_source(**PS**,**SourcePropId**,e(**S**,C,path([$\boldsymbol{P}_1$,...,$\boldsymbol{P}_n$])))

Legend: **PS** is the name of a perspective schema with this form: p([$\mathbf{B}_1$,...,$\mathbf{B}_n$],**T**), C is the name of a class or relation, $\boldsymbol{P}$, $\boldsymbol{P}_i$ are names of properties, **Op** is one of operands: \=,=,<,>,>=,<= ,**Argument1** and **Argument2** are **SourcePropIds** or values.