LOGIC-BASED INFORMATION INTEGRATION

A report submitted to the University of Manchester
for the transferal from the Master of Philosophy to the
Doctor of Philosophy degree programme
in the Faculty of Engineering and Physical Sciences

2007

By
Héctor Manuel Pérez-Urbina
School of Computer Science
Supervisor: Ian Horrocks and Ulrike Sattler
Advisor: Sean Bechhofer

Contents

Abstract

Currently there is an amazing quantity of heterogeneous data distributed over a large number of sources. Retrieving information is becoming a difficult task in which regular users have to make use of their knowledge in terms of source characteristics, data models and query languages to obtain acceptable results. Moreover, with the arrival of the World Wide Web, users were given access to a huge number of sources.

Information integration (II) is the problem of combining data from different information sources and providing the user with a unified view of these data [28]. The idea is to provide transparent integrated access to relevant data, concealing information about the sources, such as their location, data model or query language. A variety of knowledge representation (KR) formalisms have been used to design existing information integration systems. Several of such formalisms are based on Description Logics (DLs) that represent the knowledge of an application domain in terms of concepts, individuals (instances of concepts) and roles (binary relations between individuals) [5]. Research in this area has been motivated by the possibility of integrating huge amounts of data on the World Wide Web, which would contribute to the development of the Semantic Web [9].

This research is focused on studying the problem of information integration for systems modeled with DLs. The main aim of this research is to investigate whether or not the problem can be solved in such a setting. The main research hypothesis is that techniques implemented in the various existing systems can be extended and/or modified to be used in this context. During the first year of research a sound and complete query rewriting algorithm was devised for a limited scenario. The future research priority is to investigate the ways to make the algorithm more general for typical application domains and information needs. At the end, it is desirable to design and implement our algorithm in a prototype system.

Chapter 1

Introduction

1.1 Context

1.1.1 Information integration

Currently there is an amazing quantity of heterogeneous data distributed over a large number of sources. Retrieving information is becoming a difficult task in which regular users have to make use of their knowledge in terms of source characteristics, data models and query languages to obtain acceptable results. In general, in order to obtain information from several relevant sources it is necessary to (1) find relevant sources, (2) interact with each source (through its specific interface) in isolation, and (3) manually combine relevant information.

Information integration (II) is the problem of combining data from different information sources and providing the user with a unified view of these data [28]. The idea is to provide transparent integrated access to relevant data, concealing information about the sources, such as their location, data model or query language. Although the general idea behind information integration is simple (grant users with transparent access to a set of sources through a uniform view), there are a number of interesting issues and challenges to consider. Some of the main issues relate to the fact that an IIS should manage several different sources. Relevant sources could handle different data models, query languages or internal protocols (besides being spread in different locations), which implies that the system must cope with source heterogeneity and autonomy. Moreover not all considered sources necessarily have the same capabilities regarding query answering, there could be some relational databases in which all their data can be

accessed with arbitrary queries, but we could also handle some sources in which data access is limited by some query patterns. On the other hand, there could be conflicts in the meaning or in the interpretation of the same data. Moreover, another new kind of problems have to be considered, namely, multiple values of the same entity in different sources, non-observance of integrity constraints, non-conformity of the measuring units, different data formats, etc.

A basic II system (IIS) is composed of (1) a *global view* representing the application domain, (2) a set of *information source models* representing the structure of the data residing at relevant sources, and (3) a set of semantic correspondences or *mappings* relating the global view to the source models (or the source models between each other) [28]. Two main approaches for modeling mappings have been proposed in the literature [40, 31], the *global-as-view* (GaV) approach, in which the global view is defined as a set of queries over the underlying sources; and the *local-as-view* (LaV) approach, in which the sources are modeled as a set of queries over the global view. The combination of these two approaches has resulted in the GLaV approach [22], in which queries over the global view are related to queries over the sources.

Queries from users and/or applications are posed to the IIS in terms of the global view. Computing the set of answers for a given query (query processing), strongly depends on the way mappings are modeled. In the GaV approach, the problem generally resumes to some kind of unfolding [28], while in the LaV and GLaV approaches, query processing is closely related to the problem of *answering queries using views* [31, 40], and can be reduced to *view-based query rewriting* or *view-based query answering* [28].

1.1.2 The use of Description Logics

A variety of knowledge representation (KR) formalisms have been used to design existing IISs, most of which are closely related to the relational model [2]. Nonetheless, there are a number of efforts that make use of more expressive formalisms to better capture the semantics of the IIS (i.e., accurately model its three main components). Several of such richer formalisms are based on Description Logics (DLs) [5].

DLs are a family of KR formalisms that represent the knowledge of an application domain in a so-called knowledge base (KB) in terms of concepts, individuals (instances of concepts) and roles (binary relations between individuals). A KB is

composed of a terminology, so-called TBox and a set of assertions called ABox. The TBox, is a set of axioms that define a concept hierarchy. Moreover, some DLs also allow for a role hierarchy. The ABox is a collection of assertions about individuals. There are two types of assertions: (1) concept assertions, that specify that an individual is an instance of a concept, and (2) role assertions, which relate a pair of individuals with a role.

1.2 State of the art

Existing systems implementing GaV include [19, 17, 6, 24, 37]. LaV, on the other hand, is implemented in [4, 32, 23, 20], mainly focusing on query rewriting. GLaV has received less attention, and, on the contrary, current efforts on this matter [15] mainly focus on query answering. Summing up, the design and implementation of a GLaV IIS that effectively solves the problem of view-based query rewriting remains an open problem.

Therefore, the development of a sound and complete query rewriting algorithm is of major interest. It is desirable to use rich KR formalisms to model the IIS in order to capture the semantics of a given scenario as accurately as necessary. It is also desirable to provide a framework to facilitate the design of an IIS (creating the global view, adding sources, defining mappings) based on the characteristics of a specific scenario.

1.3 Aims and objectives

This research is focused on studying the problem of information integration for GLaV systems modeled with DLs, enhanced with mappings between source models. The main aim of this research is to investigate whether or not the problem of view-based query rewriting can be solved in such a setting. The main research hypothesis is that techniques implemented in the various existing systems can be extended and/or modified to be used in this general context.

The approach presented in [15], based on the II KR framework presented in [14, 13], is the closest to our goal. Nevertheless, given the fact it is used in a data warehouse context, it focuses on view-based query answering as opposed to view-based query rewriting. Unfortunately, since query answering requires the content of the sources as input, integrating different information sources with this

technique is not practical, scalable or feasible for many typical scenarios.

1.3.1 Methodology

In order to reach our research aims it is necessary to investigate:

- Different KR formalism including several DLs.
- The problem of II in general.
- The relation between II and DLs.
- Current techniques and approaches implemented in existing IISs.
- The problem of answering queries using views, its existing solutions, its complexity and its relationship with other problems.
- Typical application domains for II.
- Typical information needs (i.e., queries) w.r.t. a given domain.

1.4 Anticipated contributions

Anticipated contributions of this research include:

- An understanding of how DLs can be used in an II context.
- An investigation of the trade-off between the complexity of view-based query rewriting and the expressive power of the KR formalism used to define the global view, the source models and the mappings.
- The development of a new practical algorithm for view-based query rewriting for DL-based GLaV IISs enhanced with mappings between source models.
- The development of a framework to facilitate the design of IISs (creating the global view, adding sources, defining mappings) based on the characteristics of a specific scenario.

1.5 Structure of the document

The remainder of this document is as follows:

- chapter 2 presents background knowledge and introduces the terminology used throughout the document. This chapter is a brief introduction to Description Logics, the relational model and two considered query languages: conjunctive queries and datalog.

- chapter 3 presents the foundations of information integration. The problem of information integration is defined and several existing approaches to solving it are presented. In the last part of this chapter we describe a KR framework which abstracts the main components of an IIS.

- chapter 4 presents the formal semantics of a centralized IIS as well as that of its mappings. In the last part of this chapter we introduce the main techniques for query processing according to the type of mapping the system was modeled with.

- chapter 5 presents the generalities of the problem of answering queries using views. We present three solutions that have been proposed so far, namely, the bucket , the inverse-rule and the MiniCon algorithms.

- chapter 6 presents the various existing IISs that have been implemented so far. We analyse them w.r.t. the framework introduced in chapter 3, describing, for each system, the information integration approach it implements, the type of mappings and queries that are cosidered, as well as the distinctive features of the system.

- chapter 7 presents a sound and complete view-based query rewriting algorithm that was devised for an scenario where we consider: (1) acyclic $\mathcal{ALCH}$ [1] KBs for the global view and the source models, (2) limited GLaV and intersource mappings, and (3) atomic user queries.

- chapter 8 concludes the document presenting the research undertaken and the future work.

[1] A simple DL including role hierarchy [5].

Chapter 2

Background

2.1 Description Logics

Description Logics (DLs) are a family of KR formalisms that represent the knowledge of an application domain in a so-called knowledge base (KB) in terms of concepts, individuals (instances of concepts) and roles (binary relations between individuals) [5]. Constructors can be used to define complex concepts and/or roles. A particular Description Logic is characterised by its set of supported constructors. A naming scheme for DLs as well as the basic language for designing them was introduced in [39]. Such a language, called $\mathcal{AL}$ (Attributive concept Language) allows for negation of atomic concepts, conjunction, universal quantification, and unqualified existential quantification. For additional operators a further letter is appended, e.g., $\mathcal{AL}$ plus complex negation is called $\mathcal{ALC}$. For a complete survey on existing DLs see [5].

2.1.1 Constructors

Table 2.1 lists the constructors that are available to build more complex concept expressions from atomic concept names. The cardinality of the set S is denoted as $|S|$. Table 2.2 lists the constructors that are available to build more complex role expressions from atomic role names[1]. The inverse relation is symmetric. To avoid writing role expressions such as R^{--} we define a function inv(). inv(R) returns R^- if R is a role name, or returns S if $R = S^-$ where S is a role name.

The semantics are given by an interpretation $\mathcal{I} = \{\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}}\}$, which consists

[1]We use C and D to denote concepts, and R and S to denote roles.

Constructor	**Syntax**	**Semantics**
atomic concept	A	$A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$
top	$\top$	$\top^{\mathcal{I}} = \triangle^{\mathcal{I}}$
bottom	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$(\neg C)^{\mathcal{I}} = \triangle^{\mathcal{I}} \setminus C^{\mathcal{I}}$
nominal/oneOf	$\{a_1, \ldots, a_n\}$	$\{a_1, \ldots, a_n\}^{\mathcal{I}} = \{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$
universal quantification	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
existential quantification	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
number restriction	$\geq nR$	$(\geq nR)^{\mathcal{I}} = \{x \mid \lvert\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\}\rvert \geqslant n\}$
	$\leq nR$	$(\leq nR)^{\mathcal{I}} = \{x \mid \lvert\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\}\rvert \leqslant n\}$
qualified number restriction	$\geq nR.C$	$(\geq nR.C)^{\mathcal{I}} = \{x \mid \lvert\{y \mid \langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\rvert \geqslant n\}$
	$\leq nR.C$	$(\leq nR.C)^{\mathcal{I}} = \{x \mid \lvert\{y \mid \langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}\rvert \leqslant n\}$

Table 2.1: Syntax and semantics of concept expression constructors.

Constructor	**Syntax**	**Semantics**
atomic role R	R	$R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$
role conjunction	$R \sqcap S$	$(R \sqcap S)^{\mathcal{I}} = R^{\mathcal{I}} \cap S^{\mathcal{I}}$
role disjunction	$R \sqcup S$	$(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$
inverse role	R^-	$(R^-)^{\mathcal{I}} = \{\langle y, x\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}\}$
role composition	$R \circ S$	$(R \circ S)^{\mathcal{I}} = \{\langle x, z\rangle \mid \exists y.(\langle x, y\rangle \in R^{\mathcal{I}} \wedge \langle y, z\rangle \in S^{\mathcal{I}})\}$
transitive closure	R^+	$(R^+)^{\mathcal{I}} = \bigcup_{i>0}(R^{\mathcal{I}})^i$
reflexive transitive closure	R^*	$(R^*)^{\mathcal{I}} = \bigcup_{i\geq 0}(R^{\mathcal{I}})^i$, where $(R^{\mathcal{I}})^0 = (\mathrm{id}(\exists R.\top))^{\mathcal{I}}$, and $(R^{\mathcal{I}})^{i+1} = (R^{\mathcal{I}})^i \circ R^{\mathcal{I}}$
identity	$\mathrm{id}(C)$	$(\mathrm{id}(C))^{\mathcal{I}} = \{\langle x, x\rangle \mid x \in C^{\mathcal{I}}\}$

Table 2.2: Syntax and semantics of role expression constructors.

of a non-empty domain of interpretation $\triangle^{\mathcal{I}}$, and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function maps atomic concepts to a subset of the domain $\triangle^{\mathcal{I}}$, atomic roles to a subset of $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ and individual names to elements of $\triangle^{\mathcal{I}}$.

2.1.2 Axioms

A knowledge base $\mathcal{K}$ is a tuple of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, written as $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. $\mathcal{T}$ is composed of a set of TBox axioms (see table 2.3). $\mathcal{A}$ contains assertions about individuals in the knowledge base as a set of ABox axioms (see table 2.4). For $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, the signature of $\mathcal{K}$ is the union of concept, role, and individual names in $\mathcal{K}$.

Axiom	**Syntax**	**Semantics**
concept inclusion	$C \sqsubseteq D$	$(C \sqsubseteq D)^{\mathcal{I}} = C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept equivalence	$C \equiv D$	$(C \equiv D)^{\mathcal{I}} = C^{\mathcal{I}} = D^{\mathcal{I}}$
concept existence	$\exists C$	$(\exists C)^{\mathcal{I}} = \lvert C^{\mathcal{I}} \rvert \geq 1$
role inclusion	$R \sqsubseteq S$	$(R \sqsubseteq S)^{\mathcal{I}} = R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
role equivalence	$R \equiv S$	$(R \equiv S)^{\mathcal{I}} = R^{\mathcal{I}} = S^{\mathcal{I}}$
transitive role	$\mathrm{Trans}(R)$	$\mathrm{Trans}(R)^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$
functional role	$\mathrm{Func}(R)$	$\mathrm{Func}(R)^{\mathcal{I}} = \langle x, y \rangle \in R^{\mathcal{I}} \wedge \langle x, z \rangle \in R^{\mathcal{I}} \rightarrow x = y$

Table 2.3: TBox axioms.

According to [5], cycles in a TBox $\mathcal{T}$ are defined as follows. Let A and B be atomic concepts ocurring in $\mathcal{T}$. We say that A *directly uses* B in $\mathcal{T}$ if B appears on the right-hand side of the definition of A, and we call *uses* the transitive closure of the relation directly uses. Then $\mathcal{T}$ contains a cycle iff there exists an atomic concept in $\mathcal{T}$ that uses itself. Otherwise, $\mathcal{T}$ is called acyclic.

Some DLs assume the so-called *Unique Name Assumption* (UNA), which means that different individual names are mapped to different elements of the domain. Without the Unique Name Assumption, different individual names can be mapped to the same element of the domain. In this case, most DLs allow to state that two individuals are different or to assert that two individual names should be mapped to the same element of the domain.

Axiom	**Syntax**	**Semantics**
concept assertion R	R	$R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$
role assertion	$R \sqcap S$	$(R \sqcap S)^{\mathcal{I}} = R^{\mathcal{I}} \cap S^{\mathcal{I}}$
individual equality	$R \sqcup S$	$(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$
individual inequality	R^-	$(R^-)^{\mathcal{I}} = \{\langle y, x\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}\}$

Table 2.4: ABox axioms.

2.2 Databases

2.2.1 Relational model

Intuitively, in the relational data model, data are stored as n-ary tuples in relations or tables. Each relation has a name and is associated to a list of attributes. A database schema is simply a set of relation names (with their associated attributes). Finally a database is composed by sets of tuples (one set for every relation of its schema) [2].

Formally we consider three disjoint, countably infinite sets: **att**, which denotes an ordered set of attributes; **dom**, that contains constants of a specific domain; and **relname**, which is composed by the names of the considered relations. As already mentioned, each relation has a name and is associated to a set of attributes. In order to associate a set of attributes to each relation name, we define the function

$$\text{sort}: \textbf{relname} \rightarrow U$$

where $U \in$ **att**.

A *relation schema* $\mathcal{RS}$ is composed by a relation name R and its associated to the set of attributes sort(R). To denote a relation schema, we use the following notation:

$$\text{R}(A_1, \ldots, A_n),$$

where R $\in$ **relname** and sort(R) $= \{A_1, \ldots, A_n\}$.

A *database schema* $\mathcal{DBS}$ is a set of relation schemata. It is defined over **relname** that, as already said, comprises one symbol for each relation.

Example 2.2.1. Suppose our domain refers to restaurants. Let us define **att**

and **relname** as follows:

$$\begin{aligned}&\textbf{att} = \{\text{RestaurantName}, \text{Type}, \text{PriceClassification}, \text{PostalCode}\},\\ &\textbf{relname} = \{\text{FoodType}, \text{Price}, \text{Location}\}.\end{aligned}$$

Based on the previous sets, we can define a possible database schema $\mathcal{DBS}_1$ as a set containing the following relation schemata:[2]:

$$\begin{aligned}\mathcal{DBS}_1 = \{&\text{FoodType}(\textit{RestaurantName}, \textit{Type}),\\ &\text{Price}(\textit{RestaurantName}, \textit{PriceClassification}),\\ &\text{Location}(\textit{RestaurantName}, \textit{PostalCode})\}.\end{aligned}$$

A *tuple* t is a function that maps a subset of attributes to constants of the domain:

$$t\colon U \to \textbf{dom},$$

where $U \in \textbf{att}$.

In order to denote a tuple, we use the following notation:

$$\langle u_1 \colon c_1, \ldots, u_n \colon c_n \rangle^3,$$

where $t(u_i) = c_i$, $u_i \in U$, $c_i \in \textbf{dom}$.

A *relation* $\mathcal{R}$ for a relation schema $\mathcal{RS}$ is a set of tuples that map the attributes associated to $\mathcal{RS}$ to constants of the domain **dom**.

Example 2.2.2. Consider our domain is defined as follows:

$$\begin{aligned}\textbf{dom} = \{&\text{Chiquito}, \text{Barburrito}, \text{Chinatown}, \text{Ming}, \text{Mexican},\\ &\text{Chinese}, \text{Cheap}, \text{Expensive}, \text{L101}, \text{L102},\\ &\text{L201}, \text{L202}\}.\end{aligned}$$

Based on the relation schema of FoodType defined in the previous example, the

[2]Note that it is not necessary to know **dom** in order to define a relation schema.
[3]Given the fact that **att** is ordered, it is common not to write the attribute names.

relation FoodType could be defined as follows:

$$\begin{aligned}\text{FoodType} = \{&\langle\text{Chiquito, Mexican}\rangle, \langle\text{Barburrito, Mexican}\rangle,\\ &\langle\text{Chinatown, Chinese}\rangle, \langle\text{Ming, Chinese}\rangle\}\end{aligned}$$

Finally, a *database* $\mathcal{DB}$ for a schema $\mathcal{DBS}$ is a set of relations, one for each relation schema defined in $\mathcal{DBS}$.

Example 2.2.3. A possible database Restaurants for our previously defined database schema $\mathcal{DBS}_1$ could be composed by the relations:

$$\begin{aligned}\text{FoodType} = \{&\langle\text{Chiquito, Mexican}\rangle, \langle\text{Barburrito, Mexican}\rangle,\\ &\langle\text{Chinatown, Chinese}\rangle, \langle\text{Ming, Chinese}\rangle\},\\ \text{Price} = \{&\langle\text{Chiquito, Cheap}\rangle, \langle\text{Ming, Cheap}\rangle,\\ &\langle\text{Barburrito, Expensive}\rangle, \langle\text{Chinatown, Expensive}\rangle\},\\ \text{Location} = \{&\langle\text{Chiquito, L101}\rangle, \langle\text{Ming, L201}\rangle,\\ &\langle\text{Barburrito, L102}\rangle, \langle\text{Chinatown, L202}\rangle\}.\end{aligned}$$

2.2.2 Conjuctive queries and datalog

A conjunctive query q of arity n over a database schema $\mathcal{DBS}$ is an expression of the form:

$$Q(\vec{x}) :- \bigwedge_{1 \leq i \leq k} R_i(\vec{x}_i)$$

where:

- $R_i \in$ **relname**, i.e., R_i is the name of a relation in the database schema $\mathcal{DBS}$.
- $\vec{x}$ and $\vec{x}_i$ are tuples of variables and constants.
- Each variable of $\vec{x}$ must occur in some $\vec{x}_i$.

The atom $Q(\vec{x})$ is called the *head* of the query and refers to the answer relation. The atoms $R_1(\vec{x}_1), \ldots, R_k(\vec{x}_k)$ are the *subgoals* in the body of the query.

Given a query q and a database $\mathcal{DB}$, $q^{\mathcal{DB}}$ denotes the set of tuples in $\mathcal{DB}$ that satisfy q, i.e., the set of tuples that are valuations in **dom** for the free variables of q that make q true in $\mathcal{DB}$.

Example 2.2.4. If we were to pose the query 'cheap, mexican restaurants',

$$\text{cheapMexican}(x) : -\text{FoodType}(x, \textsc{Mexican}) \wedge \text{Price}(x, \textsc{Cheap})^4,$$

to our database Restaurants; then:

$$q^{\textsc{Restaurants}} = \{\langle \text{Chiquito} \rangle\}.$$

A *datalog* query is a set of conjunctive queries whose body predicates do not have to refer to names of relations of the database schema. In a datalog query there are two types of predicates: *EDB*, that refer to the database schema relations, and *IDB*, referring to intermediate relations. Therefore in the rules, EDB predicates appear only in the bodies, while IDB predicates can appear anywhere. There is a distinguished IDB predicate called the *query predicate* that refers to the relation of the result.

A predicate p in a datalog program is said to *depend* on another predicate q, if q appears in one of the rules whose head is p. The datalog query is said to be *recursive* if there is a cycle in the dependency graph of the predicates. A non-recursive datalog query can be rewritten as a union of conjunctive queries. Given a datalog query q and a database $\mathcal{DB}$, containing the extensions for all EDB predicated of q, $q^{\mathcal{DB}}$ denotes the least fixpoint model of q and $\mathcal{DB}$. Such a model can be computed as follows: First, rules in q are applied arbitrarily. An appication of a rule may drive new tuples for the relation denoted by the head of a given rule. Rules continue to be applied until no new tuples can be derived. $q^{\mathcal{DB}}$ is the set of tuples computed for the query predicate.

A query q_1 is said to be *contained* in a query q_2, denoted $q_1 \sqsubseteq q_2$, if for any database $mathcalDB$, ${q_1}^{\mathcal{DB}} \subseteq {q_2}^{\mathcal{DB}}$, i.e., the set of tuples computed for q_1 is a subset of those computed for q_2. The two queries are said to be equivalent if $q_1 \sqsubseteq q_2$ and $q_2 \sqsubseteq q_1$.

[4]We use THIS FONT to denote constants.

Chapter 3

Foundations of information integration

3.1 Introduction

Information integration (II) is the problem of combining data from different information sources and providing the user with a unified view of these data [28]. The idea is to provide transparent integrated access to relevant data, concealing information about the sources, such as their location, data model or query language. In this chapter we describe the different approaches that have been proposed to solve the problem of information integration. We also describe the knowledge representation framework for information integration proposed in [14, 13]. We take such a framework as a base for our own proposal for solving the problem described in chapter 7.

3.2 Approaches

3.2.1 Declarative vs. Procedural

There are two basic approaches to solving the problem of information integration, namely, declarative and procedural [14]. In the procedural approach, data are integrated in an adhoc way with respect to a set of predefined needs (v.g., typical queries or specific requirements about the sources). In this case, the idea is to develop software components that access the sources according to the predefined requirements. There are two types of such software components: wrappers, which

encapsulate the sources; and mediators, which merge and reconcile data coming from wrappers (or other mediators). The main advantage of this approach is that we could use knowledge regarding certain domain to make simplifying assumptions in order to improve the performance of the query processing, nevertheless we would have to change the system every time a new source is added of a new query pattern is identified.

On the other hand, in the declarative approach, the idea is to model the data of the sources with a suitable language in order to have a unified representation to make queries on. The main advantage of this approach is that the mechanism to answer queries is general, i.e., it is not hardcoded into the system, which provides scalability regarding the queries and the sources handled by the system.

3.2.2 Virtual vs. Materialized

Independently of the preferred approach, an information integration system (IIS) can be virtual or materialized [13]. In the virtual approach, the IIS accesses the sources every time a query is made. The IIS acts like an interface between the user and the sources. The idea is to have a representation of the content of the underlying data residing at the sources, but not the data itself. A clear advantage of this approach is that the IIS can delegate the problem of having up-to-date information to the underlying sources. Nevertheless querying is costly because, as already said, it implies accessing the sources with every query. This kind of approach is useful in situations in which the relevant sources are constantly being updated and/or scenarios where we need the information to be up-to-date.

On the other hand, in the materialized approach, the sources are accessed a priori in order to provide a replicated view of the data within the IIS. Querying is typically more efficient, nevertheless maintaining the materialized view is costly, especially if we want to provide up-to-date information. This approach results especially useful in situations in which having up-to-date information is not crucial (v.g., a data warehouse) and/or when information does not change frequently. There are some systems that provide both alternatives, i.e., the user/application can choose between the virtual and the materialized approaches.

3.2.3 Centralized vs. Peer-to-peer

The architecture of an IIS could be either *centralized* or *peer-to-peer* [29]. The centralized approach (also called mediator-based approach) implies the development of a unified view that represents the application domain and describes the data residing at the sources. The idea behind this approach is to provide a common representation or global schema of the schemata of the relevant sources for the user to pose queries on. Clearly in order to retrieve data residing at the sources, it is vital to establish a *mapping* or semantic relationship between the global schema and the schemata representing the structure of the underlying sources. The collection of such schemata is called *source* schema. In a centralized approach, users (or applications) pose queries to the IIS in terms of the global schema, then the system uses the mapping to reformulate these global queries into a set of queries in terms of the source schema. After this initial process, data is retrieved, and finally, retrieved data are integrated (i.e., cleaned, reconciled, put in a common format, etc.) and sent back to the user. Developing a centralized IIS is a challenging task, classical challenges include the construction of the global schema, the definition of the mapping between the global schema and the source schema, and the choice of the method to compute the answer to queries.

On the other hand, the peer-to-peer approach is a generalization of the former approach that considers the existence of several autonomous components called peers. Roughly, a peer-to-peer IIS can be seen as a collection of centralized systems that cooperate with each other. In a peer-to-peer approach, users (or applications) pose queries to the IIS in terms of the peer schema of a specific peer, then the system uses the mapping to reformulate these global queries into a set of queries in terms of the peer source schema, and in terms of other peer schemata. After this initial process, the process is repeated until the initial query is reformulated only in terms of the sources. Then, data is retrieved, and finally, retrieved data are integrated (i.e., cleaned, reconciled, put in a common format, etc.) and sent back to the user.

3.3 Knowledge representation framework

In this section we describe the framework to information integration proposed in [14, 13]. We chose to take this framework as reference given the fact that it serves a general setting in which different approaches to integration can be implemented,

evaluated and compared. Such framework is divided in four levels, namely, the meta level [1], the conceptual level, the logical level, and the physical level (see figure 3.1).

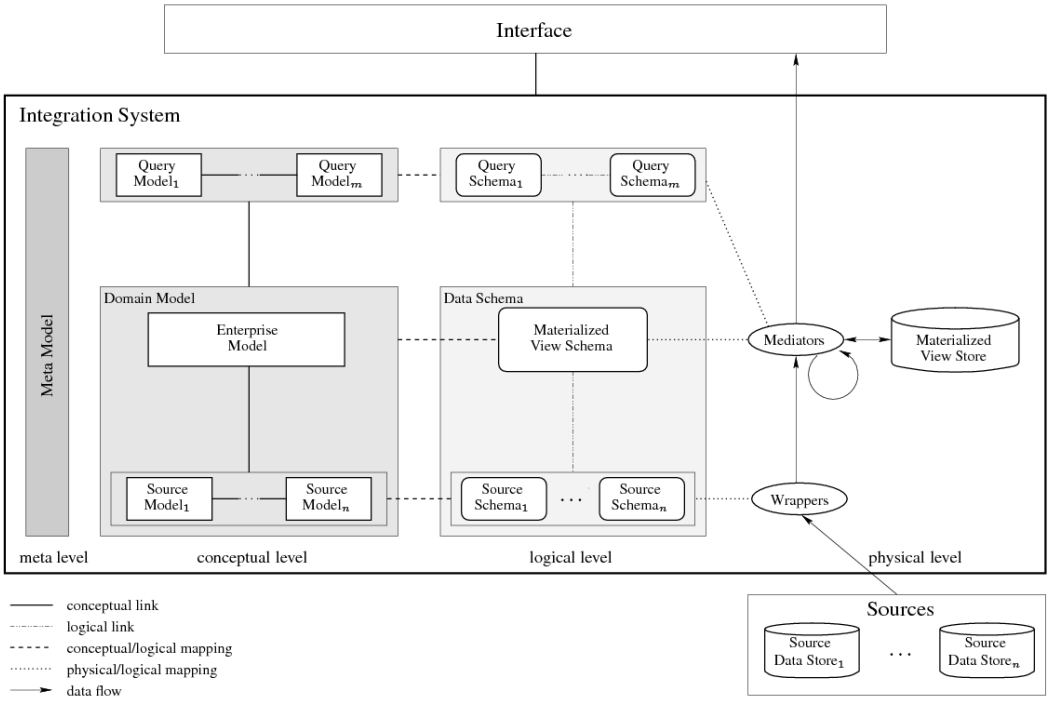

Figure 3.1: Information integration framework.

3.3.1 Conceptual level

The conceptual level contains a *formal* description of the concepts, the relationship between concepts, and the information requirements that the integration system has to deal with. The conceptual level is composed of the enterprise model, a set of source models, and a set of query models. The enterprise model is a conceptual representation of the concepts and the relationships between them that are relevant to a given IIS. The source model of a given source is simply a conceptual representation of the data residing in such source. A query model is a conceptual representation of an information need (e.g., a relational query). The enterprise model and the various source models constitute the domain model.

[1]The meta level is composed of the meta model which is used to store all meta information about the system (i.e., types of sources, location, number, etc.).

The domain model contains intermodel relationships, i.e., semantic correspondences between the enterprise model and the source models, and between the source models themselves. It is important to note that integration does not simply mean producing the enterprise model, but rather to establish the correct intermodel relationships.

3.3.2 Logical level

The logical level contains the description of the data, and the relevant queries based on the relational model (see section 2.2.1) (i.e., as a set of relations). The logical level is composed of a materialized view schema, a set of source schemata, and a set of query schemata. The materialized view schema describes the logical content of the materialized views maintained by the system, while each source schema describes the logical content of the corresponding source [2]. On the other hand, the set of query schemata express the information needs at the logical level. The materialized view schema and the various source schemata constitute the data schema.

3.3.3 Physical level

The physical level refers to the actual data managed by the system. The materialized view store contains the data the system maintains materialized. Two software components are included at this level: wrappers and mediators. A wrapper is a software module that accesses a specific source and retrieves the residing data in a way that is coherent with the logical specification of the given source. On the other hand a mediator [42] is a software module associated to a specific query, it takes as input data returned from either wrappers or other mediators, refines such data by integrating and resolving conflicts, and produces, as output, the answer of the associated query. The result of a mediator can be materialized, transferred to the interface, or passed to other mediator.

[2] Obviously, the materialized view schema is only used in case the IIS implements materialized information integration, it becomes meaningless in case the system implements a virtual approach.

3.3.4 Interlevel mappings

The conceptual, logical and physical levels described above are related through semantic correspondences called mappings. The mapping between the source models and the corresponding schemata represents the fact that there must be an explicit correspondence between the concepts of the models and the relations of the schemata. The same applies to mappings between a query model and a query schema, and between elements of the domain model and the materialized view schema. The mapping between mediators and query schemata (or the materialized view schema) represents the fact that mediators are responsible for computing the extension of specific logical objects (which can be materialized). On the other hand, wrappers are exclusively associated to source schemata elements (the ones whose extensions they are responsible to compute).

Chapter 4

Centralized information integration

4.1 Introduction

A basic II system (IIS) is composed of (1) a *global view* representing the application domain, (2) a set of *information source models* representing the structure of the data residing at relevant sources, and (3) a set of semantic correspondences or *mappings* relating the global view to the source models (or the source models between each other) [28]. Two main approaches for modeling mappings have been proposed in the literature [40, 31], the *global-as-view* (GaV) approach, in which the global view is defined as a set of queries over the underlying sources; and the *local-as-view* (LaV) approach, in which the sources are modeled as a set of queries over the global view. The combination of these two approaches has resulted in the GLaV approach [22], in which queries over the global view are related to queries over the sources. This latter approach is general enough to capture both GaV and LaV, and provides the means to express richer and more complex mappings withing an IIS, which enables to capture its semantics more accurately.

4.2 Semantics

Designing an IIS under a centralized architecture consists in defining the relationships or mappings between the enterprise model and the various source models.

In order to clearly describe the different existing approaches to define these mappings, we instantiate the framework presented in section 3.3 to formally specify a centralized IIS. The instance of such framework is based on the work of [28], in which a centralized IIS $\mathcal{I}$ is simply represented with a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where:

- $\mathcal{G}$ is the enterprise model or global view, expressed in a language $\mathcal{L}_\mathcal{G}$ over an alphabet $\mathcal{A}_\mathcal{G}$ that comprises a symbol for each element of $\mathcal{G}$.

- $\mathcal{S}$ is the set of source models, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$ (disjoint from $\mathcal{A}_\mathcal{G}$) that comprises a symbol for each element of the source models.

- $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$. In general, $\mathcal{M}$ is composed of a set of assertions of the forms:

$$q_\mathcal{S} \rightsquigarrow q_\mathcal{G},$$
$$q_\mathcal{G} \rightsquigarrow q_\mathcal{S},$$

  where $q_\mathcal{G}$ and $q_\mathcal{S}$ represent two queries over $\mathcal{G}$ and $\mathcal{S}$ respectively. Queries $q_\mathcal{S}$ are formulas expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_\mathcal{S}$. Similarly, queries $q_\mathcal{G}$ are formulas expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_\mathcal{G}$. An assertion $q_\mathcal{S} \rightsquigarrow q_\mathcal{G}$ intuitively means that the source concept represented by the query $q_\mathcal{S}$ corresponds to the global concept represented by $q_\mathcal{G}$, analogously for $q_\mathcal{G} \rightsquigarrow q_\mathcal{S}$[1]. The different types of correspondences will be discussed in section 4.3 to make this intuition precise.

In order to give semantics to $\mathcal{I}$, let us define the so-called *source database* $\mathcal{C}$ over a fixed domain Γ as a database containing the extensions of the elements of $\mathcal{S}$, i.e., $\mathcal{C}$ contains the data residing at the sources. Similarly, let us define a *global database* $\mathcal{B}$ over Γ as a database containing the extensions of the elements of $\mathcal{G}$. A global database $\mathcal{B}$ is said to be *legal* for $\mathcal{I}$ with respect to $\mathcal{C}$, if it satisfies $\mathcal{M}$ with respect to $\mathcal{C}$. The set of global databases that are legal for $\mathcal{I}$ with respect to $\mathcal{C}$ is defined as:

$$\text{sem}^\mathcal{C}(I) = \{\mathcal{B} \mid \mathcal{B} \text{ is an } \mathcal{M}\text{-model with respect to } \mathcal{C}\}.$$

[1] note that $q_\mathcal{G}$ and $q_\mathcal{S}$ must have the same arity

Intuitively, a global database $\mathcal{B}$ is part of $sem^{\mathcal{C}}(I)$ if there is a mapping between $\mathcal{B}$ and $\mathcal{C}$. The notion of mapping and the formal definition of what it means for $\mathcal{B}$ to be an $\mathcal{M}$-model with respect to $\mathcal{C}$ is presented in section 4.3.

Finally we focus on describing the semantics assigned to queries made to $\mathcal{I}$. Queries are expressed in a query language $\mathcal{L}_{\mathcal{Q}}$ and posed in terms of $\mathcal{G}$. Given a source database $\mathcal{C}$ for $\mathcal{I}$, the set of certain answers to a query q in $\mathcal{I}$ with respect to $\mathcal{C}$ is the set of tuples $\vec{c}$ of elements of the domain Γ, such that $\vec{c} \in q^{\mathcal{B}}$ for *every* global database $\mathcal{B}$ that is legal for $\mathcal{I}$ with respect to $\mathcal{C}$. Formally, the set of certain answers to a query q with respect to $\mathcal{I}$ and $\mathcal{C}$ is defined as:

$$cert(q, \mathcal{I}, \mathcal{C}) = \{\vec{c} \in q^{\mathcal{B}} \mid \forall \mathcal{B} \in sem^{\mathcal{C}}(I)\}.$$

Note that in this logical context, the problem of finding certain answers resumes to logical implication: checking whether it logically follows from the data in the sources that $\vec{c}$ satisfies q. On the other hand, finding possible answers, i.e., checking whether $\vec{c} \in q^{\mathcal{B}}$ for some global database $\mathcal{B} \in sem^{\mathcal{C}}(I)$, is a consistency problem: checking whether assuming that $\vec{c}$ is in the answer of q do not contradict data in the sources. In any case, the problem of query answering completely depends on the types of mappings used.

4.3 Mappings

In this section we describe two approaches, based on first order logic assertions, to specify the mappings between the enterprise model and the various source models: Local-as-View (LaV) and Global-as-View (GaV) [40, 31]. We also present an hybrid called Global-Local-as-View (GLaV) [22] that results from the merge of the two former approaches.

4.3.1 Local as view

In general terms, in the LaV approach (also called source-centric) the sources are defined in terms of the global view. In other words, each element of the source models is expressed in terms of elements of the enterprise model [28, 30]. The mapping $\mathcal{M}$ and the source database $\mathcal{C}$ do not provide direct information about which data satisfy the enterprise model. Sources are regarded as views, and we have to answer queries on the basis of the available data in the views [29].

Formally, in an IIS $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ based on the LaV approach, the mapping $\mathcal{M}$ associates to each element s of the source models $\mathcal{S}$ a query $q_{\mathcal{G}}$ over $\mathcal{G}$. The query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ only allows expressions representing *one* symbol of the alphabet $\mathcal{A}_{\mathcal{S}}$. Therefore, a LaV mapping is composed of a set of assertions, one for each element of $\mathcal{S}$, of the form:

$$s \rightsquigarrow q_{\mathcal{G}}$$

Example 4.3.1. To simplify the explanation, let us consider that $\mathcal{G}$ is represented with a relational schema composed of the following relations:

$$\begin{aligned}
&\text{FoodType}(\mathit{RestaurantName}, \mathit{Type}),\\
&\text{Price}(\mathit{RestaurantName}, \mathit{PriceClassification}),\\
&\text{Location}(\mathit{RestaurantName}, \mathit{PostalCode}).
\end{aligned}$$

Let $\mathcal{S}$ be composed of the following relational sources:

$$\begin{aligned}
&s_1 : \text{Mexican}(\mathit{Name}),\\
&s_2 : \text{Chinese}(\mathit{Name}),\\
&s_3 : \text{Cheap}(\mathit{Name}),\\
&s_4 : \text{Expensive}(\mathit{Name}),\\
&s_5 : \text{Location}(\mathit{Name}, \mathit{Address}),
\end{aligned}$$

where sources s_1, s_2, s_3 and s_4 contain lists of restaurants, and source s_5 contains a list of the restaurants with their addresses.

Then, an example of a LaV-based mapping $\mathcal{M}$ could be composed of the following assertions:

$$\begin{aligned}
&\text{Mexican}(N) \rightsquigarrow \text{FoodType}(N, \textsc{Mexican}),\\
&\text{Chinese}(N) \rightsquigarrow \text{FoodType}(N, \textsc{Chinese}),\\
&\text{Cheap}(N) \rightsquigarrow \text{Price}(N, \textsc{Cheap}),\\
&\text{Expensive}(N) \rightsquigarrow \text{Price}(N, \textsc{Expensive}),\\
&\text{Location}(N, A) \rightsquigarrow \text{Location}(N, A).
\end{aligned}$$

In order to better characterize $\mathcal{M}$, we use a function type[2] that associates a mapping type to each element s of $\mathcal{S}$. Formally:

$$\text{type}\colon \mathcal{S} \rightarrow \{\text{sound}, \text{complete}, \text{exact}\}.$$

With type we can specify how accurate the source element s is with respect to the associated query $q_{\mathcal{G}}$, as follows:

- If type(s) = sound, it means that the extension of s is a subset of the tuples satisfying $q_{\mathcal{G}}$. In other words, the extension of s contains *only* correct answers for $q_{\mathcal{G}}$, but *not necessarily all*. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $s \in \mathcal{S}$:

  $$s^{\mathcal{C}} \subseteq {q_{\mathcal{G}}}^{\mathcal{B}}$$

  That is, $\forall \vec{x}(s(\vec{x}) \rightarrow q_{\mathcal{G}}(\vec{x}))$.

- If type(s) = complete, it means that the extension of s is a superset of the tuples satisfying $q_{\mathcal{G}}$. In other words, the extension of s contains *all* correct answers for $q_{\mathcal{G}}$, but it may also contain *some incorrect* answers or noise. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $s \in \mathcal{S}$:

  $$s^{\mathcal{C}} \supseteq {q_{\mathcal{G}}}^{\mathcal{B}}$$

  That is, $\forall \vec{x}(q_{\mathcal{G}}(\vec{x}) \rightarrow s(\vec{x}))$.

- If type(s) = exact, it means that the extension of s is exactly the set of the tuples satisfying $q_{\mathcal{G}}$. In other words, the extension of s contains *all* and *only* correct answers for $q_{\mathcal{G}}$. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $s \in \mathcal{S}$:

  $$s^{\mathcal{C}} = {q_{\mathcal{G}}}^{\mathcal{B}}$$

  That is, $\forall \vec{x}(s(\vec{x}) \leftrightarrow q_{\mathcal{G}}(\vec{x}))$.

The main advantage of this kind of mapping is related to scalability regarding the sources. In other words, based on this mapping approach, adding sources does

[2] If not specified type(s) = sound.

not require to change the global schema, which makes the process straightforward. On the other hand, since sources are described in isolation, query processing is not trivial (see section 4.4).

4.3.2 Global as view

In the GaV approach (also called global-view-centric) the enterprise model is expressed in terms of the sources, i.e., the mapping is composed of a set of assertions that relate each element of the enterprise model with a query over the source models. Given a source database $\mathcal{C}$, $\mathcal{M}$ provides direct information about which data satisfy the elements of the global schema. Relations in $\mathcal{G}$ are views, and queries are expressed over the views [29].

In an IIS $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ based on the GaV approach, the mapping $\mathcal{M}$ associates to each element g of $\mathcal{G}$ a query $q_{\mathcal{S}}$ over $\mathcal{S}$. The query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ only allows expressions representing *one* symbol of the alphabet $\mathcal{A}_{\mathcal{G}}$. Therefore, a GaV mapping is composed of a set of assertions, one for each element of $\mathcal{G}$, of the form:

$$g \rightsquigarrow q_{\mathcal{S}}$$

Example 4.3.2. Let $\mathcal{G}$ be composed of the relations:

$$\text{Play}(\textit{Play}, \textit{Sinopsis}, \textit{TheatreName}),$$
$$\text{Location}(\textit{TheatreName}, \textit{Address}).$$

Let $\mathcal{S}$ be composed of the following sources:

$$s_1 \colon \text{Theatre}(\textit{Name}, \textit{Play}, \textit{Location}),$$
$$s_2 \colon \text{Play}(\textit{Name}, \textit{Sinopsis}),$$

where source s_1 contains a list of theatres with the plays that are showing and their addresses, and s_2 contains the sinopsis of the plays.

Then, a GaV-based mapping $\mathcal{M}$ could be composed of the following assertions:

$$\text{Play}(P, S, TN) \rightsquigarrow \text{Theatre}(TN, P, L) \wedge \text{Play}(P, S),$$
$$\text{Location}(TN, A) \rightsquigarrow \text{Theatre}(TN, P, A).$$

Similarly, in order to better characterize $\mathcal{M}$, we redefine the function type so

that, in this case, it associates a mapping type to each element g of $\mathcal{G}$. Formally:

$$\text{type}\colon \mathcal{G} \rightarrow \{\text{sound}, \text{complete}, \text{exact}\}.$$

With type we can specify how accurate is the global element g with respect to the associated query $q_{\mathcal{S}}$, as follows:

- If type(g) = sound, it means that the tuples satisfying $q_{\mathcal{S}}$ are a subset of the extension of g. In other words, the answers of $q_{\mathcal{S}}$ correspond *only* to the extension of g, but *not necessarily all*. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $g \in \mathcal{G}$:

  $$q_{\mathcal{S}}{}^{\mathcal{C}} \subseteq g^{\mathcal{B}}$$

  That is, $\forall \vec{x}(q_{\mathcal{S}}(\vec{x}) \rightarrow g(\vec{x}))$.

- If type(g) = complete, it means that the tuples satisfying $q_{\mathcal{S}}$ are a superset of the extension of g. In other words, the answers of $q_{\mathcal{S}}$ contain *all* the extension of g, but they may also include other data that *are not* part of the extension of g. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $g \in \mathcal{G}$:

  $$q_{\mathcal{S}}{}^{\mathcal{C}} \supseteq g^{\mathcal{B}}$$

  That is, $\forall \vec{x}(g(\vec{x}) \rightarrow q_{\mathcal{S}}(\vec{x}))$.

- If type(s) = exact, it means that the tuples satisfying $q_{\mathcal{S}}$ are exactly the extension of g. In other words, the correct answers of $q_{\mathcal{S}}$ correspond to *all* and *only* the extension of g. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $s \in \mathcal{S}$:

  $$q_{\mathcal{S}}{}^{\mathcal{C}} = g^{\mathcal{B}}$$

  That is, $\forall \vec{x}(q_{\mathcal{S}}(\vec{x}) \leftrightarrow g(\vec{x}))$.

The main advantage of this approach is that typically query processing can be based on some sort of simple unfolding. Nevertheless, whenever a source changes or a new one is added, we need to change the enterprise model. In particular,

given a new source, we need to figure out all the ways in which it can be used in order to obtain data for the elements of the enterprise model.

4.3.3 GLaV

Finally, there is another approach which combines the expressive power of GaV and LaV called GLaV. In GLaV, the mapping $\mathcal{M}$ is constituted by a set of assertions of the form:

$$q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$$

Example 4.3.3. Let $\mathcal{G}$ be composed of the relations:

$$\text{Accommodation}(\textit{Name}, \textit{Type}, \textit{Classification}),$$
$$\text{Price}(\textit{Classification}, \textit{PriceRange}).$$

Let $\mathcal{S}$ be composed of the following sources:

$$s_1\colon \text{Hotels}(\textit{Name}, \textit{Classification}), \text{Price}(\textit{Name}, \textit{PriceRange}),$$
$$s_2\colon \text{Hostels}(\textit{Name}, \textit{Classification}, \textit{PriceRange}),$$

where sources s_1 and s_2 contain lists of hotels and hostels with their classifications and price ranges.

Then, a GLaV-based mapping $\mathcal{M}$ could be composed of the following assertions:

$$\text{Hotels}(N, C) \wedge \text{Price}(N, PR) \rightsquigarrow \text{Accommodation}(N, \textsc{Hotel}, C) \wedge \text{Price}(C, PR),$$
$$\text{Hostels}(N, C, PR) \rightsquigarrow \text{Accommodation}(N, \textsc{Hostel}, C) \wedge \text{Price}(C, PR).$$

Again, in order to better characterize $\mathcal{M}$, we redefine the function type so that, in this case, it associates a mapping type to each query $q_{\mathcal{S}}$ of $\mathcal{M}$. Formally:

$$\text{type}\colon \mathcal{G} \rightarrow \{\text{sound}, \text{complete}, \text{exact}\}.$$

With type we can specify how accurate is the query $q_{\mathcal{S}}$ with respect to the associated query $q_{\mathcal{G}}$, as follows:

- If $\text{type}(q_{\mathcal{S}}) = \text{sound}$, it means that the extension of $q_{\mathcal{S}}$ is a subset of the

tuples satisfying $q_{\mathcal{G}}$. In other words, the extension of $q_{\mathcal{S}}$ contains *only* correct answers for $q_{\mathcal{G}}$, but *not all*. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $q_{\mathcal{S}} \in \mathcal{M}$:

$$q_{\mathcal{S}}{}^{\mathcal{C}} \subseteq q_{\mathcal{G}}{}^{\mathcal{B}}$$

That is, $\forall \vec{x}(q_{\mathcal{S}}(\vec{x}) \rightarrow q_{\mathcal{G}}(\vec{x}))$.

- If type$(q_{\mathcal{S}})$ = complete, it means that the extension of $q_{\mathcal{S}}$ is a superset of the tuples satisfying $q_{\mathcal{G}}$. In other words, the extension of $q_{\mathcal{S}}$ contains *all* correct answers for $q_{\mathcal{G}}$, but it may also contain *some incorrect* answers or noise. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $q_{\mathcal{S}} \in \mathcal{M}$:

  $$q_{\mathcal{S}}{}^{\mathcal{C}} \supseteq q_{\mathcal{G}}{}^{\mathcal{B}}$$

  That is, $\forall \vec{x}(q_{\mathcal{G}}(\vec{x}) \rightarrow q_{\mathcal{S}}(\vec{x}))$.

- If type$(q_{\mathcal{S}})$ = exact, it means that the extension of $q_{\mathcal{S}}$ is exactly the set of the tuples satisfying $q_{\mathcal{G}}$. In other words, the extension of $q_{\mathcal{S}}$ contains *all* and *only* correct answers for $q_{\mathcal{G}}$. In this case, a global database $\mathcal{B}$ is an $\mathcal{M}$-model with respect to the source database $\mathcal{C}$ if for each element $q_{\mathcal{S}} \in \mathcal{M}$:

  $$q_{\mathcal{S}}{}^{\mathcal{C}} = q_{\mathcal{G}}{}^{\mathcal{B}}$$

  That is, $\forall \vec{x}(q_{\mathcal{S}}(\vec{x}) \leftrightarrow q_{\mathcal{G}}(\vec{x}))$.

Interestingly, any GLaV system can be transformed into a GaV system [12]. The idea is that a GLaV mapping assertion can be transformed into a GaV assertion plus an inclusion dependency in $\mathcal{G}$. That is, in order to transform a GLaV system into a GaV one, for each GLaV assertion of the form:

$$q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}},$$

we add a *new* element r to $\mathcal{G}$ and we associate to r the following GaV assertion:

$$r \rightsquigarrow q_{\mathcal{S}},$$

and the following inclusion dependency:

$$r \sqsubseteq q_{\mathcal{G}}.$$

4.4 Query processing

Queries from users and/or applications are posed to the IIS in terms of the enterprise model. Computing the set of answers for a given query (query processing), strongly depends on the way mappings are modeled. In the GaV approach, the problem generally resumes to some kind of unfolding [28], while in the LaV and GLaV approaches, query processing is closely related to the problem of *answering queries using views* [31, 40].

4.4.1 GaV

In case a GaV IIS does not allow integrity constraints in the enterprise model, and mapping assertions are sound or exact, query processing can be based on a simple unfolding strategy due to the fact that the system has the single database property [28]. Basically, given a query over $\mathcal{G}$, every element of the query is substituted with the corresponding query over the sources (specified by the corresponding mapping assertion), and the resulting query is then evaluated at the sources. Nevertheless, when the language $\mathcal{L}_{\mathcal{G}}$, used for expressing $\mathcal{G}$, allows for integrity constraints and the mapping assertions are sound, then query processing in GaV becomes more complex.

The problem is that given a source database $\mathcal{C}$ and a global database $\mathcal{B}$, obtained by populating $\mathcal{G}$ according to the mapping assertions, integrity constraints may be violated in $\mathcal{B}$. We can assume that the management of key constraints is left to the designer, nevertheless, the management of foreign key constraints cannot be left to the designer, since it is strongly related to the incompleteness of the sources. Now, since foreign keys are interrelation constraints, they cannot be captured with a GaV mapping, because, by definition, it works on each element of $\mathcal{G}$ in isolation. In this case, given the assumption of sound mapping assertions, a valid strategy to enforce foreign key constraints is to add tuples to $\mathcal{B}$ every time a foreign key constraint is violated. In summary, when $\mathcal{G}$ contains foreign key constraints, the semantics of $\mathcal{I}$ must be formulated in terms of a *set* of global databases, instead of a single one. Since we are interested in the certain answers

to a query, the existence of several such databases complicates the task of query answering.

4.4.2 LaV

Given the fact that in a LaV IIS, we only know the extensions of the views associated to the sources, answering queries in LaV can be regarded as an extended form of reasoning in the presence of incomplete information [41]. Moreover, since in LaV sources are modeled as views over $\mathcal{G}$, the problem of processing a query is traditionally called view-based query rewriting and it is very closely related to the problem of answering queries using views [31, 40] (see chapter 5). There are two approaches to view-based query processing, called *view-based query rewriting* and *view-based query answering* [28].

In the former approach, given a query over $\mathcal{G}$ and a set of LaV mapping assertions, the goal is to *rewrite* the query into an expression of a fixed query language $\mathcal{L}_\mathcal{R}$, over $\mathcal{S}$, that provides the answer to the given query. It may happen that there is no expression in the chosen language $\mathcal{L}_\mathcal{R}$ that is equivalent to the original query. In this case, the objective is to compute the *maximally contained rewriting*, i.e., the most similar expression to the original query in $\mathcal{L}_\mathcal{R}$. On the other hand, in view-based query answering, besides the query and the set of mapping assertions, we are also given the extensions of the views, i.e., the contents of the sources. The goal is now to compute the set of answers the are logically implied by the system. As can be seen, in a LaV context, this is exactly the problem of computing the certain answers with respect to a source database.

The difference between the two approaches is basically that in query rewriting, query processing is divided in two steps, first the original query is rewritten in terms of a given query language over the source models, and second the rewriting is evaluated. On the contrary, in query answering, there are not limitations on how queries are processed, and the only goal is to exploit all possible information in the system in order to compute the answer to the query.

Chapter 5

Answering queries using views

5.1 Introduction

The problem of answering queries using views is to find efficient methods of answering a query using a set of previously materialized views over a database, rather than accessing the database relations. This problem is relevant to a wide variety of data management problems: query optimization, maintenance of physical data independence, data warehouse design and information integration. We will concentrate on presenting the relationship between answering queries using views and information integration. For a survey on this and the other data management problems, see [31].

Mappings in a LaV IIS are defined as a set of views over the enterprise model. As a result, the problem of rewriting a query, posed in terms of the enterprise model, into a query that refers only to the source models becomes the problem of answering queries using views. In this context, however, the number of views (i.e., sources) tends to be very large, and the sources are typically not considered to contain complete information. Given a query q and a set of view definitions (LaV mappings) $V = v_1, \ldots, v_n$, a rewriting of q using the views is a query expression q' that refers only to a set of views $V' = v_1, \ldots, v_m$ $(V' \subseteq V)$.

5.2 Rewritings

There are two types of query rewritings: equivalent and maximally-contained [31]. The rewriting q' is an *equivalent rewriting* of q using V if:

- q' refers only to the views in V, and
- $q' \cup V$ is equivalent to q.

On the other hand, q' is a maximally-contained rewriting of q using V w.r.t. a given query language L if:

- q' is a query in L that refers only to the views in V,
- $q' \cup V$ is contained in q, and
- there is no rewriting $q_1 \in L$, s.t. $q' \cup V \subseteq q_1 \cup V \subseteq q$ and $q' \cup V$ is not equivalent $q_1 \cup V$.

5.3 Existing solutions

In this section we describe three different practical algorithms that have been proposed to solve the problem of answering queries using views. We will use the scenario presented in [31] to make the explanation easier:

Example 5.3.1. Let $\mathcal{M}$ be composed of the LaV assertions shown in table 5.1. We make the assumption that type(v_i) = sound $(1 \leq i \leq 4)$, i.e., we have sound mappings.

v_1(*Student*, *Number*, *Year*)	$\rightsquigarrow$	Registered(*Student*, *Course*, *Year*)$\wedge$ Course(*Course*, *Number*)$\wedge$ *Number* $\geq$ 500$\wedge$ *Year* $\geq$ 1992,
v_2(*Student*, *Department*, *Course*)	$\rightsquigarrow$	Registered(*Student*, *Course*, *Year*)$\wedge$ Enrolled(*Student*, *Department*),
v_3(*Student*, *Course*)	$\rightsquigarrow$	Registered(*Student*, *Course*, *Year*)$\wedge$ *Year* $\leq$ 1990,
v_4(*Student*, *Course*, *Number*)	$\rightsquigarrow$	Registered(*Student*, *Course*, *Year*)$\wedge$ Course(*Course*, *Number*)$\wedge$ Enrolled(*Student*, *Department*)$\wedge$ *Number* $\leq$ 100.

Table 5.1: An example scenario.

5.3.1 The bucket algorithm

The bucket algorithm was developed in the context of the Information Manifold system (see section 6.3). Its goal is to rewrite a query, posed in terms of the enterprise model, into a query that refers only to the available sources. The algorithm requires that both, the query and the sources, are described by conjunctive queries. Each conjunct is said to be a subgoal and may include atoms of arithmetic comparison predicates. When the query does not contain any arithmetic comparison predicates the bucket algorithm is guaranteed to return the maximally contained rewriting of the query using the views [32]. The main idea underlying the bucket algorithm is that the number of query rewritings that need to be considered can be reduced if each subgoal in the query is first considered in isolation to determine which views may be relevant to each subgoal.

Given a query q, the algorithm proceeds in two phases. First, the algorithm creates a so-called bucket for each query subgoal that is not part of the comparison predicates, containing the views that are *relevant* to answering the particular subgoal. Informally, a view can be useful for a query if the set of relations it mentions overlaps of that of the query, and it selects some of the attributes selected by the query. Moreover, if the query applies comparison predicates to attributes that it has in common with the view, then the view must apply either equivalent or logically weaker predicates. More formally, a view v is considered to be relevant for a given subgoal g if the definition of v contains a subgoal g_1 such that g and g_1 can be unified through a variable mapping, and after unifying the query and the view, the predicates in q and in v are mutually satisfiable. The actual bucket contains the head of v. Nevertheless, if a subgoal g unifies with more than one subgoal in v, its bucket will contain multiple occurrences of v.

Suppose our query q is:

$$q(S, D) : -\text{Enrolled}(S, D) \wedge \text{Registered}(S, C, Y) \wedge \text{Course}(C, N) \wedge \\ N \geq 300 \wedge Y \geq 1995.$$

In the first step the algorithm creates a so-called bucket for each of the query subgoals in q. Table 5.2 shows the resulting contents of the buckets.

The bucket of Enrolled(S, D) includes v_2 and v_4 because there is a mapping unifying the subgoal in q with the corresponding subgoal: $\{S \rightsquigarrow \mathit{Student}, D \rightsquigarrow \mathit{Department}\}$. The bucket of Registered(S, C, Y) contains v_1, v_2 and v_4 because,

Enrolled(S, D)	Registered(S, C, Y)	Course(C, N)
$v_2(S, D, C')$	$v_1(S, N', Y)$	$v_1(S', N, Y')$
$v_4(S, C', N')$	$v_2(S, D', C)$	
	$v_4(S, C, N')$	

Table 5.2: The resulting buckets. Each view head in a bucket only includes variables in the domain of the mapping. Fresh variables are used for the other head variables of the source.

similarly, there is a unifying mapping: $\{S \rightsquigarrow \mathit{Student}, C \rightsquigarrow \mathit{Course}, Y \rightsquigarrow \mathit{Year}\}$. v_3 is not in the bucket of Registered(S, C, Y) because the comparison predicates $Y \geq 1995$ and $\mathit{Year} \leq 1990$ are inconsistent. The bucket of Course(C, N) includes source v_1 because of the mapping $\{C \rightsquigarrow \mathit{Course}, N \rightsquigarrow \mathit{Number}\}$. In this case v_4 is not included in because the mapping maps N to *Number*, and the comparison predicate of v_4 on the course number would be mutually inconsistent.

In the second phase, the algorithm builds a set of query rewritings, as conjunctive queries, by taking one conjunct from every bucket. For each possible combination of the cartesian product of elements of the buckets, the algorithm checks whether it is contained in the original query q. If the rewriting is contained in q or can be made to be so by adding comparison predicates, then it is added to the answer. The result of the bucket algorithm is a union of conjunctive queries.

Consider the rewriting that includes the first element of each bucket:

$$q'(S, D) : -v_2(S, D, C') \wedge v_1(S, N', Y) \wedge v_1(S', N, Y').$$

Although this rewriting is not contained in the original query, it can be made so by adding the comparison predicates $N = N'$ and $Y \geq 1995$. Therefore, the following is the first rewriting that is obtained by the algorithm[1]:

$$q'(S, D) : -v_1(S, N, Y) \wedge v_2(S, D, C') \wedge Y \geq 1995.$$

The only other interesting rewriting to consider would involve a join between v_1 and v_4. Nevertheless, such a rewriting would be dismissed because the two sources contain disjoint numbers of courses.

[1] Y and Y' have been equated as an additional optimization.

5.3.2 The inverse-rules algorithm

The inverse-rules algorithm was developed in the context of the Infomaster system (see section 6.6). The main idea of this approach is to construct a set of rules that invert the view definitions or mappings.

Example 5.3.2. Consider the following mapping:

$$v_5(\mathit{Department}, \mathit{Course}) : -\text{Enrolled}(\mathit{Student}, \mathit{Department}) \wedge \\ \text{Registered}(\mathit{Student}, \mathit{Course}).$$

From the mapping above, one inverse rule for every conjunct in the body of the view is constructed:

$$\text{Enrolled}(f_1(\mathit{Department}, X), \mathit{Department}) : -v_3(\mathit{Department}, X), \\ \text{Registered}(f_1(Y, \mathit{Course}), \mathit{Course}) : -v_3(Y, \mathit{Course}).$$

This set of rules intuitively means that a tuple in the extension of the view v_3 is a witness of tuples in the relations Enrolled and Registered. In other words, a tuple of the form $\langle \mathrm{D}_1, \mathrm{C}_1 \rangle$ tells us that the relation Enrolled contains a tuple of the form $\langle \mathrm{Z}, \mathrm{D}_1 \rangle$, for some Z, and the relation Registered contains a tuple of the form $\langle \mathrm{Z}, \mathrm{D}_1 \rangle$ for the same Z.

In such rules one function symbol is created for every existential variable that appears in the view definitions. These function symbols are used in the heads of the inverse rules, nevertheless the resulting rewriting can be rewritten in such a way that no functional terms appear. The rewriting of a query q using the set of views v is the datalog program that includes the inverse rules for v, and the query q. The inverserules algorithm returns the maximally contained rewriting of q using v [21].

Example 5.3.3. Consider the following query asking for the departments in which the students of the "Database" course are enrolled:

$$q(\mathit{Department}) : -\text{Enrolled}(\mathit{Student}, \mathit{Department}) \wedge \\ \text{Registered}(\mathit{Student}, \textsc{Database}).$$

Now, let the source v_3 include the following tuples:

$$\{\langle \text{CS}, \textsc{Database}\rangle, \langle \text{EE}, \textsc{Database}\rangle, \langle \text{CS}, \text{AI}\rangle\}.$$

The inverse rules would compute the following tuples:

$$\begin{aligned}
\text{Registered} : \{&\langle f_1(\text{CS}, \textsc{Database}), \text{CS}\rangle,\\
&\langle f_1(\text{EE}, \textsc{Database}), \text{EE}\rangle,\\
&\langle f_1(\text{CS}, \text{AI}), \text{CS}\rangle\},\\
\text{Enrolled} : \{&\langle f_1(\text{CS}, \textsc{Database}), \textsc{Database}\rangle,\\
&\langle f_1(\text{EE}, \textsc{Database}), \textsc{Database}\rangle,\\
&\langle f_1(\text{CS}, \text{AI}), \text{AI}\rangle\}.
\end{aligned}$$

Applying the query to these extensions would yield the answers CS and EE.

5.3.3 The MiniCon algorithm

The MiniCon algorithm [38] begins like the bucket algorithm, considering which views are relevant to the query subgoals. However, once the algorithm finds a possible matching between a subgoal g in the query to a subgoal g_1 in a view V, it changes perspective and looks at the variables in the query. The algorithm considers the join predicates in the query (which are specified by multiple occurrences of the same variable) and finds the minimal additional set of subgoals that need to be mapped to subgoals in V, given that g will be mapped to g_1. This set of subgoals and mapping information is called a MiniCon Description (MCD). In the second phase, the algorithm combines the MCDs to produce the rewritings. It is important to note that because of the way MCDs are constructed, the MiniCon algorithm does not require containment checks in the second phase, giving it an additional speedup compared to the bucket algorithm.

The experiments published in [38] show that:

- The MiniCon algorithm scales up to large numbers of views and significantly outperforms the other two algorithms. Moreover, in various situations the algorithm can handle thousands of views, which is a scale that is out of reach of previous algorithms.

- The bucket algorithm performs much worse than the other two algorithms in all cases.
- All three algorithms are limited in cases where the number of resulting rewritings is especially large since a complete algorithm must produce a potentially exponential number of rewritings.

Chapter 6

Existing information integration systems

6.1 Carnot

Carnot [19] is one of the first attempts to use logic-based technologies for solving the problem of information integration. Its objective is to develop a method for integrating independent information sources in order to allow these sources to be accessed and *modified* coherently. Carnot implements virtual, declarative, and centralized information integration based on a GaV approach with exact mappings between *atomic* concepts or roles.

Distinctively, Carnot does not require the development of an enterprise model over a set of independent source models, for it uses the Cyc knowledge base [27] as such. The enterprise model and each of the source models are expressed in the global context language *GCL* of Cyc (which is based on extended first-order logic). Moreover, each source has a specific local data manipulation language (e.g., SQL). Mappings between the enterprise model and the source models consist of two parts, on one hand a syntax correspondence provides a bidirectional translation for concepts of a specific source model(local schema) between *GCL* and the local data manipulation language of the corresponding source. On the other hand, a semantic correspondence takes the the form of a set of statements of equivalence between concepts of the enterprise model and concepts of the source model(local schema) called *articulation axioms*.

When a source is to be added to the system, it is necessary to develop a set of articulation axioms in a three-phase process. The fist step is to build a source

model(local schema) expressed in *GCL* that models the contents of the source (in this case, the source model(local schema) is a Cyc context or microtheory [25]). Secondly, the concepts of the built source model(local schema) are matched with corresponding concepts in the enterprise model[1]. Finally, these matches are converted into a set of articulation axioms, which then will be used to rewrite queries asked to the system. Such queries can be expressed using *GCL* or any local data manipulation language. The objective is that applications and/or users do not necessarily have to use the enterprise model (and learn/use Cyc's *GCL*) to query the system.

Given a query expressed in the source specific manipulation language, the system processes it in the following manner: First, the query is *syntactically* translated into a concept expressed in *GCL* over the source model(local schema). Then, using the set of articulation axioms associated to the resulting local concept, the system *semantically* translates it into a concept over the enterprise model. After that, the system uses the set of articulation axioms associated to the resulting global concept in reverse to translate it into a set of queries over different source models. Finally, each of these local queries is syntactically translated into its corresponding data manipulation language, and sent to the corresponding source for execution. In case the original query is expressed over the enterprise model, the system uses the set of articulation axioms associated to it to translate it into a set of queries over different source models. Similarly, each of these local queries is then syntactically translated into its corresponding data manipulation language and sent to the corresponding source for execution.

6.2 TSIMMIS

TSIMMIS (The Standford-IBM Manager of Multiple Information Sources) [17] is a project dedicated at developing tools that facilitate the rapid integration of heterogeneous sources that may include both structured and unstructured data. Distinctively, its goal is not to perform fully automated information integration, but rather to provide a framework and tools to assist users in their information processing and integration tasks. TSIMMIS implements virtual, procedural, and peer-to-peer information integration based on a GaV approach with exact mappings.

[1] In case there are not matching global concepts for the given local concepts, they are created.

The content of each source is modeled using the so-called Object Exchange Model (OEM), a simple self-describing object model. The fundamental idea is that all objects, and their subobjects, have labels that define their meaning. Each source is associated with a *translator* (wrapper) that logically converts the underlying data to OEM objects. Above wrappers lie the *mediators* which refine the information from one or more sources. A mediator embeds the knowledge that is necessary for processing a specific information request. Mappings between different source models are hardcoded withing the mediators. In other words, the way of dealing with previously identified queries is embedded in mediators, which in turn use wrappers in order to retrieve data from the sources. This approach can be regarded as being too ad hoc to be scalable; to this respect, one of the goals of TSIMMIS is to automatically or semi-automatically generate mediators from high level description of the processing they are supposed to do.

Queries posed to the system are expressed in OEM-QL, an SQL-like language extended to deal with labels and object nesting. Mediators and wrappers take as input OEM-QL queries and return OEM objects. Hence, the system forms an interconnected network of mediators and wrappers where users and mediators can obtain data either from wrappers or from other mediators. In particular, users can access the sources through the system either by writing applications that request OEM objects to a subset of mediators/wrappers, or by using of the generic browsing tools TSIMMIS provides. In such browsers, the user writes a query or selects one predefined query. If the submitted query is valid and successfully executed by one one of the system's mediator or wrapper, the answer object is returned to the user. The answer is received as a tree, whose root shows one or more levels of the answer object, with links available to take the user to portions of the answer that did not appear in the root.

Another key point is that TSIMMIS is able to handle integrity constraints. Such constraints specify semantic consistency requirements over data. In this context, i.e., a loosely coupled environment, however, it is generally not possible to guarantee that every user or application sees consistent data every time it interacts with the system. To this respect, TSIMMIS enforces constraints with weaker or so-called relaxed guarantees. By enforcing such relaxed guarantees, it is possible to know what holds and what does not, and when.

6.3 Information Manifold

The Information Manifold (IM) system [32] objective is to provide uniform access to a heterogeneous collection of sources (most of which located in the Web). One of the most important characteristics of the system is that it provides a mechanism to declaratively describe, not only the contents of the sources, but their query capabilities which enables expressing fine-grained distinctions between different sources. IM implements virtual, declarative, and centralized information integration based on a LaV approach with sound mappings.

The enterprise model is represented using CLASSIC [10, 18], i.e., the relational model augmented with certain object-oriented features (e.g., classes, attributes). Relations contain tuples while classes contain objects. In order to be able to treat relations and classes uniformly, a class is represented with a unary relation and its attributes with a set of binary relations. Similarly, the content of the sources is also modeled as a set of tuples in one or more relations, or a set of objects in one or more classes. Each source is regarded as containing tuples of a certain source relation (whose name does not belong to the set of names of the global relations). Each source relation is specified in terms of a conjunctive query over the relations of the enterprise model, describing the conditions the tuples in the source relation must satisfy. Distinctively, since the considered sources may not be able to answer arbitrary queries, each source is annotated with its query capabilities. A capability record specifies which inputs can be given to the source, the minimum and maximum number of inputs allowed, the possible outputs of the source, and the selections the source can apply.

Queries asked to the system are conjunctive queries over the enterprise model. Given a query, the system computes a so-called query plan, which in general terms represents a sequence of accesses to sources interesped with local processing operations. The query plan combines data from different sources in a way that guarantees semantically correct answers, while adhering to the various source capabilities. In more detail, a query plan is a set of conjunctive plans, where a conjunctive plan is simply a conjunctive query annotated with the inputs and outputs of every subgoal. A conjunctive plan is said to be semantically correct if its expansion, obtained by expanding the definitions of its subgoals using the mappings, is contained in the query posed to the system. Moreover, a conjunctive plan is said to be executable if it respects the various source capabilities. Finally, the answer to a query is defined as the set of tuples that can be obtained by some

executable and semantically correct conjunctive plan for it.

In order to generate an executable and semantically correct query plan for a given query, the system proceeds in two stages: first, the system generates a set of semantically correct conjunctive plans, and then it tries to order the conjuncts of each of the plans to ensure they are executable. The generation of semantically correct conjunctive plans amounts to finding a conjunctive query that uses only the source relations and is contained in the given original query. This problem is closely related to the problem of answering queries using views (see section 5), where the source relations are considered to be *sound* views. Given a query, the system implements the Bucket algorithm (see section 5.3.1) to obtain a set of semantically correct conjunctive plans. For each of these conjunctive plans the system then tries to order its subgoals to make it executable. For any conjunctive plan, if there is an ordering of its subgoals to make it executable, it is guaranteed to be found in a polynomial time in the size of the plan.

6.4 SIMS

The objective of the SIMS (Single Interface to Multiple Sources) system [4] is to access and integrate information from multiple sources, which can be relational databases or LOOM knowledge bases (for a full description of LOOM see [34]). One of its main characteristics is that it applies a variety of techniques and systems from Artificial Intelligence (e.g. planning) to build an intelligent interface to the sources. SIMS implements virtual, declarative, and centralized information integration based on a LaV approach with sound mappings.

The enterprise model and each of the source models are expressed in the LOOM knowledge representation language[2]. The enterprise model's collection of terms forms the vocabulary used to characterize the contents of the local sources. When a source is to be added to the system, it is necessary to build its model as a local LOOM model including every fact that can influence decisions concerning when and whether to use it. Such facts, besides the specification of the content, include among others whether the source is originally a LOOM knowledge base, the size of the source, and its location. Once the model is built, it is then *merged* to the enterprise model by asserting relations (e.g. is-a) between matching local

[2]LOOM combines features of both framework-based and semantic network languages, and provides reasoning facilities.

concepts and global concepts.

Queries can be posed using either global or local concepts. In any case, queries asked to the system are LOOM expressions (refering to a set of concepts). Given a query, the system identifies the relevant sources for each of the query concepts. Identification of relevant sources is straightforward in case each query concept has a set of corresponding matching local concepts. Nevertheless, when this is not the case, the system obtains a set of reformulations, that enable relevant sources to be identified, for every query concept that does not have a matching local concept. Once relevant sources have been identified, the system translates the original query into a set of local queries (which refer only to local concepts) using the asserted relations between global and local concepts. An optimized plan, or ordered sequence of the resulting local queries, is then created an executed. Put simply, given a query, the system generates a collection of local queries and then executes an optimized plan for accessing the appropriate sources.

Distinctively, SIMS uses two independent systems to achieve integration. On one hand, in order to identify relevant sources and order the sequence of a set of resulting local queries (i.e., generating the plan) SIMS uses a a means-ends analysis planner called Prodigy [16]. Prodigy has been linked to LOOM, so it can use the enterprise model as its model of the world. SIMS simply formulates the identification of relevant sources and the ordering of local queries as planning problems and passes them to Prodigy. On the other hand, in order to perform data retrieval SIMS uses the LOOM Interface Module (LIM) [35]. LIM converts a database schema into a LOOM model, and given a LOOM query grounded to a single database, automatically translates it into the appropriate database query language, executes it in the corresponding source, and returns the results as if they were LOOM instances. After SIMS has generated a set of local queries for a given query, it passes the subset of local queries that are grounded in a single database to LIM for the actual retrieval. The local queries that are grounded to a LOOM knowledge base are handled by SIMS itself.

6.5 OBSERVER

The OBSERVER (Ontology Based System Enhanced with Relationships for Vocabulary hEterogeneity Resolution) system [36] uses multiple pre-existing ontologies to access heterogeneous, distributed and independently developed sources.

Distinctive characteristics include a mechanism to estimate the possible loss of information that may occur when rewriting queries, and the exploitation of semantic interontology relationships. OBSERVER implements virtual, declarative, and peer-to-peer information integration with sound, exact and complete mappings between source models.

The content of each source is described by one or more ontologies (source models) expressed using the CLASSIC Description Logic [10, 18]. Each source model(local schema) is associated with a set of correspondences between DL expressions and the query language of the corresponding source[3]. As OBSERVER follows a peer-to-peer information integration approach, there is not an enterprise model. Hence, mappings must be defined between the concepts of different source models. Such mappings can be either synonym, hyponym or hypernim relationships, meaning that a mapping is either an equivalence or an inclusion axiom between a pair of concepts of different source models.

Queries posed to OBSERVER are CLASSIC expressions over one of the source models. OBSERVER uses inference to classify the query and determine relevant data repositories. The system then translates the query to the local query languages of the sources by using the correspondences between the schema and the underlying query language. If the user is not satisfied with the answer provided by that specific source, the scope of query can be expanded by selecting a set of target source models. Once a target source model(local schema) is selected, the query needs to be expressed over it. The system uses synonym mappings between the original source model(local schema) and the target source model(local schema) in order to substitute all the query concepts it can, by equivalent concepts of the target source model(local schema). If there are not synonym mappings for a subset of the query concepts, then the translation is only partial. The system deals with partial translations in two ways: (1) A partial translation can be combined with the partial translations obtained for other target schemata, such that the non-translated concepts may be translated across multiple schemata. (2) Each non-translated concept is substituted with the intersection of its immediate parents or by the union of its immediate children.

The resulting query can now be used to retrieve underlying data. Partial translations often result in a loss of information, however. To this respect, OBSERVER

[3]Please note that these correspondences are *not* the mappings between different source models.

proposes measures to estimate the resulting loss of information. Moreover, based on this estimation, the system chooses the translation that minimizes the loss of information. If the user is still not satisfied with the results, the process repeats itself with a new target source model(local schema).

6.6 Infomaster

Infomaster [23, 20] is an information integration system that provides integrated access to multiple distributed heterogeneous sources on the Internet. The core of the system is a so-called facilitator that determines which sources contain the information necessary to answer a given query efficiently, designs a strategy for answering it, and performs translations to convert data to a common form. Infomaster implements virtual, procedural and centralized information integration based on a LaV approach with sound and exact mappings between the enterprise model and the source models, and exact mappings between the enterprise model and the query models.

Distinctively, Infomaster makes use of various query models besides the usual use of the enterprise model and the underlying source models. Each query model represents a WWW form users can use to enter queries to the system, and it is represented with a so-called *interface* relation. On the other hand, the enterprise model is represented as a set of *base* relations, which should be chosen to be the basic 'building blocks' of a given application domain. Similarly, each source model is represented with a set of *site* relations. Both, the interface relations and the site relations are expressed in terms of the base relations.

Queries posed to the system are conjunctive queries over one of the query models. Query processing in Infomaster is a three-step process: The first step, called reduction, consists of rewriting the query into a query in terms of the enterprise model by simply unfolding its definition in terms of the base relations. In the second step, called abduction, the descriptions of the source models have to be used to translate the rewritten query into a query in terms of a subset of relevant source models. This second step is strongly related to the problem of answering queries using views (see section 5) and it is based, as it names implies, on the notion of abduction. In general, given a logical theory (composed of the description of the source models), a syntactic restriction (given the fact that we need the final rewriting to be in terms of the source models), and a

query in terms of the enterprise model; a rewritten query is considered to be an abduct of the initial one under the logical theory and the syntactic translation if: (i) the rewritten query is in terms of the source models, (ii) the logical theory including the rewritten query is consistent, and (iii) the logical theory including the rewritten query implies the initial query. Abduction is implemented in the Infomaster system using a standard model elimination prover. After abduction, the resulting query is an executable query plan, because it only refers to data that is actually available at the sources, nevertheless it could be inefficient. Therefore in the third and last step, the query plan is optimized using the descriptions of the corresponding sources. In the optimization step, the system tries to eliminate redundant source accesses, and to group the query plan in order to avoid querying the same information twice.

6.7 DWQ

The DWQ (Data Warehouse Quality) system [15, 14] is used in the context of data warehouses. In DWQ, the objective of source integration is to represent the migration of the data from the sources to a data warehouse, in order to support the design of (materialized) views that meet user requirements. DWQ implements materialized, declarative, and centralized information integration based on a GLaV approach with sound, complete and exact mappings. Distinctively DWQ includes the modeling of sound, exact and complete mappings between source models.

The enterprise model and the set of source models are expressed with $\mathcal{DLR}_{reg}$ [15]. Mappings are expressed as a set of subsumption or equivalence assertions between concepts from different models. Since DWQ is based on a materialized approach, actual data are not in the sources but reside in a set of relational schemata managed by the system at the logical level. Such schemata are intended to provide a structural description of the content of the sources (and the materialized views). The connection to the conceptual level is established by defining each relation as a relational query over the elements of the enterprise model or the source models. Finally for every relation there is a specification on how the tuples of such a relation should be constructed from a suitable set of tuples extracted from the sources. This last specification is crucial in order to load data into the data warehouse and perform data refreshment.

Queries to the system are $\mathcal{DLR}_{reg}$ concepts over the enterprise model or the source models. For each given query, the system verifies whether and how the answer can be computed from the source schemata (and materialized views). This problem is known as the query rewriting problem, which amounts to find a way to rewrite the original query into another in terms of the relations of the source schemata (or the materialized view schema). Although the system does not have a method for automatically rewriting the query, it exploits query containment checking in order to support the designer in this task. If the information contained in the various schemata at the logical level is not enough, the system starts the so-called client-driven integration. The idea is to verify whether the answer can be obtained by materializing new concepts represented in the enterprise model or the source models. In the case where neither the materialized data nor the concepts in the models are sufficient, the necessary data should be searched for in new sources, or in new portions of already analyzed sources.

Every time a new source is to be added to the system, the system starts the so-called source-driven integration process: First the corresponding source model is built and integrated into the enterprise model by adding a set of assertions between concepts of the new source model and concepts of the enterprise model or the source models. Note that the integration of a new source model can lead to changes both to the existing source models and to the enterprise model. After the first step, the source schema corresponding to the new source is produced and populated from the sources. In order to populate the new schema, the actual data have to be loaded into the system from the corresponding source. During the transfer of data, possible inconsistencies and redundancies are resolved, so that the system is able to provide and integrated and reconciled view of the data.

6.8 PICSEL

PICSEL [24] is a combination of the GaV and the LaV approaches. While it makes us of GaV mappings to relate the enterprise model with the underlying source models, it provides the mechanism to specify a restricted form of views that help characterizing the data contained in the sources. PICSEL implements virtual, declarative and centralized information integration based on a hybrid GaV-LaV approach with sound mappings.

CARIN [33], a combination of function-free Horn rules and $\mathcal{ALN}$ concept

expressions, is used to represent both the enterprise model and the set of source models. Each source is characterized by a set of source relations, and described by a CARIN knowledge base (source model) which contains a set of rules that indicate what kind of data can be found in the source, and a set of terminological and integrity constraints on the instances of the source relations. Such constraints enable a better characterization of the data residing at the sources. Although mappings between the enterprise model and the various source models follow a GaV approach, the set of constraints associated to the source models allow the expression of some kind of views (i.e., those expressible by $\mathcal{ALN}$ expressions). As a result, PICSEL combines a GaV approach with a restricted LaV approach. The reason for this limitation is that, as has been shown in [7], a full LaV approach in the setting of CARIN, does not guarantee decidability of query answering.

Queries are non-recursive CARIN rules (i.e. union of conjunctive queries over $\mathcal{ALN}$ expressions) over the enterprise model. Answering queries in PICSEL resorts to find conjunctions of source atoms (rewritings) which entail the initial query. A rewriting of a given query can be regarded as a specialized query plan that can be directly executed on the sources, and that provides answers to the initial query. In other words, the idea is to compute a representative set of all the possible rewritings of the original query, in order to get all the possible answers that can be obtained for it by accessing the available sources. For doing so, given a query, the system proceeds in two steps: query expansion and rewriting verification.

The system expands the original query to get a set of rewritings. In the first step, query expansion is achieved by using standard backward-chaining on the rules of the enterprise model in order to get a set of *ordinary* expansions for every conjunctive query composing the original query. An ordinary expansion is a set of atomic concepts and atomic roles over the enterprise model. Then, the atoms composing each ordinary expansion are again expanded using mappings, obtaining a set of *terminal* atoms. Terminal atoms either belong to a source model or, in case because they cannot be logically derived from the sources, to the enterprise model. Clearly, terminal expansions that consider only terminal atoms from the source models are valid rewritings. However, it is more subtle to understand that terminal expansions that contain atoms that do not belong to any source model, may still provide valid rewritings. In this case, if such atoms are only atomic concepts, it may be the case that their terminal expansion

still entails the original query without them. The conjunction of the remaining atoms for a given terminal expansion is considered a *candidate* rewriting. After expanding the query, for each candidate rewriting, the system checks whether it is compatible with the corresponding integrity constraints, and whether it actually entails the initial query. After the validation process, the remaining rewritings can be sent to the corresponding sources for execution.

Query processing in PICSEL is complete in the sense that the query plans, which are obtained from the query expansion step and which are checked as being valid rewritings for the verification step, completely characterize the set of rewritings of the query. Hence, the union of the answers resulting from executing such query plans provides the set of answers that can be obtained from the available sources.

6.9 TAMBIS

The Transparent Access to Multiple Bioinformatics Information Sources (TAMBIS) system [6] uses a so-called domain ontology for molecular biology and bioinformatics to integrate a set of relevant sources. TAMBIS implements virtual, declarative, centralized information integration based on a GaV approach with exact mappings.

The enterprise model is modeled using the GALEN Representation and Integration Language (GRAIL) [1] which allows the use of standard reasoning services. The enterprise model in TAMBIS, however, is more than the union of concepts of the underlying sources for it provides an abstract framework for relating, reconciling, and coordinating such concepts. Distinctively, considered sources are frequently not databases in the sense that they do not have a separate schema containing their metadata, and they do not have a declarative query language like SQL. On the contrary, most are tools, processes, or proprietary flat file structures containing embedded metadata. Sources are encapsulated by wrappers described in the functional multidatabase Collection Programming Language (CPL) [11]. Mappings relate the wrapper services in the sources with their conceptual counterparts in the enterprise model constituting the so-called Sources and Services Model (SSM).

Queries are GRAIL concept expressions of the enterprise model, hiding the sources from the user. Queries are posed to the system through a Web-based

query formulation and ontology browsing interface which ensures that only coherent queries can be expressed. When a query is posed to the system, TAMBIS executes a query translation (rewriting) and planning process that identifies appropriate sources, plans an efficient way of executing the query, and generates an execution plan for use with a middleware layer (the set of wrappers). The enterprise model is used during the query rewriting process as a semantic index to the wrapper methods, and the subsumption mechanism is used to select the most specialized mapping available. Distinctively, source-independent rewrite rules govern the choice of mappings and how the query components are combined.

6.10 TAMBIS II

TAMBIS II [37] is derived from TAMBIS and it addresses some of TAMBIS limitations. Nevertheless, as its predecessor, it implements virtual, declarative, centralized information integration based on a GaV approach with exact mappings.

The main differences with TAMBIS are as follows: (1) TAMBIS II uses the more expressive Description Logic $\mathcal{ALCQI}$ (see [5]) instead of GRAIL for modeling the enterprise model and expressing queries. This allows domains to be described more precisely in the enterprise model and allows more precise questions to be asked. (2) The reasoning services of the DL languages (i.e. $\mathcal{ALCQI}$) are used extensively during query processing to support semantic query optimization based on axioms within the enterprise model. This allows relationships between the sources and the enterprise model to be described in a declarative manner, which can then exploited by the system to detect redundant queries or to allows the use of multiple sources for the same kind of data. Finally (3) the sources are wrapped using an object model and are presented to the rest of system as being an object database, conforming to the ODMG data model [8].

6.11 SomeWhere

SomeWhere [3] is a semantic peer-to-peer information integration system that is based on simple personalized ontologies distributed at a large scale. It is based on a 'small is beautiful' philosophy in which no user imposes to others his/her own peer schema (source model). SomeWhere implements virtual, declarative, and

peer-to-peer information integration with sound, exact and complete mappings between source models.

The various source models are represented with a simple class-based data model in which the data is a set of resource identifiers, the schemata are simple definition of classes possibly constrained by inclusion, disjunction or equivalence statements. Such data model corresponds to the propositional fragment of the OWL ontology language [26]. As SomeWhere follows a peer-to-peer information integration approach, there is no notion of enterprise model. Hence, mappings must be defined between the concepts of different source models. Such mappings can be either an inclusion, equivalence or a disjunction axiom between a pair of concepts of different source models. Every source model is associated with a set of *extensional* concepts defined in terms of atomic classes of the model. Such extensional concepts are used to specify the data that is stored logically in a given peer. Axioms defining extensional concepts are restricted to be inclusion statements between an atomic extensional class and a description combining atomic concepts of the source model. In SomeWhere a new peer joins the system network through some peers that it knows (called acquaintances) by declaring mappings between its own source model and the source models of its acquaintances.

Queries are posed in terms of a given source model, answers to a query are not only instances of concepts of the local source model, but possibly instances of concepts from other source models. The systems rewrites a given query into a combination of extensional classes of different relevant source models. In other words, extensional concepts of several source models can participate in a rewriting, and thus to the answer of a query posed to a given source model. In this setting, query rewriting can be reduced to distributed reasoning over logical propositional theories by a straightforward propositional encoding of the distributed model of the system's network.

6.12 Summary

In this section we summarize the main characteristics of the analyzed IISs. For each of the considered systems, table 6.1 shows the information integration approach[4] that was implemented, the type of mappings[5] that was modeled, and the

[4]V = Virtual, M = Materialized, D = Declarative, P = Procedural, C = Centralized, P2P = Peer-to-peer.

[5]S = Sound, C = Complete, E = Exact.

KR formalism that was used to represent the system.

System	**Approach**	**Mappings**	**KR formalism**
Carnot	V, D, C	GaV E	Cyc GCL
TSIMMIS	V, P, P2P	E	OEM
IM	V, D, C	LaV S	CLASSIC
SIMS	V, D, C	LaV S	LOOM
OBSERVER	V, D, P2P	S, C, E	CLASSIC
Infomaster	V, P, C	LaV S, E	Relational model
DWQ	M, D, C	GLaV S, C, E	$\mathcal{DLR}_{reg}$
PICSEL	V, D, C	GaV-LaV S	CARIN
TAMBIS	V, D, C	GaV E	GRAIL
TAMBIS II	V, D, C	GaV E	$\mathcal{ALCQI}$
SomeWhere	V, D, P2P	S, C, E	Propositional OWL

Table 6.1: Main characteristics of analyzed IISs.

Existing systems implementing GaV include [19, 17, 6, 24, 37]. LaV, on the other hand, is implemented in [4, 32, 23, 20], mainly focusing on query rewriting. GLaV has received less attention, and, on the contrary, current efforts on this matter [15] mainly focus on query answering. Summing up, the design and implementation of a GLaV IIS that effectively solves the problem of view-based query rewriting remains an open problem.

Chapter 7

Query rewriting: The first experience

7.1 Scenario

We focus on solving the problem of view-based query rewriting (see section 4.4.2) for centralized, declarative and virtual information integration with GLaV mappings. Our approach is centralized since we consider IISs with one enterprise model, a set of source models, and a set of GLaV mappings between them. It is declarative since we make use of DLs for representing the system components. Finally it is virtual because we make the assumption that all data reside in relevant sources.

We make the simplifying assumption that user queries are atomic concepts over the enterprise model. Moreover, the enterprise model and the set of source models are considered to be acyclic $\mathcal{ALC}$[1] TBoxes composed of a set of limited concept definitions[2]. We describe the required limitations in the following section. In general the mappings, on the other hand, are composed of a set of expressions of the form: $C \sqsubseteq D$, where C is a concept expression over the source models, and D is a concept expression over the enterprise model. Given the fact that both, the enterprise model and the set of source models, are sets of concept definitions, it is possible to substitute every concept name in the mappings for its corresponding definition in the corresponding model. After making such a substitution, we only

[1] A DL allowing for concept conjunction, disjunction, negation, and universal and existential quantification [5].

[2] A concept definition is an equality axiom whose left-hand side is an atomic concept.

need to take into account the mappings to rewrite a given query.

7.2 Definitions

Let $\mathcal{G} = N_{gc} \cup N_{gr}$ $(N_{gc} \cap N_{gr} = \emptyset)$ be a global signature, where N_{gc} is a finite set of global concept names and N_{gr} is a finite set of global role names. Let $\mathcal{L} = N_{lc} \cup N_{lr}$ $(N_{lc} \cap N_{lr} = \emptyset)$ be a local signature, where N_{lc} is a finite set of local concept names and N_{lr} is a finite set of local role names. $\mathcal{G} \cap \mathcal{L} = \emptyset$.

An ABox $\mathcal{A}$ is said to be *local atomic* if it is composed of a set of assertions of the form:

$$a\!:\!C \quad \text{or} \quad (a,b)\!:\!R$$

where a, b are individual names, $C \in N_{lc}$, and $R \in N_{lr}$.

An axiom of the form $C \sqsubseteq D$ is said to be a *valid concept mapping* if C is a concept expression over $\mathcal{L}$ where the symbols $\neg$ and $\forall$ do not occur, and D is a concept expression over $\mathcal{G}$ where the symbols $\neg$, $\exists$ and $\sqcup$ do not occur. An axiom of the form $R \sqsubseteq S$ is said to be a *valid role mapping* if $R \in N_{lr}$, and $S \in N_{gr}$. An $\mathcal{ALCH}$ TBox $\mathcal{M} = \mathcal{M}_{\mathcal{C}} \cup \mathcal{M}_{\mathcal{R}}$ is said to be a *valid* set of mappings if it is composed of a set of valid concept mappings $\mathcal{M}_{\mathcal{C}}$ and a set of valid role mappings $\mathcal{M}_{\mathcal{R}}$. A mapping M of the form $C \sqsubseteq D$ holds w.r.t. $\mathcal{M}$, written $C \sqsubseteq_{\mathcal{M}} D$ if $\mathcal{M} \models M$.

A set of valid concept mappings $\mathcal{M}_{\mathcal{C}}$ is said to be *complete* if the application of any rule in table 7.1 to any $M \in \mathcal{M}_{\mathcal{C}}$ does not add any new mapping to $\mathcal{M}_{\mathcal{C}}$.

Let $\texttt{unfold}(\mathcal{M})$ be a complete set of valid concept mappings for a given valid set of mappings $\mathcal{M}$. $\texttt{unfold}(\mathcal{M})$ starts with $\mathcal{M}_{\mathcal{C}}$ and applies the expansion rules of table 7.1 to each concept mapping $M \in \mathcal{M}_{\mathcal{C}}$ until $\mathcal{M}_{\mathcal{C}}$ is complete.

The $\sqcap$-rule	
Condition:	$\mathcal{M}_{\mathcal{C}}$ contains $C \sqsubseteq D_1 \sqcap D_2$.
Action:	$\mathcal{M}_{\mathcal{C}} = \mathcal{M}_{\mathcal{C}} \cup \{C \sqsubseteq D_1, C \sqsubseteq D_2\}$.
The $\forall$-rule	
Condition:	$\mathcal{M}_{\mathcal{C}}$ contains $C \sqsubseteq \forall S.D$.
Action:	$\mathcal{M}_{\mathcal{C}} = \mathcal{M}_{\mathcal{C}} \cup \{\bigsqcup \exists R_i^-.C \sqsubseteq D,$ for every $(R_i \sqsubseteq S)$ in $\mathcal{M}_{\mathcal{R}}\}$.

Table 7.1: Expansion rules for the mappings.

Given a concept expression C over the signature of a knowledge base $\mathcal{K}$, the answer for C w.r.t. $\mathcal{K}$ is defined as $C(\mathcal{K}) = \{a \mid \mathcal{K} \models a\!:\!C\}$.

Let $\texttt{lhs}(M)$ and $\texttt{rhs}(M)$ denote the left-hand side and the right-hand side of a given mapping M respectively.

Let $\texttt{partition}(C, \mathcal{M}) = \{M \mid \texttt{rhs}(M) = C, \text{ for every } M \text{ in } \texttt{unfold}(\mathcal{M})\}$, for a given concept $C \in N_{gc}$, and a set of valid mappings $\mathcal{M}$, .

Let the rewriting of a concept $C \in N_{gc}$ w.r.t. a valid set of mappings $\mathcal{M}$ be defined as:

$$\texttt{def}(C, \mathcal{M}) = \bigsqcup_{M \in \texttt{partition}(C,\mathcal{M})} \texttt{lhs}(M).$$

7.3 View-based query rewriting algorithm

Our algorithm takes as input a set of valid mappings $\mathcal{M}$ and an atomic query Q, i.e., $Q \in N_{gc}$, and returns the rewriting $Q' = \texttt{def}(Q, \mathcal{M})$.

7.3.1 Proof of soundness and correctness

Lemma 7.3.1. *If $R_1 \sqsubseteq_{\mathcal{M}} R$, then $\exists R_1.C \sqsubseteq_{\mathcal{M}} \exists R.C$.*

Proof. If an individual a is instance of $\exists R_1.C$ w.r.t. $\mathcal{M}$, it means that $a^{\mathcal{I}} \in (\exists R_1.C)^{\mathcal{I}}$ for each model $\mathcal{I}$ of $\mathcal{M}$, and that there is another individual b s.t. $b^{\mathcal{I}} \in C^{\mathcal{I}}$, and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R_1^{\mathcal{I}}$. Now if $R_1 \sqsubseteq_{\mathcal{M}} R$, $R_1^{\mathcal{I}} \subseteq R^{\mathcal{I}}$, hence $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $a^{\mathcal{I}} \in (\exists R.C)^{\mathcal{I}}$. Since $a^{\mathcal{I}} \in (\exists R_1.C)^{\mathcal{I}}$ and $a^{\mathcal{I}} \in (\exists R.C)^{\mathcal{I}}$, $(\exists R_1.C)^{\mathcal{I}} \subseteq (\exists R.C)^{\mathcal{I}}$, therefore $\exists R_1.C \sqsubseteq_{\mathcal{M}} \exists R.C$. □

Lemma 7.3.2. *If $R \sqsubseteq_{\mathcal{M}} S$ then $R^- \sqsubseteq_{\mathcal{M}} S^-$.*

Proof. If (a, b) is an instance of R w.r.t. $\mathcal{M}$, it means that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, since $R \sqsubseteq_{\mathcal{M}} S$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in S^{\mathcal{I}}$. Now since $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, then $(b^{\mathcal{I}}, a^{\mathcal{I}}) \in R^{-\mathcal{I}}$, analogously since $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in S^{\mathcal{I}}$, then $(b^{\mathcal{I}}, a^{\mathcal{I}}) \in S^{-\mathcal{I}}$. Since $(b^{\mathcal{I}}, a^{\mathcal{I}}) \in R^{-\mathcal{I}}$ and $(b^{\mathcal{I}}, a^{\mathcal{I}}) \in S^{-\mathcal{I}}$ for any pair of individuals a and b, $R^{-\mathcal{I}} \subseteq S^{-\mathcal{I}}$, therefore $R^- \sqsubseteq_{\mathcal{M}} S^-$. □

Lemma 7.3.3. $(\bigsqcup_{C_i \sqsubseteq_{\mathcal{M}} C} C_i) \sqsubseteq_{\mathcal{M}} C$.

Proof. This lemma immediately follows from the fact that $(\bigcup_{S_i \subseteq S} S_i) \subseteq S$ for any set S and that any concept C (C_i) is interpreted as the set $C^{\mathcal{I}}$ (resp. $C_i^{\mathcal{I}}$) for any model $\mathcal{I}$ of $\mathcal{M}$. □

Proposition 7.3.1. *Every mapping $M \in \texttt{unfold}(\mathcal{M})$ holds w.r.t. $\mathcal{M}$.*

Proof. Intuitively, the concept mappings introduced by the expansion rules in table 7.1 preserve their subclass relationship. Let us analyze such rules in detail:

The $\sqcap$-rule. In this case, every time the set of mappings $\mathcal{M}$ contains a concept mapping M of the form: $C \sqsubseteq D_1 \sqcap D2$, **the $\sqcap$-rule** introduces two new concept mappings $M_1 = C \sqsubseteq D_1$ and $M_2 = C \sqsubseteq D_1$. Clearly, such mappings are implied by the first axiom, so, both M_1 and M_2 hold w.r.t. $\mathcal{M}$.

The $\forall$-rule. In this case, every time the set of mappings $\mathcal{M}$ contains a concept mapping M of the form: $C \sqsubseteq \forall S.D$, **the $\forall$-rule** introduces a new concept mapping: $M_1 = \bigsqcup \exists R_i^-.C \sqsubseteq D, \forall (R_i \sqsubseteq S) \in \mathcal{M}_\mathcal{R}$.

Let us break the construction of the M_1 in two phases: (1) in the first phase we get $M' = \exists S^-.C \sqsubseteq D$ from the original mapping M. Intuitively M says that if an individual a is an instance of C, then *all* its S-successors $b_1, \ldots, b_n$ are instances of D. On the other hand, M' says that if an individual b_i has an S-predecessor a that is instance of C, then b_i is an instance of D (because of M). It is clear to see that M' holds w.r.t $\mathcal{M}$. (2) In the second phase we get $M_1 = \bigsqcup \exists R_i^-.C \sqsubseteq D, \forall (R_i \sqsubseteq S) \in \mathcal{M}_\mathcal{R}$ from M'. Basically we replace $\texttt{lhs}(M')$ for another expression $E = \bigsqcup \exists R_i^-.C, \forall (R_i \sqsubseteq S) \in \mathcal{M}_\mathcal{R}$. From lemmata 7.3.2 and 7.3.3 it follows that $E \sqsubseteq \texttt{lhs}(M')$. Since $E \sqsubseteq \texttt{lhs}(M')$ and $\texttt{lhs}(M') \sqsubseteq D$, $E \sqsubseteq D$, so M_1 holds w.r.t. $\mathcal{M}$. □

Proposition 7.3.2. $\texttt{def}(Q, \mathcal{M}) \sqsubseteq_\mathcal{M} Q$.

Proof. The process of obtaining a definition of a global concept $C \in N_{gc}$, yields the construction of a concept expression C' that is a subclass of C w.r.t. $\mathcal{M}$.

Let us break the construction of $\texttt{def}(Q, \mathcal{M})$ in three phases: (1) In the first phase, we get $\texttt{unfold}(\mathcal{M})$. (2) Then we get a subset $\texttt{partition}(Q, \mathcal{M})$ of $\texttt{unfold}(\mathcal{M})$ by simply taking the subset of $\texttt{unfold}(\mathcal{M})$ such that the right-hand side of every $M \in \texttt{unfold}(\mathcal{M})$ is Q. (3) Finally we get

$$\texttt{def}(Q, \mathcal{M}) = \bigsqcup_{M \in \texttt{partition}(Q, \mathcal{M})} \texttt{lhs}(M).$$

From proposition 7.3.1, we know that every mapping $M \in \texttt{unfold}(\mathcal{M})$ holds w.r.t. $\mathcal{M}$. Hence every mapping $M \in \texttt{partition}(Q, \mathcal{M})$ also holds w.r.t. $\mathcal{M}$. Moreover from lemma 7.3.3 it follows that

$$\bigsqcup_{M \in \texttt{partition}(Q, \mathcal{M})} \texttt{lhs}(M) \sqsubseteq_\mathcal{M} Q,$$

therefore $\texttt{def}(Q, \mathcal{M}) \sqsubseteq_{\mathcal{M}} Q$. □

Lemma 7.3.4. *$Q'(\mathcal{K}) \subseteq Q(\mathcal{K})$ if $Q' \sqsubseteq_{\mathcal{K}} Q$, for any knowledge base $\mathcal{K}$.*

Proof. $Q' \sqsubseteq_{\mathcal{K}} Q$ means that $Q'^{\mathcal{I}} \subseteq Q^{\mathcal{I}}$ for any model $\mathcal{I}$ for $\mathcal{K}$. In other words, $a^{\mathcal{I}} \in Q'^{\mathcal{I}}$ implies $a^{\mathcal{I}} \in Q^{\mathcal{I}}$ for any individual a. Since $Q(\mathcal{K})$ (resp. $Q'(\mathcal{K})$) denotes the set of individuals which are instances of Q (resp. Q') w.r.t. $\mathcal{K}$, if $Q' \sqsubseteq_{\mathcal{K}} Q$, then $Q'(\mathcal{K}) \subseteq Q(\mathcal{K})$. □

Proposition 7.3.3. *$Q'(\langle \mathcal{M}, \mathcal{A} \rangle) \subseteq Q(\langle \mathcal{M}, \mathcal{A} \rangle)$ for any local atomic ABox $\mathcal{A}$.*

Proof. From lemma 7.3.4 we know that $Q'(\langle \mathcal{M}, \mathcal{A} \rangle) \subseteq Q(\langle \mathcal{M}, \mathcal{A} \rangle)$ if $Q' \sqsubseteq_{\langle \mathcal{M}, \mathcal{A} \rangle} Q$. From proposition 7.3.2 we know that $Q' \sqsubseteq_{\mathcal{M}} Q$, therefore $Q'(\langle \mathcal{M}, \emptyset \rangle) \subseteq Q(\langle \mathcal{M}, \emptyset \rangle)$. It trivially follows that $Q'(\langle \mathcal{M}, \mathcal{A} \rangle) \subseteq Q(\langle \mathcal{M}, \mathcal{A} \rangle)$. □

Proposition 7.3.4. *$Q(\langle \mathcal{M}, \mathcal{A} \rangle) \subseteq Q'(\langle \mathcal{M}, \mathcal{A} \rangle)$ for any local atomic ABox $\mathcal{A}$.*

Proof. Let us suppose we have a TBox $\mathcal{T}$ composed of the axiom $C' \sqsubseteq C$, for a given concept $C \in N_{gc}$, where $C' = \texttt{def}(C, \mathcal{T})$. If we were to evaluate C w.r.t. a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ (i.e., compute $C(\langle \mathcal{T}, \mathcal{A} \rangle)$), for any given local atomic ABox $\mathcal{A}$; the only way for an individual a to be an instance of C, is for it to be an instance of C', since there is *only one* mapping for C. Clearly in such a setting, for any individual a, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ then $a^{\mathcal{I}} \in C'^{\mathcal{I}}$ for any model $\mathcal{I}$ of $\mathcal{T}$. Hence, $C^{\mathcal{I}} \subseteq C'^{\mathcal{I}}$. It trivially follows that $C(\langle \mathcal{T}, \mathcal{A} \rangle) \subseteq C'(\langle \mathcal{T}, \mathcal{A} \rangle)$.

In particular, in order to be able to conclude a similar statement, $Q' \sqsubseteq_{\mathcal{M}} Q$ must hold and be the only mapping available for Q. Since Q' is constructed taking into account *all* mappings $M \in \texttt{def}(Q, \mathcal{M})$ of the form $D \sqsubseteq Q$ (where D is a concept expression over $\mathcal{L}$), we know $Q' \sqsubseteq_{\mathcal{M}} Q$ is the only available mapping for Q. Moreover we know that such mapping holds from proposition 7.3.2, hence, as can be seen, we have the same situation as that described above. Therefore, $Q(\langle \mathcal{M}, \mathcal{A} \rangle) \subseteq Q'(\langle \mathcal{M}, \mathcal{A} \rangle)$. □

Theorem 7.3.1. *$Q(\langle \mathcal{M}, \mathcal{A} \rangle) = Q'(\langle \mathcal{M}, \mathcal{A} \rangle)$ for any local atomic ABox $\mathcal{A}$.*

Proof. Since we know that $Q'(\langle \mathcal{M}, \mathcal{A} \rangle) \subseteq Q(\langle \mathcal{M}, \mathcal{A} \rangle)$ from proposition 7.3.3 and $Q(\langle \mathcal{M}, \mathcal{A} \rangle) \subseteq Q'(\langle \mathcal{M}, \mathcal{A} \rangle)$ from proposition 7.3.4, it trivially follows that $Q(\langle \mathcal{M}, \mathcal{A} \rangle) = Q'(\langle \mathcal{M}, \mathcal{A} \rangle)$. □

7.4 Lessons learned

During the development of our algorithm we discovered that relaxing the various limitations we imposed on the mappings can easily lead to situations where the problem becomes much harder. Consider the following situations in which these limitations:

- Prevent us from using recursion in the rewritings. Let us imagine $\mathcal{M}$ includes a mapping of the form:

$$C \sqsubseteq \exists R.C, \tag{7.1}$$

  where C is a concept expression over $\mathcal{G}$, and $R \in N_{gr}$. In this case, it is easy to see that if a user were to ask the query $Q = C$, trying to unfold its definition in order to get a rewriting would yield an infinite process.

- Prevent us from constructing mappings that are equivalent to those described previously. For example, let us imagine $\mathcal{M}$ contains the following mapping:

$$C \sqcap \forall R.\neg C \sqsubseteq \bot,$$

  where $C \in N_{gc}$ and $R \in N_{gr}$. A trivial transformation would show that such an axiom is equivalent to axiom 7.1, which would yield to a similar situation.

- Allow us to reduce a GLaV system into a GaV one in a straightforward manner. For example, allowing the use of conjunction on the right-hand side of mappings could permit a situation where $\mathcal{M}$ contains a mapping of the form:

$$C \sqsubseteq G_1 \sqcup G_2,$$

  where C is a concept expression over $\mathcal{L}$, $G_1 \in N_{gc}$, and $G_2 \in N_{gc}$. In this case, it is impossible to break the mapping in order to obtain independent concept definitions for G_1 and G_2.

In summary, the main idea behind our approach is that with certain limitations on the mappings, it is possible to reduce a GLaV system into a GaV one.

This allows us to base our rewriting technique on simple concept unfolding.

Chapter 8

Conclusions and future work

8.1 Research undertaken

The research undertaken in the first year can be grouped in three areas:

1. Background knowledge. A good understanding of the problem of information integration and its relation with DLs has been acquired through a thorough literature review. This review included relevant topics, such as DLs [5], information integration [28, 29], answering queries using views [31, 40], and database technologies [2].

2. Related work. A good idea of existing techniques and approaches to information integration has been gotten through an extensive literature review of existing IISs [19, 17, 6, 24, 37, 4, 32, 23, 20, 15].

3. The first attempt. A sound and complete view-based query rewriting algorithm was devised for an scenario where we consider: (1) acyclic $\mathcal{ALC}$[1] KBs for the global view and the source models, (2) limited GLaV and intersource mappings represented with an acyclic $\mathcal{ALCH}$[2], and (3) atomic user queries.

Our algorithm, presented in chapter 7, takes an atomic query in terms of the global view and returns a union of conjunctive queries over the source models. In general, given the assumed simplifications, it is possible to reduce a set of GLaV mappings to a set of GaV ones, allowing the use of unfolding to rewrite the given query.

[1] A DL allowing for concept conjunction, disjunction, negation, and universal and existential quantification [5].

[2] $\mathcal{ALC}$ with role inclusion axioms [5].

8.2 Future work

The future research is outlined in figure 8.1 and can be grouped in the following tasks:

- Write a paper for the algorithm developed in the first year, emphasizing the lessons learned. It is envisaged that the results will be published in a related workshop.

- Investigate the ways to make the algorithm more general considering: (1) more expressive KR formalisms (e.g., DLs with number restrictions) for representing the global view and the source models (as well as integrity constraints [2] within them), and (2) more expressive queries (e.g., non-atomic, conjunctive) for querying the system. The results of this investigation should be written-up in a conference paper.

- Investigate typical application domains for information integration as well as typical information needs (i.e., queries) w.r.t. a given domain. This investigation will give us insight as to what is the level of expression, w.r.t. both the KR formalism and the query language, *real* application domains need. A promising relationship between us and fellow researchers at the University of Newcastle, in the context of the ComparaGRID project[3], has been created. We plan to study their application domain in order to conduct this part of our investigation.

- Design and implement our view-based query rewriting algorithm in a prototype system.

- Evaluate the system within the context of ComparaGRID. The evaluation results, as well as the full specification of the system are envisaged to be published in an international conference paper.

- Write up the thesis and present a successful viva. The contributions of our investigation should be published in a relevant journal.

[3] http://www.comparagrid.org

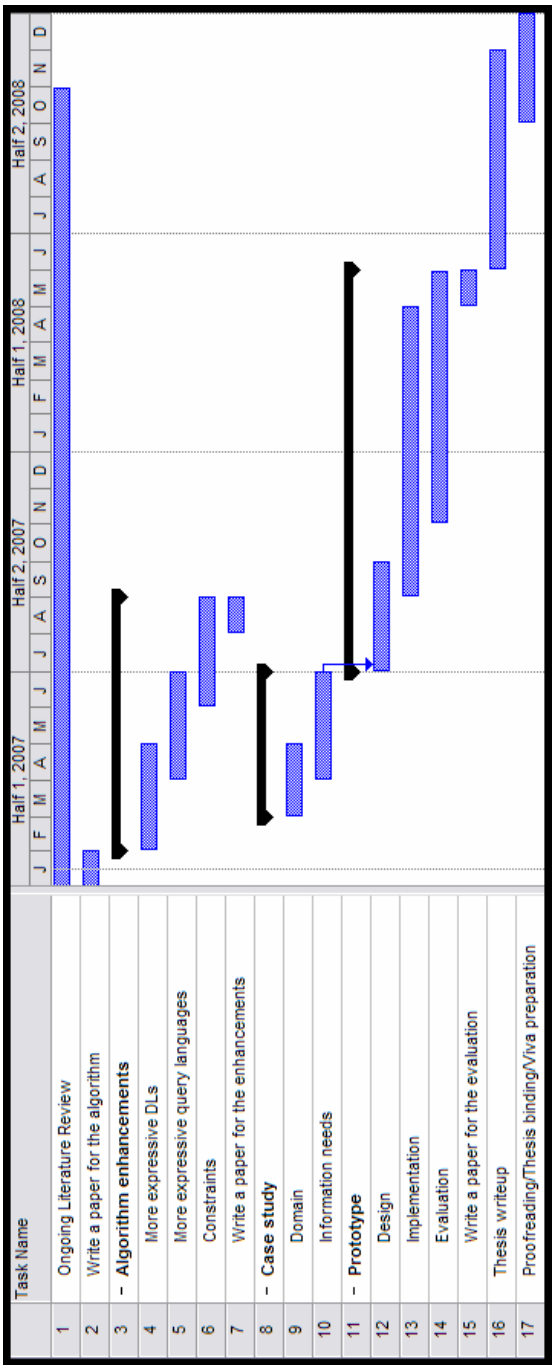

Figure 8.1: A Gantt chart outlining the timing for future research.

Bibliography

[1] C. A. Goble I. Horrocks W. A. Nolan A. L. Rector, S. K. Bechhofer and W. D. Solomon. The GRAIL Concept Modelling Language for Medical Terminology. *Artificial Intelligence in Medicine*, 9:139–171, February 1997. ISBN 01333657.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Philippe Adjiman, Philippe Chatalic, Francois Goasdou, Marie-Christine Rousset, and Laurent Simon. SomeWhere in the Semantic Web. In *International Workshop on Principles and Practice of Semantic Web Reasoning*, 2005.

[4] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1993.

[5] F. Baader and W. Nutt. *Basic Description Logics*, chapter 2, pages 47–100. Cambridge University Press, 2003.

[6] Patricia G. Baker, Andy Brass, Sean Bechhofer, Carole Goble, Norman Paton, and Robert Stevens. TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. In Janice Glasgow, Tim Littlejohn, Francis Major, Richard Lathrop, David Sankoff, and Christoph Sensen, editors, *6th Int. Conf. on Intelligent Systems for Molecular Biology*, pages 25–34, Montreal, Canada, 1998. AAAI Press, Menlo Park.

[7] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting Queries Using Views in Description Logics. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 99–108. ACM Press, 1997.

[8] Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez. *The object data standard: ODMG 3.0*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[9] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. In *Scientific American*. 2001.

[10] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. Classic: a structural data model for objects. In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 58–67, New York, NY, USA, 1989. ACM Press.

[11] P. Buneman, S. B. Davidson, K. Hart, C. Overton, and L. Wong. A Data Transformation System for Biological Data Sources. In *Proceedings of the Twenty-first International Conference on Very Large Databases*, Zurich, Switzerland, 1995. VLDB Endowment, Saratoga, Calif.

[12] Andrea Cali, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the expressive power of data integration systems. In *ER '02: Proceedings of the 21st International Conference on Conceptual Modeling*, pages 338–350, London, UK, 2002. Springer-Verlag.

[13] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Knowledge Representation Approach to Information Integration. In *Proc. of AAAI Workshop on AI and Information Integration*, pages 58–65, 1998.

[14] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description Logic Framework for Information Integration. In *Principles of Knowledge Representation and Reasoning*, pages 2–13, 1998.

[15] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Schema and Data Integration Methodology for DWQ. Technical Report DWQ-UNIROMA-004, Dipartimento di Informatica e Sistemistica, Universita di Roma La Sapienza, 1998.

[16] Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, Craig Knoblock, Steve Minton, and Manuela Veloso. PRODIGY: an integrated architecture for planning and learning. *SIGART Bull.*, 2(4):51–55, 1991.

[17] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS Project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.

[18] William W. Cohen and Haym Hirsh. Learning the CLASSIC Description Logic: Theoretical and Experimental Results. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94)*. Morgan Kaufmann, 1994.

[19] Christine Collet, Michael N. Huhns, and Wei-Min Shen. Resource Integration Using a Large Knowledge Base in Carnot. *Computer*, 24(12):55–62, 1991.

[20] O. Duschka and M. Genesereth. Infomaster - an information integration tool. In *Proceedings of the International Workshop on Intelligent Information Integration*, 1997.

[21] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 109–116, 1997.

[22] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational Plans For Data Integration. In *AAAI/IAAI*, pages 67–73, 1999.

[23] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: an Information Integration System. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 539–542, New York, NY, USA, 1997. ACM Press.

[24] François Goasdoué, Véronique Lattes, and Marie-Christine Rousset. The Use of Carin Language and Algorithms for Information Integration: The Picsel System. *International Journal of Cooperative Information Systems*, 9(4):383 – 401, December 2000.

[25] RV Guha. Micro-theories and contexts in Cyc Part I: Basic issues. Technical report, MCC, 1990.

[26] I. Horrocks, P. F. Patel-Schneider, and F. vanHarmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1), 2003.

[27] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[28] Maurizio Lenzerini. Data Integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[29] Maurizio Lenzerini. Invited tutorial on information integration. Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003, Acapulco, Mexico, August 2003.

[30] Alon Y. Levy. Logic-Based Techniques in Data Integration. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[31] Alon Y. Levy. Answering Queries Using Views: A Survey. In *Very Large Databases Journal*, volume 10, pages 270–294, 2001.

[32] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, 1996.

[33] Alon Y. Levy and Marie-Christine Rousset. Combining Horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.

[34] R. MacGregor and Raymond Bates. The LOOM knowledge representation language. In *Proceedings of the ARPA/Rome Lab 1994, Knowledge-Based Planning and Scheduling Initiative Workshop*, april 1987.

[35] Donald P. McKay, Timothy W. Finin, and Anthony O'Hare. The Intelligent Database Interface: Integrating AI and database systems. In *Proceedings of the 1990 National Conference on Artificial Intelligence*. AAAI Press, August 1990.

[36] E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. *International journal on Distributed And Parallel Databases (DAPD)*, 8(2):223–272, April 2000.

[37] Martin Peim, Enrico Franconi, Norman W. Paton, and Carole A. Goble. Query Processing with Description Logic Ontologies Over Object-Wrapped Databases. In *SSDBM '02: Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 27–36, Washington, DC, USA, 2002. IEEE Computer Society.

[38] Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.

[39] Manfred Schmidt-Schaub and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.

[40] Jeffrey D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[41] Ron van der Meyden. Logical approaches to incomplete information: a survey. pages 307–356, 1998.

[42] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.