FAKULTÄT FÜR !NFORMATIK

Implementing a Peer Data Management System

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Sebastian Skritek, BSc.
Matrikelnummer 0226286

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Univ.Prof. Dr. Reinhard Pichler
Mitwirkung: Univ.Ass. Dr. Dr. Ingo Feinerer

Wien, 9. August 2009

(Unterschrift Verfasser) (Unterschrift Betreuer)

Sebastian Skritek, Ruzickagasse 88/27 1230 Wien

"Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit — einschließlich Tabellen, Karten und Abbildungen —, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 9. August 2009 _____

(Unterschrift)

ABSTRACT

Peer Data Management Systems (PDMSs) are an approach to combine the flexibility of Peer-to-Peer (P2P) systems with the expressiveness and rich semantics of database systems. In PDMSs, data is assumed to be distributed over several autonomous peers, each of them offering (parts of) their data through its own peer schema. Similar to Data Exchange and Data Integration, every peer may define mappings between the schemas of other peers and its own schema. In contrast to Data Exchange, Data Integration or Federated Databases, however, PDMSs require no global schema and therefore no global coordination to share data. Instead, all relationships are defined only between pairs of peers.

Unfortunately, when applying the usual semantics based on first-order logic to these mappings, several important reasoning tasks, for example query answering, become undecidable over PDMSs for general settings.

Therefore, several proposals have been presented in the literature how PDMSs could be restricted to maintain decidability for typical reasoning tasks in database theory. One possibility is to restrict the topology of the network implied by the mappings and to avoid certain kinds of cycles in the mappings. But this contradicts the idea that no global coordination is required in PDMSs.

Another possibility is to restrict the expressive power of the mappings, as suggested by Calvanese et alii. Based on this idea, in 2007, De Giacomo et al. proposed a theoretical framework that allows both an arbitrary topology of the P2P network and efficient evaluation of the main reasoning tasks in Peer Data Management (PDM). Moreover it incorporates "classical" Data Exchange and Data Integration as special cases. To the best of our knowledge, this idea has not been implemented yet.

Since it seems to be a promising basis for further research, the main goal of this thesis was to study the theoretical background of PDM and to create a prototype implementation of this framework. This thesis also gives an overview over the approaches for PDMSs proposed in the literature, presents the approach of De Giacomo et al. in detail and reports on the implementation and first evaluation results of the prototype.

ZUSAMMENFASSUNG

Der Begriff "Peer Data Management System" (PDMS) bezeichnet einen Ansatz um die Flexibilität von Peer-to-Peer (P2P) Systemen mit der Ausdruckskraft und klar definierten Semantik von Datenbanksystemen zu verbinden. Dabei nimmt man an, dass die Daten auf verschiedenen, unabhängigen Knoten (Peers) verteilt sind. Jeder Peer ermöglicht über ein Peer-Schema Zugriff auf seine Daten und kann — ähnlich wie bei Datenintegration ("Data Integration") und Datenaustausch ("Data Exchange") — Abbildungen fremder Peer-Schemata auf sein eigenes Schema definieren. Im Gegensatz zu Datenintegration, Datenaustausch oder Multidatenbanksystemen benötigen PDMSs jedoch kein globales Schema und somit keine zentrale Kontrollinstanz. Alle Beziehungen werden lokal zwischen jeweils zwei Peers definiert.

Da dies im allgemeinen Fall dazu führen kann, dass etwa die Beantwortung einer Abfrage an ein PDMS unentscheidbar wird, wurden in den letzten Jahren verschiedene Ansätze entwickelt um die Entscheidbarkeit typischer Probleme im Bereich der Datenbanken zu erhalten. Deshalb die Topologie des P2P-Netzwerkes einzuschränken widerspricht jedoch der Idee von vollständig autonomen Peers.

Eine andere Möglichkeit besteht darin, die Ausdruckskraft der Abbildungen stärker zu beschränken. Basierend auf dieser Idee schlugen De Giacomo et al. 2007 ein theoretisches Modell für ein PDMS vor, welches eine beliebige Topologie des P2P Netzwerkes erlaubt und sowohl den klassischen Datenaustausch als auch die klassische Datenintegration als Spezialfälle umfasst. Nach unserem Wissenstand existiert bislang keine Implementierung dieses Ansatzes.

Ziel der vorliegenden Arbeit war es die theoretischen Grundlagen von PDMSs zu untersuchen und einen Prototypen nach dem vorgeschlagenen Modell zu entwickeln. Die Arbeit gibt weiters einen Überblick über alternative Ansätze für PDMSs in der Literatur, enthält eine detaillierte Beschreibung des Modells von De Giacomo et al. und beschreibt Ergebnisse des implementierten Prototypen.

ACKNOWLEDGMENTS

First of all I want to thank my advisor, Prof. Reinhard Pichler, for opening this topic to me and for his guidance, support and patience during the work on this thesis. I further want to thank Ingo Feinerer for his helpful feedback and all the LaTeX tips. Special thanks also to Vadim Savenkov for providing me with the CoDE system and his support therewith, and to Stefan Rümmele for his LaTeX styles and templates. All of them I want to thank for the good working atmosphere in the last months.

My gratitude goes to my parents for feeding me the last 26 years (and all the other stuff) ;-).

Last but not least, thanks to all the people that didn't mind being neglected during my working for this thesis.

This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032.

CONTENTS

1 INTRODUCTION

The idea of Peer-to-Peer (P2P) file sharing systems like Napster or Gnutella has been extremely successful in supporting the exchange of large-granular data referenced by a single identifier. This success is due to their low setup costs (i.e., their ease of use), their high flexibility and robustness with respect to failures because of their decentralised structure. On the other hand, traditional P2P systems also possess several shortcomings: They completely ignore the structure of the data as well as possible relationships within the data. Thus, they offer only a very weak semantics (if any at all), do not support any kind of data transformation and cannot be used for sharing fine-granular data. As noticed in [35], all these properties, however actually belong to the strengths of database systems.

Collaboration between autonomous local databases is traditionally achieved by techniques like federated databases [73] and multi-databases [8]. Such systems provide unified access to the participating databases and support the information sharing between them by mapping their local schemas to a global (or federated) schema. This global schema is the main drawback of such systems: Besides requiring consensus about the schema between the participants as well as some centralised infrastructure for coordination, it also introduces a single point of failure to the system. Moreover maintaining such a global schema leads to higher setup costs and makes the whole system less flexible.

The idea of *Peer Data Management* (PDM) is to combine the flexibility of P2P systems with the rich semantics of database systems. In *Peer Data Management Systems* (PDMSs), there exists no global schema, and all coordination required for data sharing is performed directly between pairs of peers. Moreover also queries are posed against a single peer which is then responsible to return the correct answer with respect to its local data and the relationships with other peers. (Sometime the term *Peer Database Systems* is used instead of PDMSs to stress that the P2P-system consists of a set of local databases. However, since there exists no uniform use of these terms we will only use PDMSs.) Many prototype systems and formalisation approaches of such systems have been presented in the literature, e.g. [2, 5, 15, 25, 26, 30, 38, 44, 46, 50, 61, 62, 72]. An often used approach for the formal description of PDMSs are schema mapping techniques that are known from data exchange and data integration.

Data exchange describes the problem of migrating data stored under some source schema to an instance under another schema. Although being an old database problem that is tackled by a variety of tools available, a formal description and consideration of its semantics has only been given recently in [20, 21]. The problems studied include questions like which instance should be materialised if there is more than one candidate, or the semantics of query answering over a materialised target instance. Moreover, the maximal

expressive power of constraints defined on the target schema are analysed such that computing solutions and performing query answering remains decidable.

Data Integration on the other hand considers the problem of providing unified access to a set of autonomous source instances, each having its own schema [41, 54]. As in data exchange, this access is provided by means of a global schema and mappings between the source and the target schema. However, unlike in data exchange, no data is actually materialised under the target schema, but all data remains only in the source instances. The global schema is only used for accessing the data: Queries are formulated against the global schema and the data integration system then returns the correct answers with respect to the data stored in the source instances and the mappings between source schemas and the global schema.

Both data exchange and data integration exhibit a distinct source and a distinct target schema. A straightforward approach for the formalisation of PDMSs would be to extend these settings by adding further schemas and let each schema be both source and target schema at the same time. Since the semantics of the mappings in these systems is described by a first-order logic (FOL) interpretation, to maintain decidability of many important reasoning tasks (like for example query answering) in such systems, the topology of the graph induced by the P2P mappings needs to be restricted. This approach has been chosen for example in the Piazza system [38].

Since such global restrictions limit the autonomy of each peer, [15] proposed a weaker semantics for the P2P mappings, that allows for an arbitrary topology of the mappings by keeping at the same time several important reasoning tasks decidable. Based on this semantics, De Giacomo et al. [30] suggested a new theoretical framework for PDMSs, that contains both data exchange and data integration as special cases. Moreover, all important problems (like query answering and data exchange) are solvable in polynomial time (data complexity) in this framework. To the best of our knowledge, however, there exists no implementation of this system yet.

1.1 GOAL AND RESULTS OF THIS THESIS

The goals of this thesis were twofold: One goal was to study and summarise the theoretical background of PDM in general and especially of the approach of De Giacomo et al., and to get an overview over the different techniques considered in the area of PDM. The other goal was to implement a prototype PDMS based on the semantics suggested in [30].

As a result, we created such a prototype implementation of the framework proposed by De Giacomo et al. [30], that works on top of traditional RDBMSs and uses the system presented in [71]. Our prototype is currently capable of computing the solution to the data exchange problem in this framework and of answering queries. Both tasks are implemented by distributed algorithms such that no centralised control or coordination is needed. Besides a description of the implementation, this thesis also presents the framework of De Giacomo et al. in detail.

This thesis also contains a summary of the main concepts and results in data exchange and data integration, since they build the theoretical background of this framework. Further, an overview of other approaches for PDMSs is given.

1.2 ORGANIZATION

This thesis is organized as follows: In Chapter 2 some basic concepts from database theory are recapitulated. In Chapter 3 we summarise the most important notions of data exchange and data integration together with complexity results for the most important tasks (computing solutions and query answering) and we give a short comparison of the concepts of data exchange and data integration. Chapter 4 introduces the idea of PDMSs and summarises some formalisation approaches for these systems from the literature. This is continued in Chapter 5 where the framework of De Giacomo et al. [30] is presented in detail. Our prototype implementation of this system is described in Chapter 6, together with some problems we encountered with the given definitions, and how they have been solved for implementation. Chapter 7 concludes this thesis.

2 PRELIMINARIES

In this chapter we summarise some basic concepts from database theory and define basic notions used throughout this thesis.

Within this thesis, we assume a fixed domain $\Gamma = \mathcal{C} \cup \mathcal{N}$, where $\mathcal{C}$ is an infinite (but countable) set of *constants* and $\mathcal{N}$ is an infinite (but countable) set of *labelled nulls*, with $\mathcal{C} \cap \mathcal{N} = \emptyset$. For constants we assume the unique name assumption (i.e. different constants denote different objects), and two labelled nulls are considered as equal if they have the same name.

A *relational schema* (or just *schema*, if "relational" is obvious from the context) $\mathbf{R}$ is a finite set of relational symbols $\{R_1, \ldots, R_k\}$, each having a name (R_i), an assigned arity $n_i \geqslant 0$ and a sequence of attributes $\langle A_{i,1}, \ldots, A_{i,n_i} \rangle$. A relational symbol and its attributes may be written as $R_i(A_{i,1}, \ldots, A_{i,n_i})$. If the attribute names need not be referenced explicitly, they may be omitted. Given a set of attributes A, $\mathtt{dom}(A)$ denotes the domain of the attributes in A, i.e. the constants allowed as values for these attributes. Moreover, if R_i is a relational symbol and A is a subset of the attributes of R_i, then $R_i[A]$ is used to refer to R_i but only considering the attributes in A.

A *fact* $R_i(t_1, \ldots, t_{n_i})$ consists of a relational symbol R_i and a tuple $t = (t_1, \ldots, t_{n_i})$ (with $t_j \in \Gamma$ for $1 \leqslant j \leqslant n_i$). As for relational symbols, if A is a (sub)set of attributes of R_i, $t[A]$ is used to only refer to those values of t corresponding to the attributes of A. Thereby the attribute set A may either be specified using the names of the attributes or their indices.

A relation corresponding to a relational symbol R_i is a set of facts $R_i(t_1, \ldots, t_{n_i})$. When the relational symbol is clear from the context, it can be omitted, and the relation is then written as the set of tuples $(t_1, \ldots, t_{n_i})$. In the following, when it is safe to do we abuse notation a little bit and use relational symbols to denote both, the symbol and the relation itself.

A *(database) instance* I of a schema $\mathbf{R}$ is a function assigning a relation to each relational symbol $R_i \in \mathbf{R}$. Given an instance I, we denote with $\mathit{const}(I)$ the constants in I and with $\mathit{nulls}(I)$ the labelled nulls in I. I is called a *definite instance* if $\mathit{nulls}(I) = \emptyset$, otherwise I is called an *indefinite instance*.

If $\mathbf{S}$ ($= \{S_1, \ldots, S_{k_s}\}$) and $\mathbf{T}$ ($= \{T_1, \ldots, T_{k_t}\}$) are two schemas with no relational symbol in common, then $\langle \mathbf{S}, \mathbf{T} \rangle$ denotes the schema $\langle S_1, \ldots, S_{k_s}, T_1, \ldots, T_{k_t} \rangle$. Also, if I is an instance of $\mathbf{S}$ and J is an instance of $\mathbf{T}$, then $\langle I, J \rangle$ denotes the corresponding instance of $\langle \mathbf{S}, \mathbf{T} \rangle$.

We use $\vec{x}$ to denote a sequence $\langle x_1, \ldots x_n, \rangle$ of variables and/or values $x_1, \ldots, x_n$.

A *conjunctive query* q is an open first-order formula $\exists \vec{y} \varphi(\vec{x}, \vec{y})$, where $\varphi(\vec{x}, \vec{y})$ is a conjunction of atoms. The free variables in $\vec{x}$ are also called the *distinguished variables* of q. An alternative representation of a conjunctive query is $\{\vec{x} \mid \exists \vec{y}\ \varphi(\vec{x}, \vec{y})\}$, that is by concentrating on the result of the query (since clear from

the context, we often omit the existential quantifier). A *Boolean conjunctive query* is a conjunctive query with $\vec{x} = \langle\rangle$.

Given a query q and a (definite or indefinite) instance I, then

- q^I denotes the (result of the) "standard" evaluation of q in I under the closed world assumption.
- $\mathrm{Eval}_{\mathrm{Null}\downarrow}(q, I)$ contains those tuples from q^I that do not contain any labelled null (that is, $\mathrm{Eval}_{\mathrm{Null}\downarrow}(q, I) = q^I \cap \mathcal{C}^k$, assuming that the arity of q is k).

Moreover, if R is a relational symbol and I is an instance, then R^I denotes the relation assigned to R by I.

The notion of a *homomorphism* is used to relate database instances to each other:

Definition 2.1 (Homomorphism between instances)**.** Let A and B be two database instances. A *homomorphism from* A *to* B is a mapping $h\colon A \to B$, that maps $\mathit{const}(A) \cup \mathit{nulls}(A)$ onto $\mathit{const}(B) \cup \mathit{nulls}(B)$ such that

- $h(c) = c$ for all $c \in \mathit{const}(A)$ and
- every fact of A is mapped to a fact of B, that is, given a fact $R(t) \in A$, $h(R(t)) \in B$ (where $h(R(t)) = R(h(t_1), \ldots, h(t_n))$)

⊣

We finally recall two important types of dependencies considered in database theory, namely inclusion dependencies and key constraints:

Definition 2.2 (Key Constraint)**.** Let R_i be a relational symbol of arity n_i. A *key constraint* is an assertion of the form

$$\mathrm{key}(R_i) = \{j_1, \ldots, j_k\} \quad (1 \leqslant j_h \leqslant n_i \text{ for } j_1 \leqslant j_h \leqslant j_k)$$

meaning that the attributes of R_i with indices $j_1, \ldots, j_k$ are a *key*[1] of R_i. ⊣

A set of key constraints is referred to as *legal key constraints* with respect to a relational schema $\mathbf{R}$ if it contains at most one key constraint for every relational symbol of $\mathbf{R}$. Note that given a set of legal key constraints one can even assume that there exists exactly one key for every relational symbol: If no key is defined explicitly for some relational symbol, one can trivially assume its whole attribute set as key. If not stated otherwise, within this thesis we always assume legal key constraints.

Definition 2.3 (Inclusion Dependency)**.** Let R_i and R_j be two relational symbols of some schema $\mathbf{R}$. Let further A be a sequence of attributes of R_i and B be a sequence of attributes of R_j with the same number of attributes as A. An *inclusion dependency* is an assertion of the form $R_i[A] \subseteq R_j[B]$. ⊣

Remember that the attributes in A may either be defined by their names or their indices. An inclusion dependency states that in every valid instance for $\mathbf{R}$, all combinations of values occurring in $R_i[A]$ must also occur in $R_j[B]$.

1 Given a relational symbol $\mathbf{R}(A_1, \ldots, A_n)$, we consider any subset $\{B_1, \ldots, B_k\} \subseteq \{A_1, \ldots, A_n\}$ as key s.t. for all facts $\mathbf{R}(t_1)$ and $\mathbf{R}(t_2)$, if $t_1 \neq t_2$, then also $t_1[B_1, \ldots, B_k] \neq t_2[B_1, \ldots, B_k]$ (i.e. a key is not required to be minimal).

3 DATA EXCHANGE AND DATA INTEGRATION

In this chapter, the formal definitions of data exchange and data integration will be presented together with a short comparison of these two concepts.

3.1 DATA EXCHANGE

In this section, the formal definitions of the data exchange problem are introduced, together with a summary of relevant basic results in data exchange from the literature.

Data exchange considers the following problem: Given two schemas, how to migrate data stored under one schema (the source schema) to the other schema (the target schema). Thereby migration means to actually materialise the exchanged data under the target schema.

As suggested by the above description, data exchange describes a very natural and old database problem, that is addressed by a variety of tools. Surprisingly, only in 2003 Fagin et al. [20, 21] (full versions in [22, 23] and an overview in [52]) proposed a formal framework to describe the semantics of data exchange based on mappings between the source and the target schema. Although this framework is sometimes referred to as schema based data exchange to underline that the data to exchange is defined by correspondences between schemas (instead of, for example rules triggered by certain actions like inserting or deleting data), it seems to be generally agreed to use this framework as formal definition for the semantics of data exchange. Also within this thesis the term "data exchange" is used to denote this framework.

In this section, following [52], the main concepts of this framework are summarised.

3.1.1 *Data Exchange Setting*

A formal semantics of the data exchange problem has to specify, given an instance of the source schema, which data has to be materialised in an instance of the target schema. In data exchange, this is done by defining a mapping between source and target schema, that expresses how the target schema corresponds to the source schema.

Definition 3.1 (Schema mapping [52])**.** A *schema mapping* is a triple $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ where $\mathbf{S}$ and $\mathbf{T}$ are two schemas with no relation symbols in common, and Σ is a set of logical formulas over $\langle \mathbf{S}, \mathbf{T} \rangle$.
$\mathbf{S}$ and $\mathbf{T}$ are referred to as *source schema* and *target schema* respectively. The formulas of Σ are called constraints or dependencies[1]. ⊣

1 In data exchange, often no formal distinction is made between constraints and dependencies, but they are interchangeably used. This is because sometimes the term "dependencies" seems

Σ will be defined a little bit later more precisely. Intuitively, using a schema mapping to describe a certain data exchange problem, given an instance I of $\mathbf{S}$, the goal is to materialise an instance J of $\mathbf{T}$ such that together I and J satisfy the formulas in Σ. Formally:

Definition 3.2 ([52]). Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ be a schema mapping.

- An instance $\langle I, J \rangle$ over $\langle \mathbf{S}, \mathbf{T} \rangle$ is an *instance of* $\mathcal{M}$ if it satisfies every $\chi \in \Sigma$ (I is called the *source instance*, J is called the *target instance*).
- $\mathtt{Inst}(\mathcal{M})$ denotes the set of all instances of $\mathcal{M}$.
- Given an instance I over $\mathbf{S}$, J is called *a solution for* I *under* $\mathcal{M}$ if $\langle I, J \rangle \in \mathtt{Inst}(\mathcal{M})$.
- $\mathtt{Sol}(\mathcal{M}, I)$ denotes the set of all solutions for I under $\mathcal{M}$.

⊣

Therefore, intuitively the goal is to materialise an instance $J \in \mathtt{Sol}(\mathcal{M}, I)$. Note that depending on I and Σ, there may exist several solutions (i.e. $|\mathtt{Sol}(\mathcal{M}, I)| \neq 1$).

Given a schema mapping $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$, the only restriction imposed on the logic used for expressing the formulas $\chi \in \Sigma$ is that whether Σ is satisfied by an instance or not is preserved under isomorphisms [52]. (Obviously, renaming of constants should not influence the correctness of a solution.) In the general case, within such a logic, there are no restrictions on the formulas in Σ.

With these definitions at hand, one could now immediately define a correspondent decision problem (given I, is $\mathtt{Sol}(\mathcal{M}, I) \neq \emptyset$?) and function problem (given I and $\mathtt{Sol}(\mathcal{M}, I) \neq \emptyset$, to compute some $J \in \mathtt{Sol}(\mathcal{M}, I)$) for schema mappings. In data exchange however, the formulas in Σ are restricted to two types of formulas: *tuple generating dependencies* (that generalise inclusion dependencies) and *equality generating dependencies* (that generalise functional dependencies).

Definition 3.3 (tuple generating dependency [20]). A *tuple generating dependency* (short *TGD*) is a closed logical formula

$$(\forall \vec{x})\ (\varphi(\vec{x}) \rightarrow (\exists \vec{y})\ \psi(\vec{x}, \vec{y}))$$

where $\varphi(\vec{x})$ and $\psi(\vec{x}, \vec{y})$ are conjunctions of atoms such that $\varphi(\vec{x})$ contains only variables from $\vec{x}$, and $\psi(\vec{x}, \vec{y})$ contains only variables from $\vec{x}$ and $\vec{y}$. It is further assumed, that all $x \in \vec{x}$ occur at least once in $\varphi(\vec{x})$, that is no $x \in \vec{x}$ occurs only in $\psi(\vec{x}, \vec{y})$.
A *full TGD* is a TGD where $\vec{y} = \emptyset$. ⊣

to be more appropriate because they describe how one instance depends on another. On the other hand, sometimes "constraints" seems to fit better because they imply restrictions on valid instances. However, especially to avoid confusion in certain P2P settings described later, we use "dependency" to denote expressions over two different schemas and "constraints" to denote expressions over a single schema.

Definition 3.4 (equality generating dependency [20])**.** An *equality generating dependency* (short *EGD*) is a closed logical formula

$$(\forall \vec{x})\ (\varphi(\vec{x}) \rightarrow (x_i = x_j)) \quad (i, j \leqslant |\vec{x}|)$$

where $\varphi(\vec{x})$ is a conjunction of atoms that contains only variables from $\vec{x}$, and every $x \in \vec{x}$ occurs at least once in $\varphi(\vec{x})$. ⊣

Intuitively, as indicated by their name, TGDs define which data has to be present in an instance based on its current content, while EGDs enforce equalities between attributes.

Instead of defining TGDs as logical formulas, $\varphi(\vec{x})$ and $\psi(\vec{x}, \vec{y})$ can be interpreted as conjunctive queries q_1 and q_2 of the same arity with the distinguished variables $\{\ x \mid x \in \vec{x} \ \wedge\ x \in \psi(\vec{x}, \vec{y})\}$ (see e.g. [30]). A TGD is then written as $q_1 \rightarrow q_2$[2]. Unlike the definition as logical sentence, this definition does not immediately give rise to a notion of satisfiability of a TGD. To achieve the same semantics, a TGD $q_1 \rightarrow q_2$ is considered to be satisfied if $q_1^I \subseteq q_2^I$.

In a similar way, EGDs can be defined as constraints $q \rightarrow x_1 = x_2$ where x_1 and x_2 are the distinguished variables of q, and an EGD is considered to be satisfied by an instance I if for each $(X_1, X_2) \in q^I$, $X_1 = X_2$.

Note that in the further parts of this thesis, when using the representation of TGDs and EGDs as logical sentences, both, the universal and the existential quantifiers are omitted for better readability, since the quantification of the variables is clear from the context (all variables appearing only in $\psi(\vec{x}, \vec{y})$ are existentially, all others are universally quantified). Moreover, in the following, given a TGD $\varphi(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$, $\varphi(\vec{x})$ may be referred to as the *left hand side* of the TGD and $\psi(\vec{x}, \vec{y})$ may be referred to as the *right hand side* of the TGD.

As stated above, in the general case of a schema mapping $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ there are no restrictions on the formulas in Σ. For *data exchange settings*, further restrictions on the form of the TGDs and EGDs in Σ are assumed:

Definition 3.5 (Data Exchange Setting)**.** A *data exchange setting* is a schema mapping $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ where

- Σ_{st} is a set of TGDs $q_1 \rightarrow q_2$, where q_1 is a conjunctive query over $\mathbf{S}$, and q_2 is a conjunctive query over $\mathbf{T}$ (called *source-to-target TGDs*).
- Σ_t is a set of TGDs $q_1 \rightarrow q_2$ and EGDs $q \rightarrow x_1 = x_2$, where q_1, q_2 and q are conjunctive queries over $\mathbf{T}$ (called *target TGDs* and *target EGDs*).

⊣

There are several reasons for these restrictions. First, restricting to TGDs and EGDs is justified by the fact that these two dependency types cover together the most important types of constraints considered in relational database theory, by having the same expressive power as the class of embedded implicational

2 To be exact, if there exists some $x \in \vec{x}$ that does not appear in $\psi(\vec{x}, \vec{y})$, $\varphi(\vec{x})$ and $\psi(\vec{x}, \vec{y})$ do not have the same arity. But as can be easily verified, replacing $\varphi(\vec{x})$ by $(\exists \vec{z})\ \varphi(\vec{x}, \vec{z})$ in Definition 3.3 and restricting $\vec{x}$ to those values appearing in both, $\varphi(\vec{x})$ and $\psi(\vec{x}, \vec{y})$ does not change the semantics of the definition.

dependencies discussed in [3, 19]. Further, constraints over the source schema are omitted since it is assumed that they are already satisfied by the data in the source instance. Additionally, allowing other kinds of EGDs and TGDs would increase the expressiveness of the data exchange setting, but, in most of such less restricted settings, would also increase the complexity of computing solutions and query answering (which, as is stated later, is tractable for the setting defined above).

An example for a more general setting is Peer Data Exchange [29], where $\Sigma = \Sigma_{st} \cup \Sigma_t \cup \Sigma_{ts}$. Thereby Σ_{ts} is a set of *target-to-source TGDs* that are used to restrict the data materialised in the target instance. Computing solutions in this setting, however is NP-complete, and query answering is coNP-complete.

Example 3.6. The following example illustrates the idea of a data exchange setting: Assume two schemas representing the library of a university (Lib_{UB}) and the library of a department of the university (Lib_{Dep}):

- $\mathrm{Lib}_{UB} = \{\mathrm{Authorised}(\mathrm{StudentId}, \mathrm{FirstName}, \mathrm{LastName}, \mathrm{IdFOS}),$ $\mathrm{Registered}(\mathrm{StudentId}, \mathrm{CardId})\}$
- $\mathrm{Lib}_{Dep} = \{\mathrm{Allowed}(\mathrm{StudentId}, \mathrm{CardId}, \mathrm{IdFOS}),$ $\mathrm{Surcharge}(\mathrm{CardId}, \mathrm{Price}),$ $\mathrm{Contacts}(\mathrm{StudentId}, \mathrm{IdFos}, \mathrm{Email}, \mathrm{Phone})\}$

Suppose that one is interested in filling the Lib_{Dep} according to the data from Lib_{UB}. This could be described by a data exchange setting consisting of the source schema Lib_{UB}, the target schema Lib_{Dep}, a source-to-target TGD (i.e. Σ_{st})

1. $\mathrm{Authorised}(\mathrm{SID}, \mathrm{FN}, \mathrm{LN}, \mathrm{FOS}) \wedge \mathrm{Registered}(\mathrm{SID}, \mathrm{CardId}) \rightarrow$ $\mathrm{Allowed}(\mathrm{SID}, \mathrm{CardId}, \mathrm{FOS}) \wedge \mathrm{Surcharge}(\mathrm{CardId}, \text{`0'})$

and one target TGD and one target EGD (Σ_t)

2. $\mathrm{Allowed}(\mathrm{SID}, \mathrm{CardId}, \mathrm{FOS}) \rightarrow \mathrm{Contacts}(\mathrm{SID}, \mathrm{FOS}, \mathrm{Email}, \mathrm{Phone})$
3. $\mathrm{Contacts}(\mathrm{SID}, \mathrm{FOS}, \mathrm{Email}_1, \mathrm{Phone}) \wedge$ $\mathrm{Contacts}(\mathrm{SID}, \mathrm{FOS}, \mathrm{Email}_2, \mathrm{Phone}) \rightarrow \mathrm{Email}_1 = \mathrm{Email}_2$

We will use this data exchange setting as running example throughout this section. ◇

Definition 3.7 (Data Exchange Problem [52])**.** Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a data exchange setting, where Σ_{st} is a set of source-to-target TGDs and Σ_t is a set of target TGDs and EGDs. Given an instance I of $\mathbf{S}$, the *data exchange problem associated with* $\mathcal{M}$ is to compute an instance J of $\mathbf{T}$, such that both, Σ_{st} and Σ_t are satisfied in $\langle I, J \rangle$.
The corresponding decision problem is whether $\mathrm{Sol}(\mathcal{M}, I) \neq \emptyset$. ⊣

Note that the schema mapping is regarded to be fixed, hence the complexity of the function and decision problem is the data complexity. The general structure of a data exchange setting and the corresponding problem is depicted in Figure 1.

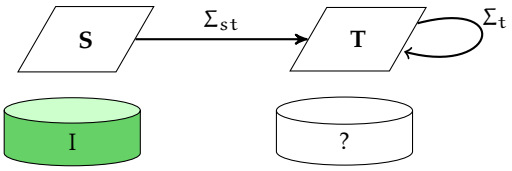

Figure 1: Structure of a data exchange setting. Given a source instance I for **S**, the problem is to find an appropriate target instance for **T**.

Example 3.8. Assume the data exchange setting from Example 3.6 and a target instance $I = \{\mathtt{Authorised}(\text{'0912345'}, \text{'alice'}, \text{'a.'}, \text{'001'}),$
$\mathtt{Authorised}(\text{'0912345'}, \text{'alice'}, \text{'a.'}, \text{'002'}), \mathtt{Registered}(\text{'0921345'}, \text{'a1'}),$
$\mathtt{Authorised}(\text{'621005'}, \text{'bob'}, \text{'bond'}, \text{'007'})\}$ for $\mathrm{Lib}_{\mathrm{UB}}$.

Then each of the following instances for $\mathrm{Lib}_{\mathrm{Dep}}$ is a valid solution for the data exchange problem:

$J = \{\mathtt{Allowed}(\text{'0912345'}, \text{'a1'}, \text{'001'}), \mathtt{Allowed}(\text{'0912435'}, \text{'a1'}, \text{'002'}),$
$\mathtt{Surcharge}(\text{'a1'}, \text{'0'}), \mathtt{Contacts}(\text{'0912345'}, \text{'001'}, \mathtt{Email}_1, \mathtt{Phone}_1),$
$\mathtt{Contacts}(\text{'0912345'}, \text{'002'}, \mathtt{Email}_1, \mathtt{Phone}_2)\}$

$J_1 = \{\mathtt{Allowed}(\text{'0912345'}, \text{'a1'}, \text{'001'}), \mathtt{Allowed}(\text{'0912435'}, \text{'a1'}, \text{'002'}),$
$\mathtt{Surcharge}(\text{'a1'}, \text{'0'}), \mathtt{Contacts}(\text{'0912345'}, \text{'001'}, \mathtt{Email}_1, \mathtt{Phone}_1),$
$\mathtt{Contacts}(\text{'0912345'}, \text{'002'}, \mathtt{Email}_1, \mathtt{Phone}_1)\}$

$J_2 = \{\mathtt{Allowed}(\text{'0912345'}, \text{'a1'}, \text{'001'}), \mathtt{Allowed}(\text{'0912435'}, \text{'a1'}, \text{'002'}),$
$\mathtt{Surcharge}(\text{'a1'}, \text{'0'}),$
$\mathtt{Contacts}(\text{'0912345'}, \text{'001'}, \text{'alice@pgp.sec'}, \mathtt{Phone}_1),$
$\mathtt{Contacts}(\text{'0912345'}, \text{'002'}, \text{'alice@pgp.sec'}, \mathtt{Phone}_2)\}$

$J_3 = \{\mathtt{Allowed}(\text{'0912345'}, \text{'a1'}, \text{'001'}), \mathtt{Allowed}(\text{'0912435'}, \text{'a1'}, \text{'002'}),$
$\mathtt{Surcharge}(\text{'a1'}, \text{'0'}), \mathtt{Contacts}(\text{'0912345'}, \text{'001'}, \mathtt{Email}_1, \mathtt{Phone}_1),$
$\mathtt{Contacts}(\text{'0912345'}, \text{'002'}, \mathtt{Email}_1, \mathtt{Phone}_2),$
$\mathtt{Allowed}(\text{'621005'}, \text{'mi6'}, \text{'007'})\}$ ◇

As already mentioned, for a given data exchange problem there may be more than one solution: There may be more than one instance J such that all constraints in Σ are satisfied by $\langle I, J\rangle$. In fact, it is even likely that there is an infinite number of such instances. In such situations the question arises which of the possible solutions should be materialised for data exchange. As Example 3.8 shows, some of the solutions are more general than others: For some of the solutions, assumptions are made that are not justified by the schema mapping (e.g. in J_1 that the phone numbers, although unknown, are the same in both Contacts facts, or in J_2 that the e-mail address is 'alice@pgp.sec'). Other solutions may contain tuples that are not necessary to satisfy the schema mapping (having no relation to the schema mapping or to the data in the source instance), like e.g. in J_3.

A natural requirement on the materialised solution therefore is that it neither contains unjustified assumptions nor data (that is, no information that cannot be derived from the source instance by the schema mapping). These requirements are formalised by the notion of *universal solutions*:

Definition 3.9 (Universal Solution [52])**.** Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a schema mapping. Given an instance I of $\mathbf{S}$, a *universal solution* to the data exchange problem associated with $\mathcal{M}$ is an instance $J \in Sol(\mathcal{M}, I)$, such that for all $J' \in Sol(\mathcal{M}, I)$ there exists a homomorphism h such that $h(J) = J'$. ⊣

Such universal solutions are the most general solutions for I under $\mathcal{M}$. That means that every solution J' for I under $\mathcal{M}$ can be obtained from a universal solution J by a variable mapping and adding additional facts.

Unfortunately, also universal solutions need not be unique (up to isomorphism). For example, the solution J in Example 3.8 is a universal solution, but so is $J \cup \{Surcharge(CardId_1, Price_1)\}$. But as the next proposition shows, all universal solutions are at least homomorphically equivalent.

Proposition 3.10 ([22])**.** *Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a data exchange setting, and I be a source instance. If J and J' are universal solutions for I under $\mathcal{M}$, then J and J' are homomorphically equivalent.*

In fact, the above proposition follows directly from the definition of universal solutions.

Another question is, whether there exists always a universal solution whenever there exists some solution (i.e. if $Sol(\mathcal{M}, I) \neq \emptyset$). As it will be stated below, Fagin et al. [22] showed that this is at least the case for those data exchange settings where computing a universal solution is tractable.

3.1.2 *Computing Universal Solutions — The Chase*

With these results at hand, indicating that universal solutions are a good choice for materialisation, the next problem is whether a universal solution can be (efficiently) computed. Since checking a certain solution for whether there exist homomorphisms to all other solution is not feasible, this is not immediately obvious. However, for a broad class of data exchange systems universal solutions can indeed be efficiently computed by adopting the *chase* procedure [3, 58] to create universal solutions [20]. The idea is to start with an instance $\langle I, \emptyset \rangle$ and to create incrementally a target instance, using TGDs to instantiate tuples and EGDs to enforce equalities between terms. The formal definition given below is more in the style of [30] than of [20], since it is reused in Chapter 5 within the context of [30].

Definition 3.11 (Chase Rule)**.** Let $\mathbf{R}$ be a schema and I an instance of $\mathbf{R}$.

TGD rule: A TGD ϕ: $\varphi(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$ is *applicable* to I if there exists a tuple $t \in \varphi(\vec{x})^I$ such that $t \notin \psi(\vec{x}, \vec{y})^I$.
If ϕ is applicable to I, ϕ is *applied* to I by adding the facts corresponding to $\psi(t, \vec{Y})$ to I, where $\vec{Y}$ contains a fresh (that is not yet present in I) labelled null for every $y \in \vec{y}$.

EGD rule: An EGD ϕ: $\varphi(x_1, x_2) \rightarrow x_1 = x_2$ is *applicable* to I if there exists a pair $(t_1, t_2) \in \varphi(x_1, x_2)^I$ where $t_1 \neq t_2$.
If ϕ is applicable to I, there are two possibilities how ϕ is *applied* to I:

- If at least one of t_1 and t_2 is a labelled null (w.l.o.g. assume t_1), replace all occurrences of t_1 in I by t_2. (If both are labeled nulls, it does not matter which one is kept and which one is replaced.)
- If both, t_1 and t_2 are constants, then stop and output FAIL.

⊣

Within the above definition, "the facts corresponding to $\psi(t, \vec{Y})$" refers to those facts consisting of the same relation symbols as the atoms in the conjunction $\psi(\vec{x}, \vec{y})$, where all occurrences of some $x \in \vec{x}$ in the bodies of these facts are replaced by the corresponding value in t. Moreover, all occurrences of some $y \in \vec{y}$ are replaced by the corresponding fresh labelled null in $\vec{Y}$.

Informally the meaning of those rules is: If there is a TGD violated by an instance I, then add tuples to I in such a way that the TGD is satisfied (i.e. do exactly what the name "tuple generating" suggests). For the existentially quantified variables, new labelled nulls are created since except the fact that there must exist some value, nothing else is known about them. Hence no assumptions are made about these values. On the other hand, the values from the distinguished variables are known from the result of $\varphi(\vec{x})^I$. If an EGD is violated, try to resolve this violation by unification of labelled nulls. If this is not possible, then I violates the corresponding constraint, hence I is inconsistent.

The idea of the chase is to repeat these steps until all violations are resolved, or an inconsistency is detected. This is formalised by the notion of a chase sequence:

Definition 3.12 (Chase Sequence)**.** Let Σ be set of TGDs and EGDs.
A *chase sequence* is a (possible infinite) sequence of instances $I_0, \ldots, I_i, I_{i+1}, \ldots$ where each I_{k+1} is obtained from I_k by applying a TGD or EGD from Σ to I_k. The transition from I_k to I_{k+1} via the application of a chase rule is called a *chase step*.
A chase sequence is called finite if, after a finite number of steps, a fixpoint is reached, that is some instance I_f where no more element from Σ is applicable, otherwise it is called infinite.
If a chase sequence starting with instance I_0 is finite, then I_f is called the *result of the chase of* I_0. ⊣

Example 3.13. Assume the data exchange setting from Example 3.6 and the source instance I from Example 3.8. Starting with $\langle I, \emptyset \rangle$, a possible chase sequence is given below. The numbers at the beginning of each line denote the dependency applied in this step. (We only state the content of the target schema and omit the values of I.)

$J_0 = \emptyset$

1 $J_1 = \{\mathtt{Allowed}(\text{'0912345'}, \text{'a1'}, \text{'001'}), \mathtt{Surcharge}(\text{'a1'}, \text{'0'})\}$

1 $J_2 = J_1 \cup \{\mathtt{Allowed}(\text{'0912345'}, \text{'a1'}, \text{'002'}), \mathtt{Surcharge}(\text{'a1'}, \text{'0'})\}$

2 $J_3 = J_2 \cup \{\mathtt{Contacts}(\text{'0912345'}, \text{'001'}, \mathtt{Email}_1, \mathtt{Phone}_1)\}$

2 $J_4 = J_3 \cup \{\mathtt{Contacts}(\text{'0912345'}, \text{'002'}, \mathtt{Email}_2, \mathtt{Phone}_2)\}$

3 $J_5 = J_4[\mathtt{Email}_2 \to \mathtt{Email}_1] = \{\mathtt{Allowed}(\text{'0912345'}, \text{'a1'}, \text{'001'}),$
$\mathtt{Allowed}(\text{'0912435'}, \text{'a1'}, \text{'002'}), \mathtt{Surcharge}(\text{'a1'}, \text{'0'}),$
$\mathtt{Contacts}(\text{'0912345'}, \text{'001'}, \mathtt{Email}_1, \mathtt{Phone}_1),$
$\mathtt{Contacts}(\text{'0912345'}, \text{'002'}, \mathtt{Email}_1, \mathtt{Phone}_2)\}$

Since in $\langle I, J_5\rangle$ no more dependency is applicable, the chase sequence is finite and the chase terminates with the result J_5. Note that J_5 is exactly the solution J of Example 3.8. ◇

Hence, let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t\rangle$ be a schema mapping, given a definite instance I the idea for computing a universal solution is to start with the instance $\langle I, \emptyset\rangle$ and to apply dependencies from $\Sigma_{st} \cup \Sigma_t$ until no more rule is applicable, hence until no more dependency is violated (or an inconsistency is discovered).

Note that defining the chase for indefinite source instances may be a little bit problematic due to the effect of target EGDs because the source instance must not be changed. A more procedural representation of the chase (for definite instances) is given in Algorithm 1.

Alg. 1 The Chase Procedure for a data exchange setting

Input: a source instance I, a set Σ_{st} of TGDs and a set Σ_t of TGDs and EGDs.
Output: a universal solution or FAIL
$J \leftarrow \emptyset$
while there exists $\phi : \varphi(\vec{x}) \to \psi(\vec{x}, \vec{y}) \in \Sigma_{st}$ that is applicable to $\langle I, J\rangle$ **do**
  let $\vec{X} \in \varphi(\vec{x})^{\langle I,J\rangle}$, $\vec{X} \notin \psi(\vec{x}, \vec{y})^{\langle I,J\rangle}$ and $\vec{Y}$ be a vector of fresh labelled nulls
  $J \leftarrow J \cup \psi(\vec{X}, \vec{Y})$ // apply ϕ to $\langle I, J\rangle$
while there exists $\phi \in \Sigma_t$ that is applicable to $\langle I, J\rangle$ **do**
  if ϕ is a TGD $\varphi(\vec{x}) \to \psi(\vec{x}, \vec{y})$ **then**
    let $\vec{X} \in \varphi(\vec{x})^{\langle I,J\rangle}$, $\vec{X} \notin \psi(\vec{x}, \vec{y})^{\langle I,J\rangle}$ and $\vec{Y}$ fresh labelled nulls
    $J \leftarrow J \cup \psi(\vec{X}, \vec{Y})$ //apply ϕ to $\langle I, J\rangle$
  else if ϕ is an EGD $\varphi(x_1, x_2) \to x_1 = x_2$ **then**
    let $(X_1, X_2) \in \varphi(x_1, x_2)^{\langle I,J\rangle}$, $X_1 \neq X_2$
    if at least one of X_1, X_2 is a labelled null **then**
      $J \leftarrow J[X_1 \to X_2]$ // or $J \leftarrow J[X_2 \to X_1]$ resp.
    else
      return FAIL
return J

The correctness of the chase is given by the following theorem:

Theorem 3.14 ([20]). *Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t\rangle$ be a data exchange setting, where Σ_{st} is a set of TGDs and Σ_t is a set of TGDs and EGDs. Given some source instance* I,

1. *let $\langle I, J\rangle$ be the result of some successful finite chase of $\langle I, \emptyset\rangle$ with $\Sigma_{st} \cup \Sigma_t$. Then* J *is a universal solution.*

2. *if there exists some failing finite chase of* $\langle I, \emptyset \rangle$ *with* $\Sigma_{st} \cup \Sigma_t$*, then there exists no solution.*

Hence, if the chase returns a solution, it is correct: either it is a universal solution, or there exists no solution at all. Therefore, the solution J from Example 3.8 is indeed a universal solution, since as shown in Example 3.13, it is the solution of chasing the source instance.

Some remarks concerning the chase: The reason why nothing is stated about the order in which the dependencies should be applied is that the order does not matter. Any order will yield a correct result, although different orders may lead to different universal solution. The different solution may differ by more than just isomorphisms, but as stated above, universal solutions need not be unique up to isomorphism.

As done in Algorithm 1, it is possible first to chase Σ_{st} until no more $\phi \in \Sigma_{st}$ is applicable, and only after that chase Σ_t. This works because chasing Σ_t cannot remove or change any data that would make any dependency in Σ_{st} applicable again: For a $\phi \in \Sigma_{st}$ that was satisfied in some instance $\langle I, J_k \rangle$ to become applicable again in some instance $\langle I, J_l \rangle$ later in the chase sequence, it is required that there exists some $\vec{X} \in \varphi(\vec{x})^{\langle I, J_l \rangle}$ such that $\vec{X} \notin \psi(\vec{x}, \vec{y})^{\langle I, J_l \rangle}$. But such an $\vec{X}$ cannot exist:

- Because I is not changed, $\varphi(\vec{x})^{\langle I, J_i \rangle}$ always returns the same results.
- Conjunctive queries without negation or even inequalities are monotone. Hence adding tuples as results of the application of TGDs cannot remove $\vec{X}$ from $\psi(\vec{x}, \vec{y})^{\langle I, J_i \rangle}$ $(i \geqslant k)$.
- Moreover, because conjunctive queries without inequalities are closed under homomorphisms, applying EGDs cannot result in some $\vec{X}$ being removed from $\psi(\vec{x}, \vec{y})^{\langle I, J_i \rangle}$ $(i \geqslant k)$.

Note that this only holds for definite source instances.

Although Theorem 3.14 assures that if the chase stops, it returns a correct result (either a universal solution, or there exists no solution at all), nothing is stated about the termination of the chase. In fact, the chase may not terminate, as shown in the following example:

Example 3.15. Assume a variation of the setting in Example 3.13. Let $\mathrm{Lib}_{UB} = \{\mathrm{Registered}(\mathrm{StudentId}, \mathrm{CardId})\}$ and let $\mathrm{Lib}_{Dep} = \{\mathrm{Allowed}(\mathrm{CardId}, \mathrm{UserId}), \mathrm{InvitedBy}(\mathrm{InvitedUserId}, \mathrm{UserId})\}$. Moreover, assume the following dependencies:

1. $\mathrm{Registered}(\mathrm{SID},\ \mathrm{CardId}) \rightarrow \mathrm{Allowed}(\mathrm{CardId},\ \mathrm{UID})$ $(\in \Sigma_{st})$
2. $\mathrm{Allowed}(\mathrm{CardId},\ \mathrm{UID}) \rightarrow \mathrm{InvitedBy}(\mathrm{UID},\ \mathrm{MemberUID})$ $(\in \Sigma_t)$
3. $\mathrm{InvitedBy}(\mathrm{UID},\ \mathrm{MemberUID}) \rightarrow$
   $\mathrm{Allowed}(\mathrm{MemberCardID},\ \mathrm{MemberUID})$ $(\in \Sigma_t)$

and a source instance $I = \{\mathrm{Registered}(\text{'0912345'}, \text{'001'})\}$.
Chasing $\langle I, \emptyset \rangle$ gives the following sequence of target instances:

$$J_0 = \emptyset$$

$$1\ J_1 = \{Allowed(\text{'0912345'}, UID_1)\}$$

$$2\ J_2 = J_1 \cup \{InvitedBy(UID_1, UID_2)\}$$

$$3\ J_3 = J_2 \cup \{Allowed(CardId_1, UID_2)\}$$

$$2\ J_4 = J_3 \cup \{InvitedBy(UID_2, UID_3)\}$$

$$3\ J_5 = J_4 \cup \{Allowed(CardId_2, UID_3)\}$$

$$\ldots$$

(the numbers at the beginning of each line again denote the number of the applied dependency)

It is obvious, that this gives an indefinite chase sequence. ◇

As it can be seen from the example, the nontermination arises from certain cycles in the TGDs, where the introduction of some fresh labelled null at a certain position in one chase step results in a later chase step to introduce again another fresh labelled null at the same position with the same effect.

This notion of cycles in a set of TGDs is studied in terms of the *dependency graph* of a set of TGDs:

Definition 3.16 (Dependency Graph [22])**.** Let Σ be a set of TGDs $\varphi(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$ over some fixed schema $\mathbf{S}$.
A *position* is a pair (R_i, A_{i_j}), where $R_i(A_1, \ldots, A_{n_i})$ is a relation in $\mathbf{S}$ and $(A_1, \ldots, A_{n_i})$ are the attributes of R_i.
The *dependency graph* is a directed graph $D = (V, E \cup E^*)$ whose set of directed edges contains edges (E) and *special edges* (E^*)[3]:

- The set V of nodes contains one node for each position (R_i, A_{i_j}) in $\mathbf{S}$.
- The set E contains the following edges: For each variable x that appears on the left hand side and the right hand side of some TGD in Σ, there is an edge between each position on which x appears on the left hand side and each position where x appears on the right hand side.
- The set E^* contains the following special edges: For each variable x that appears on both sides of some TGD in Σ, there is one special edge between each position of x in $\varphi(\vec{x})$ and any position of some $y \in \vec{y}$ in the right hand side of the same TGD.

⊣

Intuitively, the dependency graph encounters how data is propagated by the TGDs. Thereby edges indicate that an existing value is propagated, and special edges represent the introduction of a fresh labelled null.

3 Note that in graph theory the set of directed edges (arcs) is typically denoted by A, while E is used for undirected edges. However, to not confuse attributes and arcs, we use E also for directed edges.

Example 3.17. Figure 2 shows the dependency graph of the data exchange setting of Example 3.15 when omitting the source schema.

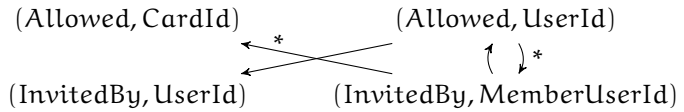

Figure 2: The dependency graph of the data exchange setting described in Example 3.15

◇

Example 3.17 shows that the problem described above as reason for the nontermination in Example 3.15 shows up in the dependency graph as the cycle $(\mathtt{Allowed}, \mathtt{UserId}) \xrightarrow{*} (\mathtt{InvitedBy}, \mathtt{MemberUserId}) \to (\mathtt{Allowed}, \mathtt{UserId})$ containing a special edge.

Those sets of TGDs that do not give rise to such cycles are called *weakly acyclic*, and are indeed the less restrictive class of TGDs for which the chase terminates.

Definition 3.18 (Weakly acyclic set of TGDs)**.** A set of TGDs is *weakly acyclic*, if in the corresponding dependency graph there is no cycle that contains a special edge. ⊣

Example 3.19. Assume the data exchange setting introduced in Example 3.6 extended by an additional target TGD $\mathtt{Contacts}(\mathsf{SID}, \mathsf{FOS}, \mathtt{Email}, \mathtt{Phone}) \to \mathtt{Allowed}(\mathsf{SID}, \mathtt{CardId}, \mathsf{FOS})$. The corresponding dependency graph (when omitting the source schema and the relational symbol Surcharge) is shown in Figure 3

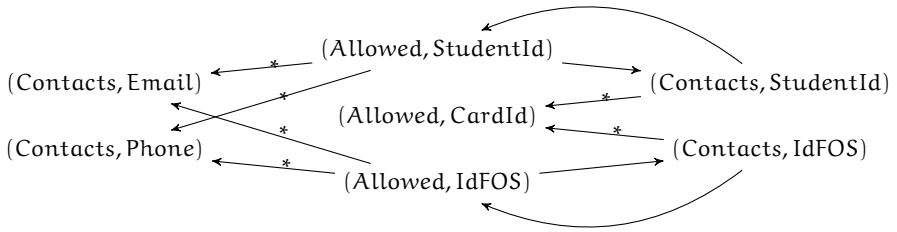

Figure 3: The dependency graph of an extension of the data exchange setting described in Example 3.15

◇

Note that the dependency graph in example 3.19 contains a cycle. Weak acyclicity does not forbid the existence of cycles in the dependency graph in general, but only those cycles containing a special edge. Intuitively this means that within such a cycle without special edge only values that have already been there at the beginning of the chase are propagated. Therefore only a finite amount of tuples can be created by the corresponding TGDs.

Given a data exchange setting $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$, obviously Σ_{st} always is a weakly acyclic set of TGDs, since the relations in the left hand side and right hand side of each TGD form two disjoint sets. Therefore a data exchange setting is called weakly acyclic if the TGDs in Σ_t form a weakly acyclic set.

It was shown by the following result that weak acyclicity is sufficient to guarantee the termination of the chase.

Theorem 3.20 ([20]). *Let Σ be the union of a set of EGDs and a set of weakly acyclic TGDs. Given a definite instance I, the length of the chase sequence over Σ starting with I is polynomially bounded in the size of I.*

Moreover, in [53] it was shown that weak acyclicity is the less restrictive case where the chase terminates, in the sense that allowing for a single non-weakly acyclic TGD makes checking an instance for consistency undecidable. (Nontermination directly follows from the correctness of the chase.)

Theorem 3.21 ([53]). *There exists a schema mapping $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ where Σ_{st} contains a single full source-to-target TGD and Σ_t contains one target EGD, one target full TGD and one non-weakly acyclic target TGD, such that for a given instance I deciding whether $\mathtt{Sol}(\mathcal{M}, I) \neq \emptyset$ is undecidable.*

From Theorem 3.20 and the correctness of the chase it immediately follows that a weakly acyclic schema mapping enjoys some nice properties:

Corollary 3.22 ([52]). *Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a schema mapping such that Σ_{st} is a set of source-to-target TGDs and Σ_t is the union of a set of targets EGDs with a weakly acyclic set of target TGDs. Then*

- *given a source instance I, a universal solution for I exists if and only if a solution for I exists.*
- *there exists a polynomial-time algorithm that, given a source instance I, tests whether a solution for I exists and if so, it produces a universal solution J for I.*

Both results rely on the chase: For a weakly acyclic set of TGDs it is guaranteed to halt in polynomial time. Therefore it outputs a universal solution whenever a solution exists and returns FAIL if no solution exists.

Summarising the properties of universal solutions, it is reasonable to consider universal solutions as the correct answer to a data exchange problem, hence as those instances to materialise: Universal solutions are the most general solutions (i.e. there exist homomorphism from a universal solution to all solutions). Moreover, in those cases where checking a data exchange problem instance for consistency is decidable, universal solutions are guaranteed to exist whenever some solution exists. In the next section, another reason why universal solutions are a good choice for materialisation is given.

3.1.3 *Query Answering*

Beside just materialising a (universal) solution for a data exchange problem instance, another interesting task is how queries posed to the target schema can be processed.

Given a data exchange setting $\mathcal{M}$, a source instance I and a query q over $\mathbf{T}$, the answer to q should be computable over the materialised (universal) solution, since after migrating the data the source instance may no longer be available. But at the same time the answer should ideally not depend on the specific (universal) solution actually materialised (as, assuming $|\mathtt{Sol}(\mathcal{M}, I)| > 1$, simply evaluating q over a materialised solution for I under $\mathcal{M}$ may yield different answers depending on the chosen solution).

To reflect that the result of the query depends on $\mathcal{M}$ and I, and not on a particular solution, in data exchange the result of a query q over $\mathbf{T}$ is defined as the set of *certain answers*.

Definition 3.23 (Certain Answers)**.** Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a data exchange setting and q a query over $\mathbf{T}$.
Then the set of *certain answers* with respect to $\mathcal{M}$ is defined as

$$\mathtt{certain}_{\mathcal{M}}(q, I) = \bigcap_{J \in \mathtt{Sol}(\mathcal{M}, I)} q^J$$

⊣

That is, the certain answers to a query q are those tuples that are part of the answer to the query over all solutions J for I under $\mathcal{M}$. Since — just as checking for a universal solution by looking for homomorphisms to all other solutions — computing the intersection over a possible infinite set of solutions is not feasible, some other way for computing the certain answers is needed.

When restricting q to the class of unions of conjunctive queries (UCQ), computing the certain answers on a data exchange setting can be done by evaluating q over any universal solution:

Theorem 3.24 ([22])**.** *Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a data exchange setting and q be a UCQ over the target schema $\boldsymbol{T}$. Given a source instance I, $\mathtt{certain}_{\mathcal{M}}(q, I) = \mathtt{Eval}_{\mathtt{Null}\downarrow}(q, J)$ for every universal solution $J \in \mathtt{Sol}(\mathcal{M}, I)$.*

The plausibility of this result can be easily checked by taking into account the intuition of universal solutions as those solutions that only contain information implied by the source instance and the mapping, without unjustified assumptions. Obviously, this and only this information must be part of any solution.

From the above theorem, it immediately follows that query answering for UCQs can be done in polynomial time (data complexity), given that the TGDs in Σ_t are weakly acyclic. The border between tractable and intractable queries is well studied, as shown by the following results.

Theorem 3.25 ([22])**.** *Assume that $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ is a schema mapping such that Σ_{st} is a set of source-to-target TGDs and Σ_t is the union of a set of target EGDs and target TGDs.*
If q is a union of conjunctive queries with at most one inequality per conjunctive query, then the certain answers of q are polynomial-time computable.

Although computing certain answers to UCQs with one inequality per disjunct is tractable, it cannot be done by the method described above, that

is by simply evaluating some query q over a universal solution, but requires specialised algorithms:

Theorem 3.26 ([22]). *There exists a data exchange setting $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ with $\Sigma_t = \emptyset$ and all TGDs in Σ_{st} are of the form $A(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$ where $A(\vec{x})$ is a single atom and all $x \in \vec{x}$ occur in $\psi(\vec{x}, \vec{y})$ and a Boolean conjunctive query q with one inequality, such that there exists no first-order query q' over a canonical universal solution J with $\mathtt{certain}_{\mathcal{M}}(q, I) = q'^{J}$.*

Finally, the following result justifies that more than one inequality per disjunct is intractable.

Theorem 3.27 ([57]). *Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a restricted data exchange setting where $\Sigma_t = \emptyset$ and all TGDs in Σ_{st} are of the form $A(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$ where $A(\vec{x})$ is a single atom and all $x \in \vec{x}$ occur in $\psi(\vec{x}, \vec{y})$.*
Then computing the certain answers to a given Boolean conjunctive query with two or more inequalities is coNP-complete.

3.1.4 *The Core of a Universal Solution*

Since there may be more than one universal solution, the question arises whether some (or one) universal solutions are to some extend "better" than others and should therefore be regarded as the preferred instances to be materialised. In fact, it turned out that there exists a universal solution that can be regarded as a minimal universal solution. It is called the core of a universal solution.

Example 3.28. Remember the data exchange setting from Example 3.6. It was already mentioned before, that while the solution J is a universal solution, so is $J \cup \{\mathtt{Surcharge}(\mathtt{CardId}_1, \mathtt{Price}_1)\}$. However, J is obviously the smaller universal solution (in fact, J even is a core), and the additional fact $\mathtt{Surcharge}(\mathtt{CardId}_1, \mathtt{Price}_1)$ does not give any new information, since the existence of some tuple in the Surcharge relation is already known from the fact Surcharge('a1', '0'). ◇

The concept of cores is known from other areas, [42] for example studies cores of graphs by considering the concept of a structure and defining the concept of cores of structures. According to [23], this definition can be translated to define cores of instances. Therefore it is necessary to first define the notion of a *subinstance* of a database instance.

Definition 3.29 (Subinstance [23]). A database instance C is a *subinstance* of a database instance A if the set of facts of C is a subset of the set of facts in A. ⊣

Definition 3.30 (Core of a database instance [23]). A subinstance C of a database instance A is called a *core of* A if there is a homomorphism from A to C, but there is no homomorphism from A to a proper subinstance of C.
A database instance C is a *core* if it is a core of itself (that is, there is no homomorphism from C to a proper substructure of C). ⊣

According to [23], the following results proven in [42] for cores of structures carry over to cores of instances:

Proposition 3.31 ([23]). *The following statements hold:*

- *Every finite structure has a core, and all cores of the same finite structure are isomorphic.*
- *If* C *is the core of a finite structure* A*, then there is a homomorphism* $h\colon A \to C$ *such that* $h(v) = v$ *for every member of the universe of* C*.*

Translated to database instances and data exchange settings, this means

- if there exists a universal solution, then there exists a core,
- ihe core of all universal solutions is unique up to isomorphism, and
- the functions mapping universal solutions to their cores map constants to themselves, hence they are homomorphisms between instances.

The next result from [23] justifies that the core is a valid choice as solution of a data exchange problem.

Proposition 3.32 ([23]). *Let* $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ *be a data exchange setting,* Σ_{st} *a set of TGDs and* Σ_t *a set of TGDs and EGDs.*
If I *is a source instance and* J *is a solution for* I *under* $\mathcal{M}$*, then* $\mathtt{core}(J)$ *is a solution for* I *under* $\mathcal{M}$*.*
Consequently, if J *is a universal solution for* I*, then also* $\mathtt{core}(J)$ *is a universal solution for* I*.*

From the results stated above, it follows that for any data exchange setting $\mathcal{M}$, if $\mathtt{Sol}(\mathcal{M}, I) \neq \emptyset$ there exists a unique (up to isomorphism) core. Because of this uniqueness it is also referred to as *the core of the data exchange problem*.

From that, the question arises whether the core of a universal solution can be efficiently computed. Although computing the core of an arbitrary structure is intractable (more precisely, given two structures A and B, deciding whether $\mathtt{core}(A) = B$ is DP-complete [23]), [23] states a polynomial time algorithm to compute the core of a universal solution of a data exchange setting where Σ_t contains only EGDs. This result is extended in [31, 32] by a polynomial time algorithm that computes the core of a data exchange setting (that is, its minimal universal solution) where Σ_t is allowed to contain both EGDs and a weakly acyclic set of TGDs. However, this algorithm does not handle EGDs directly, but simulates each EGD by a set of full TGDs. Also, it is not allowed to perform the chase steps in an arbitrary order, but a so-called "nice" order is required. According to [71], the simulation of EGDs has certain disadvantages. Finally, [71] states a polynomial time algorithm to compute a minimal universal solution of a data exchange problem instance that handles EGDs directly. The idea behind all these algorithms is to use information collected during the chase to efficiently compute the core of the created solution.

Although running in polynomial time, [71] reports that core computation is a quite extensive task.

3.2 DATA INTEGRATION

The general idea of data integration as regarded in this thesis is the following: Data is assumed to be stored at a set of different (maybe autonomous) sources. The task of a data integration system is to provide a unified view of this data. Throughout this thesis it is assumed that this unified view is given by a so-called *global schema*. Users can only access this global schema (and not the sources directly). The global schema is only virtual, no data is materialised under this schema. Instead, all data remains in the sources, and the data integration system has to compute the answers to queries posed against the global schema by posing appropriate queries over the sources. Thereby, the two main challenges are

- how to define the mappings between the sources and the global schema, and
- how to translate queries over the global schema to queries over the sources.

This section introduces some theoretical concepts and results regarding these issues, thereby following Lenzerinis overview paper [54], but not covering all issues mentioned there. A short "comparison" of data exchange and data integration is given in the next section.

3.2.1 *Data Integration System*

The first step when considering data integration [41, 54] is to formally define the notion of a data integration system. [54] gives a logical framework for modelling data integration systems based on a global schema:

Definition 3.33 (Data Integration System [54])**.** A *data integration system* $\mathcal{I}$ is a triple $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where

- $\mathcal{G}$ is the *global schema* (or *mediated schema*),
- $\mathcal{S}$ is the *source schema*, and
- $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$.

Thereby $\mathcal{M}$ contains assertions of the form $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ and $q_{\mathcal{G}} \rightsquigarrow q_{\mathcal{S}}$, where $q_{\mathcal{S}}$ is a query over the source schema and $q_{\mathcal{G}}$ is a query over the global schema. ⊣

As stated above, user-queries are only posed against $\mathcal{G}$. The intuitive meaning of the assertions in $\mathcal{M}$ is that the concept of one schema expressed through the query on the left hand side corresponds to the concept on the other schema expressed through the query in the right hand side.

Although, according to [54], this definition of a data integration system captures all approaches introduced in the literature, within this thesis $\mathcal{G}$ and $\mathcal{S}$ are assumed to be relational schemas only, but allowing for constraints over them. (With a little abuse of notation, $\mathcal{G}$ and $\mathcal{S}$ are used to denote the schema directly as well as to refer to the schema and the constraints.)

Definition 3.34 ([54])**.** Let $\mathcal{M} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration setting.
An instance D is a *source instance for* $\mathcal{I}$ if it is an instance of $\mathcal{S}$ and satisfies all constraints defined over $\mathcal{S}$. A *global instance* B is an instance B of $\mathcal{G}$. A global instance B is *legal with respect to* D if satisfies all constraints over $\mathcal{G}$ and the mapping $\mathcal{M}$ with respect to D. ⊣

It depends on the interpretation of the assertions in $\mathcal{M}$, under which conditions $\mathcal{M}$ is considered to be satisfied, but in general, given a source instance D, there may be more than one legal global instances B with respect to D. Let $\mathtt{Mod}(\mathcal{I}, D)$ denote the set of all valid global instances for $\mathcal{I}$ with respect to D.

Before introducing the most important interpretations for the mappings, it is necessary to first define the semantics of query answering over a data integration system. Since, given a source instance D, there may be more than one legal global instance B, answers to queries over a data integration systems, just as in data exchange are defined using the concept of certain answers:

Definition 3.35 (Certain Answers [54])**.** Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, and q a query over $\mathcal{G}$. Given a source instance D, the *certain answers* to q over $\mathcal{I}$ with respect to D are defined as

$$\mathtt{certain}(q, \mathcal{I}, D) = \bigcap_{B \in \mathtt{Mod}(\mathcal{I}, D)} q^B \ .$$

⊣

That is, the answer to a query over a data integration system consists of all tuples that are part of the answer to q over every legal global instance.

3.2.2 *Schema Mappings in Data Integration*

According to [54], there exist two main approaches for modelling the mapping $\mathcal{M}$ of a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$: GAV (where each element of $\mathcal{G}$ is expressed as a view over $\mathcal{S}$) and LAV (where each element of $\mathcal{S}$ is expressed as as view over $\mathcal{G}$). In addition, GLAV, a generalisation of GAV and LAV will be shortly introduced because of its close relationship to data exchange.

While GAV, LAV and GLAV are approaches to define the form of the assertions $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ and $q_{\mathcal{G}} \rightsquigarrow q_{\mathcal{S}}$ of a mapping $\mathcal{M}$, also different interpretations of these assertions have been considered in the literature. The three types of semantics proposed for when such an assertion should be considered as satisfied are listed below:

Given a source instance D, a global instance B satisfies $\mathcal{M}$

- in a *sound mapping* if $q_{\mathcal{S}}^D \subseteq q_{\mathcal{G}}^B$;
- in an *exact mapping* if $q_{\mathcal{S}}^D = q_{\mathcal{G}}^B$;
- in a *complete mapping* if $q_{\mathcal{S}}^D \supseteq q_{\mathcal{G}}^B$

for every assertion $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$ and $q_{\mathcal{G}} \rightsquigarrow q_{\mathcal{S}}$ in $\mathcal{M}$.

Intuitively, a sound mapping is satisfied if B contains at least the data implied by the assertions, an exact mapping if B contains exactly the implied

data and a complete mapping is satisfied if $\mathbb{B}$ contains at most the implied data.

Global as View — GAV

Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, the idea of GAV mappings is to express the content of the global schema by assigning to each relational symbol of $\mathcal{G}$ a query over the source schema $\mathcal{S}$, that is each relation of the global schema is defined as a view over the source schema. More formally, $\mathcal{M}$ contains only assertions of the form $g \rightsquigarrow q_{\mathcal{S}}$, one for each $g \in \mathcal{G}$. Applying the different interpretations of $\mathcal{M}$, this means that a global instance $\mathbb{B}$ satisfies an assertion $g \rightsquigarrow q_{\mathcal{S}}$ under a sound GAV mapping if $q_{\mathcal{S}}^{\mathbb{D}} \subseteq g^{\mathbb{B}}$, under an exact GAV mapping if $q_{\mathcal{S}}^{\mathbb{D}} = g^{\mathbb{B}}$ and under a complete GAV mapping if $q_{\mathcal{S}}^{\mathbb{D}} \supseteq g^{\mathbb{B}}$.

This can also be expressed by the first-order sentences $\forall \vec{x}\ q_{\mathcal{S}}(\vec{x}) \rightarrow g(\vec{x})$ (for sound GAV mappings), $\forall \vec{x}\ q_{\mathcal{S}}(\vec{x}) \leftrightarrow g(\vec{x})$ (for exact GAV mappings) and $\forall \vec{x}\ g(\vec{x}) \rightarrow q_{\mathcal{S}}(\vec{x})$ (for complete GAV mappings).

Example 3.36. Sticking to the examples of a university library and departments libraries of this university, assume a scenario with two departments and the following schemas:

- $\mathrm{Lib}_{\mathrm{UB}} = \{\mathrm{Books}(\mathrm{ISBN}, \mathrm{Title}, \mathrm{Author}, \mathrm{Id}, \mathrm{Status})\}$
- $\mathrm{Lib}_{\mathrm{Dep}_1} = \{\mathrm{Books}_{\mathrm{Dep}_1}(\mathrm{ISBN}, \mathrm{Title}, \mathrm{Author}, \mathrm{Id}),\ \mathrm{Status}(\mathrm{Id}, \mathrm{Lend})\}$
- $\mathrm{Lib}_{\mathrm{Dep}_2} = \{\mathrm{Books}_{\mathrm{Dep}_2}(\mathrm{Id}, \mathrm{Title}, \mathrm{Author}, \mathrm{Status}),\ \mathrm{Ids}(\mathrm{ISBN}, \mathrm{UBNR})\}$

From this schemas, assume a data integration system with
$\mathcal{G} = \mathrm{Lib}_{\mathrm{UB}}$,
$\mathcal{S} = \mathrm{Lib}_{\mathrm{Dep}_1} \cup \mathrm{Lib}_{\mathrm{Dep}_2}$ and
$\mathcal{M} = \{$
$\{(\mathrm{ISBN}, \mathsf{T}, \mathsf{A}, \mathrm{Id}, \mathsf{S}) \mid \mathrm{Books}_{\mathrm{Dep}_1}(\mathrm{ISBN}, \mathsf{T}, \mathsf{A}, \mathrm{Id}) \wedge \mathrm{Status}(\mathrm{Id}, \mathsf{S})\} \cup$
$\{(\mathrm{ISBN}, \mathsf{T}, \mathsf{A}, \mathrm{Id}, \mathsf{S}) \mid \mathrm{Books}_{\mathrm{Dep}_2}(\mathrm{ISBN}, \mathsf{T}, \mathsf{A}, \mathsf{S}) \wedge \mathrm{Ids}(\mathrm{ISBN}, \mathrm{Id})\}$
$\rightsquigarrow \{(\mathrm{ISBN}, \mathsf{T}, \mathsf{A}, \mathrm{Id}, \mathsf{S}) \mid \mathrm{Books}(\mathrm{ISBN}, \mathsf{T}, \mathsf{A}, \mathrm{Id}, \mathsf{S})\}\}$

Consider a simple source instance $\mathbb{D}$ for $\mathcal{S}$:
$\{\mathrm{Books}_{\mathrm{Dep}_1}(\text{'001'},\text{'THEbook'},\text{'THEauthor'},\text{'b1'}),\ \mathrm{Status}(\text{'b1'},\text{'false'}),$
$\mathrm{Books}_{\mathrm{Dep}_2}(\text{'d2b1'},\text{'alsobook'},\text{'alsoauthor'},\text{'false'}),\ \mathrm{Ids}(\text{'011'},\text{'d2b1'})\}$

And the following instances for $\mathcal{G}$:
$\mathbb{B}_1 = \{\mathrm{Books}(\text{'001'},\text{'THEbook'},\text{'THEauthor'},\text{'b1'},\text{'false'}),$
$\mathrm{Books}(\text{'011'},\text{'alsobook'},\text{'alsoauthor'},\text{'d2b1'},\text{'false'}),$
$\mathrm{Books}(\text{'010'},\text{'abook'},\text{'anauthor'},\text{'b2'},\text{'true'})\}$
$\mathbb{B}_2 = \{\mathrm{Books}(\text{'001'},\text{'THEbook'},\text{'THEauthor'},\text{'b1'},\text{'false'}),$
$\mathrm{Books}(\text{'011'},\text{'alsobook'},\text{'alsoauthor'},\text{'d2b1'},\text{'false'})\}$
$\mathbb{B}_3 = \emptyset$

Thereby $\mathbb{B}_1$ and $\mathbb{B}_2$ would be valid global instances if the mapping is regarded to be sound, $\mathbb{B}_2$ would be a valid global instance under an exact mapping, and $\mathbb{B}_2$ and $\mathbb{B}_3$ are valid global instances under a complete mapping. ◇

This means, for a global instance B of $\mathcal{G}$ to satisfy a sound GAV mapping, every relation of $\mathcal{G}$ has to contain at least the results of the corresponding query over $\mathcal{S}$. To satisfy an exact mapping, B must contain exactly those tuples occurring in these results, and for a complete mapping B must not contain any tuple not present in the results of the corresponding queries.

As a result, exact GAV mappings possess the *single database property*: Given a source instance D, there exists exactly one legal global instance B with respect to D, which consists exactly of the results from evaluating the queries over $\mathcal{S}$ (i.e., it coincides with the view extension of $\mathcal{G}$).

As discussed later, allowing for constraints on the global schema enhances the expressiveness of a GAV system compared to GAV systems where no constraints are allowed over $\mathcal{G}$.

In GAV systems, adding or removing a source may require to reformulate many of the view definitions in $\mathcal{M}$. On the other hand, defining the global schema as a view over the source schema helps query answering, since the view definitions suggest how to retrieve the interesting data from the sources. Hence a GAV system may be a good choice when the set of sources is stable or if the global schema is unstable for some reasons.

Local as View — LAV

LAV mappings have been introduced in [55]. Unlike the straightforward approach of GAV mappings to express the relations of the global schema as views over the sources, the idea of LAV mappings is (given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$) to express the relations of $\mathcal{S}$ (where the data actually resides) as views over the global schema $\mathcal{G}$.

Formally, $\mathcal{M}$ contains only assertions of the form $s \rightsquigarrow q_{\mathcal{G}}$, one for each $s \in \mathcal{S}$. Applying the three different interpretations for $\mathcal{M}$, this means that a global instance B satisfies an assertion $s \rightsquigarrow q_{\mathcal{G}}$ under a sound LAV mapping if $s^D \subseteq q_{\mathcal{G}}^B$, under an exact LAV mapping if $s^D = q_{\mathcal{G}}^B$ and under a complete LAV mapping if $s^D \supseteq q_{\mathcal{G}}^B$.

Just as for GAV mappings, this can be expressed by the first-order sentences $\forall \vec{x}\, s(\vec{x}) \rightarrow q_{\mathcal{G}}(\vec{x})$ (for sound LAV mappings), $\forall \vec{x}\, s(\vec{x}) \leftrightarrow q_{\mathcal{G}}(\vec{x})$ (for exact LAV mappings) and $\forall \vec{x}\, q_{\mathcal{G}}(\vec{x}) \rightarrow s(\vec{x})$ (for complete LAV mappings).

Example 3.37. Assume the same scenario as in Example 3.36, but now defined as LAV mapping, i.e. let $\mathcal{M} = \{$
$\{(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{Id}) \mid \mathrm{Books}_{\mathrm{Dep}_1}(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{Id})\} \rightsquigarrow$
$\quad \{(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{Id}) \mid \mathrm{Books}(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{Id}, \mathrm{S})\}$,
$\{(\mathrm{Id}, \mathrm{S}) \mid \mathrm{Status}(\mathrm{Id}, \mathrm{S})\} \rightsquigarrow \{(\mathrm{Id}, \mathrm{S}) \mid \mathrm{Books}(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{Id}, \mathrm{S})\}$,
$\{(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{S}) \mid \mathrm{Books}_{\mathrm{Dep}_2}(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{S})\} \rightsquigarrow$
$\quad \{(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{S}) \mid \mathrm{Books}(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{Id}, \mathrm{S})\}$,
$\{(\mathrm{ISBN}, \mathrm{Id}) \mid \mathrm{Ids}(\mathrm{ISBN}, \mathrm{Id})\} \rightsquigarrow \{(\mathrm{ISBN}, \mathrm{Id}) \mid \mathrm{Books}(\mathrm{ISBN}, \mathrm{T}, \mathrm{A}, \mathrm{Id}, \mathrm{S})\}\}$

When assuming the same source instance D as in Example 3.36, and also the same target instances B_1, B_2 and B_3, again B_1 and B_2 are valid global instances for a sound mapping, B_2 is a valid global instance under an exact mapping and B_2 and B_3 are valid global instances when assuming a complete mapping. ◇

That is, to satisfy a sound LAV mapping with respect to some source instance D, the content of a target instance B of the global schema must be such that for each relation $S \in \mathcal{S}$, all tuples contained in S must be also contained in the result of the corresponding query over $\mathcal{G}$. On the other hand, a global instance B satisfies a complete LAV mapping if for every source relation $S \in \mathcal{S}$ the result of the corresponding query is contained in S, and B satisfies an exact LAV mapping if the result of the query and the content of S coincide. Obviously, even for exact LAV mappings the single database property does not hold.

Just as for GAV mappings, the expressive power of a LAV system can be enhanced by allowing for constraints on $\mathcal{G}$. LAV settings with relational integrity constraints have been considered for example in [17, 36].

As can be clearly seen from the previous examples, in contrast to GAV systems, adding or removing sources in LAV settings can be done very easily by just adding or removing the corresponding view definitions, without the need to consider the description of the other sources. Hence LAV mappings may be a good choice when the global schema is assumed to be stable, but the set of sources is not. As a result, although query answering requires some more effort than in GAV systems, LAV mappings have been used very successfully in many data integration scenarios (see e.g. [41] for an overview of the impact of LAV).

Global and Local as View — GLAV

GLAV [28] is sometimes referred to as a combination of GAV and LAV, but in fact it is an extension of both. GLAV no longer associates relations of $\mathcal{G}$ or $\mathcal{S}$ with a single query over the other schema, but relates the result of queries over $\mathcal{S}$ to queries over $\mathcal{G}$.

Formally, all assertions in $\mathcal{M}$ are of the form $q_{\mathcal{S}} \rightsquigarrow q_{\mathcal{G}}$, that is both sides of the assertions consist of conjunctive queries. A global instance B satisfies such an assertion with respect to some source instance D if $q_{\mathcal{S}}^{D} \subseteq q_{\mathcal{G}}^{B}$. Hence GLAV mappings can be regarded as sound mappings.

Example 3.38. Assume the same scenario as in the previous two examples. Using GLAV mappings, the mappings between $\mathcal{S}$ and $\mathcal{G}$ could be expressed as

$$\begin{aligned}\mathcal{M} = \{ & \\ & \{(\mathrm{ISBN}, T, A, \mathrm{Id}, S) \mid \mathrm{Books}_{\mathrm{Dep}_1}(\mathrm{ISBN}, T, A, \mathrm{Id}) \wedge \mathrm{Status}(\mathrm{Id}, S)\} \\ & \quad \rightsquigarrow \{(\mathrm{ISBN}, T, A, \mathrm{Id}, S) \mid \mathrm{Books}(\mathrm{ISBN}, T, A, \mathrm{Id}, S)\}, \\ & \{(\mathrm{ISBN}, T, A, \mathrm{Id}, S) \mid \mathrm{Books}_{\mathrm{Dep}_2}(\mathrm{ISBN}, T, A, S) \wedge \mathrm{Ids}(\mathrm{ISBN}, \mathrm{Id})\} \\ & \quad \rightsquigarrow \{(\mathrm{ISBN}, T, A, \mathrm{Id}, S) \mid \mathrm{Books}(\mathrm{ISBN}, T, A, \mathrm{Id}, S)\}\}\end{aligned}$$

Assuming again D from Example 3.36 as source instance, and the global instances B_1, B_2 and B_3, then only B_1 and B_2 satisfy the above mapping. ◇

3.2.3 *Comparing GAV and LAV*

GAV style mappings are often referred to as possessing a procedural style for mapping definition, because they define how to retrieve the data for the global instance using queries over the sources. Contrary, LAV style mappings

are referred to as a declarative way of defining mappings, since they describe the data in the sources with respect to the global schema, but on the first sight state nothing about how to retrieve the data.

Accordingly, query answering in LAV systems conceptually is no easy task, while in most of the GAV settings query answering consists of an unfolding of the query according to the view definitions (for example in exact GAV settings, because of the single database property of such settings, query answering is the same as answering a query over a view in an RDBMS).

Therefore, the question whether a LAV system can be transformed into a GAV system (or vice versa) is not only interesting for comparing the expressive power of the two approaches, since such transformations would also allow to translate a declarative description of a data integration system into a more procedural one (or vice-versa).

Such transformations between GAV and LAV systems in a restricted setting have been studied in [9, 11]. Thereby, two systems are considered equivalent when they yield the same results for query answering, that is when both return the same results for all queries (given the same source instance). This is formalised by the following definition:

Definition 3.39 (Query preserving [11])**.** Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and $\mathcal{I}' = \langle \mathcal{G}', \mathcal{S}, \mathcal{M}' \rangle$ be two data integration systems over the same source schema, and let $\mathcal{G} \subseteq \mathcal{G}'$. Then $\mathcal{I}'$ is *query-preserving with respect to* $\mathcal{I}$, if for every query q to $\mathcal{I}$ and every source instance D it holds that $\mathtt{certain}(q, \mathcal{I}, D) = \mathtt{certain}(q, \mathcal{I}', D)$. ⊣

Hence the idea is to compare two data integration systems with respect to their behaviour towards a user querying the system.

Definition 3.40 (Query reducible [11])**.** Let C_1, C_2 be two classes of data integration systems. C_1 is *query-reducible* to C_2 if there exists a function $f\colon C_1 \to C_2$ such that for each $\mathcal{I} \in C_1$, $f(C_1)$ is query preserving with respect to $\mathcal{I}$. ⊣

Hence a class of data integration systems C_1 is query-reducible to a class of data integration systems C_2 if all data integration systems of C_1 can be expressed query preserving as a system of C_2, that is if C_2 has at least the expressive power of C_1.

In [9, 11], only sound mappings and relational schemas have been considered, and all queries (those used for defining the views and those posed over $\mathcal{G}$) are restricted to be conjunctive queries. If in such settings, there are no constraints allowed over $\mathcal{G}$, then the class of GAV and the class of LAV systems are incomparable.

Theorem 3.41 ([11])**.** *The class of GAV data integration systems is not query-reducible to the class of LAV systems.*

Theorem 3.42 ([11])**.** *The class of LAV data integration systems is not query-reducible to the class of GAV systems.*

As already stated before, allowing for constraints on $\mathcal{G}$ enhances the expressive power of GAV and LAV systems. This can be shown by consideration of the following two types of constraints on $\mathcal{G}$.

Definition 3.43 ([11])**.**

- A *single-head full TGD* is a TGD of the form $\varphi(\vec{x}) \rightarrow R(\vec{x})$,
- a *simple EGD* is an EGD of the form $P(\vec{x}) \rightarrow x_i = x_j$,

where R and P are single relational symbols. ⊣

Then the following results were shown by [11]:

Theorem 3.44 ([11])**.** *The class of GAV data integration systems is query reducible to the class of LAV systems where only single-head full TGDs are allowed on $\mathcal{G}$.*

Theorem 3.45 ([11])**.** *The class of LAV data integration systems is query reducible to the class of GAV systems where only inclusion dependencies and simple EGDs are allowed on $\mathcal{G}$.*

Moreover it was shown that the size of the resulting systems is linear in the size of the original system.

It was further stated above that GLAV mappings can be regarded as a kind of generalisation of GAV mappings. This intuition is supported by a result in [11] stating that each GLAV system is query-reducible to a GAV system with inclusion dependencies and simple EGDs. Therefore, by enhancing the expressive power of GAV systems they are able to express GLAV systems.

3.2.4 *Query Answering*

Recall that given a data integration system $\mathcal{I}$ and a source instance D (since there may be more than one legal global instance B for $\mathcal{I}$) the certain answer semantics is applied for query answering (Definition 3.35). Therefore, considering query answering under complete semantics is trivial, since $\emptyset$ is always a valid global instance, and therefore $\mathtt{certain}(q, \mathcal{I}, D)$ is always $\emptyset$. Consequently in the following section only the exact and sound semantics are considered.

Query Answering under GAV Mappings

In exact GAV systems without constraints on $\mathcal{G}$, because of the single database property, query answering can be done by simply unfolding the query according to the view definitions. This means, given a query q over $\mathcal{G}$, replace in q every relational symbol $g \in \mathcal{G}$ by the query from the corresponding view definition $g \rightsquigarrow q_S$. The resulting query over $\mathcal{S}$ returns the certain answers to q over $\mathcal{G}$. For monotone queries (in particular for queries not using negation), the same strategy can be applied to sound GAV mappings [10], as long as there are no constraints on $\mathcal{G}$.

Allowing for constraints on $\mathcal{G}$ has only little effect on query answering under exact mappings: Either the single legal global instance B that satisfies the mappings also satisfies the constraints on $\mathcal{G}$, then query answering gives the same results as without those constraints. Otherwise no legal global instance exists, and query answering becomes trivial.

Assuming sound mappings, the situation is different: Since sound mappings allow legal global instances to contain facts not immediately derivable from

the mappings, the constraints may imply additional answers that cannot be derived when given only the source relations and the view definitions. Therefore simple unfolding of the queries may no longer suffice to compute the answers to a query. Such a case, assuming keys and foreign keys on $\mathcal{G}$, is given in the following example.

Example 3.46. Assume a variation of the scenario in Example 3.36: Consider a data integration system with a global schema $\mathcal{G} = \{\mathsf{Books}(\mathsf{ISBN}, \mathsf{Title}, \mathsf{Author}, \mathsf{Id}, \mathsf{Status}), \mathsf{OldBooks}(\mathsf{ISBN}, \mathsf{Title}, \mathsf{Author}, \mathsf{Id}, \mathsf{Status})\}$ with a key constraint $\mathsf{key}(\mathsf{Books}) = \{\mathsf{Id}\}$ and a foreign key dependency $\mathsf{OldBooks}[\mathsf{Id}] \subseteq \mathsf{Books}[\mathsf{Id}]$ defined over $\mathcal{G}$.

Further assume that from some source instance D' of the source schema $\mathcal{S}$, by the mapping $\mathcal{M}$ the fact $\mathsf{OldBooks}(\text{'old011'}, \text{'oldBook'}, \text{'diedalready'}, \text{'00'}, \text{'true'})$ can be derived.

Given a query $\mathsf{q} = \{\mathsf{Id} \mid \mathsf{Books}(\mathsf{ISBN}, \mathsf{Title}, \mathsf{Author}, \mathsf{Id}, \mathsf{Status})\}$, because every instance for $\mathcal{G}$ satisfying both, $\mathcal{M}$ (since assuming a sound mapping) and the foreign key dependency has to contain some fact Book with $\mathsf{Id} = \text{'old011'}$, it can be derived that ('old001') must be part of the certain answers to q over $\mathcal{G}$. ◇

Therefore other techniques for query answering are needed. Moreover, it remains to be justified for which constraints over $\mathcal{G}$ query answering is indeed decidable and in which of these cases it is even tractable.

As it will be mentioned in Section 3.3, sound GAV mappings can be interpreted as a restricted form of source to target TGDs. Hence by assuming target TGDs and target EGDs as constraints on $\mathcal{G}$, one possibility for query answering is to chase the source instance according to the mappings and constraints. So the idea is - just for query answering - to materialise the data in $\mathcal{G}$, process the query over the materialised instance and then to drop the instance again. In this case the same results for decidability and tractability of the problem apply as in data exchange.

However, for data integration settings most of the time a different kind of constraints has been considered over the global schema, namely key constraints and inclusion dependencies. When allowing only those types of dependencies over $\mathcal{G}$, the border between decidability and undecidability is well known. Moreover, computing certain answers in the decidable cases can be done in polynomial time (data complexity). In the following, a summary of some of these results is given.

Many of these results have not been derived for a data integration setting but for single schemas with constraints only. But by considering $\mathcal{G}$ as single schema and interpreting the view extensions (that is the results when evaluating the queries from the view definitions over $\mathcal{S}$) as an instance for $\mathcal{G}$, the results carry immediately over to data integration systems with sound GAV mappings.

In the general case, query answering under key constraints and inclusion dependencies is undecidable. However, for restricted classes of inclusion dependencies the problem becomes decidable. The next two definitions show the border between decidability and undecidability.

Definition 3.47 (Non key conflicting inclusion dependency [13])**.** Let $\mathbf{R}$ be a relational schema and $\mathcal{K}$ a set of key dependencies over $\mathbf{R}$. An inclusion dependency $R_i[A_1] \subseteq R_j[A_2]$ ($R_i, R_j \in \mathbf{R}$) is a *non key conflicting inclusion dependency (NKCID) with respect to* $\mathcal{K}$ if either

- no key dependency is defined for R_j in $\mathcal{K}$ or
- a key dependency $key(R_2) = K$ is defined in $\mathcal{K}$, and A_2 is not a strict superset of K ($K \not\subset A_2$).

Moreover, let $\mathcal{D}$ be a set of inclusion dependencies over $\mathbf{R}$, $\mathbf{R}$ is *non key conflicting (NKC)* (with respect to $\mathcal{D}$ and $\mathcal{K}$) if all $d \in \mathcal{D}$ are NKCIDs with respect to $\mathcal{K}$. ⊣

Note that, under set semantics of instances, whenever there is no key defined for a relation, the set of all attributes of this relation can be regarded as key.

Intuitively, NKCIDs do not propagate supersets of keys of the relation on their right hand side. As will be discussed later in more detail, this has the effect that adding NKCIDs to a schema cannot lead to key violations. Non key conflicting inclusion dependencies can be regarded as a generalisation of *foreign keys*, which can be expressed as the class of inclusion dependencies $R_i[A_1] \subseteq R_j[A_2]$ such that $key(R_j) = A_2$. Another class of NKCIDs are the *foreign key dependencies*, that are inclusion dependencies $R_i[A_1] \subseteq R_j[A_2]$ such that $A_2 \subseteq key(R_j)$.

A slightly less restrictive class than NKCIDs are the *1 key conflicting inclusion dependencies*.

Definition 3.48 (1 key conflicting inclusion dependencies (1KCIDs))**.** Let $\mathbf{R}$ be a relational schema and $\mathcal{K}$ a set of key dependencies over $\mathbf{R}$. An inclusion dependency $R_i[A_1] \subseteq R_j[A_2]$ ($R_i, R_j \in \mathbf{R}$) is a *1 key conflicting inclusion dependency (1KCID) with respect to* $\mathcal{K}$ if either

- it is a NKCID or
- there exists a key constraint $key(R_2) = K$ in $\mathcal{K}$ such that $K \subset A_2$ and A_2 contains a single additional attribute of R_j not in K.

Moreover, let $\mathcal{D}$ be a set of inclusion dependencies over $\mathbf{R}$, then $\mathbf{R}$ is *one key conflicting (1KC)* (with respect to $\mathcal{D}$ and $\mathcal{K}$) if all $d \in \mathcal{D}$ are 1KCIDs with respect to $\mathcal{K}$. ⊣

In [13] it was shown that while computing the certain answers to a conjunctive query over a relational schema with key dependencies and NKCIDs is decidable and can be even done in polynomial time (data complexity), query answering under 1KCIDs is undecidable.

Theorem 3.49 ([13])**.** *Let $\boldsymbol{R}$ be a 1KC relational schema with respect to the set of key dependencies $\mathcal{K}$ and the set of 1-key-conflicting inclusion dependencies $\mathcal{D}$. Let further q be a conjunctive query over $\boldsymbol{R}$. Given an instance R for $\boldsymbol{R}$, computing $certain(q, \boldsymbol{R}, R)$ is undecidable.*

The proof of this theorem is done by reducing the problem to the implication problem for key dependencies and 1KCIDs, which was shown to be undecidable (by reduction from the implication problem of functional dependencies and inclusion dependencies) within the same paper.

Theorem 3.50 ([13]). *Let $\boldsymbol{R}$ be a NKC relational schema with respect to the set of key dependencies $\mathcal{K}$ and the set of non-key-conflicting inclusion dependencies $\mathcal{D}$. Let further be q a conjunctive query over $\boldsymbol{R}$. Given an instance R for $\boldsymbol{R}$, computing $\mathtt{certain}(q, \boldsymbol{R}, R)$ is solvable in polynomial time (data complexity).*

This theorem is proven by providing an algorithm that computes the correct answer. Both, the algorithm and the proof that the algorithm is correct rely heavily on the results of [45]. The idea is to extend the chase (see Section 3.1.2) by a chase rule for inclusion dependencies. Although for cyclic inclusion dependencies the corresponding chase sequence does not terminate, it was shown in [45] that for certain problems it is sufficient to consider only a finite part of the beginning of the chase sequence.

Note that as stated before, these two results immediately apply to sound GAV systems with the corresponding constraints over $\mathcal{G}$ as well.

Another important result that is not only used in the proof of Theorem 3.50, but which is the basis for many results concerning key dependencies and NKCIDs is the separation property:

Theorem 3.51 (Separation [45]). *Let $\boldsymbol{R}$ and $\boldsymbol{S}$ be two identical relational schemas, $\mathcal{K}$ be a set of key dependencies over $\boldsymbol{R}$ and $\mathcal{D}$ a set of NKCIDs over $\boldsymbol{R}$, respectively a set of inclusion dependencies over $\boldsymbol{S}$. Let further q be a conjunctive query over $\boldsymbol{R}$ (and therefore also $\boldsymbol{S}$).*

Given an instance D for $\boldsymbol{R}$ and $\boldsymbol{S}$, it holds that $\mathtt{certain}(q, \boldsymbol{R}, D) = \mathtt{certain}(q, \boldsymbol{S}, D)$ iff D is consistent with $\mathcal{K}$.

This theorem justifies the already mentioned idea of the nice property of NKCIDs that they cannot introduce a key violation: If an instance is consistent with respect to key dependencies, then adding NKCIDs cannot make the instance inconsistent. For query answering under key dependencies and NKCIDs, this means that the key dependencies and the NKCIDs can be considered separately: It suffices to first check the given instance for being consistent with the key dependencies, and if this is the case then the NKCIDs can be handled without needing to take care for constraint violations.

The results presented in [13] are rather of theoretical interest, since the stated algorithm for query answering is quite involved and probably not very efficient (although running in polynomial time data complexity).

A more practical result for query answering is given in [12]. There the problem of query answering in a sound GAV setting with key constraints and inclusion dependencies over $\mathcal{G}$ is considered. For the case when the inclusion dependencies are restricted to NKCIDs (that is for the maximal class of inclusion dependencies for which query answering is still decidable), an algorithm is given, that, given a UCQ q over $\mathcal{G}$ computes a perfect rewriting q' of q (w.r.t. the certain answers) over the key constraints and inclusion dependencies. Thereby *perfect rewriting* means that if q' is evaluated over

some instance D of $\mathcal{G}$, it returns exactly the certain answers to q over D regardless of whether D satisfies the constraints over $\mathcal{G}$ or not. The result is again a UCQ over $\mathcal{G}$ that can then be translated using unfolding to queries over the source schema $\mathcal{S}$. Because of this, the rewriting algorithm is not only applicable to GAV settings. Based on the separation property (Theorem 3.51), the algorithm consists of two parts, one handling the inclusion dependencies and one handling the key constraints. In Section 5.6.1 query rewriting with respect to inclusion dependencies will be introduced in detail, as this is an important part of the implementation described in Chapter 6. Because the handling of key constraints is not needed, there will be no more detailed description of it. The idea behind the rewriting with respect to the key constraints is to check whether some key constraint is violated, and if this is the case to ensure that the answer to the query contains every possible tuple over the active domain of the instance the query is evaluated on. If there is no key violation, the key constraints have no effect, as suggested by the separation property.

In [70], the border between decidability and undecidability of query answering under key constraints and inclusion dependencies has been studied even further. Given a schema $\mathbf{R}$, a set $\mathcal{K}$ of key constraints and inclusion dependencies over $\mathbf{R}$ and an instance D for $\mathbf{R}$, the paper distinguishes between computing the set of certain answers to a query q over $\mathbf{R}$ and computing the set of answers that belong to the result of the query over all finite instances D' of $\mathbf{R}$ that satisfy D and $\mathcal{K}$ (note that the certain answers are the set of answers that belong to the result of the query over all — finite and infinite — instances of $\mathbf{R}$ that satisfy D and $\mathcal{K}$).

It is shown, while for sets of key constraints and foreign key dependencies these two kinds of answers coincide, this no longer holds for sets of key dependencies and NKCIDs. Moreover, it was shown that computing the set of answers only over finite instances is undecidable for NKCIDs. More formally this is described by the following theorem:

Theorem 3.52 ([70]). *Let $\mathbf{R}$ be a schema, and $\mathcal{K}$ be a set of key dependencies and foreign key dependencies. Given an instance D of $\mathbf{R}$, denote with $\mathcal{D}$ the set of all instances D' of $\mathbf{R}$ that satisfy $\mathcal{K}$ s.t. there exists a homomorphism h from D to D'. Moreover let $\mathcal{D}_f$ contain all finite instances in $\mathcal{D}$. Given a conjunctive query q, then*

$$\bigcap_{B \in \mathcal{D}} q^B = \bigcap_{B \in \mathcal{D}_f} q^B \ .$$

From this result together with the result from Theorem 3.50 it follows immediately that computing the set of answers over all finite instances is tractable.

Theorem 3.53 ([70]). *Let $\mathbf{R}$ be a schema, and $\mathcal{K}$ be a set of key dependencies and NKCIDs. Given an instance D of $\mathbf{R}$, denote with $\mathcal{D}_f$ the set of all finite instances D' of $\mathbf{R}$ that satisfy $\mathcal{K}$ s.t. there exists a homomorphism h from D to D'. Then computing*

$$\bigcap_{B \in \mathcal{D}_f} q^B$$

is undecidable.

Note that for the unrestricted case (i.e. computing the certain answers), query answering under NKCIDs is still decidable, while becoming undecidable only under 1KCIDs.

For all the results in this section concerning query answering under GAV mappings and constraints on the global schema, the language of the query q has been considered to be fixed as the language of conjunctive queries, and only different types of constraints over the global schema have been considered. As the following result from [70] shows, the expressive power of the query language cannot be extended.

Theorem 3.54 ([70])**.** *Let* q *be a UCQ with inequalities,* D *an instance of a relational schema* $\boldsymbol{R}$ *and* $\mathcal{K}$ *a set of inclusion dependencies over* $\boldsymbol{R}$*. Then computing the certain answers to* q *over* R *under* $\mathcal{D}$ *with respect to* D *is undecidable for both cases: when considering only finite instances and when considering all instances.*

Note that in the above theorem, no key constraints occur. As mentioned earlier, since in this case the set of all attributes of each relation can be regarded as key, this corresponds to a setting with foreign key dependencies.

Query Answering under LAV Mappings

Query answering in LAV systems is regarded to be more complicated than in GAV systems. This is due to the fact that only the content of the views is known, and from this the content of the base tables (i.e. the global schema) has to be inferred. This kind of query processing is referred to as *view based query processing*. There exist two main approaches to view based query processing: *view based query answering* and *view based query rewriting*.

Not surprisingly, the complexity of query processing in LAV systems depends on three parameters: The query language used for defining the views, the query language used for expressing the query q and the semantics considered for the mappings (sound, complete or exact). Thereby often sound mappings in combination with relatively simply query languages are considered. A possible reason for this are the complexity results presented below for view based query answering.

While sound mappings correspond to applying the open world assumption on the data in $\mathcal{S}$, using exact mappings means to apply the closed world assumption to the data in the view extensions, which in many cases makes query answering harder. In the following, assume again a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$.

VIEW BASED QUERY REWRITING In view based query rewriting, the task is, given a query q over $\mathcal{G}$ together with the assertions in $\mathcal{M}$, to transform q into some query q^* over $\mathcal{S}$. Thereby the language of q^* is fixed (but it need not be the same as the language of q), and processing q^* over $\mathcal{S}$ should ideally return the certain answers for q over $\mathcal{G}$. A rewriting of q that satisfies these requirements is called a *perfect rewriting*. In the following we will use the term rewriting to denote both, the actual rewriting process and its result, i.e. the rewritten query q^*. Note that the perfect rewriting of a query is independent of the source data.

sound	CQ	$CQ^{\neq}$	UCQ	Datalog	FOL
CQ	in P	coNP	in P	in P	undec.
$CQ^{\neq}$	in P	coNP	in P	in P	undec.
UCQ	coNP	coNP	coNP	coNP	undec.
Datalog	coNP	undec.	coNP	undec.	undec.
FOL	undec.	undec.	undec.	undec.	undec.
exact	CQ	$CQ^{\neq}$	UCQ	Datalog	FOL
CQ	coNP	coNP	coNP	coNP	undec.
$CQ^{\neq}$	coNP	coNP	coNP	coNP	undec.
UCQ	coNP	coNP	coNP	coNP	undec.
Datalog	undec.	undec.	undec.	undec.	undec.
FOL	undec.	undec.	undec.	undec.	undec.

Table 1: Data complexity of view based query answering in LAV data integration systems under sound and exact mappings for different query languages as stated in [1]. The languages used for querying are listed horizontally, those used within the view definitions vertically.

However, it may be possible that such a query q^* does not exist, for example if the required query cannot be expressed in the language of q^*. Then the goal is to achieve a *maximally contained rewriting*, which is regarded to be the best possible rewriting. More formally: A maximally contained rewriting q^* is a rewriting such that for every source instance D over $\mathcal{S}$, ${q^*}^D \subseteq q^D$ and for all queries q' (in the same language as q^*) it holds that if ${q^*}^D \subseteq {q'}^D \subseteq q^D$, then q^* and q' are equivalent. Note that every perfect rewriting trivially is a maximally contained one.

Obviously those classes of queries are of special interest whose maximally contained rewriting (being perfect or not) can be expressed by a query language that can be evaluated efficiently over $\mathcal{S}$. This is for example the case for conjunctive queries if the views are also defined only using conjunctive queries. Then their perfect rewriting is a UCQ [54]. [54] also states that *"already for very simple query languages containing union"*([54], page 7) the perfect rewriting cannot be evaluated efficiently in general. Moreover, it gives many references to results concerning query rewriting in several different settings.

VIEW BASED QUERY ANSWERING Although looking similar to view based query rewriting, view based query answering is something different. In view based query answering, in addition to the query q over $\mathcal{G}$, and the view definitions, also the extensions of the views are given and the task is to compute the certain answers of q over $\mathcal{G}$. Unlike in view based query rewriting, where computing the solutions is clearly divided into two steps, of which the first step is independent of the data in the views, in view based query answering there are no restrictions on how the answers are derived, and

the view extensions can be always used for computing the answers. Moreover, another way to define a perfect rewriting q^* is to demand that the result of q^* over $\mathcal{S}$ retrieves the same tuples as retrieved by view based answering.

In [1], the complexity of view based query answering in LAV settings under sound and exact semantics is studied. The results reported there are summarised in Table 1.

It follows immediately that query processing using view based query rewriting, which creates perfect rewritings and then evaluates this rewriting over the source schema, cannot perform better than these results, since otherwise the same strategy could be used when processing the query using view based query answering.

Beside relational global schemas, view based query answering has been also studied for semistructured global schemas (and related problems), but here only the relational cases are considered.

3.3 "COMPARISON" OF DATA EXCHANGE WITH DATA INTEGRATION

In this section, we summarise some of the important differences between data exchange and data integration and on the other side draw some connections between them. First of all, data exchange and data integration describe two different settings and cover different tasks: In data exchange, one wants to

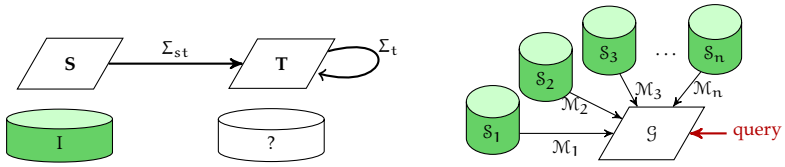

Figure 4: The overall structure of a data exchange setting (left) and a data integration system (right)

migrate data between two different schemas. Therefore the goal is, given a source instance, to materialise an instance which represents the source data as best as possible (according to the mapping defined between source and target schema). The materialised instance can then be used completely independently and autonomously from the source data. For example, query answering can be performed only on the materialised target instance. The main problem in data exchange is to compute such an instance. Therefore one is interested under which circumstances and for which kinds of data exchange settings such a solution exists at all and can be (efficiently) computed. Moreover, if there are several possible solutions, one wants to know which one should be selected for materialisation.

In data integration on the other hand, the goal is to provide a unified view to a set of autonomous sources. Although being also modeled as a source schema (conceptually, there is no problem with defining different, autonomous sources by a single source schema) and a target schema (called *global schema*) with a mapping defined between them, the task is not to materialise any data

under the global schema. Instead, all data resides in the sources. The user on the other hand does not interact with the sources directly, but all queries are posed over the global schema. The task of the data integration system is then to extract the correct results from the sources and to return those results again expressed in terms of the global schema to the user.

Although both approaches are quite different from each other, the formalisms used in both areas can be related to each other up to some extent.

First of all it should be noted that for query answering, in both data exchange and data integration, the notion of certain answers, which originates in the study of incomplete databases, has been adopted.

The more interesting relations however exist between the kinds of mappings used in both areas (see e.g. [52]): In Section 3.2.2, it was stated that a sound GAV mapping $g \rightsquigarrow q_S$ (interpreted as $q_S^D \subseteq g^B$ for a source instance D and a legal global instance B) can be expressed by the first-order sentence $\forall \vec{x}\,(q_S(\vec{x}) \rightarrow g(\vec{x}))$. Both, the description of the interpretation and this logical description resemble a restricted form of source-to-target TGDs. A sound GAV mapping can therefore be expressed by source-to-target TGDs of the form $\varphi(\vec{x}) \rightarrow T(\vec{x})$, where T is a single relational symbol of the global schema. Moreover, all $x \in \vec{x}$ appearing in $T(\vec{x})$ must also occur in $\varphi(\vec{x})$ (i.e. $T(\vec{x})$ must not contain existentially quantified variables). According to this equivalence, data exchange settings without target dependencies where all source-to-target dependencies are of the above form are sometimes referred to as a GAV setting.

For sound LAV mappings $s \rightsquigarrow q_G$, the situation is similar. They are interpreted as $s^D \subseteq q_G^B$, and can be expressed by the first-order sentence $\forall \vec{x}\,(s(\vec{x}) \rightarrow q_G^B)$. Therefore they can be expressed by source-to-target TGDs of the form $S(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$, where S is a single relational symbol of the source schema, and all $x \in \vec{x}$ actually have to occur in $\psi(\vec{x}, \vec{y})$. Accordingly, data exchange settings without target dependencies where all source-to-target dependencies are of the above form are sometimes referred to as LAV settings.

It is immediate to see, that a data exchange setting without target dependencies, but arbitrary source-to-target TGDs generalises both sound GAV and sound LAV settings from data integration. Moreover, such a setting corresponds exactly to a sound GLAV setting in data integration.

Data exchange and data integration also differ with respect to the constraints allowed over the target/global schema. While in data integration most of the time only key constraints and inclusion dependencies over the global schema are considered, for data exchange settings the more general forms of TGDs and EGDs are studied. This leads to different concepts for the set of maximal expressive constraints such that query answering remains decidable. While in data exchange weakly-acyclic sets of TGDs are the least restrictive types of constraints that allow query answering to be tractable, in data integration key dependencies and NKCIDs are considered as the maximal expressive but yet tractable types of constraints over the global schema.

Note that these two types of constraints are incomparable: A set of key constraints and inclusion dependencies can be obviously modeled by a set of EGDs and TGDs. But since in data integration no restrictions are imposed

on the cyclicity of the NKCIDs, the corresponding set of TGDs needs not to be weakly acyclic. On the other hand, even with acyclic TGDs it is obviously possible to express constraints that are not expressible by inclusion dependencies. Moreover the constraints that can be defined by a set of EGDs include all constraints definable by a set of key constraints.

4 PEER DATA MANAGEMENT

Data exchange and data integration settings always consist of two participants[1], each having a certain role: There exists one dedicated source and one dedicated target. Both are described by a schema, and the relationship between the source and the target is expressed by a mapping between these two schemas (as depicted in Figure 4 in the last chapter). A natural extension of both settings is to allow for more than only two participants in a setting, and not to restrict each of them to a certain role (source or target), but to allow them to offer and retrieve data at the same time (i.e. to be source and target). In such an extended setting, each member in the system (called *peer*) may offer some data through its own public schema and relate its own data to the data offered by other peers by defining mappings between the public schemas of some other peers and its own schema. Such a setting is depicted in Figure 5. As an extension of data exchange and data integration, beside

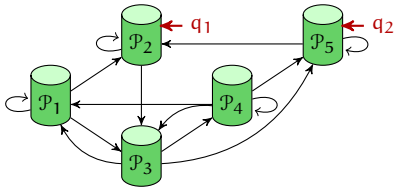

Figure 5: Idea of a Peer Data Management setting: Every peer may define local constraints on its schema, dependencies between peer schemas are expressed by mappings, and queries are posed against the schema of a single peer.

mappings between peers, also local constraints over the schema of each peer may be possible. In such a setting, queries are posed over the public schema of a single peer, and are answered by taking into account both: the data actually stored at that peer, as well as all data derivable by the mappings from the other peers.

4.1 PEER DATA MANAGEMENT SYSTEMS

The idea sketched above is captured by the notion of Peer Data Management Systems (PDMSs). The goal of PDMSs is to provide support for data sharing between autonomous sources. Thereby the setup costs for initialising and joining such a system should be as low as possible.

The goal of this section is to introduce the idea of PDMSs by first referring to systems covering the same tasks but having certain drawbacks, and then by

1 Although in data integration the data is assumed to reside in many different sources, all these sources are modeled by a single schema.

showing how PDMSs try to overcome these drawbacks. In the second part of this section, three general classes of PDMSs proposed in the literature are presented.

4.1.1 *Related Techniques*

Peer-to-Peer Systems

Peer-to-peer (P2P) filesharing systems like Napster or Gnutella match most of the requirements identified at the beginning of this section and have been already used extremely successfully. The strength of such P2P systems stems from their flexible architecture: They do not distinguish between dedicated servers (sources that provide data) and clients (targets that consume data). Instead, they, conceptually[2], consist of a network of equal nodes (called *peers*), with each peer providing the same functionality and acting as both, server (source) and client (target) at the same time. Therefore P2P systems are very flexible with respect to changes in the membership of peers (every peer may join and leave at any time) and robust with respect to failures of peers. Additionally, because of their decentralised architecture, they do not need any centralised coordination, administration or infrastructure.

The major drawback of P2P filesharing systems is their applicability to only large granular data which is most of the time only referenced by an identifier (like e.g. files that are identified by their filenames). They also possess only very limited data management capabilities.

Example 4.1. Assume a P2P filesharing network whose members share scientific articles.

Retrieving an article whose author and title are known is a typical task supported by such systems. On the other hand, queries like: 'return a list of abstracts of all articles concerning "Peer Data Management" ordered by the publishing dates of the articles' are not supported.

Moreover assume that several biology research groups want to share the contents of their genome databases. This would be far out of the scope of traditional P2P filesharing systems. ◇

As it was noticed in [35], these shortcomings of traditional P2P systems are due to their lack of considering the semantics of the data: They neither take into account the structure of the data nor do they allow to express relationships between data items. Because of this, they offer only very weak semantics (if any at all), and do not support the management and sharing of fine-granular data. Therefore they are also unable to support even simple data transformation (like returning the ordered list of abstracts mentioned in Example 4.1). It was further observed that supporting all theses properties belongs exactly to the strengths of database systems.

2 The concrete implementations of such systems may (e.g. for performance reasons) depend on some centralised infrastructure or distinguish certain super-peers.

Multi-Databases/Federated Databases

Given a set of autonomous database instances, according to [7] collaboration between them is achieved in terms of federated databases (FDBSs) [73] and multi-databases (MDBSs) [8]. Within these systems, coordination and collaboration between the already existing database systems is based on a federated (or global) schema: The global schema is used to relate the local schemas of the participating databases to each other. This is done by mapping the local schemas using GAV and LAV mappings to the global schema. Like in data integration, this global schema is then used to control and coordinate the access to the data stored in the local database instances. The main difference between FDBSs and MDBSs is that the latter allows for several different global schemas, while FDBSs only contain a single global schema. The different global schemas in MDBSs allow for more autonomy of the member databases.

However, with respect to flexibility and low setup costs, the need for a global schema is the major drawback of these systems. First of all, it has to be possible to express an appropriate schema at all. If this can be done, the participants then have to agree on a certain global schema. Here the possibility offered by MDBSs to use several schemas for different tasks or for data sharing with different member databases may help to overcome some of these problems and at the same time allow for more autonomy of the different members. However, someone has to be responsible for setting up and maintaining the global schema, and since queries to the combined system are posed over this global schema, it also requires some infrastructure that controls query answering (as well as for the other coordination tasks). A possible setting where it may be problematic to find someone taking the responsibility for maintaining the global schema could be for example when different companies want to share some of their data, but none of them wants to be responsible [41].

Relationship to PDMSs

The idea of PDMSs[3] is to combine the flexibility and level of distribution of the peers from P2P systems with the strong semantics and high expressive power of database systems. Therefore in PDMSs the global schema is omitted, and all coordination is done between pairs of peers only. Instead of mapping the local (peer) schemas to a global schema, the mappings are now defined directly between pairs of peer schemas (see Figure 6). Another difference between PDMSs and MDBSs is how query answering works: In MDBSs, the query is posed over the global schema. The MDBS then issues corresponding subqueries (according to the mappings) to each local database, collects the

3 As already mentioned in the introduction, the term *PDMS(s)* is not used unambiguously. For example, in [7], PDMS is used to denote every kind of P2P system used for data sharing, including P2P filesharing systems. In such cases, a collection of database systems connected by the described P2P approach is often referred to as *Peer Database System (PDBS)*, to stress that the nodes of the network are actually databases. Other authors (e.g. [43]) use the term PDMSs to only denote the set of PDBSs. Sometimes PDMS is even used to only refer to a very special kind of a PDBS. To avoid unnecessary confusion, in this thesis only the term PDMS(s) is used as introduced in this chapter (i.e. with the same meaning as the term PDBS in [43]).

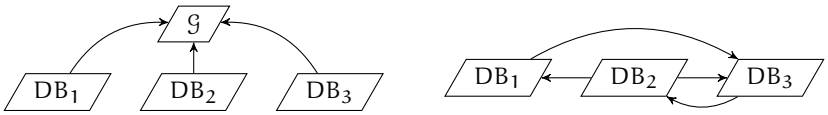

Figure 6: Structure of federated/multi database systems (left) and PDMSs (right).

answers and returns them as results to the initial query. In PDMSs, every query is posed against the schema of a single peer. This peer then evaluates the query over its local schema, and decides whether (parts of) the query needs to be forwarded (maybe in a rewritten form) to some of its neighbours. If the query is indeed forwarded, then the receiving peer performs exactly the same steps and returns the results to the first peer.

In [7], a classification of distributed database systems presented in [66] is extended to also cover PDMSs (see Figure 7). In this classification, each system is characterised along the three dimensions *distribution*, *autonomy* and *heterogeneity*. The classifications of some distributed database systems are

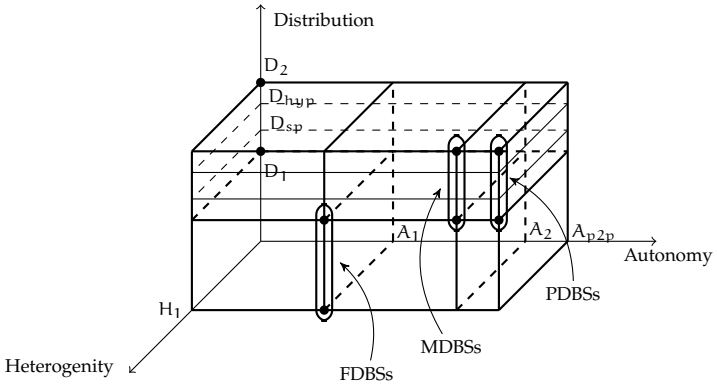

Figure 7: Classification of distributed data management systems along the dimensions distribution, autonomy and heterogeneity [7].

marked in Figure 7. For a more detailed description of these classifications please refer to [7, 66], as in the following we will only comment shortly on the classification of PDMSs.

While [66] considers only three levels of autonomy, [7] adds another level (A_{p2p}) to point out that for the reasons mentioned above (especially with respect to query answering), the autonomy of the members in a PDMS is higher than those of members in a MDBS. A_{p2p} is defined to reflect *full autonomy*. Moreover, with respect to heterogeneity, PDMSs are considered to be fully heterogeneous, since it is by no means required that all peers are equal, not even that they all implement the same data model. Finally, PDMSs are considered to span several levels of distribution to underline that there have been several proposals in the literature that are based for example on super-peers, make use of distributed hashtables or maintain some global

indices. The PDMS described in Chapter 5 and implemented as part of this thesis can be classified (w.r.t. to the level of distribution) as pure P2P system (D_2), requiring no centralised coordination.

4.1.2 *Classes of PDMSs*

Above, PDMSs were introduced as combinations of MDBSs and P2P systems. In the literature, several proposals for PDMSs have been made. On the one hand, concrete prototypes have been presented (a selection is introduced in Section 4.3), and on the other hand formalisation techniques and frameworks for modeling and describing PDMSs were suggested (see Section 4.2). As a rule of thumb, within all these suggestions, three different classes of PDMS can be distinguished according to the information exchanged by the peers: queries, data or updates.

The most common approach to PDMSs is to exchange queries between the peers (e.g. [15, 38]). Such systems can be seen as a generalisation of data integration systems: The mappings between the peers are only "virtual", meaning that they do not require data to be materialised in order to satisfy them. Instead, all data resides at its source. If a query is posed to a peer, on the one hand it evaluates this query over its local schema, and on the other hand it issues subqueries to its neighbours in order to retrieve those answers to the query implied by the mappings. These steps are then repeated by each peer that receives such a subquery. Once the primary peer collected all implied answers, it returns them as results of the query. In such settings where all data resides at the sources, and only queries and their answers are exchanged on demand between the peers, each peer is often modelled as a data integration system itself: A peer may contain local relations where the data is actually stored, while the public peer relations are only virtual. A mapping between the local relations and the peer relation defines which data the peer is willing to share. As depicted in Figure 8, it may not be required that a peer contributes local relations, but it may as well act as mediator only.

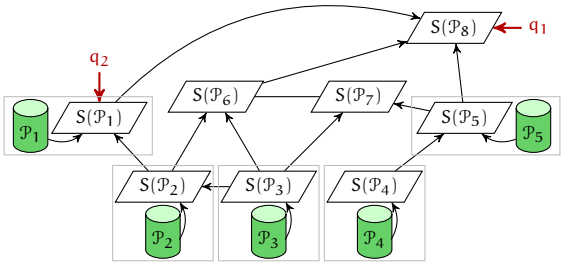

Figure 8: An example structure of a PDMS where peers exchange queries: Some peers contribute data to the system and are modelled as data integration systems themselves, while others act as mediators only.

While the former approach can be seen as a generalisation of data integration, other approaches try to generalise the data exchange setting (e.g. [30], or to some lesser extent [26]). Thereby, just as in data exchange, the mappings are interpreted as constraints that require certain facts to be materialised in order to satisfy the mappings. Since this requires data to be materialised under the peer schemas, in general no distinction between local relations and peer schema is made. Query answering in such settings only requires to evaluate the query locally at the peer to which it is posed, since all data implied by the mappings is already materialised.

The third class of PDMS differs slightly more from the other two classes. Instead of exchanging information about the data actually stored in the sources, only information on the updates performed on the local databases are shared by the peers (e.g. [76], or the ECA rules — see Section 4.2.3). How this is done specifically and the effect of updates on the instance of one peer to the instances of the other peers depends on the different approaches. For example, when using ECA rules, they are used to express the mappings between the peers, and each update triggers under certain conditions immediately a set of clearly defined actions on the instances of other peers. On the other hand in the ORCHESTRA system (for a short summary see Section 4.3.7), schema mappings between the peers are used to transform updates made at one peer to updates over the schemas of the neighbouring peers. Moreover, updates are not published and imported immediately when they occur, but they are stored until the user decides to publish them (or to apply updates published by other peers). So the general idea is that the user works on an isolated database instance and may import changes made by other peers, but there are no notions like satisfying a mapping. Updates made at other peers can be accepted or rejected.

In the remainder of this chapter, first an overview of some of the techniques and formalisation approaches proposed in the literature for modelling and describing PDMSs is given, followed by a short overview of some PDMS prototypes presented in the literature is given (by making no claim to be complete, but trying to capture the variety of proposed systems). More detailed summaries of suggested prototype systems and their properties, as well as classifications and further research challenges can be found in [43] and [7]. Finally in Section 4.4 one PDMS, Piazza, is presented in a little bit more detail.

4.2 FORMALISATION APPROACHES AND TECHNIQUES

In Chapter 3 schema mappings have been introduced as a mapping formalism between database schemas. Therefore these kinds of schema mappings are one possibility to express the dependencies between peers in a PDMS. However, in the literature several additional approaches have been considered. In this section, some of the most important of these formalisation approaches are summarised. This includes techniques for modelling PDMSs as well as different methods to define mappings between peer schemas.

In Section 4.2.4, some of the problems occurring when extending data exchange and data integration to a P2P setting as described at the beginning of this chapter and solutions for these problems are discussed. This is of special interest because the semantics for PDMSs proposed in Section 4.2.4 is the semantics that was adopted in the PDMS implemented as part of this thesis.

4.2.1 *Local Relational Model (LRM)*

The *Local Relational Model (LRM)* [4, 72] was one of the first suggestions for how to formally model a PDMS and to describe its semantics. It has therefore been also used (e.g. in [4]) to describe the idea of PDMSs.

The LRM is a proposal for a data model for PDMSs, and is a generalisation of the model-theoretic semantics proposed in [68]. The idea is to describe the P2P system as a *relational space*, where each peer is modelled as a local relational database, and to express dependencies between these local databases by so called *coordination formulas*. When two or more databases model the same part of the world, this is referred to as *overlapping databases*, and *domain relations* are used to express such overlappings.

More formally, a *local database* is a relational database, identified within a relational space by an index $i \in I$, where I is the the set of indices for a relational space. Because LRM uses a model-based semantics, the schema of database i is defined over some logical language L_i and the set of valid instances db_i for the database i is defined by first-order interpretations of L_i on dom_i (the domain of database i).

Given a set of local databases, the LRM does not define a notion of global (in)consistency, but only for local consistency of single databases i, based on the number of valid instances for i. A database i is complete if $|db_i| = 1$, incomplete if $|db_i| > 1$ and inconsistent if $db_i = \emptyset$.

Because whether two databases overlap or not does not depend on the constants they contain, but only on the concepts represented by those constants, *domain relations* are used to represent overlappings. A domain relation $r_{i,j}$ is any subset of $dom_i \times dom_j$. As domain relations need neither be functions nor symmetric, each overlapping between two databases db_i and db_j is expressed by the two domain relations $r_{i,j}$ and $r_{j,i}$.

A *relational space* is defined as a pair $\langle db, r \rangle$ where db is a set of local databases with indices from $i \in I$, and r is a function assigning to each pair $(i, j) \in I \times I$ a domain relation $r_{i,j}$. (Despite of the definition, it is neither necessary that all pairs of local databases must overlap, nor must there exist a mapping between all pairs of local databases. For such cases $r_{i,j} = r_{j,i} = \emptyset$.)

Coordination formulas used to define semantic dependencies between two local databases are defined as follows:

Definition 4.2 (Coordination Formula [4])**.** The set of *coordination formulas* CF on the family of languages L_i ($i \in I$, L_i is used to express the schema of database i) is defined as
$CF ::= i\colon \phi \mid CF \rightarrow CF \mid CF \wedge CF \mid CF \vee CF \mid \exists i\colon x.CF \mid \forall i\colon x.CF$
(where $i \in I$ and ϕ is a formula of L_i). ⊣

Thereby $i\colon \phi$ means that ϕ is satisfied in database i, the connectives have their usual meaning. The quantifier expressions $\forall i\colon x.CF$ and $\exists i\colon x.CF$ mean that for all elements in dom_i, CF evaluates to true (or that there exists an element in dom_i such that CF evaluates to true resp.). If within the scope of some quantifier the context of the variable is changed by some expression $j : \phi(x)$, then the values of the domain relations $r_{i,j}$ and $r_{j,i}$ are used to map the value of x under dom_i to a value of dom_j.

Moreover, ϕ is considered to be satisfied in database i, if ϕ holds for all instances $d \in db_i$. Based on this notion, satisfiability of the other operators is defined as expected.

Coordination formulas can be either used to express constraints on a relational space (for example to state that a certain piece of information must be present in some database, but without determining in which one), or to define how to derive new information from the data present in certain databases (that is, for defining queries). Queries can either be used to express relationships between databases, or to pose user queries over the schema of one database i.

Definition 4.3 (i-query [4])**.** An i-*query* is a coordination formula of the form $A(\vec{x}) \rightarrow i\colon q(\vec{x})$, where $A(\vec{x})$ is a coordination formula, q is a new n-ary relational symbol and $\vec{x}$ contains n variables. ⊣

Definition 4.4 (Global Answer to an i-query [4])**.** Let $\langle db, r\rangle$ be a relational space. The *global answer of an* i-*query* of the form $A(\vec{x}) \rightarrow i\colon q(\vec{x})$ in $\langle db, r\rangle$ is the set: $\{t \in dom_i^n \mid \langle db, r\rangle \vDash \exists i\colon \vec{x}.(A(\vec{x}) \wedge i\colon \vec{x} = t)\}$ ⊣

More details about the LRM can be found in [4, 72]. While [72] gives a more detailed introduction to the LRM than [4] and contains further examples, the latter describes an architecture for an implementation of the LRM.

4.2.2 *Mapping Tables*

Mapping tables [49, 50, 59, 67] coincide with the domain relations in the LRM and are used to express correspondences of values between different domains. This can be either used to describe which values from different domains refer to the same concepts, or to relate concepts (for example in biology by relating genes to proteins). Thereby such a mapping needs not to express some general relationship, but does only mean that for the two database instances the mapping table is defined on, there exists some correspondence.

In the simplest case, a mapping table is a binary relation where each row defines an association between two values. Note that mapping tables neither need to encode functions nor are they restricted to map one value onto exactly one other value. Although it would be possible to describe the mapping between the domains of two complete databases within a single relation, most of the time one column in a mapping table corresponds to one attribute in a relational schema, and therefore mapping tables provide also some rudimentary relation between the schemas (by relating attributes from one schema to attributes of the other schema). In the following, only such kinds of mapping tables are considered.

The intuition of two possible applications of mapping tables is depicted in Example 4.5:

Example 4.5. By sticking to the general library scenario from the last chapter, assume the following two schemas:
$\mathrm{Lib}_{\mathrm{UB}} = \{\mathrm{Author}_1(\mathrm{FirstName}, \mathrm{LastName}), \mathrm{Bookstatus}_1(\mathrm{Book}, \mathrm{Status})\}$ and $\mathrm{Lib}_{\mathrm{Dep}_1} = \{\mathrm{Author}_2(\mathrm{Name}), \mathrm{Bookstatus}_2(\mathrm{Book}, \mathrm{Status})\}$. Then the following two mapping tables could be used to relate those two schemas:

			$\mathbf{Status}_1$	$\mathbf{Status}_2$
fn	**ln**	**name**	'0'	'unknown'
'THE'	'author'	'author, THE'	'1'	'in archive'
'also'	'author'	'author, also'	'2'	'checked out'

◇

More formally, one row in such a table can be defined as mapping:

Definition 4.6 (Mapping [50])**.** Let U be a set of attributes and let A denote a single attribute. A tuple t over these attributes is a mapping over U, if $t[A]$ is either a value from $\mathrm{dom}(A)$, a variable or of the form $v - S$ (v being a variable and $S \subseteq \mathrm{dom}(A)$) for all $A \in U$. ⊣

Hence in Example 4.5 every row in one of the tables corresponds to one mapping, while all $t[A]$ contain only values from $\mathrm{dom}(A)$.

Variables are used to easily express mappings between ranges of values. For example the identity of the values between two attributes can be expressed by a simple mapping (x_1, x_1), where x_1 is a variable. The expression $v - S$ denotes that v may not be instantiated with a value in S (i.e. this expression can be used to restrict the domain of a variable).

A mapping table consists of several mappings, with the restriction that variables are local with respect to the mapping, that means no two mappings (hence rows) share some variable.

Definition 4.7 (Mapping Table [50])**.** Let X and Y be nonempty disjoint sets of attributes. A *mapping table* m from X to Y is a finite set of mappings over $X \cup Y$ such that each variable appears in at most one mapping. ⊣

Different possibilities for the semantical interpretation of mapping tables exist, depending on whether an open or a closed world semantics is applied. In an open world semantics, every value x for an attribute in X can be mapped to any value for an attribute in Y, independent of whether x is present in the mapping table or not. Under closed world semantics, if x appears in the mapping table, it can only be mapped to those values indicated by the mappings. If x does not appear in the table, then it must not be mapped to any value at all. Since it may be advantageous to apply a different semantics to data present in the mapping table than to data not present, this gives rise to four possible semantics. But only two of them, CO-world (closed-open-world: all values present in the mapping table must only be mapped according to the mappings in the table, values not in the table may be mapped to any value of Y) and CC-world (closed-closed-world: only those mappings specified in the mapping table are allowed) are of practical use.

Having a set of mapping tables, it might be of interest whether they are consistent, or whether they imply further mappings, not explicitly stated. To be able to perform such reasoning over mapping tables, they are regarded as *mapping constraints*. This is done based on the observation that given a tuple t and a mapping table M, the table M restricts the set of tuples onto which t may be mapped. This gives rise to the notion of *mapping constraints*. By combining mapping constraints using $\wedge, \vee$, and $\neg$, one obtains *mapping constraint formulas (MCF)*. With these notions it is possible to define consistency and implication of mapping tables. If ϕ is a mapping constraint formula over a set of attributes U, ϕ is *consistent* if there exists some nonempty instance I for U that satisfies ϕ. Moreover, let $\Sigma \cup \{\phi\}$ be a set of mapping constraint formulas. Then Σ *implies* ϕ, if for every instance I for U, if I satisfies Σ, then I satisfies also ϕ. In [50] it is stated that deciding whether a given MCF is consistent is NP-complete. Since this is equivalent to deciding whether a set of MCFs implies a given MCF, the result holds for the latter problem as well. Nevertheless [50] identifies a restricted case where these problems are efficiently solvable and gives an algorithm for it. In [67] this algorithm is extended to a less restrictive but still tractable case.

Another important issue is how mapping tables can be actually used for query rewriting. This problem is considered for example in [49], where mapping translations for both, sound and complete semantics of the query are considered, and an algorithm for computing such translations is presented. In [59] a different setting also containing mapping tables is considered, and a query rewriting algorithm for this setting is proposed.

4.2.3 *ECA Rules*

Event-Condition-Action (ECA) rules [46, 47, 77] are similar to the concept of SQL triggers. As such, they allow to specify actions to be performed when a certain event (for example the insertion or deletion of a tuple into/from a database) occurs and a set of conditions is satisfied. But unlike SQL triggers, instead of being defined over a single database only, they are intended to alter the data of one peer according to an action occurring in a neighbouring peer. Therefore ECA rules are used as a mechanism for keeping database instances at different peers consistent with each other, that is to coordinate the data exchange between different peers. Obviously for ECA rules it does not matter whether two peers use the same or different schemas. Unlike in "traditional" data exchange, instead of defining dependencies between schemas that restrict valid instances over those schemas, they encode rules determining which actions have to be taken to coordinate the data at different peers.

The generic form of ECA rules is

$$\textbf{when} < event >, \textbf{if} < condition >, \textbf{then} < action > , \text{where}$$

$event$ describes an event like insertion, update or deletion of a tuple that triggers the rules and $condition$ is a Boolean expression encoding the constraints that need to be satisfied for an ECA rule to be active. The **if**-part of the rule may be omitted, which is equivalent to the condition $true$. Once a

rule is triggered by the event specified by *event*, *condition* is checked, and if it is satisfied, *action* is performed. Executing the defined action will result in updating the data of some peer according to the event and condition that triggered the rule.

In [46] an algorithm for deriving and exchanging ECA rules between peers in a setting also including mapping tables is discussed. In [47], besides a detailed description of possible languages for defining *event*, *condition* and *action* (a discussion of them is beyond the scope of this thesis) in an ECA rule, also an algorithm for processing these rules in a distributed[4] setting is proposed: An ECA rule is typically defined at a single peer that is responsible for executing it. But since a rule may include events and conditions of several peers, in a first step the rule is decomposed into subrules that are sent to the corresponding peers. The action defined in these new subrules consists of sending the result of the evaluation of the rule to the peer responsible for executing the original rule. Based on the results this peer receives, it then decides about further actions.

[77] considers the rewriting of ECA rules according to schema mappings. The idea is to simplify the creation of ECA rules by providing sets of rules between standard schemas for certain domains (like for example one global schema for hospitals, one for pharmacies, ...). Between these schemas typical ECA rules are defined and stored in a "library" of such rules. Like in data integration, the concrete schema of a peer is mapped to the appropriate global schema (using exact GAV mappings). When two peers, each having mapped its local schema to some of the global schemas, decide that they want to coordinate their data, they pick an ECA rule of this library. This rule between the two global schemas is then rewritten using the mappings between the peer schemas and the global schemas to a rule directly between the schemas of the two peers.

4.2.4 *A Weaker Semantics for Schema Mappings*

As mentioned at the beginning of this chapter, one way to model PDMSs is to consider them as generalisations of data exchange and data integration settings. This means to apply schema mappings as used in these systems as P2P mappings (i.e. as mappings between two peers) in PDMSs. Such mappings have the advantage to possess a strong and well understood semantics and a high expressive power. Unfortunately, their expressive power is even too high, such that in a general P2P setting several important reasoning tasks (like query answering or data exchange) are undecidable. This can be easily seen for the case when P2P mappings are modelled as TGDs. While in a data exchange setting the set of source-to-target TGDs is assured to be weakly acyclic, this is no longer the case when allowing an arbitrary topology for P2P mappings. Therefore the data exchange problem becomes undecidable in such settings. The same is obviously true when using GLAV mappings. Considering only combinations of GAV and LAV mappings, query answering

4 Although the setting considered in this paper is a multidatabase setting, the authors state the algorithm is intended to be used in a P2P setting.

may also become undecidable for an arbitrary topology of these mappings. The reason for this is that in data exchange and data integration, a first-order logic (FOL) semantics is applied to the mappings, which allows for reasoning over the whole set of mappings.

Therefore three possibilities exist to overcome this problem of undecidability: One can restrict the topology of the P2P mappings, such that reasoning over them remains decidable (for example if the mappings are TGDs, one has to forbid that the P2P mappings of a PDMS form a not weakly acyclic set). Such an approach (for a combination of GAV and LAV mappings) has been chosen for the *Peer Programming Language* used in the Piazza system which is described in more detail in the Section 4.4. Drawback of this approach is that the peers are no longer completely autonomous. Coordination cannot be done only between pairs of peers, since when defining mappings between two peers, global knowledge is needed to avoid a forbidden cycle.

Another possibility is to restrict the kinds of allowed mappings such that no dangerous cycles can be created. For example, if only allowing full TGDs, the important reasoning tasks remain decidable. But this would restrict the expressive power of the queries unnecessarily.

The third approach is to apply a weaker interpretation than FOL to the P2P mappings, such that reasoning remains decidable even in the presence of arbitrary cycles. This is also the approach taken in the theoretical framework described in the next chapter that was implemented as part of this thesis.

Therefore this approach is now described in more detail. In the following, first the intuition of this interpretation is explained, and then four formalisations of this idea are presented. Three of them are given in this section, while the fourth approach is stated in the next chapter when describing the framework for PDMSs proposed in [30]. In the remainder of this section, mappings between peers are assumed to be expressed as TGDs.

The weaker semantics for PDMSs has been first suggested in [14]. Its basic idea is not to use mappings for reasoning over the complete system, but to use them only to fetch data directly from other peers. This difference is best shown by the following example.

Example 4.8. This is a reformulation of a standard example (see e.g. [25]) in terms of the university library scenario used throughout this thesis.

Assume three peers with the corresponding schemas
$\mathrm{Lib}_{\mathrm{UB}} = \{\mathrm{Books}(\mathrm{ISBN}, \mathrm{Title}, \mathrm{Author}, \mathrm{Id}, \mathrm{Status})\}$
$\mathrm{Lib}_{\mathrm{UB}_2} = \{\mathrm{Archive}(\mathrm{Title}, \mathrm{Status}), \mathrm{Elsewhere}(\mathrm{Title}, \mathrm{Status})\}$
$\mathrm{Website} = \{\mathrm{Books}_{ws}(\mathrm{Title}, \mathrm{Status})\}$
and the following mappings:

1. $\mathrm{Books}(\mathrm{ISBN}, \mathrm{Tit}, \mathrm{Aut}, \mathrm{Id}, \mathrm{Stat}) \rightarrow$
   $\mathrm{Archive}(\mathrm{Tit}, \mathrm{Stat}) \vee \mathrm{Elsewhere}(\mathrm{Tit}, \mathrm{Stat})$
2. $\mathrm{Archive}(\mathrm{Tit}, \mathrm{Stat}) \rightarrow \mathrm{Books}_{ws}(\mathrm{Tit}, \mathrm{Stat})$
3. $\mathrm{Elsewhere}(\mathrm{Tit}, \mathrm{Stat}) \rightarrow \mathrm{Books}_{ws}(\mathrm{Tit}, \mathrm{Stat})$

Then, by applying the "standard" FOL interpretation, these mappings imply that every book stored in the Books table also belongs to the Books_{ws} table:

For every book in the Books table it is known (mapping 1) that it is either in the Archive or in the Elsewhere table. In both cases however a corresponding entry in the Books_{ws} table is implied either by mapping 2 or 3.

Under the weaker semantics, entries in the Books table do not imply any entry in the Books_{ws} table. This is because under this semantics, the mappings only propagate data that is known by a peer. But just because some fact (e.g. Books('001', 'THEbook', 'THEauthor', 'b1', '0')) is contained in the Books relation, from the dependency 1 neither Archive('THEauthor', '0') nor Elsewhere('THEauthor', '0') is known by peer Lib_{UB_2} (i.e. neither of those facts is implied by dependency 1). Therefore neither dependency 2 nor 3 does imply some fact Books_{ws}('THEauthor', '0'). Hence the existence of some row in the Books table does not imply any data in the Books_{ws} table. ◇

Hence, under this semantics only facts are exchanged that are known by a peer (and in the above example, neither Archive('THEauthor', '0') nor Elsewhere('THEauthor', '0') is known by Lib_{UB_2}). Thereby a fact is considered to be known if it is part of every interpretation of the peer, that is if it holds in every possible world. Therefore, the semantics can also be described as peers only exchanging certain answers. The following example shows another difference in the behaviour of the FOL and the weaker semantics.

Example 4.9. Consider a similar setting as in Example 4.8:
$\text{Lib}_{Dep} = \{\text{Books}_{Dep}(\text{ISBN}, \text{Title}, \text{Author}, \text{Id})\}$
$\text{Lib}_{\text{UB}} = \{\text{Books}(\text{ISBN}, \text{Title}, \text{Author}, \text{Id}, \text{Status})\}$
$\text{Website} = \{\text{Books}_{ws}(\text{Title}, \text{Status})\}$
and the two mappings

1. $\text{Books}_{Dep}(\text{ISBN}, \text{Tit}, \text{Aut}, \text{Id}) \rightarrow$
   $\text{Books}(\text{ISBN}, \text{Tit}, \text{Aut}, \text{Id}, \text{Stat})$
2. $\text{Books}(\text{ISBN}, \text{Tit}, \text{Aut}, \text{Id}, \text{Stat}) \rightarrow$
   $\text{Books}_{ws}(\text{Tit}, \text{Stat})$

(Note that dependency 1 contains an existentially quantified variable on its right hand side.)

Moreover consider a CQ $q = \{(\text{Tit}) \mid \text{Books}_{ws}(\text{Tit}, \text{Stat})\}$ over the peer Website and an instance D for Books_{Dep} containing a single fact Books_{Dep}('001', 'THEbook', 'THEauthor', 'b1').

Then, under the FOL interpretation of the mappings, q returns the answer {('THEbook')}. Under the weaker semantics however, the answer to q is $\emptyset$, although the Title attribute is never involved in any uncertainty. But because of the existentially quantified variable in dependency 1, no tuple (Title, Status) is implied by the mappings, and hence the Books_{ws} relation contains no data and the query returns the empty set as answer. ◇

Hence, mappings interpreted under this semantics only exchange tuples that contain no existentially quantified variables (i.e. labelled nulls), and a mapping is satisfied by an instance under this semantics if all tuples without labelled nulls are exchanged. This reflects exactly the intuition of this semantics for the same reason why certain answers cannot contain labelled nulls. Since they

may be interpreted as different values in different possible worlds, the value of positions containing a labelled null are not known (and as mentioned above, one intuition of this semantics is that peers only exchange certain answers).

Several suggestions how to formalise this intuition of exchanging only "known" facts have been made. They are summarised below together with some proposed extensions to handle inconsistency. Recall that a GLAV mapping consists of assertions $q_S \rightsquigarrow q_G$, where q_S and q_G are conjunctive queries of the same arity.

The suggestions presented below deal with yet another drawback of the semantical description of a PDMS using classical FOL. As it will be shown some paragraphs later, when using classical FOL the whole PDMS is modelled as one flat FOL theory. This one theory describing the whole PDMS does not express the structure and modularity of the modelled system. This means that the single peers and the structure of the network are lost in such a theory. It is therefore also a goal of these alternative suggestions to provide a formalisation that models both the peers and their connections nicely (i.e. that reflects their structure).

Epistemic Semantics

The notion of a weaker semantics for P2P mappings has been first introduced in [14]. In this paper, the semantics of PDMSs that can be seen as an extension of data integration systems is considered: In this setting, every peer (P_i) consists of a local schema ($\mathcal{S}_{P_i}$) and a peer schema ($\mathcal{G}_{P_i}$)[5]. While the local schema describes the data stored at a peer, the peer schema describes the data a peer is willing to share. The peer schema of each peer is related to its local schema by sound GLAV mappings, called *local mappings*. Moreover, dependencies between peer schemas of different peers can be expressed by GLAV mappings as well.

The semantics of each peer is described by an own FOL theory T_{P_i}, that contains all FOL formulas needed to express $\mathcal{G}_{P_i}$ (i.e. formulas to describe the relations and, if defined, constraints over $\mathcal{G}_{P_i}$) and one formula $\forall \vec{x} \, (\exists \vec{y} \, (q_S(\vec{x}, \vec{y})) \rightarrow \exists \vec{z} \, (q_G(\vec{x}, \vec{z})))$ for each local mapping $q_S \rightsquigarrow q_G$. Then, assuming a finite instance D of $\mathcal{G}_{P_i}$, an instance $\langle D, B \rangle$ for $\langle \mathcal{S}_{P_i}, \mathcal{G}_{P_i} \rangle$ (i.e. an interpretation of T_{P_i}) is called a *model of* P_i *based on* D if it is a model of T_{P_i}.

Using the classical FOL semantics, the semantics of a complete PDMS would be defined as follows: Denote T_P as the union of all peer theories T_{P_i}. An interpretation $\mathcal{I}$ of T_P is a FOL model of the PDMS with respect to a set of instances $\{D_i\}$ for the local schemas $\mathcal{S}_{P_i}$ if it is a model of P_i based on D_i for each peer P_i and it is further a model of a set of formulas $\forall \vec{x} \, (\exists \vec{y} \, (q_1(\vec{x}, \vec{y})) \rightarrow \exists \vec{z} \, (q_2(\vec{x}, \vec{z})))$, one for each P2P mapping $q_1 \rightsquigarrow q_2$.

The language of epistemic logic extends the language of FOL by allowing for one additional form of atoms, namely $\mathbf{K}\phi$, where ϕ is an epistemic formula. Intuitively, $\mathbf{K}\phi$ describes those objects that are known to satisfy ϕ. Because of space restrictions it is not possible to give an introduction into epistemic logic.

5 For simplicity, it is assumed that all local and peer schemas are pairwise disjoint.

Short introductions and references to more detailed descriptions can be found in [14, 15].

Using epistemic logic, the semantics of a PDMS $\mathcal{P}$ is defined in terms of an epistemic theory for the PDMS. This theory T_e consists of T_P (as defined above) and one axiom $\forall\vec{x}\,(\mathbf{K}(\exists\vec{y}\,(q_1(\vec{x},\vec{y})) \rightarrow \exists\vec{z}\,(q_2(\vec{x},\vec{z}))))$ for each P2P mapping $q_1 \rightsquigarrow q_2$ (denote this set of axioms with M_P). Note that just by the intuition of the meaning of an atom $\mathbf{K}\phi$ described above, this exactly expresses the idea that peers only exchange data that is known by them. Just as in the FO-case, assume a set of instances $\{D_i\}$ for the local schemas $\mathcal{S}_{P_i}$. Then an epistemic interpretation $(\mathcal{I},\mathcal{W})$ is an *epistemic model of* $\mathcal{P}$ *based on* $\{D_i\}$, if each $W \in \mathcal{W}$ is a model of P_i based on D_i for each peer P_i, and $(\mathcal{I},\mathcal{W})$ is an epistemic model of M_P.

This definition guarantees that, given an epistemic model $(\mathcal{I},\mathcal{W})$ of $\mathcal{P}$ based on $\{D_i\}$, for every P2P mapping $q_1 \rightsquigarrow q_2$, whenever a tuple $t \in \bigcap_{W\in\mathcal{W}} q_1^W$, then also $t \in \bigcap_{W\in\mathcal{W}} q_2^W$, which covers exactly the intuition mentioned above that peers exchange certain answers. (Note that unlike in the FOL semantics, the semantics of P2P mappings is not defined for a single FOL model, but for a set of FOL models, namely $\mathcal{W}$.)

In [15], this semantics is compared to the FOL semantics with respect to properties like modularity, generality and decidability, which are argued to be desirable properties for a PDMS. It is then shown that with respect to these properties, the FOL based semantics shows some undesirable behaviour, and it is argued that the semantics based on epistemic logic shows a more preferable one.

Moreover an algorithm for query answering in a PDMS based on this semantics is given, which runs in polynomial time (data complexity).

Local Semantics

In [25], three different (but equivalent) formalisations are presented that are all equivalent to the semantics based on epistemic logic. Here, only one of them will be mentioned, namely the *Local Semantics*. Unlike in the setting described above, in [25] every peer consists of only a single schema $\mathcal{P}_i$ that is used for both, describing the data locally stored at the peer and defining mappings between peers. Each peer is identified with its schema $\mathcal{P}_i$. A PDMS is therefore modelled as a sequence of peers $\langle\mathcal{P}_1,\ldots,\mathcal{P}_n\rangle$, and a set of P2P mappings. Mappings are expressed in [25] as GAV mappings $q_{\mathcal{P}} \rightsquigarrow q_{\mathcal{P}_i}$, where $q_{\mathcal{P}}$ is a conjunctive query over all peer schemas $\mathcal{P}_i$ and $q_{\mathcal{P}_i}$ is a conjunctive query over the peer schema of a single peer $\mathcal{P}_i$ containing only a single atom and no variable not occurring in $q_{\mathcal{P}}$. Note however, that the definition of a local model given below can be immediately extended to arbitrary TGDs.

A *local model* M of a PDMS is a sequence $\langle M_1,\ldots,M_n\rangle$ such that each M_i is a non empty set of instances m_i for $\mathcal{P}_i$ (i.e. $\forall m_i \in M_i : m_i \vDash \mathcal{P}_i$). Moreover, for every mapping $b_1(\vec{x}_1,\vec{y}_1)\wedge\cdots\wedge b_k(\vec{x}_k,\vec{y}_k) \rightarrow h(\vec{x})$ (where b_i

is a relational symbol from $\mathcal{P}_{j_i}$, and h from some peer schema P_h) and every assignment of $\vec{x}$ it holds that

$$\begin{aligned}(\forall m_{i_1} \in M_{i_1} : (m_{i_1} \vDash \exists \vec{y} : b_1(\vec{x}_1, \vec{y}))) \wedge \\ \cdots \wedge \\ (\forall m_{i_k} \in M_{i_k} : (m_{i_k} \vDash \exists \vec{y} : b_k(\vec{x}_k, \vec{y}))) \rightarrow \\ (\forall m_h \in M_h : (m_h \vDash h(\vec{x})))\end{aligned}$$

Again, this definition captures the intuition that only facts that are known by the peers are exchanged. Just as in the epistemic case, the semantics is defined using a set of models instead of a single model, and only facts that hold in each of these models are exchanged.

In [25], also the problem of *local inconsistency* is investigated, and the local semantics is extended to be robust with respect to local inconsistency. A database of a peer $\mathcal{P}_i$ is considered to be locally inconsistent if the corresponding set M_i contains no instance in any local model of the PDMS (i.e. if $M_i = \emptyset$ in any local model). Under the local semantics, this would immediately imply that no model of the overall system exists. Therefore, by applying the fact that from an inconsistent source everything can be derived it is defined that every relation $\mathbf{R}$ (of arity n) that depends on an inconsistent peer $\mathcal{P}_i$ (i.e. there exists a mapping that contains $\mathbf{R}$ on its right hand side and a relational symbol of $\mathcal{P}_i$ on its left hand side), contains every tuple from Δ^n. Thereby Δ denotes the domain that is used for the interpretations of the PDMS, so depending on the concrete setting most probably the active domain.

Moreover, the paper presents two algorithms for query answering, a centralised one that requires global knowledge about all mappings in the system, and a decentralised, distributed algorithm that does not have this shortcoming. Both algorithms run in polynomial time with respect to data complexity.

Multimodal Epistemic Semantics

In [16], yet another formalisation of the general idea of the weaker semantics is presented. It is not completely equivalent with the other approaches, because the aim of the proposed semantics is to handle inconsistency. Therefore, two different types of inconsistencies are considered: *local inconsistency* and *P2P inconsistency*. Local inconsistency describes the fact that the content of a local peer database is inconsistent. This means that local inconsistency is a problem of single peers. P2P inconsistency on the other hand occurs if, for a peer whose local data is consistent, contradicting data is implied by some P2P mapping (either because information implied by two different mappings contradict or because the implied data is inconsistent with respect to the data stored locally).

The idea suggested in [16] is on the one hand to isolate locally inconsistent peers (i.e. to "disable" all mappings originating in such a peer), such that the inconsistency of this peer is not propagated to the whole system. On the other hand, peers reject data derivable by P2P mappings that would lead to inconsistency at this peer.

The considered setting is the same as the one in [14], described above when introducing the formalisation using epistemic semantics. To better reflect the structure of a PDMS, i.e. that it consists of a set of autonomous peers, the authors choose a multimodal epistemic logic to model such a system. More precisely, the logic $K45_n$ is used to describe the semantics of a PDMS. Because of space restriction, no introduction to this logic is given here. A nice introduction and further references can be found in [16]. Thereby $K45_n$ is similar to the epistemic logic described above. But instead of a single modal operator $\mathbf{K}$, it contains a set of modal operators $\mathbf{K_1} \dots \mathbf{K_n}$ (intuitively, one for each peer). These modal operators are used in the same way as the $\mathbf{K}$ operator may be used, that is whenever ϕ is a formula, $\mathbf{K_i}\phi$ is a valid formula. But the modal operators have a different semantics. While the intended meaning of an atom $\mathbf{K}\phi$ above was that ϕ is known to be true, $\mathbf{K_i}\phi$ expresses that peer $\mathcal{P}_i$ believes that ϕ is true. (The change from knowledge to beliefs is made in order to be able to handle inconsistencies.) More formally, while $\mathbf{K}\phi \rightarrow \phi$ always holds in the above semantics, $\mathbf{K_i}\phi \rightarrow \phi$ need not to be true.

A PDMS $\mathcal{P}$ is then described by a $K45_n$ theory T. For each peer $\mathcal{P}_i$, T contains for each FOL formula ϕ needed to describe $\mathcal{G}_{P_i}$ (i.e. the formulas describing the schema and the constraints over the schema) a formula $\mathbf{K_i}\phi$, and for each local mapping $q_S \rightsquigarrow q_G$ a formula $\mathbf{K_i}(\forall \vec{x}\ \exists \vec{y}\ q_S(\vec{x}, \vec{y}) \rightarrow \exists \vec{z}\ q_G(\vec{x}, \vec{z}))$. Intuitively, these formulas state that every peer actually believes the content of its local database. Moreover, for every P2P mapping $q_i \rightarrow q_j$, a formula $\forall \vec{x}\ (\mathbf{K_i}(\exists \vec{y}\ q_i(\vec{x}, \vec{y}) \rightarrow \mathbf{K_j}(\exists \vec{z}\ q_j(\vec{x}, \vec{z}))$ is added to T, with the intuitive meaning that if some tuple t is believed by peer $\mathcal{P}_i$ to belong to the answer to q_i, then $\mathcal{P}_j$ must believe that t belongs to the answer of q_j (which again resembles the intuition of the weaker semantics for P2P settings).

Models of such theories are then defined analogous as for the epistemic semantics from above with respect to a set of given instances for the local schemas.

Note that this formalisation is not yet capable of handling inconsistencies. To manage this, the logic $K45_n$ is extended to a nonmonotonic logic $K45_n^A$. Syntactically, $K45_n^A$ differs from $K45_n$ by a second set of modal operators $\mathbf{A_1} \dots \mathbf{A_n}$. These operators correspond to the complement of negation as failure (known as *justified assumption*). As this operator will only be used negated, this expresses negation as failure.

With these operators, to handle local inconsistency in the way described above, every P2P mapping is described by a formula $\forall \vec{x}(\neg \mathbf{A_i} \perp_i \wedge \mathbf{K_i}(\exists \vec{y} q_i(\vec{x}, \vec{y}) \rightarrow \mathbf{K_j}(\exists \vec{z} q_j(\vec{x}, \vec{z}))))$, instead of the formulas for P2P mappings under $K45_n$. Intuitively, $\neg \mathbf{A_i} \perp_i$ is true iff peer $\mathcal{P}_i$ is not locally inconsistent. Therefore, as soon as $\mathcal{P}_i$ becomes locally inconsistent, this mapping becomes inactive. If $\mathcal{P}_i$ is not locally inconsistent, then the mapping has the same semantics as under the logic $K45_n$.

To handle also P2P inconsistency, the formulas describing P2P mappings are extended by another condition, such that P2P mappings are described by formulas of the form $\forall \vec{x}(\neg \mathbf{A_i} \perp_i \wedge \mathbf{K_i}(\exists \vec{y} q_i(\vec{x}, \vec{y}) \wedge \neg \mathbf{A_j}(\neg \exists \vec{z}\ q_j(\vec{x}, \vec{z})) \rightarrow \mathbf{K_j}(\exists \vec{z} q_j(\vec{x}, \vec{z}))))$. Intuitively, the new condition $\neg \mathbf{A_j}(\neg \exists \vec{z}\ q_j(\vec{x}, \vec{z}))$ states that the values implied by this mapping are consistent with the current knowledge

of $\mathcal{P}_j$, expressed by stating that the negation of this data cannot be derived. This ensures that $\mathcal{P}_j$ remains consistent. If neither $\mathcal{P}_i$ is locally inconsistent nor does the implied data contradict the data in $\mathcal{P}_j$, then the semantics of the mapping is the same as under the logic $\mathsf{K45}_n$. Otherwise this mapping is inactive. Moreover, the semantics of $\mathsf{K45}_n^A$ is defined in such a way, that the amount of data rejected by a peer is minimised.

The formal description of the semantics of these theories is omitted here due to space restrictions.

For a setting where the P2P mappings are only GAV mappings and only key constraints are allowed over the peer schemas, the authors show that query answering under the semantics based on the $\mathsf{K45}_n^A$ logic is decidable. Moreover, they show that query answering in such a setting is coNP-complete, and give a coNP-algorithm for this problem.

4.3 PDMS PROTOTYPES

In this section some prototype PDMSs presented in the literature are introduced. This list of prototype systems does not aspire to be complete, but the selection ought to give an overview of the variety of approaches considered for PDMSs. PDMSs have been proposed based on different data models (e.g. the relational model, XML or RDF) and use different kinds of mappings. Based on the way how relationships are defined between peers, peers are coupled tighter in some systems than in others (for example in PeerDB coupling is loose while it is stronger in e.g. coDB), and also systems have different strengths of semantics.

4.3.1 *Piazza*

The Piazza system [35, 37]) is an early and one of the most cited PDMSs. It is a generalisation of traditional data integration settings and defines dependencies between peers using schema mappings. The semantics of these mappings is defined by classical FOL interpretations.

Because of its importance and as an example for a PDMS applying classical FOL semantics to schema mappings, a more detailed description of the Piazza system and the formal methods used for describing its semantics is given in Section 4.4. (A detailed description of a PDMS applying the weaker semantics to schema mappings — the approach of De Giacomo et al. — is given in Chapter 5.)

4.3.2 *PeerDB*

PeerDB [62, 65] uses an approach for expressing and deriving mappings between schemas completely different from the approaches described above and of those used by the other systems presented in this section. While in the other systems the neighbours of a peer and the dependencies between peer schemas are (at least to some extent) predefined “by hand” in a configuration

step by a user, PeerDB tries to establish these mappings at runtime: In PeerDB, every node of the P2P network shares data stored under a relational schema. However, the set of neighbours is not fixed, and no explicit mappings exist between the schemas of different peers. Each relational symbol and each attribute of a peer schema can have attached keywords. These keywords should describe the relation or attribute, and are used to derive mappings between the relations of different peers: If a user poses a query over its local schema (every user can only query its local database), software agents are sent to a dynamic set of neighbours of the peer that try to match the tables and attributes appearing in the query with the relations and attributes in the schemas of other the peers. Thereby a possible match is assumed for example if the description of two attributes share a common keyword. When a software agent has checked a certain peer for all possible mappings, it is sent further to the neighbours of this peer. This propagation stops when the time to life of the agent has expired.

All possible mappings found by the agents are returned to the user for selecting the correct ones or those the user is interested in. Then the original query is rewritten according to the selected mappings, and sent to the corresponding peers, where it is executed. Finally, the answers to the queries are returned to the initiating peer and are then presented to the user.

PeerDB further provides access control and caching mechanisms, and tries to update the topology of the P2P network according to the derived schema mappings (for example, if for most of the queries there are some connections between the schemas of two peers that are no direct neighbours, those peers are connected as neighbours).

4.3.3 *Edutella*

Edutella [56, 61, 64] was developed to share (educational) resources that are described by RDF metadata. Therefore in Edutella, an instance contains several RDF statements, that are queried by a user to retrieve those resources that meet the requirements. Because of the big number of existing RDF query languages, Edutella provides an abstraction layer for the query languages. Before a query (posed in some language over a peer) is relayed to other peers, it is first translated into Edutellas own exchange query language. There exist different levels of this exchange query language, each having a different expressive power, and each peer can define up to which level it is able to accept queries. Internally, the peers use a Datalog based model to represent those queries.

To avoid flooding the whole P2P network with a query (i.e. to simply forward a query to all neighbours which repeat this step until it is guaranteed that the query has reached all peers, whereby peers that can contribute to the result of query return their contribution to the initiating peer), Edutella implements a super-peer based approach. Thereby each peer is connected to a certain super-peer, which allows to implement more intelligent methods for query routing between the super-peers than just flooding. Moreover super peers maintain indices about the data stored in the peers connected to them.

Edutella also contains an own "mapping-service" that allows the translation of queries between different schemas. In fact, several techniques have been suggested to deal with heterogeneous schemas. One idea presented in [56] is that of *local correspondences*: Every super-peer defines a global schema and a set of constraints. A peer that wants to connect to a super-peer has to satisfy these constraints. The schemas of those peers connected to a certain super-peer are mapped to this super-peer schema by first defining views over the peer schema, and then defining correspondences between these views and the elements of the super-peer schema.

In [64], also mappings between the domains of different peers, similar to attribute correspondences and mapping tables are described.

4.3.4 *Hyperion*

The Hyperion project [2, 48, 51, 63, 69] investigated several aspects of PDMSs, for example the work on mapping tables and ECA rules introduced above has been done within this project. Within this project, also the Hyperion PDMS was developed. The Hyperion PDMS assumes relational databases at each peer and allows for both, exchanging data between peers by altering the instance of one peer according to the database instance of another peer, as well as defining only virtual mappings. Those virtual mappings do not define any constraints on what data must be present in a valid database instances, but are only used to allow the posing of global queries. These queries are not only evaluated over the schema of a single peer but are also forwarded to the other peers according to the virtual peer mappings.

Thereby different mechanisms for defining the mappings for these two tasks are used: For defining relationships between data that are only considered for query processing, *mapping tables* and *mapping expressions* are used. Mapping expressions are a generalised form of GLAV expressions that express relationships at the schema level, while mapping tables mainly define correspondences on the data level (although they also provide very little information about correspondences at the schema level). In the Hyperion PDMS, the main focus lies on the mapping tables. Query processing is done by rewriting the query according to the mapping tables and mapping expressions and by forwarding the query along such mappings.

Coordination rules are used to express which data has to be materialised in a peer database according to the content of a neighbouring peer instance. These rules are ECA rules, hence instead of only defining relationships between instances that have to be satisfied, coordination rules define actions that have to be performed under certain conditions. Of course, mapping tables are combined with coordination rules to translate the data exchanged between two domains from one domain into the other.

One idea considered in Hyperion are *interest groups*: An interest group provides a common schema for the field of interest it covers. Between different interest groups there exist predefined mappings and coordination rules. When a peer connects a certain interest group, its schema is mapped to the global schema of the group. Based on this information, between selected peers,

the tables of one peer are tried to be expressed as views over the tables of other peers. From these views, mapping tables are inferred. Finally mapping expressions are created and the mapping tables are populated. For peers from other interest groups, the predefined mappings between the global schemas of the groups are used to infer the relationships.

According to the authors, the main focus of the Hyperion PDMS lies on the mappings on the data level, not on mappings on the schema level. Therefore mapping expressions play only a minor role in their considerations.

4.3.5 *HepToX*

HepToX [5, 6] is a PDMS working on XML-structured data. The relationship of the data stored in two peers is expressed by *mapping expressions*, datalog like rules between the DTDs describing the XML data of each peer. Although these mappings intuitively describe how to transform instances between two peers, no data exchange is performed in HepToX, but the mappings are only used for query answering. Therefore HepToX could also be described as a P2P Data Integration System. To create the mapping expressions, the user is not required to really encode them as rules. A graph representation of the DTDs of the two peers is shown, and tags representing the same concepts in both DTDs can be related by connecting them with an arrow. Moreover several tags can be grouped if the same concept is represented by more than one tag in one of the schemas. Thereby it is not necessary that all tags in the DTDs are matched, since the mappings need not define how to transform an instance from one schema into the other schema. From these arrows and boxes drawn by the user, the mapping expressions are automatically generated. Within the three main steps of this algorithm, first the groups of nodes are identified for which a mapping expression needs to be defined. Then the subgraph induced by the nodes of a group in the DTD is transformed into a tree and an according tree expression, and finally mappings are defined between these rules.

With these mapping expressions at hand, a query posed over one peer database can be translated according to the mappings to queries over the databases of peer neighbours. Thereby, given a mapping from peer P_1 to P_2, the mapping can be used for both, translating a query over P_2 to a query over P_1 and vice versa. For both cases the semantics is defined as well as an algorithm stated that performs the transformation. This algorithm again consists of three steps: first the query is expanded such that it matches a mapping expression, then the expanded query is translated according to the mapping expression, and in the final step redundant parts of the translated query are removed.

As query language a fragment of XQuery is considered. More details about these two algorithms, as well as about the mapping expressions and an evaluation of a prototype implementation are given in [5].

4.3.6 *coDB*

The coDB system [27] is based on the "local semantics" presented in [25] (see Section 4.2.4, roughly spoken this means that peers do only exchange certain answers, which allows for cycles in the P2P mappings between the peers) and uses the distributed update algorithm presented in [26]. Dependencies between peers are expressed as GLAV-mappings between the peer schemas.

Query answering is based on the idea to use the mappings between the peers (called *coordination rules* and relating queries over schemas of different peers) to fetch data from the neighbours, to store this data locally and to answer queries posed to a peer also locally according to the data that has been fetched before. Thereby data can be either fetched "on the fly" if a query is posed to a peer, or a global update can be performed. In the first case, a peer tries only to fetch that data it requires for answering a given query, in the latter case all peers exchange all data as defined by the coordination rules. In both cases the same update algorithm presented in [26] is used. Thereby one peer initiates an update by sending the queries from the left hand sides of its coordination rules to the corresponding neighbours. There these queries are evaluated over the local data and the result is immediately returned to the initiator. Moreover, the peers forward the update request by asking their neighbours to send the data according to their coordination rules (either for only those tables involved in queries received from the initiator of the request or simply all data). Hence they act as initiators of their own. Whenever some peer retrieves new data as result to an request issued, it immediately checks whether all the required information implied by this data has already been sent to the peer that issued the request. If not, this data is immediately propagated.

This update algorithm does not require any global knowledge or control over the system, but runs locally at each peer. Moreover it is capable of handling changes in the network (i.e. adding or removing of coordination rules) while the algorithm is running. Under the assumption that there are only finitely many changes between those nodes of the network that are involved in the current update, the algorithm is guaranteed to terminate and to return the correct results.

Evaluation results of the coDB system are presented in [27]. There also a more detailed description of the system architecture is given. As stated, detailed information about the semantics implemented in coDB can be found in [25], and the update algorithm is described in detail in [26].

4.3.7 *ORCHESTRA*

ORCHESTRA [33, 34, 44, 76] is a prototype implementation of what the authors call *collaborative data sharing system (CDSS)*. The idea of CDSS differs slightly from the idea of (peer) data exchange and data integration systems. Just as in these settings, a CDSS consists of several peers, each having a local database, and of pairwise mappings between the schemas. But unlike those settings, the goal is not to exchange data until some globally consistent state

is reached or to use these mappings for query answering. The mappings are only used to exchange updates made at one peer with the other members of the network: The idea is that every user only works on its local database instance. Queries to the local database are only processed locally, and updates also have an effect to the local database only. Instead, a local log of all the updates is maintained. From time to time, whenever the user decides to share its data with the other members of the network, this update log is published. Again, this has no direct effect to the local databases of the other peers. Only if a member decides to import all the data published since he/she has imported data for the last time, those updates are applied to its local database.

The idea of ORCHESTRA is to implement a system that supports such a setting. In ORCHESTRA, the mappings between the schemas of the peers are expressed as GLAV-mappings, that is by TGDs. Valid mappings are restricted to sets of weakly acyclic TGDs, hence arbitrary cycles in the mappings are forbidden. As stated above, unlike in data exchange or data integration, the mappings are not used to directly exchange data, but to translate updates (like insertion, deletion or the changing of a certain tuple) performed on an instance of one schema to an update expression over another schema. A detailed description of how ORCHESTRA performs this update translation is given in [34].

Moreover ORCHESTRA allows the definition of so called *trust policies*. The idea of trust policies is to enable users to express whether they trust some sources more than others, since it is often the case that users regard certain sources more reliable (w.r.t. the correctness of their data) than others. These trust policies are defined by assigning trust levels to the different sources for which mapping to the local schema exist. A peer may also completely distrust some members of the network. (Note that just defining no mapping from such a member to the local database is not the same, since data from such a source may be retrieved via some intermediate peers. Therefore allowing a peer to define that data originating at a certain peer is not trusted gives additional expressive power.)

When a peer decides to import (the authors call this step *reconciliation*) all updates published by the other peers since its last reconciliation, first the update sequence is flattened. This means that for sequences of depending updates by the same peer, the intermediate steps are omitted and only the final result is visible. For example, if some peer changed the value of a certain tuple several times, and each time published these updates, then all these changes are replaced by a single update that leads to the final result. Then ORCHESTRA has to decide which of these updates shall be applied to the local instance and which updates have to be rejected. Once an updingate has been rejected, this update and all updates that depend on it will be rejected in future reconciliation steps too. There are several reasons why updates may be rejected: Because the update is inconsistent, because the peer distrusts the source of this update, or because it depends on an already rejected update.

An update can be inconsistent for two reasons: The result of the update could conflict with constraints on the local schema, or two updates contradict each other (like two updates that change the same attribute in a tuple to two

different values). With respect to the first case, note that the instances at the different peers need to be only locally consistent, but not with respect to the instances at the other peers. Updates that are inconsistent with the local instance are rejected. If some updates are conflicting, ORCHESTRA tries to decide which update to accept and which to reject using the trust policies, by accepting the update from the source that is more trusted. If this is not possible, ORCHESTRA marks the updates, and all dependent updates as *deferred*, and the user has to decide what shall be done with the updates.

On the other hand, to decide whether an update is trusted or not, not the peer from which the update has been received is taken into account, but the peer where the corresponding data originated. This is necessary since it may happen that some peer P_i trusts P_j but distrusts P_k, while P_j trusts P_k. If P_j accepts an update of P_k, and publishes the application of this update to its local database, the update would arrive at P_i. In such a situation, there are two possibilities: If there are updates originating from a trusted source that are dependent on the untrusted update, it has to be accepted (since otherwise the trusted updates would need to be rejected too), except it has already been rejected in some earlier reconciliation step. If there are no dependent trusted updates, the untrusted update is rejected.

For both cases, it is not only necessary to determine where the data originated from, but also how (by which mappings) it arrived at the peer. Moreover, this is an important information for translating updates according to schema mappings. To be able to determine this information, ORCHESTRA stores for each tuple provenance information, that does not only refer to where the tuple comes from, but also how it has been derived.

The reconciliation algorithm is described in detail [76], together with different possibilities for its implementation (discussing a centralised and a distributed algorithm, each combined with either a centralised or distributed global update store) and an evaluation of a prototype implementation.

[44] gives a general overview over the ORCHESTRA system, while [34] describes it in detail and presents an evaluation of a prototype implementation of the system.

4.3.8 *Discussion*

At the end of this section, we shortly summarise some of the properties of the presented PDMS prototypes.

Systems have been proposed for several different data models: For relational databases (like e.g. coDB, PeerDB, Hyperion and ORCHESTRA), for XML based data (e.g. HepToX) or RDF (like Edutella).

Based on the different techniques and formalisations used to express relationships between schemas (e.g. schema mappings, mapping tables, ECA rules, update translations), sometimes all peers in the system are assumed to share the same domain, while sometimes also different domains at the peers are considered.

According to the different semantics applied to schema mappings (if schema mappings are used at all), some systems require restrictions on the topology

of the P2P network to obtain decidability of important reasoning tasks (e.g. Piazza and ORCHESTRA), while other systems apply a weaker semantics, therefore allowing for arbitrary topologies (e.g. coDB).

Independent of whether global information is needed to satisfy restrictions on the topology of the P2P network, reasoning (e.g. query answering) over some systems require global information (like in the Piazza system). Other systems use super-peers to structure the P2P network (like Edutella), and several systems are completely decentralised.

In most of the systems, the neighbourhood of a peer and the relationship to its neighbours are predefined by the user, offering a well defined semantics of the resulting PDMS. Some systems (like e.g. HepToX) support the user in setting up the mappings between schemas by creating them semi-automatically according to some easy to do user input. In contrast, PeerDB does not require to define the neighbourhood and the mappings beforehand, but tries to determine them at query time. As a result user feedback is needed during query processing to decide which mappings found are correct. Moreover the system does not offer a strict semantics.

Also the tasks supported by the systems differ. While most of them focus on query answering (e.g. coDB, PeerDB, Piazza, HepToX, Hyperion), the goal of Edutella is to help locating resources in the network, and systems like ORCHESTRA try to support the synchronisation of independent, hence possibly inconsistent database instances.

4.4 PEER-PROGRAMMING LANGUAGE (PPL)/PIAZZA

The Piazza peer data management system [35, 37, 38, 39, 40, 75] is one of the most cited PDMS prototypes. Although it is intended to work on XML data, and the prototype implementation actually uses XML, its basic semantics has been described and introduced using the relational model. Within this summary of the basic ideas of Piazza, only relational aspects are considered, and not their extensions to data structured using XML.

As common in PDMSs, Piazza assumes data to be stored at peers which offer data through a schema, and define dependencies between schemas of other peers and their own schema. Thereby, the Piazza system implements a strict generalisation of data integration systems: Like in data integration systems, data is assumed to reside in certain source relations (and to remain there) that are related by mappings to virtual schemas. Queries are posed over these virtual schemas and are then rewritten to queries over the source relations. But unlike the traditional data integration systems described in Section 3.2, there exists not a single virtual global schema, but every peer defines its own schema.

The Piazza system assumes a unique domain shared by all peers, hence no mapping of constants is performed between peers (like by using mapping tables). As query language, conjunctive queries are considered.

4.4.1 *System Definition*

A peer consists of its *peer schema*, the corresponding *peer relations* and a set of *stored relations*. The peer schemas are used to access the system. That is all queries posed to the system are formulated over the peer schema of a single peer. Using the mappings defined within the system, the query is then reformulated such that it refers only to stored relations, where the actual data resides. Hence, each peer may contribute data to the system via the content of its stored relations. Thereby it is not required that the set of stored relations of each peer is nonempty, i.e. a peer is not required to contribute data, but may consist of a peer schema only too.

Figure 8 at the beginning of this chapter (see page 41) sketches this structure.

Syntax of $\mathcal{PPL}$

For defining mappings between those schemas, the authors introduce the *Peer-Programming Language* ($\mathcal{PPL}$, that shall be pronounced as "people", being the formalism used to define the semantics of Piazza), that offers two kinds of mappings: *storage descriptions* and *peer mappings*.

Storage descriptions are used to describe the content of the stored relations of a peer with respect to its peer relations. More precisely, the content of the stored relations is defined using the answer of a query q over the peer schema. There exist two kind of storage descriptions to either express that a stored relations contains exactly the result of q (*equality description*, corresponding to the CWA) or only a subset of it (*inclusion description*, corresponding to the OWA).

Definition 4.10 (Storage Description [38])**.**

- An *equality description* is a storage description of the form $A : R = q$,
- an *inclusion description* is a storage description of the form $A : R \subseteq q$,

where A is peer, R is a stored relation of A and q is a conjunctive query over the peer schema of A. ⊣

As can be immediately seen, equality descriptions correspond to exact LAV mappings in data integration, while inclusion descriptions correspond to sound LAV mappings.

Peer mappings are used to define semantic mappings between the peer schemas of different peers. Thereby each mapping may define a connection between an arbitrary number of peers. Similar to the storage descriptions, there exist inclusion and equality mappings.

Definition 4.11 (Peer Mapping [38])**.** Let A_1 and A_2 be sets of peers, and denote with q_1 (q_2) a conjunctive query over the peer schemas of the peers in A_1 (A_2), such that q_1 and q_2 have the same arity.

- An *equality peer mapping* is a peer mapping of the form $q_1 = q_2$.
- An *inclusion peer mapping* is a peer mapping of the form $q_1 \subseteq q_2$.

⊣

It is again immediate that these mappings are able to express (exact and sound) GAV and LAV mappings between peer schemas.

There exists another type of peer mappings, called *definitional mappings*.

Definition 4.12 (Definitional Mapping [38])**.** A *definitional mapping* is a datalog rule that contains only peer relations in the head and the body. ⊣

As long as a peer relation is only contained in the head of one definitional mapping, this definitional mapping could be also written as equality peer mapping (e.g. $R(\vec{x}) :- R_1(\vec{x})$ is equivalent to $R(\vec{x}) = R_1(\vec{x})$, when considering $R(\vec{x})$ and $R_1(\vec{x})$ as conjunctive queries over the relations R and R_1 with the distinguished variables $\vec{x}$). But using definitional mappings allows to express disjunction by defining more than one such mapping with the same relation in the head (e.g. the extension of the above example by $R(\vec{x}) : -R_2(\vec{x})$ cannot be expressed by equality peer mappings any more). Moreover, restricting equality mappings to definitional mappings has advantages with respect to the complexity of query evaluation.

Having fixed the definition of the mappings, for the Piazza system a PDMS is given by a tuple $\langle \mathcal{P}, \mathcal{S}, m, \mathcal{R}, \mathcal{L}_N, \mathcal{D}_N \rangle$, where $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a set of peers, $\mathcal{S} = \{S_1, \ldots, S_j\}$ is a set of peer schemas, m is a function mapping peers to peer schemas, $\mathcal{R} = \{R_1, \ldots, R_n\}$ is a set of stored relations R_i for peer P_i (where R_i is allowed to be equal $\emptyset$), $\mathcal{L}_N$ is a set of peer mappings and $\mathcal{D}_N$ is a set of storage descriptions.

In the remainder of this section, such a system will be referred to as setting.

Semantics of $\mathcal{PPL}$

As for data integration, the interesting task in Piazza is query answering. Since, given a concrete problem instance, the mappings of $\mathcal{PPL}$ may give rise to more than one possible solution, the result of a query is defined by the notion of certain answers.

Let $N = \langle \mathcal{P}, \mathcal{S}, m, \mathcal{R}, \mathcal{L}_N, \mathcal{D}_N \rangle$ be a setting and D be an instance of the stored relations in $\mathcal{D}_N$. A *data instance* I for N assigns a set of tuples to each relation in N, that is to all peer schemas and to all stored relations.

Definition 4.13 (Consistent data instance [38])**.** Let I be a data instance for a setting N and D be an instance of the stored relations. Then I is *consistent* with N and D if

- for every equality description $A\colon R = q$ in $\mathcal{D}_N$, $R^D = q^I$,
- for every inclusion description $A\colon R \subseteq q$ in $\mathcal{D}_N$, $R^D \subseteq q^I$,
- for every equality peer mapping $q_1 = q_2$ in $\mathcal{L}_N$, $q_1^I = q_2^I$,
- for every inclusion peer mapping $q_1 \subseteq q_2$ in $\mathcal{L}_N$, $q_1^I \subseteq q_2^I$, and
- for every definitional mapping ϕ the following holds: Let p be the peer relation in the head of ϕ. Denote with $\phi_1, \ldots, \phi_k$ all definitional mappings that contain p in their head. Then $p^I = \bigcup_{1 \leqslant i \leqslant k} \phi_i^I$.

⊣

It is again obvious that there may be more than one consistent data instance for a given problem instance. Hence the certain answers to a query q with respect to N and D ($\mathtt{certain}(q, N, D)$) are defined straightforward to contain all those tuples present in the result of q over every data instance consistent with respect to N and D.

On the other hand, the question is whether there exists always at least one consistent data instance. For arbitrary PDMSs, this need not be the case:

Theorem 4.14 ([40])**.** *Let* N *be a PDMS specified in* $\mathcal{PPL}$*. If all storage descriptions in* N *are only inclusion storage descriptions, then for every instance* D *of the storage descriptions in* N *there exists a consistent data instance with respect to* N *and* D*.*

If equality storage descriptions are allowed, then there are PDMSs N *and instances* D *for the storage descriptions such that there exists no consistent data instance with respect to* N *and* D*.*

4.4.2 *Complexity of Query Answering*

The problem of *query answering* is to compute $\mathtt{certain}(q, N, D)$.

As throughout this thesis, only data complexity is considered. Not surprisingly, the complexity of query answering depends heavily on which restrictions are assumed on the mappings. One possible restriction is acyclicity of the mappings.

Definition 4.15 (Acyclic Inclusion Peer Mappings [38])**.** Let $\mathcal{L}$ be a set of inclusion peer mappings in $\mathcal{PPL}$. Construct a directed graph $D = (V, A)$ as follows: V contains a node for every relation occurring in any mapping in $\mathcal{L}$. A contains an arc from a node corresponding to relation R to a node corresponding to relation S if there exists an inclusion peer mapping $q_1 \subseteq q_2$ where R appears in q_1 and S appears in q_2.

$\mathcal{L}$ is *acyclic* if D contains no cycles. ⊣

The following was shown in [40]:

Theorem 4.16 ([40])**.** *Let* N *be a PDMS specified in* $\mathcal{PPL}$*.*

1. *The problem of finding all certain answers to a conjunctive query* q*, for given* N*, is undecidable.*
2. *If* N *includes only inclusion peer mappings and inclusion storage descriptions, and the inclusion peer mappings are acyclic, then a conjunctive query* q *can be answered in polynomial time with respect to data complexity.*

Hence cycles within the inclusion peer mappings are responsible for query answering becoming undecidable. It should be noted that equality peer mappings immediately introduce cycles, since each equality peer mapping $q_1 = q_2$ is equivalent to the inclusion peer mappings $q_1 \subseteq q_2$ and $q_2 \subseteq q_1$.

Acyclic inclusion dependencies are a very strong restriction on the topology of the network, and it turns out that it can be weakened without losing

tractability. An equality mapping $q_1 = q_2$ is called *projection free* if neither q_1 nor q_2 project out any attribute. Cycles induced by projection free equality peer mappings are tractable:

Theorem 4.17 ([38]). *Let* N *be a setting for which all inclusion peer mappings are acyclic, but that may contain equality peer mappings.*

1. *If (a) all equality storage descriptions and all equality peer mappings in* N *are projection free and (b) every peer relation that is in the head of any definitional mapping does not appear on the right hand side of any other mapping, then query answering can be done in polynomial time with respect to data complexity.*
2. *If the above conditions hold, except that equality storage descriptions are allowed to contain projections, then query answering becomes coNP-complete.*
3. *If the conditions of 1. hold, except that on the right hand side of the peer mappings UCQs are allowed, then query answering becomes coNP-complete.*

The first part of the theorem states the maximal class of mappings in $\mathcal{PPL}$ for which query answering is tractable. The hardness results of (2) and (3) follow immediately from the complexity results for query answering in LAV data integration settings given in Chapter 3 (in Table 3.2.4 on page 33):
(2) corresponds to the case of answering conjunctive queries over views defined using only conjunctive queries under CWA (that is, assuming exact LAV mappings). Given such a data integration setting $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, it can be modeled using $\mathcal{PPL}$ by a single peer P with the peer schema $\mathcal{G}$, the stored relations $\mathcal{S}$ and by adding one equality storage description P: $s_i = q_i$ for every LAV style mapping $s_i \rightsquigarrow q_i \in \mathcal{M}$.
(3) on the other hand corresponds to answering conjunctive queries over views defined using UCQs under OWA (that is, assuming sound LAV mappings). Given such a data integration setting $\mathcal{I}$, it can be modeled in $\mathcal{PPL}$ by two peers P_1 and P_2: P_1 contains one stored relation s_s and one peer relation p_s for every $s \in \mathcal{S}$, and the peer schema of P_2 equals $\mathcal{G}$. Further there exists one (projection free) equality storage description $\mathsf{P}_1 : s_s = p_s$ for every $s \in \mathcal{S}$, and one inclusion peer mapping $p_s \subseteq q_s$ for every LAV style mapping $s \rightsquigarrow q_s \in \mathcal{M}$ (where q_s is a UCQ).

The above results have been considered for queries without comparison predicates. In addition to these results, also the effect of comparison predicates on the complexity of query answering has been reported.

Theorem 4.18 ([38]). *Let* N *be a maximal expressive* $\mathcal{PPL}$ *setting as defined in Theorem 4.17 for which query answering is tractable, and* q *be a conjunctive query.*

- *If comparison predicates appear only in storage descriptions or in the bodies of definitional mappings, but not in* q*, then query answering can still be done in polynomial time.*
- *Otherwise (that is either* q *contains comparison predicates or they are used in equality or inclusion peer mappings), query answering becomes a coNP problem.*

4.4.3 *Query Reformulation*

Query answering in the Piazza system is done by reformulating a query q over some peer schema into a query q' that only refers to stored relations. This rewriting is a maximally contained rewriting, and moreover when query answering is tractable q' is a perfect rewriting of q (see Section 3.2.4 for the definitions of maximally contained and perfect rewriting). Describing this reformulation and the corresponding algorithm in detail is beyond the scope of this thesis, such that in the following only the main properties of such a rewriting are summarised.

The algorithm computing the rewriting is a centralised algorithm, that means that it runs locally on the peer to which the query is posed. Since the algorithm requires global knowledge about about all mappings in the system, each peer has access to this information. Hence, the algorithm takes as input the query q together with all peer mappings and storage descriptions, and it outputs a query q' referring only to stored relations that is guaranteed to only return certain answers. Moreover, if computing all certain answers is feasible, q' returns all certain answers. In the case if query answering is not feasible, approximation techniques can be used to retrieve at least some (and only correct) certain answers.

The reformulation consists of two steps. First, q is reformulated according to the peer mappings. Once there is no more rewriting possible using peer mappings, the retrieved result is rewritten in the second step using the storage descriptions. One of the main ideas for the first step is to regard all peer mappings either as pure LAV or as pure GAV mappings, and therefore to use techniques either from LAV or GAV query rewriting (i.e. techniques for answering queries using views or unfolding strategies). To achieve this, in a preprocessing step all equality mappings are rewritten as two inclusion mappings. Then, each inclusion mapping of the form $q_1 \subseteq q_2$ is transformed into the mappings $V \subseteq q_2$ and $V{:-}q_1$, where V is a fresh relational symbol. After this, each peer mapping is either a sound LAV style mapping $s \subseteq q_s$, where s is either a peer relation or one of the fresh relational symbols, or a GAV style mapping $s{:-}q_s$. The first step of the reformulation then is building a *rule-goal tree* (although the authors point out that in fact it is more likely to become a DAG), whose root consists of the goals of the queries. As long there are subgoals in the tree that can be expanded with respect to some peer mapping, this is done by either applying rules for GAV or LAV query rewriting, depending on the applicable rule.

Once no more rule can be applied to any subgoal, the resulting subgoals (over the peer schemas) can be rewritten using the storage descriptions to queries over the stored relations that then need to be combined according to the structure of the rule-goal tree.

As indicated by Theorem 4.18, this algorithm can be extended to handle comparison predicates as well. This is done by annotating the branches of the rule-goal tree according to the constraints defined by the predicates.

Because the rule-goal tree may become both, deep and highly branching, optimisation techniques are considered and implemented. Besides memoisa-

tion techniques and methods for pruning the tree (detection of dead ends and redundant paths), also strategies for selecting an order of rule application such that the effect of pruning techniques can be maximised are considered. [39, 40] contain more information about the optimisation techniques considered.

Experiments with the prototype implementation [38] showed that one bottleneck of the rewriting is the second rewriting step, that is the transformation of the rule-goal tree to queries over the stored relations, while building the rule-goal tree scales quite well.

Another important aspect is that the query reformulation algorithm should find some first reformulations (and hence answers) quickly, that is it should return the first results while the rule-goal tree is still expanded, such that the user does not need to wait until the complete rewriting has finished before the first results arrive.

4.4.4 *Further Considerations*

Equivalence

For performing global optimisations of an instance of a PDMS, like removing redundant mappings or composing mappings to speedup query answering, it is necessary to have a notion of equivalence of different settings to be able to decide whether some optimisation does not change the semantics of the optimised system.

Definition 4.19 (Equivalence [40])**.** Let N_1 and N_2 be two PDMS. W.l.o.g. assume that they have the same set of peers, peer relations and stored relations. N_1 is *equivalent* to N_2 if for every instance D of the stored relations (in N_1 and N_2) and every query q over the peer schema of one peer, $certain(q, N_1, D) = certain(q, N_2, D)$. ⊣

Using this definition, equivalence of systems depends on the query language of q. For example, while systems may not be equivalent for queries expressed in FOL (*FOL-equivalence*), they may be equivalent for CQs (*CQ-equivalence*).

Another notion of equivalence is that of *relative equivalence*.

Definition 4.20 (Relative Equivalence [40])**.** Let N_1, N_2 be two PDMS, and $\bar{P} = \{P_1, \ldots, P_m\}$ be a set of peer relations. N_1 and N_2 are *equivalent relative to* $\bar{P}$, if for every instance D of the stored relations and any query q over the peer relations in $\bar{P}$, $certain(q, N_1, D) = certain(q, N_2, D)$ ⊣

Relative equivalence is very interesting in practice, since a PDMS is considered to be dynamic, that is peers can leave and join at will. If such a change of the system occurs, relative equivalence expresses whether this influences the result of a query over a certain peer.

[40] states a class of systems for which relative equivalence is decidable.

Theorem 4.21 ([40])**.** *Let N_1, N_2 be two PDMS with only acyclic inclusion peer mappings, and where the storage descriptions do not contain both, inclusion and equality storage descriptions. Then*

- *it is decidable whether* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *are FOL-equivalent.*
- *If all the peer mappings in* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *contain only a single relational symbol on their left hand side, then for every set of* $\bar{\mathcal{P}}$ *of peer relations, it is decidable whether* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *are CQ-equivalent relative to* $\bar{\mathcal{P}}$*.*

More details about equivalence of $\mathcal{PPL}$ systems can be found in [40].

Miscellaneous

Within the Piazza project further issues have been considered:

- To support efficient search in a PDMS, index structures for the stored data are considered [75].
- Piazza shall also allow users to define access policies to their data, that is to restrict the data access according to user defined rules [60, 75].
- In [75], also some ideas are summarised how the creation of schema mappings could be supported by the system.
- How Piazza (and $\mathcal{PPL}$) is extended to XML is especially described in [37] and [39].

As already pointed out, besides developing the logical model of Piazza and query answering algorithms, there exists also a prototype implementation with experimental results reported in [37, 38, 40].

Since this will be an important property when discussing the approach of De Giacomo et al. in the next chapter, for query answering in a PDMS described using $\mathcal{PPL}$ to be tractable, some global restrictions on the system are required. Hence, the peers cannot choose their mappings independently, but require global knowledge to create a tractable instance.

5 THE APPROACH OF DE GIACOMO ET AL.

In this chapter, a detailed description of the framework proposed in [30] is given. A prototype implementation of this approach has been created as part of this thesis and is described in the next chapter.

In [30], two main results are presented: First, the data exchange setting described in Section 3.1 is generalised to a P2P based setting by using the semantics described in Section 4.2.4 to model the mappings between different peers: Instead of one source and one target schema with a mapping defined between them, the proposed *P2P data exchange system* (PDE-system) consists of a set of peer schemas with pairwise mappings between them. Like in data exchange, the goal is — given initial instances for each peer — to materialise instances of the peer schemas such that all mappings are satisfied. To the best of our knowledge, such a straight forward generalisation of data exchange has not been proposed before.

PDE-systems are then enhanced by "virtual" mappings, corresponding to the mappings used in data integration systems (Section 3.2). Although expressed by the same constructs (namely TGDs and EGDs) as data exchange mappings, their meaning is different. While data exchange mappings are used to express constraints that have to be satisfied by the data stored physically in a database, virtual mappings express relationships that are only considered for query answering, but that do not directly affect the data actually stored in a database. Hence the resulting *P2P data exchange and data integration system* (PDEI-system) combines the ideas of data exchange, data integration and peer data management systems. Moreover, it contains both, data exchange and data integration as special cases.

To allow for an arbitrary topology of the P2P mappings (since global restrictions on a network of autonomous peers seem unrealistic or at least not desirable), instead of using the first-order logic interpretation for these mappings (like e.g. in $\mathcal{PPL}$), the semantics of the P2P mappings is defined by adopting the epistemic logic approach mentioned in Section 4.2.4.

In the following, first PDE- and PDEI-systems are introduced formally. Then an extension of the chase is presented for solving the data exchange problem in these settings, together with the corresponding complexity results. Finally query answering in PDEI-systems is described.

5.1 BASIC DEFINITIONS

Before defining PDE- and PDEI-systems formally, it is necessary to introduce two more definitions:

Definition 5.1 (Instantiation [30])**.** Let **R** be a schema, and assume an indefinite instance $\mathbb{B}$ for **R**.
A definite instance $\mathbb{D}$ is an *instantiation* of $\mathbb{B}$ if there exists an injective

function $f\colon \mathit{const}(B) \cup \mathit{nulls}(B) \to \mathcal{C}$ such that $\forall c \in \mathit{const}(B)\colon f(c) = c$, and $D = \{r(f(t_1), \ldots, f(t_n)) \mid r(t_1, \ldots, t_n) \in B\}$. Moreover, no $x \in \mathit{nulls}(B)$ is mapped to some $c \in \mathit{const}(B)$. ⊣

Note that f is injective, hence every labelled null in B is mapped to a different constant not already present in B. Moreover it follows immediately that if D is an instantiation of B, then there exists a homomorphism $h\colon B \to D$. On the other hand, just because there exists a homomorphism $h\colon B \to D$ does not mean that D is an instantiation of B, as among other restrictions an instantiation is not allowed to contain additional facts.

Let D be a definite instance over some schema $\mathbf{R}$. Recall from Section 3.1 that a TGD $q_i \to q_j$ over $\mathbf{R}$ is *satisfied* in D if $q_i^D \subseteq q_j^D$, and an EGD $q(x_1, x_2) \to x_1 = x_2$ over $\mathbf{R}$ is *satisfied* in D if for each answer $\langle t_1, t_2 \rangle \in q^D$, $t_1 = t_2$.

As already mentioned above, the suggested framework does not pose any limitations to the topology of the mappings between the peers (like for example acyclicity). The intention of this is that such limitations mean that a single peer, when joining the system and defining its mappings to the other peers, requires global knowledge about the existing mappings to avoid the creation of forbidden cycles. This is seen as both an unrealistic and undesirable assumption. Instead, defining the mappings of a peer shall be possible using only local information. Because the mappings between the peers are expressed by TGDs, without global constraints, non weakly acyclic settings cannot be avoided. Hence, to obtain decidability, a different notion of satisfiability for TGDs than the above mentioned (corresponding to a FOL interpretation) is needed. Therefore, the alternative semantics first proposed in [15] and already introduced in Section 4.2.4 is used for PDE- and PDEI-systems. Its intuition is that peers only exchange certain answers, that is information that is definitely known by the peers. Informally, a TGD $q_i \to q_j$ is satisfied under this semantics if $\mathtt{certain}(q_i, \mathcal{D}) \subseteq \mathtt{certain}(q_j, \mathcal{D})$, where $\mathcal{D}$ is a set of instances, and $\mathtt{certain}(q, \mathcal{D})$ denotes the certain answers to q with respect to $\mathcal{D}$.

Without stressing the use of epistemic logic, in [30] this semantics is formalised using the notion of *CERT-satisfiability*:

Definition 5.2 (CERT-satisfied [30])**.** Let $\mathcal{D}$ be a set of definite instances, and $\phi\colon q_i \to q_j$ be a TGD. Then ϕ is *CERT-satisfied in* $\mathcal{D}$ if

$$\bigcap_{D \in \mathcal{D}} q_i^D \subseteq \bigcap_{D \in \mathcal{D}} q_j^D \ .$$

⊣

Note that by this definition, a TGD is not satisfied by a single instance, but by a set of definite instances, just as for the formalisations presented in Section 4.2.4.

For both, PDE- and PDEI-systems (as for all PDMS) a query q is posed over the schema of a single peer. Moreover, as long as not stated otherwise, q is assumed to be a UCQ.

5.2 PDE-SYSTEM

As already stated, PDE-systems are a generalisation of data exchange settings: Instead of a single source containing data, and a single (empty) target where data shall be materialised, a PDE-setting consists of several peers, connected to each other by TGDs and being both, source and target at the same time.

Formally:

Definition 5.3 (PDE-system [30])**.** A *P2P data exchange system* (PDE-system) is a triple $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$ where

- $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a set of peers, each containing a relational schema (the schemas of the peers are assumed to be pairwise disjoint),
- $\mathcal{C}_E$ is a set of *(local) constraints*, that is a set of TGDs and EGDs, where each constraint is expressed over the schema of a single peer, and
- $\mathcal{M}_E$ is a set of *P2P mappings*, that is a set of TGDs $q_i \to q_j$, where q_i, q_j are expressed over the schemas of P_i, P_j resp.

⊣

Example 5.4. The following simple PDE-system will be used as running example in this chapter: Assume a slight variation of the setting in Example 3.6, consisting of the three peers $\mathtt{Uni}$, $\mathtt{Lib}_{\mathtt{UB}}$, $\mathtt{Lib}_{\mathtt{Dep}}$ with the following schemas:
$S(\mathtt{Uni}) = \{\mathtt{Staff}(\mathtt{FirstName}, \mathtt{LastName}, \mathtt{Departement})\}$
$S(\mathtt{Lib}_{\mathtt{UB}}) = \{\mathtt{Authorised}(\mathtt{StudentId}, \mathtt{FirstName}, \mathtt{LastName})\}$
$S(\mathtt{Lib}_{\mathtt{Dep}}) = \{\mathtt{Allowed}(\mathtt{StudentId}, \mathtt{FirstName}, \mathtt{LastName}),$
$\mathtt{Contact}(\mathtt{StudentId}, \mathtt{Email})\}$
Moreover assume the following constraints and mappings:

1 $\mathtt{Authorised}(\mathtt{SID}, \mathtt{FN}, \mathtt{LN}) \to \mathtt{Allowed}(\mathtt{ID}, \mathtt{FN}, \mathtt{LN})$ $(\in \mathcal{M}_E)$

2 $\mathtt{Allowed}(\mathtt{SID}, \mathtt{FN}, \mathtt{LN}) \to \mathtt{Authorised}(\mathtt{SID}, \mathtt{FN}_1, \mathtt{LN}_1)$ $(\in \mathcal{M}_E)$

3 $\mathtt{Staff}(\mathtt{FN}, \mathtt{LN}, \mathtt{Dep}) \to \mathtt{Authorised}(\mathtt{SID}, \mathtt{FN}, \mathtt{LN})$ $(\in \mathcal{M}_E)$

4 $\mathtt{Allowed}(\mathtt{SID}, \mathtt{FN}, \mathtt{LN}) \to \mathtt{Contact}(\mathtt{SID}, \mathtt{Email})$ $(\in \mathcal{C}_E)$

5 $\mathtt{Contact}(\mathtt{SID}, \mathtt{Email}) \to \mathtt{Allowed}(\mathtt{SID}, \mathtt{FN}, \mathtt{LN})$ $(\in \mathcal{C}_E)$

Hence the PDE-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ is defined by

- $\mathcal{P} = \{\ \mathtt{Uni}, \mathtt{Lib}_{\mathtt{UB}}, \mathtt{Lib}_{\mathtt{Dep}}\ \}$
- $\mathcal{M}_E = \{\ 1, 2, 3\ \}$
- $\mathcal{C}_E = \{\ 4, 5\ \}$

(where the numbers refer to the list of TGDs above). ◇

The instances over the schemas of the peers of a PDE-system are called states:

Definition 5.5. Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$ be a PDE-system, and denote with $S(\mathcal{P}) = \langle S(P_1), S(P_2), \ldots, S(P_n) \rangle$ the union of the schemas of the peers in $\mathcal{P}$. A *state* B for $\mathcal{S}$ is an instance of $S(\mathcal{P})$.
According to the terminology for instances, if $\mathit{nulls}(B) = \emptyset$, then B is a *definite state*, otherwise B is an *indefinite state*. ⊣

Like in data exchange, since given some PDE-system and state, the goal is to materialise some valid instance, it needs to be clarified under which condition a PDE-system is satisfied (with respect to a given state).

As already mentioned in Section 4.2.4, PDE-systems adopt the alternative semantics for P2P mappings. The idea of peers exchanging only certain answers is formally defined in [30] as follows:

Definition 5.6 ([30])**.** Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$ be a PDE-system, B a state for $\mathcal{S}$ and $\mathcal{D}$ a set of definite states for $\mathcal{S}$. $\mathcal{D}$ *satisfies* $\mathcal{S}$ and B if

1. for each $D \in \mathcal{D}$ there exists a homomorphism from B to D,
2. for each $D \in \mathcal{D}$ and for each TGD or EGD $\phi \in \mathcal{C}_E$, ϕ is satisfied in D, and
3. for each $\phi \in \mathcal{M}_E$, ϕ is CERT-satisfied in $\mathcal{D}$.

⊣

Note that this definition describes exactly the semantics of the epistemic model of a PDMS described in Section 4.2.4: An isolated peer, only with respect to its local constraints is described by a FOL theory. With respect to the P2P mappings on the other hand, a PDE-system is no longer satisfied by a single instance, but by a set of instances (not every single $D \in \mathcal{D}$, but $\mathcal{D}$ satisfies $\mathcal{S}$ and B).

Let $\mathcal{D}$ and $\mathcal{D}'$ be two sets of definite states that satisfy $\mathcal{S}$ and B. Then it is easy to check that also $\mathcal{D} \cup \mathcal{D}'$ satisfies $\mathcal{S}$ and B. Hence it follows immediately that there exists a unique maximal set of definite states that satisfies $\mathcal{S}$ and B, namely the union over all such sets. Denote this maximal set with $\mathit{Sem}(\mathcal{S}, B)$. Intuitively, $\mathit{Sem}(\mathcal{S}, B)$ contains all possible solutions for $\mathcal{S}$ with respect to B.

With these notions settled, several properties of states can be considered:

Definition 5.7 ($\mathcal{S}$-Consistent State [30])**.** Let $\mathcal{S}$ be a PDE-system, and B a state for $\mathcal{S}$. Then B is $\mathcal{S}$-*consistent* if $\mathit{Sem}(\mathcal{S}, B) \neq \emptyset$. ⊣

This means that a state B is $\mathcal{S}$-consistent if all mappings and constraints of $\mathcal{S}$ are satisfiable with respect to B, i.e. if there exists a solution to the data exchange problem.

Definition 5.8 ($\mathcal{S}$-Admissible State [30])**.** Let $\mathcal{S}$ be a PDE-system, and B a state for $\mathcal{S}$. Then B is $\mathcal{S}$-*admissible* if

1. B is $\mathcal{S}$-consistent and
2. every instantiation of B belongs to $\mathit{Sem}(\mathcal{S}, B)$.

⊣

Intuitively, an $\mathcal{S}$-admissible state is a state where all data exchange defined by $\mathcal{S}$ has taken place. Note that formally the P2P mappings in $\mathcal{S}$ are not satisfied by a single state, but by a set of definite states. However, by the intuition of this semantics, a mapping is satisfied if all data known by a peer is exchanged. Thereby a value is regarded to be "known" if it is the same in every possible world, and the set of possible worlds is exactly described by $Sem(\mathcal{S}, B)$. Since there exists a homomorphism from B to every $D \in \mathcal{D}$, exactly those information expressed by constants in B are regarded to be known. Hence intuitively an $\mathcal{S}$-admissible state is a state such that for each P2P mapping $q_i \rightarrow q_j$, every definite tuple t from the answer of q_i over peer $\mathcal{P}_i$ is also contained in the answer of q_j over peer $\mathcal{P}_j$.

Definition 5.9 (Universal $\mathcal{S}$-Solution [30])**.** Let $\mathcal{S}$ be a PDE-system, and B a $\mathcal{S}$-consistent state. A state B' for $\mathcal{S}$ is a *universal $\mathcal{S}$-solution* of B if

1. B' is $\mathcal{S}$-admissible and
2. $Sem(\mathcal{S}, B') = Sem(\mathcal{S}, B)$.

⊣

That is, given a state B, a universal $\mathcal{S}$-solution is a state where all data exchange defined by $\mathcal{S}$ based on B has taken place, such that the set of definite states that satisfy $\mathcal{S}$ and B is not changed. Therefore, on the one hand no information present in B is removed, and on the other hand no data is added to B' that is not justified by B and the mappings and constraints in $\mathcal{S}$.

Example 5.10. Consider the PDE-system $\mathcal{S}$ from Example 5.4. The following state is $\mathcal{S}$-consistent:
$B = \{Staff(\text{'eve'},\text{'e.'},\text{'unkn.'}), Authorised(\text{'0912345'},\text{'alice'},\text{'a.'}),$
$Allowed(\text{'621005'},\text{'bob'},\text{'bond'}), Contact(\text{'9000'},\text{'dave@hal.net'})\}$.
Note that since $\mathcal{C}_E$ does not contain any EGD, in fact all states are $\mathcal{S}$-consistent. But it is easy to think of a system with EGDs and a corresponding state that is not $\mathcal{S}$-consistent.

Although being $\mathcal{S}$-consistent, B is not $\mathcal{S}$-admissible, as there are several facts implied by the mappings and constraints that are not materialised in B. An example for an $\mathcal{S}$-admissible state is B':
$B' = \{Staff(\text{'eve'},\text{'e.'},\text{'unkn.'}), Authorised(\text{'0912345'},\text{'alice'},\text{'a.'}),$
$Authorised(\text{'621005'}, FN_2, LN_2), Authorised(\text{'9000'}, FN_3, LN_3),$
$Authorised(SID_2, \text{'eve'},\text{'e.'}),$
$Allowed(\text{'621005'},\text{'bob'},\text{'bond'}), Allowed(SID_1, \text{'alice'}, \text{'a.'}),$
$Allowed(\text{'9000'}, FN_1, LN_1), Allowed(SID_3, \text{'eve'}, \text{'e.'}),$
$Contact(\text{'9000'},\text{'dave@hal.net'}), Contact(SID_1, Email_1),$
$Contact(Contact(SID_3, Email_2)\}$
Note that the TGDs from $\mathcal{M}_E$ are not satisfied under FOL semantics, as for example SID_1 and SID_3 do not appear in the $Authorised$ table. However, all P2P mappings are satisfied with respect to the weaker semantics. Note further that the two TGDs 1 and 2 (see Example 5.4) do not form a weakly acyclic set. Hence under FOL semantics, there would not exist a finite universal solution for B.

$\mathbb{B}'$ however is not just some $\mathbb{S}$-admissible state, but even a universal $\mathbb{S}$-solution of $\mathbb{B}$. ◇

Just as in data exchange, also for PDE-systems the notion of a minimal universal solution exists:

Definition 5.11 ($\mathbb{S}$-Core [30])**.** Let $\mathbb{S}$ be a PDE-system and $\mathbb{B}$ an $\mathbb{S}$-consistent state. A state $\mathbb{B}'$ is an $\mathbb{S}$-*core of* $\mathbb{B}$ if

1. $\mathbb{B}'$ is a universal $\mathbb{S}$-solution of $\mathbb{B}$ and
2. there exists no $\mathbb{B}'' \subset \mathbb{B}'$ such that $\mathbb{B}''$ is a universal $\mathbb{S}$-solution of $\mathbb{B}$.

⊣

Note that also for PDE-systems the cores of all universal solutions are unique up to isomorphisms.

For better understanding of the intuition of these definitions, it might be helpful to relate them to the concepts used in data exchange. In data exchange, a solution to an instance of the data exchange problem is some target instance J, such that $\langle I, J\rangle$ (let I be the given source instance) satisfies $\mathcal{M}$ (Section 3.1). As stated above, the mappings of a PDE-system are not satisfied by a single instance, but only by a set of definite instances. Hence, intuitively, the solution for the correspondence of the data exchange problem over a PDE setting is a set of definite instances. Therefore, an $\mathbb{S}$-consistent state corresponds to a source instance I in data exchange such that $\mathrm{Sol}(\mathcal{M}, I) \neq \emptyset$, so the question whether a given state is $\mathbb{S}$-consistent corresponds to the question whether $\mathrm{Sol}(\mathcal{M}, I) \neq \emptyset$ in data exchange.

An $\mathbb{S}$-admissible state simply corresponds to an instance $\langle I, J\rangle$ in data exchange where no mapping and no constraint is violated, that is all data that can be derived by the mappings and constraints from the current state is actually materialised.

In data exchange, it was shown that some solutions are more general than others, and on the other hand that given a source instance, some target instances contain facts not justified by the mapping $\mathcal{M}$. The universal solutions have been those solutions that are general enough to capture all solutions of a given problem instance. This is exactly the intuition of universal $\mathbb{S}$-solutions: They contain all positive information that is shared by all definite instances in $\mathrm{Sem}(\mathbb{S}, \mathbb{B})$, but not more.

The definition of the $\mathbb{S}$-core is only stated for completeness, as core computation will be omitted in this thesis.

As universal $\mathbb{S}$-solutions express exactly the (positive) information shared by all instances in $\mathrm{Sem}(\mathbb{S}, \mathbb{B})$, like in data exchange, the question arises, given that $\mathrm{Sem}(\mathbb{S}, \mathbb{B}) \neq \emptyset$, whether such a state always exists (corresponding to Corollary 3.22). This is actually the case, and just as in data exchange, all universal $\mathbb{S}$-solutions are homomorphically equivalent, as shown by the following result:

Proposition 5.12 ([30])**.** *Let $\mathbb{S}$ be a PDE-system and $\mathbb{B}$ an $\mathbb{S}$-consistent state. Then*

1. *there exists at least one universal $\mathbb{S}$-solution of $\mathbb{B}$ and*
2. *all universal $\mathbb{S}$-solutions of $\mathbb{B}$ are homomorphically equivalent.*

Moreover, for PDE-systems, the core, that is the minimal universal $\mathcal{S}$-solution, is unique up to isomorphism.

Proposition 5.13 ([30]). *Let $\mathcal{S}$ be a PDE-system and B an $\mathcal{S}$-consistent state. Then, there exists a unique (up to isomorphism) $\mathcal{S}$-core of B.*

5.2.1 *Relationship with Data Exchange*

In the last section, the concepts of PDE-systems have only been related intuitively to those in data exchange. However, a PDE-system is a real generalisation of a data exchange setting. This means that the data exchange setting appears as a special case of a PDE-system (hence, every data exchange setting can be transformed into an equivalent PDE-system):

Given a data exchange setting $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$, denote with $\mathcal{S}_{\mathcal{M}}$ the corresponding PDE-system $\mathcal{S}_{\mathcal{M}} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$, where

- $\mathcal{P} = \{P_s, P_t\}$ with $S(P_s) = \mathbf{S}$ and $S(P_t) = \mathbf{T}$,
- $\mathcal{C}_E = \Sigma_t$, and
- $\mathcal{M}_E = \Sigma_{st}$.

That is, just as described at the beginning of Chapter 4, source and target are modeled as peers. Given a source instance I for $\mathbf{S}$, this corresponds to a definite instance for the schema of P_s, while the instance of P_t is empty.

As stated by the next result, all important concepts of data exchange occur in the corresponding PDE-system:

Theorem 5.14 ([30]). *Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a data exchange setting, I a finite definite instance for $\boldsymbol{S}$, and $\mathcal{S}_{\mathcal{M}}$ the PDE-system corresponding to $\mathcal{M}$. Then*

1. $\mathsf{Sem}(\mathcal{S}, \langle I, \emptyset \rangle) \neq \emptyset$ *iff* $\mathsf{Sol}(\mathcal{M}, I) \neq \emptyset$.
2. *If I' is a finite universal $\mathcal{S}$-solution for I, then $(I' \setminus I)$ is a universal solution for I in $\mathcal{M}$ ($(I' \setminus I)$ restricts the instance I' to an instance of $\boldsymbol{T}$). Vice versa, if J is a universal solution for I in $\mathcal{M}$, then $\langle I, J \rangle$ is a (finite) universal $\mathcal{S}$-solution for I.*
3. *If I' is the $\mathcal{S}$-core of I, then $(I' \setminus I)$ is the core of the universal solutions for I in $\mathcal{M}$. Vice versa, if J is the core of the universal solutions for I in $\mathcal{M}$, then $\langle I, J \rangle$ is the $\mathcal{S}$-core of I.*

Note that these correspondences only hold for definite source instances I, since for a definite instance the set of answers and certain answers (to the same query q) coincide, and therefore the difference between the semantics applied to the source-to-target TGDs in data exchange and the one applied to the P2P mappings in PDE-systems does not show up.

The above theorem shows that the concepts of universal $\mathcal{S}$-solutions and $\mathcal{S}$-cores are real generalisations of the corresponding concepts in data exchange.

5.2.2 *Certain Answers*

For PDE-systems it typically holds that if $Sem(\mathcal{S}, B) \neq \emptyset$, then even $|Sem(\mathcal{S}, B)| > 1$. And just as in data exchange, the results of query answering should not depend on which of these states has been chosen for materialisation. That is the answer to a query should not depend on the state it was evaluated on, but on all answers in $Sem(\mathcal{S}, B)$. This immediately gives rise to the notion of certain answers for a PDE-system:

Definition 5.15 (Certain Answers [30])**.** Let $\mathcal{S}$ be a PDE-system, B an $\mathcal{S}$-admissible state and q a query over $\mathcal{S}$. Then the set of *certain answers* to q in $\mathcal{S}$ and B is defined as

$$certain(q, \mathcal{S}, B) = \bigcap_{D \in Sem(\mathcal{S}, B)} q^D \ .$$

⊣

In data exchange, computing the certain answers to a query q over $\mathbf{T}$ can be done by evaluating q over a universal solution and then removing all tuples from the result that contain labelled nulls (see Theorem 3.24). By the close relationship between data exchange and PDE-systems it is not surprising that a similar result holds for PDE-systems as well.

Theorem 5.16 ([30])**.** *Let $\mathcal{S}$ be a PDE-system, B an $\mathcal{S}$-admissible state for $\mathcal{S}$ and q a query over $\mathcal{S}$. Then* $certain(q, \mathcal{S}, B) = Eval_{Null\downarrow}(q, B)$.

Note that B is assumed to be an $\mathcal{S}$-admissible state (and not as in data exchange a universal solution). But as can be easily seen, every $\mathcal{S}$-admissible state is a universal $\mathcal{S}$-solution of itself. Moreover, by definition, every universal $\mathcal{S}$-solution is an $\mathcal{S}$-admissible state. Hence, given a state that is not $\mathcal{S}$-admissible, query answering still consists of first finding a universal $\mathcal{S}$-solution, and then evaluating the query over the resulting state. (How a universal $\mathcal{S}$-solution for a given state B can be computed is described in Section 5.4.)

5.3 PDEI-SYSTEM

Although, as stated in Section 3.3, the mappings used in data integration (in the following referred to as "virtual mappings") can be also expressed by TGDs, their meaning is completely different. Dependencies expressed by virtual mappings are not regarded as conditions that have to be enforced on the data materialised in some database, but the data implied by virtual mappings shall be considered for query answering. The same is true for virtual constraints over a schema. An instance of the schema need not satisfy all those constraints, but for computing answers to a query, their effects must be taken into account.

Hence when extending the PDE-system by virtual mappings, one has to distinguish between data exchange (i.e. mappings to be enforced on the data) and virtual mappings (and constraints). Therefore, we refer to virtual

mappings as *i-mappings*, while we call data exchange mappings *e-mappings*. Constraints over the schema of a single peer are called *i-constraints* and *e-constraints*, respectively.

This necessary distinction is also reflected in the definition of PDEI-systems:

Definition 5.17 (PDEI-system [30])**.** A *P2P data exchange and integration system* (PDEI-system) is a 5-tuple $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ where

- $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a set of peers, each containing a relational schema (the schemas of the peers are assumed to be pairwise disjoint),
- $\mathcal{C}_E$ (*e-constraints*) and $\mathcal{C}_I$ (*i-constraints*) are two sets of TGDs and EGDs where each constraint is expressed over the schema of a single peer, and
- $\mathcal{M}_E$ (*e-mappings*) and $\mathcal{M}_I$ (*i-mappings*) are two sets of P2P mappings, that is sets of TGDs $q_i \to q_j$ where q_i,q_j are expressed over the schema of P_i, P_j, respectively.

⊣

For defining the semantics of PDEI-systems, it is necessary to settle on when a PDEI-system is regarded to be satisfied (with respect to a given state).

Definition 5.18 ([30])**.** Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ be a PDEI-system and B a state for $\mathcal{S}$. A set $\mathcal{D}$ of definite instances satisfies $\mathcal{S}$ and B if

1. for each $D \in \mathcal{D}$, there exists a homomorphism from B to D,
2. for each $D \in \mathcal{D}$ and for each TGD or EGD $\phi \in \mathcal{C}_E \cup \mathcal{C}_I$, ϕ is satisfied in D, and
3. for each $\phi \in \mathcal{M}_E \cup \mathcal{M}_I$, ϕ is CERT-satisfied in $\mathcal{D}$.

⊣

This means that both, data exchange and virtual mappings have the same semantics, and only differ in their behaviour with respect to materialisation of data. Hence, for the same reason as for PDE-systems, for every PDEI-system $\mathcal{S}$ and state B for $\mathcal{S}$, there exists a unique maximal set of definite instances that satisfy $\mathcal{S}$ and B. Just as for PDE-systems, this is denoted as $Sem(\mathcal{S}, B)$.

Example 5.19. The following PDEI-system is derived from the PDE-system of Example 5.4 by two minor changes:

Assume the same set of peers $\{ Uni, Lib_{UB}, Lib_{Dep} \}$ with the same schemas as in Example 5.4: $S(Uni) = \{Staff(FirstName, LastName, Departement)\}$
$S(Lib_{UB}) = \{Authorised(StudentId, FirstName, LastName)\}$
$S(Lib_{Dep}) = \{Allowed(StudentId, FirstName, LastName),$
$\quad Contact(StudentId, Email)\}$
Moreover assume the following constraints and mappings:

1 $Authorised(SID, FN, LN) \to Allowed(ID, FN, LN)$ $(\in \mathcal{M}_E)$

2 $Allowed(SID, FN, LN) \to Authorised(SID, FN_1, LN_1)$ $(\in \mathcal{M}_E)$

3 $Staff(FN, LN, Dep) \to Authorised(SID, FN, LN)$ $(\in \mathcal{M}_E)$

4 $\mathtt{Allowed}(\mathsf{SID}, \mathsf{FN}, \mathsf{LN}) \to \mathtt{Contact}(\mathsf{SID}, \mathtt{Email})$ $(\in \mathcal{C}_E)$

5 $\mathtt{Contact}(\mathsf{SID}, \mathtt{Email}) \to \mathtt{Allowed}(\mathsf{SID}, \mathsf{FN}, \mathsf{LN})$ $(\in \mathcal{C}_I)$

6 $\mathtt{Staff}(\mathsf{FN}, \mathsf{Ln}, \mathsf{Dep}) \to \mathtt{Allowed}(\mathsf{ID}, \mathsf{FN}, \mathsf{LN})$ $(\in \mathcal{M}_I)$

Notice that the dependencies 1 – 4 are equal to the dependencies 1 – 4 in Example 5.4. Dependency 5 is also the same as in the previous example, but instead of being a data exchange constraint, it is now virtual. The virtual mapping 6 is new.

Hence the PDEI-system $\mathcal{S} = \langle \mathcal{P}, \emptyset, \emptyset, \mathcal{C}_I, \mathcal{M}_I \rangle$ is defined by

- $\mathcal{P} = \{ \mathtt{Uni}, \mathtt{Lib}_{\mathtt{UB}}, \mathtt{Lib}_{\mathtt{Dep}} \}$
- $\mathcal{M}_E = \{ 1, 2, 3 \}$
- $\mathcal{C}_E = \{ 4 \}$
- $\mathcal{M}_I = \{ 6 \}$
- $\mathcal{C}_I = \{ 5 \}$

(where the numbers refer to the list of TGDs above). ◇

With this settled, the definition of an $\mathcal{S}$-consistent state (Definition 5.7) for PDE-systems carries over to PDEI-systems, when using the notion of $\mathit{Sem}(\mathcal{S}, B)$ for PDEI-systems.

In a PDEI-system, given some state, both, virtual and data exchange mappings and constraints imply certain facts. While data exchange dependencies require that these facts are materialised in the database, this is not necessary for data implied by virtual dependencies. The question is what to do with data that can be derived using both, virtual and data exchange dependencies. De Giacomo et al. argument that such facts shall not be materialised. Hence in a PDEI-system, it is not required that all data derivable by data exchange dependencies is materialised, but only those facts that are not logically implied by the virtual dependencies.

Like the $\mathcal{S}$-admissible state for a PDE-system, a state should be $\mathcal{S}$-admissible for a PDEI-system if all required data exchange has taken place. By the above discussion, this means that all facts implied by the e-mappings and e-constraints have been materialised, except those that are already implicitly given by the i-mappings or i-constraints. To state it another way, a state B is $\mathcal{S}$-admissible, if in the state B' that is obtained from B by adding all facts derivable from B by the i-mappings and i-constraints, all e-mappings and e-constraints are satisfied.

For a formal definition of such a state, some notations need to be fixed first: Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ be a PDEI-system. Then denote with $\mathcal{S}^E$ the PDEI-system derived from $\mathcal{S}$ by removing all i-constraints and i-mappings, denote with $\mathcal{S}^I$ the PDEI-system derived from $\mathcal{S}$ by removing all e-constraints and e-mappings, and denote with $\mathcal{S}^{I \to E}$ the PDEI-system derived from $\mathcal{S}$ by replacing the e-mappings with the i-mappings, the e-constraints with the i-constraints, and removing all i-constraints and i-mappings. That is,

- $\mathcal{S}^{E} = \langle \mathcal{P}, \mathcal{C}_{E}, \mathcal{M}_{E}, \emptyset, \emptyset \rangle$
- $\mathcal{S}^{I} = \langle \mathcal{P}, \emptyset, \emptyset, \mathcal{C}_{I}, \mathcal{M}_{I} \rangle$
- $\mathcal{S}^{I \to E} = \langle \mathcal{P}, \mathcal{C}_{I}, \mathcal{M}_{I}, \emptyset, \emptyset \rangle$.

Then the informal description of an $\mathcal{S}$-admissible state from above can be formally defined as:

Definition 5.20 ($\mathcal{S}$-Admissible State for PDEI-systems [30])**.** Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_{E}, \mathcal{M}_{E}, \mathcal{C}_{I}, \mathcal{M}_{I} \rangle$ be a PDEI-system and B a state for $\mathcal{S}$. B is $\mathcal{S}$-admissible if

1. B is $\mathcal{S}$-consistent and
2. there exists a universal $\mathcal{S}^{I \to E}$-solution of B that is $\mathcal{S}^{E}$-admissible.

⊣

Based on this concept, the definitions of *universal $\mathcal{S}$-solution*, *$\mathcal{S}$-core* and *certain answers* for PDEI-systems are the same as for PDE-systems, except that they use the notions of $Sem(\mathcal{S}, B)$ and $\mathcal{S}$-admissible state for PDEI-systems instead of those for PDE-systems. As it was shown in [30], the $\mathcal{S}$-core of a PDEI-system need not be unique up to isomorphism, and is therefore not further considered in this thesis.

Example 5.21. Consider the PDEI-system $\mathcal{S}$ of Example 5.19 and the state B from Example 5.10 for the PDE-system considered there. Further recall that it differs only in two mappings from $\mathcal{S}$. Then B is obviously also a $\mathcal{S}$-consistent state for $\mathcal{S}$, but also no $\mathcal{S}$-admissible state.

B'' is an example for an $\mathcal{S}$-admissible state, and at the same time is a universal $\mathcal{S}$-solution for B:

$B' = \{\mathtt{Staff}(\text{'eve'},\text{'e.'},\text{'unkn.'}), \mathtt{Authorised}(\text{'0912345'},\text{'alice'},\text{'a.'}),$
$\mathtt{Authorised}(\text{'621005'}, \mathtt{FN}_2, \mathtt{LN}_2), \mathtt{Authorised}(\text{'9000'}, \mathtt{FN}_3, \mathtt{LN}_3),$
$\mathtt{Authorised}(\mathtt{SID}_2, \text{'eve'},\text{'e.'}),$
$\mathtt{Allowed}(\text{'621005'},\text{'bob'},\text{'bond'}), \mathtt{Allowed}(\mathtt{SID}_1, \text{'alice'}, \text{'a.'}),$
$\mathtt{Contact}(\text{'9000'},\text{'dave@hal.net'}), \mathtt{Contact}(\mathtt{SID}_1, \mathtt{Email}_1),$
$\mathtt{Contact}(\mathtt{SID}_3, \mathtt{Email}_2)\}$

When comparing B'' with B' from Example 5.10, one notices that the facts $\mathtt{Allowed}(\text{'9000'}, \mathtt{FN}_1, \mathtt{LN}_1)$ and $\mathtt{Allowed}(\mathtt{SID}_3, \text{'eve'}, \text{'e.'})$ of B' do not belong to B''. This is because they are already implied by the virtual dependencies: The first one is implied by dependency 6, and the second one is implied by dependency 5. ◇

Concerning query answering, it is clear that the result from Theorem 5.16 for computing the certain answers for PDE-systems is not valid for PDEI-systems, since in the latter an $\mathcal{S}$-admissible state does not contain any information about the data implied by i-mappings and i-constraints. Therefore the detailed discussion of query answering in PDEI-systems is deferred to a later section (see Section 5.6). Although query answering is only considered for a restricted class of PDEI-systems (as it is undecidable for general PDEI-systems), according to [30], even for this restricted class of PDEI-systems, given a PDEI-system $\mathcal{S}$ and some state B for $\mathcal{S}$, there exists no finite state B' such that computing

the certain answers to some query q over $\mathcal{S}$ can be done by evaluating q on B'. On the other hand, the next theorem states that when computing the certain answers to q over an $\mathcal{S}$-admissible state, the e-mappings and e-constraints have no effect on the query any more, but only the i-mappings and i-constraints need to be considered.

Theorem 5.22 ([30]). *Let $\mathcal{S}$ be a PDEI-system, q a query over $\mathcal{S}$ and B an $\mathcal{S}$-admissible state. Then* $\mathtt{certain}(q, \mathcal{S}, B) = \mathtt{certain}(q, \mathcal{S}^I, B)$.

PDEI-systems are a generalisation of PDE systems: Given a PDE-system $\mathcal{S}_E = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$, this system is equivalent to $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \emptyset, \emptyset \rangle$. It can be easily checked that for such PDEI-systems, all definitions for PDEI-systems coincide with the corresponding definitions on PDE-systems. To see that in this case also the definitions of $\mathcal{S}$-admissible states coincide, one should consider that if $\mathcal{C}_I = \mathcal{M}_I = \emptyset$, B is the only universal $\mathcal{S}^{I \to E}$-solution of B.

5.3.1 *Relationship with Data Integration*

As stated in Section 5.2.1, the data exchange setting (Section 3.1) occurs as special case of a PDE system. Because PDE-systems are special cases of PDEI-systems, data exchange settings occur as special case of PDEI-systems as well. Moreover, PDEI-systems also extend data integration systems (Section 3.2), that is, every data integration setting can be transformed into an equivalent PDEI-system.

Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system where constraints over $\mathcal{G}$ are allowed. Denote the set of constraints with Σ. The corresponding PDEI-system $\mathcal{S}_\mathcal{I} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ is defined as follows:

- $\mathcal{P} = \{P_\mathcal{S}, P_\mathcal{G}\}$,
- $S(P_\mathcal{S}) = \mathcal{S}$, $S(P_\mathcal{G}) = \mathcal{G}$,
- $\mathcal{C}_E = \mathcal{M}_E = \emptyset$,
- $\mathcal{C}_I = \Sigma$, and
- $\mathcal{M}_I = \mathcal{M}$.

Recall that $\mathtt{Mod}(\mathcal{I}, D)$ denotes the set of legal global databases to $\mathcal{I}$ with respect to a source instance D. Because the source instances for a data integration system are considered to be definite, the difference between the FOL semantics applied to data integration systems and the semantics of PDEI-systems does not show up. This is why the following theorem holds.

Theorem 5.23 ([30]). *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, and let $\mathcal{S}_\mathcal{I}$ be the corresponding PDEI-system. Then for each source instance D for $\mathcal{S}$,* $\mathtt{Mod}(\mathcal{I}, D) = \mathtt{Sem}(\mathcal{S}_\mathcal{I}, D)$.

Hence, the certain answers to a query q posed over $P_\mathcal{G}$ are the same as the certain answers to q over $\mathcal{G}$. Note that one could also model each source of the data integration system as an own peer.

5.4 THE E-CHASE

The different kinds of states considered above ($\mathcal{S}$-consistent, $\mathcal{S}$-admissible, universal $\mathcal{S}$-solution) give rise to several natural problems: Given a PDE-system $\mathcal{S}$ and a state B for $\mathcal{S}$:

- Is B $\mathcal{S}$-consistent?
- Is B $\mathcal{S}$-admissible?
- If B is $\mathcal{S}$-consistent, compute a universal $\mathcal{S}$-solution for B.

These questions can be solved by using a variant of the chase procedure, called E-CHASE.

Definition 5.24 (E-CHASE [30])**.** Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$ be a PDE-system and B a state for $\mathcal{S}$.
Consider the following chase rules:

TGD rule: A TGD ϕ: $\varphi(\vec{x}, \vec{z}) \rightarrow \psi(\vec{x}, \vec{y}) \in \mathcal{C}_E$ is *applicable* to B if there exists a tuple $t \in \varphi(\vec{x}, \vec{z})^B$ such that $t \notin \psi(\vec{x}, \vec{y})^B$.
If ϕ is applicable to B, ϕ is *applied* to B by adding facts corresponding to $\psi(t, \vec{Y})$ to B, where $\vec{Y}$ contains a fresh (that is not yet present in B) labelled null for every $y \in \vec{y}$.

EGD rule: An EGD ϕ: $\varphi(x_1, x_2) \rightarrow x_1 = x_2 \in \mathcal{C}_E$ is *applicable* to B if there exists a pair $(t_1, t_2) \in \varphi(x_1, x_2)^B$ where $t_1 \neq t_2$.
If ϕ is applicable to B, there are two possibilities how ϕ is *applied* to B:

- If at least one of t_1, t_2 is a labelled null (w.l.o.g. assume t_1), replace all occurrences of t_1 in B by t_2. (If both, t_1 and t_2 are labelled nulls, it does not matter which one is kept and which one is replaced.)
- If both, t_1 and t_2 are constants, set $B = \text{FAIL}$.

P2P-mapping rule: A TGD ϕ: $\varphi(\vec{x}, \vec{z}) \rightarrow \psi(\vec{x}, \vec{y}) \in \mathcal{M}_E$ is *applicable* to B if there exists a tuple $t \in \mathrm{Eval}_{\mathrm{Null}\downarrow}(\varphi(\vec{x}, \vec{z}), B)$ such that $t \notin \mathrm{Eval}_{\mathrm{Null}\downarrow}(\psi(\vec{x}, \vec{y}), B)$.
If ϕ is applicable to B, ϕ is *applied* to B by adding facts corresponding to $\psi(t, \vec{Y})$ to B, where $\vec{Y}$ contains a fresh (that is not yet present in B) labelled null for every $y \in \vec{y}$.

The corresponding *chase sequence* is a (possibly infinite) sequence $B_0, \ldots, B_i, B_{i+1}, \ldots$ where each B_{k+1} is obtained from B_k by applying either a TGD or EGD from $\mathcal{C}_E$ or a TGD from $\mathcal{M}_E$ to B_k, and $B_0 = B$. The transition from B_k to B_{k+1} via the application of a chase rule is called a *chase step*. A chase sequence is called finite if, after a finite number of steps, a fixpoint is reached, that is some instance B_f where no more TGD or EGD from $\mathcal{C}_E \cup \mathcal{M}_E$ is applicable, otherwise it is called infinite.

If, for some i, $B_i = \text{FAIL}$, then E-CHASE$(\mathcal{S}, B) = \text{FAIL}$.
If the chase sequence is finite, and $B_f \neq \text{FAIL}$, then E-CHASE$(\mathcal{S}, B) = B_f$.
If the chase sequence is infinite, then E-CHASE$(\mathcal{S}, B) = \bigcup_{i \in \mathbb{N}} B_i$. ⊣

Note that the TGD and EGD rule are the same as for the chase in data exchange (Definition 3.11), due to the fact that to the local constraints the FOL semantics is applied. Further, the P2P-mapping rule only exchanges ground tuples between peers, which meets exactly the intuition of the used semantics as described in Section 5.2. Moreover, as shown by Theorem 5.16 and the result stated next, this corresponds exactly to the peers exchanging certain answers. Note further, that in cases of infinite chase sequences, the definition of E-CHASE$(\mathcal{S}, B)$ describes an infinite state.

A more procedural description of the E-CHASE is given in Algorithm 2. Note that in the case of infinite chase sequences, the result of the algorithm differs from Definition 5.24, since it simply does not terminate.

Alg. 2 E-CHASE

Input: a PDE-system $\mathcal{S}$ and a state B for $\mathcal{S}$
Output: E-CHASE $(\mathcal{S}, B)$
$Ch_{\mathcal{S},B} \leftarrow \emptyset$; $Chnew_{\mathcal{S},B} \leftarrow B$
while $Ch_{\mathcal{S},B} \neq Chnew_{\mathcal{S},B}$ **do**
{
$Ch_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B}$;
for all $\phi \in \mathcal{C}_E \cup \mathcal{M}_E$ **do**
{
if $\phi = (\varphi(\vec{x},\vec{z}) \rightarrow \psi(\vec{x},\vec{y})) \in \mathcal{C}_E$
and $\vec{t} \in \varphi(\vec{x},\vec{z})^{Chnew_{\mathcal{S},B}}$
and $\vec{t} \notin \psi(\vec{x},\vec{y})^{Chnew_{\mathcal{S},B}}$
then
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B} \cup \{\psi(\vec{t},\vec{Y})\}$
if $\phi = (\varphi(x_1,x_2) \rightarrow x_1 = x_2) \in \mathcal{C}_E$
and $\langle t_1,t_2\rangle \in \varphi(x_1,x_2)^{Chnew_{\mathcal{S},B}}$
and $t_1 \neq t_2$
then
if $t_1 \in \mathcal{N}$ **then**
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B}[t_1 \leftarrow t_2]$;
else if $t_2 \in$ *const* **then return** FAIL
else
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B}[t_2 \leftarrow t_1]$;
if $\phi = (\varphi(\vec{x},\vec{z}) \rightarrow \psi(\vec{x},\vec{y})) \in \mathcal{M}_E$
and $\vec{t} \in Eval_{Null\downarrow}(\varphi(\vec{x},\vec{z}), Chnew_{\mathcal{S},B})$
and $\vec{t} \notin Eval_{Null\downarrow}(\psi(\vec{x},\vec{y}), Chnew_{\mathcal{S},B})$
then
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B} \cup \{\psi(\vec{t},\vec{Y})\}$
}
}
return $Chnew_{\mathcal{S},B}$

($\vec{Y}$ contains a fresh labelled null for each $y \in \vec{y}$)

The following result justifies that the E-CHASE is an appropriate tool for tackling the problems sketched above:

Theorem 5.25 ([30]). *Let* $\mathcal{S}$ *be a PDE-system and* $\mathbb{B}$ *a finite state for* $\mathcal{S}$*. Then*

- $\mathbb{B}$ *is* $\mathcal{S}$*-consistent iff* E-CHASE$(\mathcal{S}, \mathbb{B}) \neq$ *FAIL,*
- *if* E-CHASE$(\mathcal{S}, \mathbb{B}) \neq$ *FAIL, then* E-CHASE$(\mathcal{S}, \mathbb{B})$ *is a universal* $\mathcal{S}$*-solution of* $\mathbb{B}$*, and*
- $\mathbb{B}$ *is* $\mathcal{S}$*-admissible iff* E-CHASE$(\mathcal{S}, \mathbb{B}) = \mathbb{B}$.

Because of the definition of E-CHASE$(\mathcal{S}, \mathbb{B})$, if the chase sequence is infinite, then E-CHASE$(\mathcal{S}, \mathbb{B}) \neq$ FAIL. That is in this case $\mathbb{B}$ is regarded as a consistent state ($\mathsf{Sem}(\mathcal{S}, \mathbb{B})$ just contains no finite definite state). Moreover, in such a case, E-CHASE$(\mathcal{S}, \mathbb{B})$ denotes an infinite universal $\mathcal{S}$-solution for $\mathbb{B}$.

The following example sketches how the universal $\mathcal{S}$-solution of Example 5.10 can be retrieved using the chase.

Example 5.26. Consider the PDE-system and the states $\mathbb{B}$ and $\mathbb{B}'$ of Example 5.10. There exists a chase sequence s.t. E-CHASE$(\mathcal{S}, \mathbb{B}) = \mathbb{B}'$. For convenience, only the facts added in each iteration are listed below. The numbers identify the applied dependency:

1 $\mathtt{Allowed}(\mathsf{SID}_1, \text{'alice'}, \text{'a.'})$

4 $\mathtt{Contact}(\mathsf{SID}_1, \mathtt{Email}_1)$

5 $\mathtt{Allowed}(\text{'9000'}, \mathsf{FN}_1, \mathsf{LN}_1)$

2 $\mathtt{Authorised}(\text{'621005'}, \mathsf{FN}_2, \mathsf{LN}_2)$

2 $\mathtt{Authorised}(\text{'9000'}, \mathsf{FN}_3, \mathsf{LN}_3)$

3 $\mathtt{Authorised}(\mathsf{SID}_2, \text{'eve'}, \text{'e.'})$

1 $\mathtt{Allowed}(\mathsf{SID}_3, \text{'eve'}, \text{'e.'})$

4 $\mathtt{Contact}(\mathtt{Contact}(\mathsf{SID}_3, \mathtt{Email}_2)$

As can be easily verified, no more dependency is applicable. ◇

5.4.1 *Weakly Acyclic PDE-Systems*

Although Theorem 5.25 suggests that the E-CHASE might be a useful tool for reasoning over PDE-systems, it states nothing about the complexity of computing E-CHASE $(\mathcal{S}, \mathbb{B})$. It is not even guaranteed that the result of E-CHASE $(\mathcal{S}, \mathbb{B})$ is a finite state, that is whether the chase terminates.

In fact, because the results in Section 5.2.1, showing that PDE-systems generalise data exchange settings, are not restricted to data exchange settings where the set of TGDs in $\mathcal{M}$ is weakly acyclic, it follows immediately (from the discussion in Section 3.1.2) that there exist PDE-systems such that the chase sequence of the E-CHASE is infinite. Hence in the general setting, deciding

whether a state is $\mathcal{S}$-consistent, $\mathcal{S}$-admissible, or computing a universal $\mathcal{S}$-solution is undecidable.

Therefore, if one is interested only in PDE-systems where these problems are indeed decidable, the class of PDE-systems under consideration has to be restricted. As the problem arises from the cyclicity of the TGDs in $\mathcal{C}_E$, like in data exchange only weakly acyclic TGDs are allowed in $\mathcal{C}_E$.

Definition 5.27 (Weakly Acyclic PDE-System [30])**.** A PDE-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$ is a *weakly acyclic PDE-system* iff the TGDs in $\mathcal{C}_E$ form a weakly acyclic set of TGDs. ⊣

Note that the property of weak acyclicity can be guaranteed by each peer with only local information: if $\mathcal{C}_E$ of each peer is weakly acyclic, so is the complete PDE-system. Therefore every peer remains completely autonomous when setting up the mappings.

This definition of weakly acyclic PDE-systems still allows for not weakly acyclic sets of TGDs, for example $\mathcal{M}_E$ or $\mathcal{M}_E \cup \mathcal{C}_E$. This indeed is a problem when applying the classical FOL interpretation (like for example used in $\mathcal{PPL}$) on $\mathcal{M}_E$, which would require to impose global restrictions on the structure of a PDE-system. Because of applying the alternative semantics to the P2P mappings, hence requiring them only to be CERT-satisfied, these kinds of not weakly acyclic sets of TGDs do not cause undecidability: For weakly acyclic PDE-systems, all the considered problems are decidable and can even be solved efficiently.

Theorem 5.28. *[30] Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E \rangle$ be a weakly acyclic PDE-system and B be a finite state for $\mathcal{S}$. Then*

1. *Deciding whether B is $\mathcal{S}$-consistent can be done in polynomial time (data complexity).*
2. *Deciding whether B is $\mathcal{S}$-admissible can be done in polynomial time (data complexity).*
3. *If B is $\mathcal{S}$-consistent, then a universal $\mathcal{S}$-solution of B can be computed in polynomial time (data complexity).*
4. *If B is $\mathcal{S}$-admissible, then query answering can be solved in logarithmic space (data complexity).*

Proof sketch. (following [30]) (4) follows directly from Theorem 5.16 and the well known fact that answering conjunctive queries can be done in logarithmic space, together with the observation that dropping all tuples containing labelled nulls from the result (that is, deciding whether a tuple contains a labelled null) is obviously feasible in logarithmic space as well.

By Theorem 5.25, the proofs for (1) – (3) reduce to showing that for a weakly acyclic PDE-setting, computing E-CHASE$(\mathcal{S}, B)$ is feasible in polynomial time (data complexity). To prove this, it is necessary to recall that by Theorem 3.22, chasing a set of EGDs and weakly acyclic TGDs is feasible in polynomial time. Hence chasing $\mathcal{C}_E$ requires only polynomial time. Further, checking whether a TGD in $\mathcal{M}_E$ is applicable, and if, applying it can be also done obviously in

polynomial time. But chasing a TGD in $\mathcal{M}_E$ propagates only tuples containing values from $\mathit{const}(B)$. Therefore, denoting the arity of the tuples propagated by some TGD $\phi \in \mathcal{M}_E$ with n, every ϕ is at most $|\mathit{const}(B)|^n$ times applicable. Since the number of TGDs in $\mathcal{M}_E$ is considered as fixed, this only gives a polynomial number of applications of TGDs in $\mathcal{M}_E$. In the worst case, $\mathcal{C}_E$ is chased after each such application of an e-mapping, which still gives an overall polynomial running time with respect to data complexity. □

5.5 THE EI-CHASE

Given a PDEI-system $\mathcal{S}$ and a state B for $\mathcal{S}$, the same problems are of interest as for a PDE-system. But the E-CHASE is no longer suitable for computing the solutions for these problems, since it does not take into account the i-mappings and i-constraints. Although data implied directly by those mappings is not materialised, this data has to be considered for deciding whether an e-mapping or e-constraint is applicable or not.

Example 5.29. Consider a slight variation of the PDEI-system of Example 5.19, and assume a PDEI-system $\mathcal{S} = \langle \mathcal{P}, \emptyset, \emptyset, \mathcal{C}_I, \mathcal{M}_I \rangle$ where

- $\mathcal{P} = \{\mathrm{Uni}, \mathrm{Lib}_{UB}, \mathrm{Lib}_{Dep}\}$ with
  $S(\mathrm{Uni}) = \{\mathrm{Registered}(\mathrm{StudentId}, \mathrm{Name})\}$,
  $S(\mathrm{Lib}_{UB}) = \{\mathrm{Authorised}(\mathrm{StudentId}, \mathrm{Name})\}$,
  $S(\mathrm{Lib}_{Dep}) = \{\mathrm{Allowed}(\mathrm{StudentId}, \mathrm{Name})\}$,
- $\mathcal{M}_I = \{\mathrm{Registered}(\mathrm{SID}, \mathrm{N}) \rightarrow \mathrm{Authorised}(\mathrm{SID}, \mathrm{N})\}$,
- $\mathcal{M}_E = \{\mathrm{Authorised}(\mathrm{SID}, \mathrm{N}) \rightarrow \mathrm{Allowed}(\mathrm{SID}, \mathrm{N})\}$, and
- $\mathcal{C}_E = \mathcal{C}_I = \emptyset$.

Given a state $B = \mathrm{Registered}(\text{'0912345'}, \text{'alice a.'})$, a universal $\mathcal{S}$-solution of B should contain $\mathrm{Allowed}(\text{'0912345'}, \text{'alice a.'})$, but not $\mathrm{Authorised}(\text{'0912345'}, \text{'alice a.'})$.

Also, assuming a different PDEI-system $\mathcal{S}' = \langle \mathcal{P}', \mathcal{C}'_E, \mathcal{M}'_E, \mathcal{C}'_I, \mathcal{M}'_I \rangle$ where

- $\mathcal{P}' = \{\mathrm{Lib}_{UB}, \mathrm{Lib}_{Dep}\}$ with
  $S(\mathrm{Lib}_{UB}) = \{\mathrm{Authorised}(\mathrm{StudentId}, \mathrm{Name})\}$,
  $S(\mathrm{Lib}_{Dep}) = \{\mathrm{Allowed}(\mathrm{StudentId}, \mathrm{Name})\}$,
- $\mathcal{M}'_I = \{\mathrm{Authorised}(\mathrm{SID}, \mathrm{N}) \rightarrow \mathrm{Allowed}(\mathrm{SID}, \mathrm{N})\}$,
- $\mathcal{M}'_E = \{\mathrm{Authorised}(\mathrm{SID}, \mathrm{N}) \rightarrow \mathrm{Allowed}(\mathrm{SID}, \mathrm{N})\}$, and
- $\mathcal{C}'_E = \mathcal{C}'_I = \emptyset$.

and given a state $B' = \mathrm{Authorised}(\text{'0912345'}, \text{'alice a.'})$, a universal $\mathcal{S}$-solution of B' should not contain $\mathrm{Allowed}(\text{'0912345'}, \text{'alica a.'})$, since this fact can be derived by the i-mapping.

Obviously, the same holds with respect to the constraints in $\mathcal{C}_E$ and $\mathcal{C}_I$. ◇

The EI-CHASE is an extension of the E-CHASE that also considers the effect of virtual dependencies. This is achieved by taking into account beside the materialised data also the data implied by virtual dependencies when computing the answers to the queries $\varphi(\vec{x},\vec{z})$ and $\psi(\vec{x},\vec{y})$ of a TGD, resp. the query $\varphi(x_1,x_2)$ of an EGD. This is expressed by the notion of $certain_{Null}(q,\mathcal{S}',B)$, where $\mathcal{S}'$ is a PDEI-system of the form $\langle\mathcal{P},\emptyset,\emptyset,\mathcal{C}_I,\mathcal{M}_I\rangle$. Intuitively, $certain_{Null}(q,\mathcal{S}',B)$ computes the certain answers to q in $\mathcal{S}'$ and B when considering all labelled nulls in B as constants after the equalities implied by the EGDs in $\mathcal{C}_I$ have been materialised.

More formally, let $\mathcal{S}'$ be a PDEI-system of the form $\langle\mathcal{P},\emptyset,\emptyset,\mathcal{C}_I,\mathcal{M}_I\rangle$, q a conjunctive query with arity k and B a finite state for $\mathcal{S}'$. Further use σ to denote a function $\mathit{nulls}(B) \to \mathit{const}(B) \cup \mathit{nulls}(B)$, and let $\sigma(B)$ be the state obtained by applying σ to B. That is, σ substitutes the null values in B. Finally, let $\sigma_{\mathcal{S}'}$ be the most general of such substitutions such that $\sigma_{\mathcal{S}'}(B)$ is $\mathcal{S}'$-consistent when the remaining labelled nulls in $\sigma_{\mathcal{S}'}(B)$ are considered as constants[1]. Obviously, such a substitution $\sigma_{\mathcal{S}'}$ only exists if B is $\mathcal{S}'$-consistent. In this case, $certain_{Null}(q,\mathcal{S}',B)$ is defined as the set of certain answers of q to $\mathcal{S}'$ and $\sigma_{\mathcal{S}'}(B)$ when considering all labelled nulls in $\sigma_{\mathcal{S}'}(B)$ as constants. When B is not $\mathcal{S}'$-consistent, then $certain_{Null}(q,\mathcal{S}',B)$ is defined to equal Γ^k. (Recall from Section 2 that Γ denotes the domain of the PDEI-system.)

Definition 5.30 (EI-CHASE [30])**.** Let $\mathcal{S} = \langle\mathcal{P},\mathcal{C}_E,\mathcal{M}_E,\mathcal{C}_I,\mathcal{M}_I\rangle$ be a PDEI-system and B a state for $\mathcal{S}$.
Consider the following chase rules:

TGD rule: A TGD ϕ: $\varphi(\vec{x},\vec{z}) \to \psi(\vec{x},\vec{y}) \in \mathcal{C}_E$ is *applicable* to B if there exists a tuple $t \in certain_{Null}(\varphi(\vec{x},\vec{z}),\mathcal{S}^I,B)$ such that $t \notin certain_{Null}(\psi(\vec{x},\vec{y}),\mathcal{S}^I,B)$.
If ϕ is applicable to B, ϕ is *applied* to B by adding facts corresponding to $\psi(t,\vec{Y})$ to B, where $\vec{Y}$ contains a fresh (that is not yet present in B) labelled null for every $y \in \vec{y}$.

EGD rule : An EGD ϕ: $\varphi(x_1,x_2) \to x_1 = x_2 \in \mathcal{C}_E$ is *applicable* to B if there exists a pair $(t_1,t_2) \in certain_{Null}(\varphi(x_1,x_2),\mathcal{S}^I,B)$ where $t_1 \neq t_2$.
If ϕ is applicable to B, there are two possibilities how ϕ is *applied* to B:

- If at least one of t_1,t_2 is a labelled null (w.l.o.g. assume t_1), replace all occurrences of t_1 in B by t_2. (If both t_1 and t_2 are labelled nulls, it does not matter which one is kept and which one is replaced.)
- If both, t_1 and t_2 are constants, set $B = \text{FAIL}$.

P2P-mapping rule: A TGD ϕ: $\varphi(\vec{x},\vec{z}) \to \psi(\vec{x},\vec{y}) \in \mathcal{M}_E$ is *applicable* to B if there exists a tuple $t \in certain(\varphi(\vec{x},\vec{z}),\mathcal{S}^I,B)$ such that $t \notin certain(\psi(\vec{x},\vec{y}),\mathcal{S}^I,B)$.
If ϕ is applicable to B, ϕ is *applied* to B by adding facts corresponding to $\psi(t,\vec{Y})$ to B, where $\vec{Y}$ contains a fresh (that is not yet present in B) labelled null for every $y \in \vec{y}$.

1 We discuss some problems encountered with this definition in Section 6.2.2.

The corresponding *chase sequence* is a (possibly infinite) sequence $B_0, \ldots, B_i, B_{i+1}, \ldots$ where each B_{k+1} is obtained from B_k by applying either a TGD or EGD from $\mathcal{C}_E$ or a TGD from $\mathcal{M}_E$ to B_k, and $B_0 = B$. The transition from B_k to B_{k+1} via the application of a chase rule is called a *chase step*. A chase sequence is called finite if, after a finite number of steps, a fixpoint is reached, that is some instance B_f where no more TGD or EGD from $\mathcal{C}_E \cup \mathcal{M}_E$ is applicable. Otherwise it is called infinite.

If for some i $B_i = \text{FAIL}$, then EI-CHASE$(\mathcal{S}, B) = \text{FAIL}$.
If the chase sequence is finite, $B_f \neq \text{FAIL}$ and B_f is not $\mathcal{S}^I$-consistent, then EI-CHASE$(\mathcal{S}, B) = \text{FAIL}$.
If the chase sequence is finite, $B_f \neq \text{FAIL}$ and B_f is $\mathcal{S}^I$-consistent, then EI-CHASE$(\mathcal{S}, B) = B_f$.
If the chase sequence is infinite and $\bigcup_{i\in\mathbb{N}} B_i$ is not $\mathcal{S}^I$ consistent, then EI-CHASE$(\mathcal{S}, B) = \text{FAIL}$.
If the chase sequence is infinite and $\bigcup_{i\in\mathbb{N}} B_i$ is $\mathcal{S}^I$ consistent, then EI-CHASE$(\mathcal{S}, B) = \bigcup_{i\in\mathbb{N}} B_i$. ⊣

Some remarks concerning the above definition: Note that the FOL based semantics is applied again to the peers, just as in E-CHASE. Moreover, compared to the E-CHASE, the use of $\mathtt{Eval}_{\mathtt{Null}}$ takes the virtual mappings into account. Just as described in the intuition of the alternative semantics for P2P mappings, what is exchanged by the rule for the P2P mapping are certain answers. Also just like for the E-CHASE, in case of an infinite chase sequence, EI-CHASE$(\mathcal{S}, B)$ describes an infinite state (as long as this state is $\mathcal{S}^I$-consistent). While the definition of the E-CHASE, using the notions of q^B and $\mathtt{Eval}_{\mathtt{Null}\downarrow}(q, B)$, gives immediately rise to a possible implementation, the case is not as obvious for $\mathtt{certain}_{\mathtt{Null}}(q, \mathcal{S}^I, B)$ and $\mathtt{certain}(q, \mathcal{S}^I, B)$. The discussion of one possible way to implement these notions is deferred to the next chapter (see Section 6.3.4), where also a possible way to check a state for $\mathcal{S}^I$-consistency is described.

A more procedural description of the EI-CHASE is given in Algorithm 3. Note that because a concrete algorithm cannot handle infinite instances, in the case of an infinite chase sequence the result returned by the algorithm differs from Definition 5.30. In fact, the corresponding program will not terminate. The chase sequence stated in Example 5.26 is almost a chase sequence of an EI-CHASE computing the universal $\mathcal{S}$-solution of B (from the same example) w.r.t. the PDEI-system defined in Example 5.21. Only the two facts implied by the virtual mappings have to be removed.

Just as for the E-CHASE, it can be also shown for the EI-CHASE that it is a good choice for reasoning over PDEI-systems:

Theorem 5.31 ([30]). *Let $\mathcal{S}$ be a PDEI-system and B a finite state for $\mathcal{S}$. Then*

- *B is $\mathcal{S}$-consistent iff* EI-CHASE$(\mathcal{S}, B) \neq$ *FAIL.*
- *If* EI-CHASE$(\mathcal{S}, B) \neq$ *FAIL, then* EI-CHASE$(\mathcal{S}, B)$ *is a universal $\mathcal{S}$-solution of B.*
- *B is $\mathcal{S}$-admissible iff* EI-CHASE$(\mathcal{S}, B) = B$.

Alg. 3 EI-CHASE

Input: a PDEI-system $\mathcal{S}$ and a state B for $\mathcal{S}$
Output: EI-CHASE $(\mathcal{S}, B)$
$Ch_{\mathcal{S},B} \leftarrow \emptyset$; $Chnew_{\mathcal{S},B} \leftarrow B$
while $Ch_{\mathcal{S},B} \neq Chnew_{\mathcal{S},B}$ **do**
{
$Ch_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B}$;
for all $\phi \in \mathcal{C}_E \cup \mathcal{M}_E$ **do**
{
if $\phi = (\varphi(\vec{x},\vec{z}) \rightarrow \psi(\vec{x},\vec{y})) \in \mathcal{C}_E$
and $\vec{t} \in certain_{Null}(\varphi(\vec{x},\vec{z}), \mathcal{S}^I, Chnew_{\mathcal{S},B})$
and $\vec{t} \notin certain_{Null}(\psi(\vec{x},\vec{y}), \mathcal{S}^I, Chnew_{\mathcal{S},B})$
then
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B} \cup \{\psi(\vec{t},\vec{Y})\}$
if $\phi = (\varphi(x_1,x_2) \rightarrow x_1 = x_2) \in \mathcal{C}_E$
and $\langle t_1, t_2\rangle \in certain_{Null}(\varphi(x_1,x_2), \mathcal{S}^I, Chnew_{\mathcal{S},B})$
and $t_1 \neq t_2$
then
if $t_1 \in \mathcal{N}$ **then**
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B}[t_1 \leftarrow t_2]$;
else if $t_2 \in$ *const* **then return** FAIL
else
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B}[t_2 \leftarrow t_1]$;
if $\phi = (\varphi(\vec{x},\vec{z}) \rightarrow \psi(\vec{x},\vec{y})) \in \mathcal{M}_E$
and $\vec{t} \in certain(\varphi(\vec{x},\vec{z}), \mathcal{S}^I, Chnew_{\mathcal{S},B})$
and $\vec{t} \notin certain(\psi(\vec{x},\vec{y}), \mathcal{S}^I, Chnew_{\mathcal{S},B})$
then
$Chnew_{\mathcal{S},B} \leftarrow Chnew_{\mathcal{S},B} \cup \{\psi(\vec{t},\vec{Y})\}$
}
}
if $Chnew_{\mathcal{S},B}$ is $\mathcal{S}^I$-consistent **then return** $Chnew_{\mathcal{S},B}$
return FAIL

($\vec{Y}$ contains a fresh labelled null for each $y \in \vec{y}$)

The same remarks as for the corresponding results for the E-CHASE apply here: If the chase sequence is infinite and the resulting infinite state is $\mathcal{S}^I$-consistent, then the resulting state is considered to be $\mathcal{S}$-consistent (i.e. EI-CHASE $(\mathcal{S}, B) \neq$ FAIL). Moreover in such a case EI-CHASE $(\mathcal{S}, B)$ denotes an infinite universal $\mathcal{S}$-solution.

5.5.1 *Stratified PDEI-Systems*

As it was the case for PDE-systems, also for general PDEI-systems the decision problem whether a given state B is $\mathcal{S}$-consistent is undecidable. Therefore, if

one is interested only in settings where this problem can be decided (and also in such settings where the $\mathcal{S}$-consistency of B implies the existence of a finite universal $\mathcal{S}$-solution), suitable limitations must be applied to the allowed sets of TGDs. Clearly, for the TGDs in $\mathcal{C}_E$ the same restriction as in PDE-systems is applied, that is they are restricted to a weakly acyclic set of TGDs. The same restriction could be applied to the TGDs in $\mathcal{C}_I$. But as mentioned in Section 3.2, in data integration most of the time not weakly acyclic TGDs have been considered as constraints over the global schema, but instead sets of key constraints and inclusion dependencies (which in turn can be expressed as EGDs and TGDs).

Therefore, the constraints allowed in $\mathcal{C}_I$ are also restricted in terms of key constraints and inclusion dependencies: The EGDs in $\mathcal{C}_I$ are restricted to legal key constraints, and the TGDs in $\mathcal{C}_I$ are restricted to foreign key dependencies with respect to the key constraints in $\mathcal{C}_I$.

Thereby [30] gives no explanation why PDEI-systems are restricted to foreign key dependencies only. In Section 3.2.4 (Theorem 3.50), results were summarised that state that the most expressive class of key constraints and inclusion dependencies that allow for efficient query answering is the class of NKCIDs. However, this result holds only when considering unrestricted database instances. When allowing for finite database instances only, query answering over NKCIDs becomes undecidable. Nevertheless we think that query answering over PDEI-systems that allow for NKCIDs still falls into the decidable case. Hence the use of NKCIDs would still allow for efficient query answering in PDEI-systems.

However, it can be immediately (see Section 3.3) verified that those two approaches — weakly acyclic TGDs and EGDs on the one hand, legal key constraints and foreign key dependencies on the other hand — are incomparable with respect to their expressive power.

But just restricting $\mathcal{C}_E$ and $\mathcal{C}_I$ separately is not enough to guarantee decidability. Example 5.32 shows such a case. It can be easily verified that there exists no finite universal solution for this setting, hence from Theorem 5.31 the undecidability of checking a state for $\mathcal{S}$-consistency follows. Intuitively, the problem arises because there is still data materialised according to TGDs that belong to a not weakly acyclic set of TGDs.

Example 5.32 ([30])**.** Consider a PDEI-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ where

- $\mathcal{P} = \{\mathcal{P}_1\}$ with $S(\mathcal{P}_1) = \{E(X), R(X,Y), F(X), Q(X,Y)\}$,
- $\mathcal{C}_E = \{E(x) \rightarrow R(x,y)\}$,
- $\mathcal{C}_I = \{R[2] \subseteq F[1]; F[1] \subseteq Q[2]; Q[1] \subseteq E[1]\}$,
- $\mathcal{M}_E = \mathcal{M}_I = \emptyset$,

and a state $B = \{E('a')\}$. Then it can be easily verified that there exists no finite universal $\mathcal{S}$-solution for B.

Note that in the dependency graph of $\mathcal{S}$, there exists a cycle containing special edges, namely $E_1 \xrightarrow{*} R_2 \rightarrow F_1 \xrightarrow{*} Q_1 \rightarrow E_1$. From the fact that the

mappings in $\mathcal{C}_E$ and $\mathcal{C}_I$ are interpreted under the classical FOL semantics, undecidability follows immediately. ◇

Although the TGDs in $\mathcal{C}_E$ are weakly acyclic, this does not hold any more when combining them with the TGDs in $\mathcal{C}_I$. To avoid such situations, also the combinations of TGDs from $\mathcal{C}_E$ with TGDs from $\mathcal{C}_I$ need to be restricted:

Definition 5.33 (Stratified PDEI-System [30])**.** A PDEI-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ is a *stratified PDEI-system*, if the TGDs in $\mathcal{C}_E$ form a weakly acyclic set, $\mathcal{C}_I$ corresponds to a set of legal key constraints and foreign key dependencies, and further no head of a foreign key dependency in $\mathcal{C}_I$ appears on the left hand side of any TGD in $\mathcal{C}_E$. ⊣

Note that $\mathcal{C}_I$ may still contain a set of acyclic TGDs. But now no position for which data is materialised belongs to a cycle in the dependency graph (Definition 3.16) containing a special edge. The following result from [30] shows that this is indeed sufficient to guarantee the existence of a finite universal $\mathcal{S}$-solution.

Theorem 5.34 ([30])**.** *Let $\mathcal{S}$ be a stratified PDEI-system and B a finite $\mathcal{S}$-consistent state. Then there exists a finite universal $\mathcal{S}$-solution of B.*

Moreover, such a universal $\mathcal{S}$-solution can be computed efficiently, just as checking whether a state is $\mathcal{S}$-consistent or $\mathcal{S}$-admissible. This is based on the fact that for stratified $\mathcal{S}$-systems EI-CHASE$(\mathcal{S}, B)$ can be computed in polynomial time (data complexity).

Theorem 5.35 ([30])**.** *Let $\mathcal{S}$ be a stratified PDEI-system and B a finite state for $\mathcal{S}$. Then*

1. *whether B is $\mathcal{S}$-consistent can be decided in polynomial time (data complexity),*
2. *whether B is $\mathcal{S}$-admissible can be decided in polynomial time (data complexity), and*
3. *if B is $\mathcal{S}$-consistent, then a universal $\mathcal{S}$-solution can be computed in polynomial time (data complexity).*

Under the assumption that $\mathtt{certain}_{Null}$ and $\mathtt{certain}$ for a PDEI-system can be computed efficiently, the same arguments as for the corresponding results for the E-CHASE prove the correctness of this theorem. An efficient algorithm for computing the certain answers over a stratified PDEI-system is presented in the next section. For an extended notion of stratified PDEI-systems, a method for efficiently computing $\mathtt{certain}_{Null}$ is presented in the next chapter (Section 6.3.4).

5.6 QUERY ANSWERING IN STRATIFIED PDEI-SYSTEMS

Although for $\mathcal{S}$-consistent stratified PDEI-systems there always exists a finite universal $\mathcal{S}$-solution, [30] states that it can be shown that there exists no finite state B' such that computing the certain answers to a query q with respect

to $\mathcal{S}$ and B can be done by simply evaluating q over B'. Therefore other methods for query answering are needed. By the result of Theorem 5.22, these methods need only take into account the dependencies and constraints from $\mathcal{M}_I$ and $\mathcal{C}_I$, but need not consider $\mathcal{M}_E$ and $\mathcal{C}_E$. Hence the problem corresponds to the problem of query answering in GLAV data integration systems with constraints on the global schema.

In Section 3.2.4, the notion of a perfect rewriting of a query has been presented and the rewriting algorithm presented in [12] is mentioned. In a GAV data integration system that allows non-key conflicting inclusion dependencies and key constraints over the global schema $\mathcal{G}$, given a UCQ q over the target schema, this algorithm computes a perfect rewriting of q. This means it returns a UCQ q', such that evaluating q' over the view extensions of $\mathcal{G}$ returns the certain answers to q.

Since a GAV data integration system with inclusion dependencies over the global schema is expressive enough to model a GLAV data integration system (see Section 3.2.3), this rewriting algorithm can be also used for query answering in PDEI-systems.

Before the application of this algorithm to PDEI-systems is described in detail, the idea is roughly sketched: The schema of each peer can be considered as global schema in a GLAV data integration system with foreign key dependencies and key constraints allowed over this schema. Thereby $\mathcal{C}_I$ corresponds to the constraints on the global schema, and the relations at the peer can be regarded as the view extensions. A UCQ q can then be rewritten into a query q' that, evaluated over the peer relations, returns the certain answers to q with respect to the peer instance. The data implied by the TGDs $q_i \rightarrow q_j$ in $\mathcal{M}_I$ (assume q_i over peer P_i and q_j over peer P_j) is taken into account by simply materialising this data in the relations of P_j. To include consideration of the data implied by $\mathcal{C}_I$ when evaluating the TGDs in $\mathcal{M}_I$, the queries on the left hand side of the P2P mappings are also rewritten.

5.6.1 *Rewriting Conjunctive Queries under Inclusion Dependencies*

As mentioned above, in [12] a rewriting algorithm is proposed for a GAV data integration system with key constraints (KDs) and non key conflicting inclusion dependencies (NKCIDs) allowed over the global schema. Given a UCQ q over the global schema, q is transformed into a query q' according to this algorithm such that evaluating q' on the view extension of the global schema returns the certain answers to q.

This algorithm utilises the separation property (Theorem 3.51), that roughly states that given an instance D of a schema $\mathbf{R}$ with a set Σ_k of KDs and a set Σ_I of NKCIDs over $\mathbf{R}$, then if D is consistent with Σ_k, no data implied from D by Σ_I violates any key constraint in Σ_k. This observation allows to split the rewriting algorithm into two parts, one that rewrites the query q according only to the NKCIDs, working completely independently from the KDs (even working without any KDs), and another one that "checks" whether the instance D on which query answering shall be performed is consistent with the KDs (in [12], the instance D corresponds to the view extensions,

while in a PDEI-system $\mathcal{S}$, the instance D is a state for $\mathcal{S}$). This second part adds additional subqueries to the result of the first part of the algorithm that ensure that the result of the query contains every possible n-tuple (where n is the arity of q) over the values present in D if D is not consistent with the KDs.

For query answering in PDEI-systems, only the rewriting under NKCIDs only (that is without taking care of the KDs) is of interest. This is because in [12], a GAV setting is assumed, hence the view extensions correspond to a definite instance D of $\mathcal{G}$. As a result, there is no need to check for unifications of labelled nulls to satisfy the KDs (since D contains none): either D satisfies Σ_k or not. But for PDEI-systems also unification of labelled nulls with other values needs to be considered to determine whether a state is consistent. Therefore checking the consistency of the current state is done separately. Consequently, here only query rewriting under inclusion dependencies (IDs) is considered.

The basic idea of the rewriting algorithm is, given some UCQ q, to regard inclusion dependencies $R_i[A_1] \subseteq R_i[A_2]$ as rewriting rules for adding new subqueries to q: If there exists some subquery q_i of q with a goal g_i corresponding to the right hand side of some ID, then add a new subquery q_j to q, created from q_i by replacing g_i with the left hand side of the ID. The following example shall give a better intuition of this idea:

Example 5.36. To concentrate on the rewriting algorithm, this and the next example are expressed over a single schema instead of a complete PDEI-system.

Assume the same setting as in Example 3.46 and consider the schema $\mathcal{G}$ = {Books(ISBN, Title, Author, Id, Status), OldBooks(ISBN, Title, Author, Id, Status)} with a key constraint key(Books) = {Id} and a foreign key dependency ϕ: OldBooks[Id] $\subseteq$ Books[Id] defined over $\mathcal{G}$. Hence in every instance D′ satisfying ϕ, all ids[2] stored in the OldBooks relations need to occur in the Books relation as well.

Denote with D some instance of $\mathcal{G}$ that not necessarily satisfies ϕ, and let q = {Id | Books(ISBN, Title, Author, Id, Status)} whose certain answers contain those ids shared by all instances of the Books schema that satisfy ϕ with respect to D.

Therefore, the answer to q does not only contain those ids contained in the Books relation in D, but also those from the OldBooks relation.

To consider this in the query evaluated over D, the idea is to use ϕ as a rewriting rule and to add the subquery {Id | OldBooks(ISBN, Title, Author, Id, Status)} to q. q therefore becomes a UCQ with those two subqueries. ◇

To give a formal description of this algorithm, it is first necessary to define when a goal of a query and the right hand side of an ID correspond, and how to derive a new goal from such a correspondence. In [12] this correspondence between a subgoal g (which in case of conjunctive queries is an atom) and an ID I is referred to as I being *applicable* to g.

2 Note that while we use "IDs" as abbreviation for "inclusion dependencies", "ids" denotes the values of the Id attribute of relations.

But first some properties of variables in conjunctive queries need to be fixed: Let q be a conjunctive query. A variable x occurring in q is *bound* if it occurs at least twice in q, otherwise, that is if it occurs only once, it is *unbound*. All variables in the head of q must be bound, since they must occur at least once in the body of q. A *bound term* is either a bound variable or a constant. To be able to denote unbound variables more easily, all unbound variables of a query are referred to as ξ in the following. For the description of the rewriting, inclusion dependencies are represented as $r[i_1, \ldots, i_h] \subset s[j_i, \ldots, j_h]$ where i_k and j_k are no attributes, but they denote the positions of attributes in r resp. s.

Definition 5.37 ([12])**.** Let $g = s(x_1, \ldots, x_l)$ be an atom and $I = r[i_1, \ldots, i_h] \subset s[j_i, \ldots, j_h]$ an inclusion dependency. Then I is *applicable* to g if $\forall l\ (1 \leqslant l \leqslant n)$: if $x_l \neq \xi$, then $\exists h$ s.t. $j_h = l$. ⊣

The intuition of this rule is that if the value at some position in g is of interest for the query (i.e. it is $\neq \xi$), then also those values that are propagated by an inclusion dependency to this position must be considered. This is, an ID I is applicable to an atom g if the position of every bound term in g appears in the rhs of I. Moreover, this implies that for every bound term there exists a correspondence on the lhs of I. More precisely, this correspondence means that I maps data from r to each bound term. And obviously the relational symbols on the rhs of I and of g must be equal.

The result of applying I to g is denoted with $gr(g, I)$ and is defined as follows[3]:

Definition 5.38 ([12])**.** Let g and I be as above, and assume that I is applicable to g. Then $gr(g, I)$ is defined as $gr(g, I) = r(y_1, \ldots, y_m)$ (where m is the arity of r), where for each $\forall l(1 \leqslant l \leqslant m)$: $y_l = \begin{cases} x_{j_h} & \text{if } \exists h \text{ s.t. } i_h = l \\ \xi & \text{otherwise} \end{cases}$ ⊣

But only adding new subqueries by replacing goals according to the above rewriting rule would not find all solutions, as the following example shows:

Example 5.39. Consider the following schema $S = \{a/3, b/2, c/2\}$ with the two inclusion dependencies $c[1, 2] \subseteq a[1, 3]$ and $a[2, 3] \subseteq b[1, 2]$. (Assuming according key constraints.)

Given the query $q = \{(A, B) \mid a(A, B, \_) \wedge b(B, C)\}$, rewriting $b(B, C)$ adds the subgoal $\{(A, C) \mid a(A, B, \_) \wedge a(\_, B, C)\}$. Now no ID is applicable any more. But the resulting query is no perfect rewriting, since it completely misses the content of c. But what can be done is adding a subquery $\{(A, C) \mid a(A, B, C)\}$, since the result of this query is already included in the result of the above two subgoals. However, in this query B is no longer a bound term, hence it can

3 In fact, in [12] the definition of $gr(g, I)$ is "the atom $s(Y_1, \ldots, Y_m)$ (m is the arity of s [..]" ([12], page 3). We assume that this is a typing error, since this definition does not only contradict the intuition of "a rewriting rule whose direction is right-to-left", but simply returns wrong results: Assume e.g. the relation symbols $a/1, b/1$, $I = a[1] \subseteq b[1]$, and $q = \{A \mid b(A)\}$. Applying the original definition from [12] to $b(A)$ would return $b(A)$. In the following, this would return $\{A \mid b(A)\}$ as the perfect rewriting of q which is obviously incorrect.

be rewritten using the second ID, giving the new subquery $\{(A, C) \mid c(A, C)\}$. Altogether, these three subgoals give a correct perfect rewriting $q' = \{(A, B) \mid a(A, B, \_) \wedge b(B, C)\} \cup \{(A, C) \mid a(A, B, \_) \wedge a(\_, B, C)\} \cup \{(A, C) \mid c(A, C)\}$. $\diamond$

The idea used in the example above is to identify certain restricted cases of the subqueries, to make them explicit and then to check whether this gives rise to some further rewriting. Formally, this is done by trying to unify two subgoals within the body of the same subquery.

Definition 5.40 ([12])**.** Given two atoms with the same relational symbol, $g_1 = s(x_1, \ldots, x_n)$ and $g_2 = s(y_1, \ldots, y_n)$, g_1 and g_2 *unify* if for each i ($1 \leqslant i \leqslant n$) either $x_i = y_i$, or $x_i = \xi$ or $y_i = \xi$.
If g_1 and g_2 unify, then $U(g_1, g_2) = s(z_1, \ldots, z_n)$ where (for $1 \leqslant i \leqslant n$)

$$z_i = \begin{cases} x_i & \text{if } x_i \neq \xi \text{ (i.e. either } x_i = y_i \text{ or } y_i = \xi) \\ y_i & \text{if } x_i = \xi \end{cases}$$

$\dashv$

With these notions defined, the algorithm *rewrite* (see Algorithm 4) performs as follows: Given a UCQ q, the following two rules are applied as long as they create new subqueries. Once they have no more effect, the union of all subqueries (those created and those given as input) is a perfect rewriting of q. According to the above definitions, the two rules either unify two atoms or replace an atom according to an ID:

UNIFICATION If there exists a subquery q' of q that contains two atoms g_1 and g_2 in its body that unify, create a new subquery (denoted with $\mathit{reduce}(q, g_1, g_2)$ in Algorithm 4) from q' by removing both, g_1 and g_2 from the body and adding $U(g_1, g_2)$ instead. Finally[4] all variables in $\mathit{reduce}(q, g_1, g_2)$ that appear only once after replacing g_1 and g_2 but have been bound terms before are marked as unbound terms (that is, they are replaced by ξ. This is expressed by the function τ in Algorithm 4), and the new subquery is added to q.

ID APPLICATION If there exists a subquery q' of q that contains a goal g in its body, such that some inclusion dependency ϕ is applicable to g, then a new subquery (denoted by $q'[g/gr(g, I)]$ in Algorithm 4) is added to q that is obtained from q' by replacing g with $gr(g, I)$. The new subquery is then added to q.

Termination of the algorithm follows from the following facts:

- The maximal number of goals in the body of each subquery is fixed and equal to the maximal number of goals in the body of any subquery of the input query q.
- The number of relational symbols is fixed, since only relational symbols from q and Σ_I can be used.

4 [12] states that first "the substitution obtained in the computation of $U(g_1, g_2)$" shall be applied to the complete new subquery. However, this substitution either replaces a bound term by itself ($x_i = y_i$), or an unbound term by something else. But by definition, each unbound term appears only once in the whole query, hence no other element is affected by those substitutions. Therefore this step is omitted here.

Alg. 4 $rewrite(q, I)$ — compute a perfect reformulation [12]

Input: a relational schema **R**, a set of inclusion dependencies Σ_I, an UCQ q
Output: a perfect rewriting of q w.r.t. Σ_I $\bar{q} \leftarrow q$
repeat
  $q_l \leftarrow \bar{q}$
  for all $q' \in q_l$ **do**
    //unification
    for all $g_1, g_2 \in body(q')$ **do**
      if g_1 and g_2 unify **then**
        $\bar{q} \leftarrow \bar{q} \cup \{\tau(reduce(q', g_1, g_2))\}$
    //ID application
    for all $g \in body(q')$ **do**
      for all $\phi \in \Sigma_I$ **do**
        if ϕ is applicable to g **then**
          $\bar{q} \leftarrow \bar{q} \cup \{q'[g/gr(g, \phi)]\}$
until $q_l = \bar{q}$ **return** $\bar{q}$

- The number of bound terms is finite and restricted to those bound terms from q, and all unbound terms are marked with the same symbol, namely ξ.

Therefore the number of atoms that can be created by the algorithm is finite, which implies that the number of subqueries created is finite. Therefore the algorithm terminates. Moreover, the resulting query is a perfect rewriting of the query q given as input.

Theorem 5.41. *Let **R** be a relational schema and Σ_I be a set of inclusion dependencies over **R**. Given a UCQ q over **R**, $rewrite(q, \Sigma_I)$ is a perfect rewriting of q with respect to Σ_I.*

Proof. The result follows immediately from Theorem 3.3 in [12]. □

5.6.2 *Computing the Certain Answers*

So far, the following is known for computing the certain answers to a query q over a PDEI-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ and a state D for $\mathcal{S}$: If D is $\mathcal{S}$-admissible, then $\mathcal{C}_E$ and $\mathcal{M}_E$ have no effect on the result and can be neglected. That is, query answering reduces to computing the certain answers to q over $\mathcal{S}^I$. Moreover a method for query answering under inclusion dependencies over a consistent instance was shown in the last section.

Putting all this together, it remains to clarify how to take the mappings in $\mathcal{M}_I$ into account, and how to ensure that the current state is consistent with respect to the key constraints in $\mathcal{C}_I$.

The idea is the following: Given an $\mathcal{S}$-admissible state D, a state D' is materialised which is obtained from D by materialising all the data implied by the mappings in $\mathcal{M}_I$. Moreover, the key constraints in $\mathcal{C}_I$ are enforced over this state. Therefore the result is either a state that is consistent with respect

to the key constraints and contains all data derivable by the mappings in $\mathcal{M}_I$, or one gets the information that the state is inconsistent (and query answering is therefore trivial). If D' is consistent, then it satisfies all prerequisites such that computing the certain answers to a given query q reduces to evaluating $rewrite(q, \mathcal{I})$ (where $\mathcal{I}$ denotes the set of foreign key dependencies in $\mathcal{C}_I$) over D'.

More formally: Denote with $\mathcal{I}$ a set of inclusion dependencies and use $\mathcal{J}$ to denote a set of key constraints. If ϕ is a TGD of the form $q_i \rightarrow q_j$, then the rewriting of ϕ is defined as $Expand(\phi, \mathcal{I}) = \{q' \rightarrow q_j \mid q' \in rewrite(q_i, \mathcal{I})\}$.

For a set $\mathcal{T}$ of TGDs, the result of rewriting is defined as

$$Expand(\mathcal{T}, \mathcal{I}) = \bigcup_{\phi \in \mathcal{T}} Expand(\phi, \mathcal{I}) \ .$$

Since the goal is to materialise all data implied by the i-mappings, the queries on the lhs of the TGDs must also take into account the data implied by the inclusion dependencies, and not only the data materialised in the current state. This is the purpose of $Expand$.

Definition 5.42 ([30])**.** Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ be a PDEI-system, and let $\mathcal{I}$ denote the set of inclusion dependencies in $\mathcal{C}_I$. Then let $\tau(\mathcal{S})$ be the following PDE-system[5]:

$$\tau(\mathcal{S}) = \langle \mathcal{P}, \emptyset, Expand(\mathcal{M}_I, \mathcal{I}) \rangle \ .$$

⊣

With all these notions introduced, computing the certain answers over a PDEI-system is characterised by the following result:

Theorem 5.43 ([30])**.** *Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ be a stratified PDEI-system, B a finite $\mathcal{S}$-admissible state and q a query over $\mathcal{S}$. Denote with $\mathcal{I}$ the set of inclusion dependencies in $\mathcal{C}_I$. Then*
$certain(q, \mathcal{S}, B) = Eval_{Null\downarrow}(rewrite(q, \mathcal{I}), \text{E-CHASE}(\tau(\mathcal{S}), B)) \ .$

Concerning the complexity of query answering, it is first observed that $rewrite(q, \mathcal{I})$ runs obviously in polynomial time (data complexity), since it is in fact independent of the data in the input instance. Then, by the complexity results for E-CHASE (see Theorem 5.28) and the well known complexity of evaluating UCQs over an instance, the complexity of query answering follows immediately.

Theorem 5.44 ([30])**.** *Let $\mathcal{S}$-be a stratified PDEI-system and B a finite $\mathcal{S}$-admissible state. Then query answering can be solved in polynomial time (data complexity).*

5 We discuss this definition later in Section 6.2.1 since we think it is not completely correct (w.r.t. to Theorem 5.43).

6 IMPLEMENTATION

We created a prototype implementation of a Peer Data Management System based on the semantical framework proposed in [30] and summarised in Chapter 5. There have been several reasons for the implementation of this particular framework:

- Since the framework generalises both, data exchange and virtual data integration, it provides greater flexibility than PDMSs that support either only data exchange or data integration. It allows the user to decide at configuration time whether to model a dependency as a data exchange or data integration mapping.
- Data exchange and data integration dependencies are modelled by schema mappings using TGDs and EGDs. These very powerful formalisms are easy to use and offer a well defined, extensively studied and well understood semantics.
- For reasonably restricted settings, many of the important reasoning tasks are solvable in polynomial time (data complexity): Checking a state for consistency, computing universal solutions (what could be referred to as computing the solution to the data exchange problem) and computing the certain answers of a query with respect to the virtual mappings (what could be referred to as computing the solution to the data integration problem).
- The restrictions imposed to obtain tractability seem feasible for practical use. Most important, these restrictions apply only to the local dependencies of a peer, and are independent of the overall P2P network induced by the P2P mappings (e.g., there is no restriction like acyclicity required on the topology of the network). Therefore, to guarantee the restrictions to hold for the overall system, it suffices that each peer satisfies these constraints locally. Hence the compliance with the restrictions can be checked locally by each peer without the need of global information. Moreover, the use of global information would not give additional expressive power.
- To the best of our knowledge, this approach has not been implemented yet. We are interested to which extend the good theoretical properties (polynomial time algorithms for many reasoning tasks) also hold in "practice".
- The framework of De Giacomo et al. seems to be a promising basis for further research planned at the department. Having a prototype that can be used for proof of concept implementations and evaluation of new ideas and extensions seems eligible.

Our implementation is based on CoDE ("Core computation in Data Exchange"), a prototype system for data exchange and core computation presented in [71]. Each peer is built on top of a traditional DBMS that is used to store and access the data residing at that peer efficiently and with only very little additional programming effort. Moreover, CoDE provides an implementation of the chase steps as used in data exchange (see Definitions 3.11 and 3.12) that can be used either directly or with some modifications for the implementation of the E-CHASE and EI-CHASE. This allows us to focus more on the coordination of the peers and the problems introduced by the addition of (virtual) data integration mappings. We tested our implementation using HSQLDB[1] as database back-end because it very easily allows to set up and run several database instances, which enables us without much effort to provide each peer with its own database. However, the system can be adopted easily to run with other DBMSs as well (CoDE already provides connectors to Oracle and PostgreSQL, but they have not yet been tested for our PDMS implementation).

In the next section (Section 6.1), a short overview of the implementation is given. Section 6.2 discusses some problems detected with the definitions presented in the last chapter. Afterwards we describe our main design decisions in Section 6.3 and stae some implementation details in Section 6.4. We conclude this chapter with a discussion of our implementation in Section 6.5.

6.1 GENERAL SYSTEM ARCHITECTURE

In our PDMS, all peers are conceptually equal. This means there are no super-peers or other specialised nodes present in the P2P network, and also no kind of centralised control is needed. Instead, the network only consists of a set of equal (with respect to their functionality) peers. Therefore the implementation of the PDMS reduces to the implementation of such a peer, that we realised as a Java program. Of course, the peers may differ with respect to their schemas, stored data and constraints/mappings.

The general structure of our peer implementation is shown in Figure 9. Every peer defines a schema, constraints over this schema and mappings from the schemas of other peers to the own peer schema that define in which data stored at the other peers a peer is interested in, and how the content of the local instance depends on the other peer instances. This information is, among others, specified within an XML configuration file handed to the peer at startup. The schema describes the structure of the RDBMS that is used for storing the local data of the peer. To support database platforms using different SQL dialects, CoDE implements its own database connectors that are used for rewriting the SQL queries into the correct dialect before issuing them to the database system, which is done by using the functionality provided by the spring framework [74]. These connectors are the reason why our implementation does not work on all databases with JDBC drivers (but first needs an own connector for each database).

1 http://hsqldb.org/

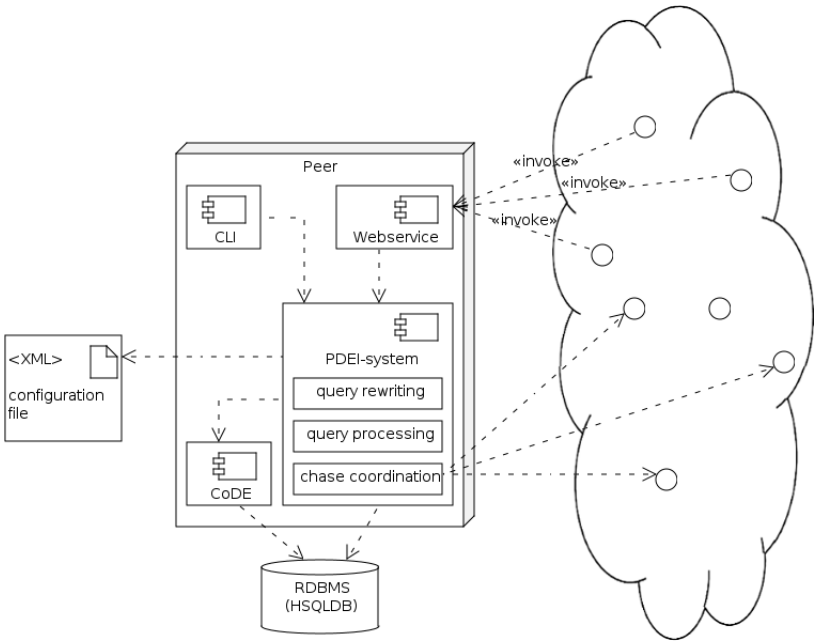

Figure 9: Overview of the peer structure in our PDMS implementation

To use a platform and programming language independent mechanism, communication between the peers is implemented using webservices. Every peer publishes a set of webservices that are then called by the other peers in order to exchange information. Although in our setting, each peer is an instance of the same implementation, by using webservices for communication also different peer implementations could join the network, as long as they implement the webservice interface (and the correct protocols). All the information required to communicate with a certain peer is also specified in the configuration file, such that no lookup service or other mechanisms to locate a certain peer are needed.

Since, as shown in Chapter 5, all important reasoning tasks over PDE- and PDEI-systems are based on the chase procedure, besides computing query rewritings and posing queries to the database backend, the most important task of the PDEI implementation is to control the chase. Thereby for processing the local chase (i.e. chasing e-constraints and i-constraints) parts of the CoDE system are used. CoDE transforms the XML specifications of the dependencies using XSLT templates into SQL commands that chase the local database instance. By using a mechanism sketched in [15], also most of the work of chasing the P2P mappings can be done by using the local chase implementation of CoDE. Moreover, CoDE is used for managing most of the database access.

While CoDE checks dependencies for being applicable, and if so applies them, the PDEI-system coordinates the local chase with the propagation of the chase to the neighbouring peers. This includes sending the correct

data according to the P2P mappings to the neighbours and keeping track about which neighbours have already finished their chase. Concerning query answering, the PDEI-system rewrites the query, initialises the necessary chase and returns the certain answers to the user.

Finally a command-line interface provides some basic control of the peer, like posing queries or initiating chases, and displays results or other feedback to the user.

6.2 PROBLEMS ARISING FROM AMBIGUOUS DEFINITIONS

In this section we mention and discuss some problems encountered during the implementation because of some slight inaccuracies in the definitions of certain notions given in [30]. Moreover we introduce and justify the assumptions made to overcome these problems.

6.2.1 *Query Answering in PDEI-Systems*

In section 5.6.2 we introduced some results concerning query answering in a PDEI-system. The main result was Theorem 5.43, stating that in a PDEI-system $\mathcal{S}$, for an $\mathcal{S}$-admissible state B and a conjunctive query q, $certain(q, \mathcal{S}, B) = Eval_{Null\downarrow}(rewrite(q, \mathcal{I}), \text{E-CHASE}(\tau(\mathcal{S}), B))$ (where $\mathcal{I}$ denotes the set of inclusion dependencies in $\mathcal{C}_I$).

At least according to our understanding of how to apply the rewriting algorithm in this situation, this theorem is not correct, as shown by the following counter-example:

Example 6.1. Let $\mathcal{S}$ denote the PDEI-system $\langle\{P_1\}, \emptyset, \emptyset, \{key(a) = \{1\}\}, \emptyset\rangle$ with $S(P_1) = \{a/3\}$. Assume further the indefinite state $B = \{$ a('1','a',C), a('1',A,'c')$\}$ where $1, a, c \in \mathcal{C}$ and $A, C \in \mathcal{N}$ (i.e. A and C are labelled nulls).

Given the query $q = \{(K, A, C) \mid a(K, A, C)\}$, then $certain(q, \mathcal{S}, B) \neq Eval_{Null\downarrow}(rewrite(q, \mathcal{I}), \text{E-CHASE}(\tau(\mathcal{S}), B))$:

- $Eval_{Null\downarrow}(rewrite(q, \mathcal{I}), \text{E-CHASE}(\tau(\mathcal{S}), B)) = \emptyset$:
  It first needs to be checked that B is an $\mathcal{S}$-admissible state. By Theorem 5.31 this holds iff $\text{EI-CHASE}(\mathcal{S}, B) = B$. Since $\mathcal{C}_E = \mathcal{M}_E = \emptyset$, no chase rule can be applied to B, hence $\text{EI-CHASE}(\mathcal{S}, B) \neq B$ only if B is not $\mathcal{S}^I$-consistent and hence $\text{EI-CHASE}(\mathcal{S}, B) = \text{FAIL}$. But since for example the state $D = \{$a('1','a','c')$\}$ satisfies $\mathcal{S}^I$ and B, $Sem(\mathcal{S}^I, B) \neq \emptyset$, and B is $\mathcal{S}^I$-consistent. Therefore $\text{EI-CHASE}(\mathcal{S}, B) = B$, hence B is $\mathcal{S}$-admissible[2].
  To compute $Eval_{Null\downarrow}(rewrite(q, \mathcal{I}), \text{E-CHASE}(\tau(\mathcal{S}), B))$, consider the two arguments of $Eval_{Null\downarrow}$:
  - $\text{E-CHASE}(\tau(\mathcal{S}), B) = B$:
    This holds because from $\mathcal{M}_I = \emptyset$ it follows that $Expand(\mathcal{M}_I, \mathcal{I}) = \emptyset$, hence $\tau(\mathcal{S}) = \langle\mathcal{P}, \emptyset, \emptyset\rangle$, and there are no rules that could be applied during the chase. Therefore $\text{E-CHASE}(\tau(\mathcal{S}), B) = B$.

2 This could be also verified by checking the conditions of Definition 5.20. Being more lengthy, this yields the same result.

– $rewrite(q, \mathcal{I}) = q$:
$\mathcal{I}$ are the foreign key dependencies in $\mathcal{C}_I$. Because $\mathcal{C}_I$ contains only a single key constraint, $\mathcal{I} = \emptyset$. Therefore there is trivially no inclusion dependency to apply to q. Moreover, since the body of q contains only a single atom, no unification can be performed either. Therefore $rewrite(q, \mathcal{I}) = q$.

Therefore,
$Eval_{Null\downarrow}(rewrite(q, \mathcal{I}), \text{E-CHASE}(\tau(\mathcal{S}), B)) = Eval_{Null\downarrow}(q, B)$. Because of $q^B = \{(\text{'1'},\text{'a'},C), (\text{'1'},A,\text{'c'})\}$, which contains no tuple without a labelled null, $Eval_{Null\downarrow}(q, B) = \emptyset$.

- $certain(q, \mathcal{S}, B) = \{(\text{'1'},\text{'a'},\text{'c'})\}$:
This follows from the fact that every definite instance in $Sem(\mathcal{S}, B)$ has to contain a fact $a(\text{'1'},\text{'a'},\text{'c'})$: For every instance $D \in Sem(\mathcal{S}, B)$, there must exist a homomorphism from B to D. Because every homomorphism maps constants onto themselves, every instance $D \in Sem(\mathcal{S}, B)$ has to contain some fact $a_1 = a(K, A, C)$ with $a[1] = \text{'1'}$ and $a[2] = \text{'a'}$ and some arbitrary value for $a[3]$ and some fact $a_2 = a(K_1, A_1, C_1)$ with $a[1] = \text{'1'}$ and $a[3] = \text{'c'}$ and some arbitrary value for $a[2]$. Moreover also the key constraint $key(a) = 1$ has to be satisfied. Since the only possibility to achieve this is that $a_1 = a_2 = a(\text{'1'},\text{'a'},\text{'c'})$, it follows that $a(\text{'1'},\text{'a'},\text{'c'}) \in D$, $\forall D \in Sem(\mathcal{S}, B)$, which proves the claim.

Hence Theorem 5.43 gives $\emptyset = \{(\text{'1'},\text{'a'},\text{'c'})\}$ and therefore does not apply in this setting. ◇

The problem as we see it is that the key constraints from $\mathcal{C}_I$ are never taken into account for query answering, neither for query rewriting nor in $\tau(\mathcal{S})$. To overcome this problem, we based our implementation on a slightly different definition of $\tau(\mathcal{S})$:

Definition 6.2 ($\tau(\mathcal{S})$)**.** Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ be a PDEI-system. Let $\mathcal{I}$ denote the set of inclusion dependencies in $\mathcal{C}_I$ and $\mathcal{J}$ denote the set of key constraints in $\mathcal{C}_I$. Then $\tau(\mathcal{S})$ is defined as the following PDE-system:

$$\tau(\mathcal{S}) = \langle \mathcal{P}, \mathcal{J}, Expand(\mathcal{M}_I, \mathcal{I}) \rangle \ .$$

⊣

Using this definition for $\tau(\mathcal{S})$, E-CHASE$(\tau(\mathcal{S}), B)$ also enforces the equalities implied by the key constraints in $\mathcal{C}_I$ on B. If such a chase fails, then B is not consistent for $\mathcal{S}$, and query answering is trivial. On the other hand, if the data materialised during this chase does not violate any key, then, by the separation property, also the data implied by the foreign key constraints can not violate any key constraint.

Together with the correctness of the *rewrite* algorithm, this ensures that all tuples implied by $\mathcal{C}_I$ and $\mathcal{M}_I$ are exchanged between the peers, and that $Eval_{Null\downarrow}(rewrite(q, \mathcal{I}), \text{E-CHASE}(\tau(\mathcal{S}), B))$ indeed returns the certain answers.

6.2.2 *Semantics of* $certain_{Null}(q, \mathcal{S}^I, B)$

In Section 5.5, $certain_{Null}(q, \mathcal{S}^I, B)$ was introduced as presented in [30]. Given a PDEI-system $\mathcal{S}' = \langle \mathcal{P}, \emptyset, \emptyset, \mathcal{C}_I, \mathcal{M}_I \rangle$[3] and a state B for $\mathcal{S}$, assuming that B is $\mathcal{S}'$-consistent, $certain_{Null}(q, \mathcal{S}^I, B)$ was defined as the "set of certain answers to q in $\mathcal{S}'$ and $\sigma_{\mathcal{S}}(B)$ computed under the assumption that the null values in $\sigma_{\mathcal{S}'}(B)$ are ordinary constants" [30].

This definition leaves some room for interpretation, which is the reason why we justify our interpretation here. The problems arise from the phrase "under the assumption that the null values [...] are ordinary constants". The differences between labelled nulls and constants are that labelled nulls do not appear in definite instances and that they may be mapped (by homomorphisms or instantiations) to other values. Therefore assuming labelled nulls to be constants means that they can no longer be unified or mapped to other values, and that they are allowed in definite instances, hence to appear in universal solutions.

The inaccuracies we encountered are with respect to the situations in which labelled nulls should be considered as constants, and in these situations, which labelled nulls should be actually interpreted as constants.

When to Consider Labelled Nulls as Constants

The first ambiguity we encountered is for which situations labelled nulls should be considered as constants. It can be best described when considering a concrete method for computing the certain answers over a PDEI-system, for example the one described in Theorem 5.43. There, the question arises whether the labelled nulls should be already considered as constants during the computation of E-CHASE$(\tau(\mathcal{S}'), \sigma_{\mathcal{S}}(B))$, or only for the evaluation of $rewrite(q)$ over the result of the E-CHASE.

More generally, the question is whether labelled nulls should be only considered as constants for the final evaluation of the given query over some state (i.e. for deciding which tuples from $q^{B'}$ actually belong to the certain answers, that is contain no labelled nulls), or whether the labelled nulls should be also considered as constants while computing which information is implied by the mappings in $\mathcal{M}_I$.

We argue that although the definition describes the second case, the first case is the correct one. This is due to the fact that it captures the intuitive semantics of PDEI-systems (peers only exchange certain answers), and that the following example shows that the second interpretation gives wrong results.

Example 6.3. Consider the following PDEI-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ with

- $\mathcal{P} = \{P_1, P_2\}$,
- $S(P_1) = \{a/2\}$, $S(P_2) = \{b/2, c/2\}$,
- $\mathcal{C}_E = \{b(A, B) \rightarrow c(A, B)\}$,

3 We use $\mathcal{S}'$ to emphasise that $\mathcal{C}_E = \mathcal{M}_E = \emptyset$ for these considerations.

- $\mathcal{M}_I = \{a(A,B) \rightarrow b(A,B)\}$, and
- $\mathcal{C}_I = \mathcal{M}_E = \emptyset$.

Note that because of $\mathcal{C}_I = \emptyset$, $rewrite(q) = q$ for all UCQs q.
Given a state $B = \{a(A,'a')\}$, where A is a labelled null, and a query $q = \{(B) \mid c(\_,B)\}$, then $certain(q,\mathcal{S},B) = \emptyset$ (as $Sem(\mathcal{S},B)$ contains definite instances without any tuples for the relations b and c).
But when using the second interpretation of $certain_{Null}(q,\mathcal{S}^I,B)$, EI-CHASE$(\mathcal{S},B) = \{a(A,'a'), c(A,'a')\}$ (because $(A,'a')$ would be an answer to $certain_{Null}(\{(A,B) \mid b(A,B)\},\mathcal{S}^I,B)$). Therefore the answer to q would be $\{('a')\}$, which does not match the correct result from above. ◇

Note that for the semantics of the local constraints there is no difference between these two interpretations, since conflicts with key constraints are already considered by $\sigma_{\mathcal{S}'}$, and because of the FOL interpretation of the foreign key dependencies it makes no difference for their semantics at all.

Therefore, when computing $certain_{Null}(q,\mathcal{S}^I,B)$, for determining the effect of the mappings in $\mathcal{M}_I$, we consider the labelled nulls as labelled nulls, and only when evaluating q over some state where all the information implied by the P2P mappings is already materialised we consider the labelled nulls as constants.

Labelled Nulls to Consider as Constants

The more subtle ambiguity is which labelled nulls should be actually interpreted as constants.

The problem is that considering only those labelled nulls as constants that are already materialised in B, as suggested by defining $\sigma_{\mathcal{S}'}$ only over *nulls*(B), does not give the expected results, as demonstrated by the next two examples.

Example 6.4. Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ be a PDEI-system where

- $\mathcal{P} = \{P_1\}$,
- $S(P_1) = \{a/3, b/2\}$,
- $\mathcal{C}_E = \{a(A,\_,C_1), b(A,C_2) \rightarrow C_1 = C_2\}$,
- $\mathcal{C}_I = \{a[1] \subseteq b[1], key(b) = \{1\}\}$, and
- $\mathcal{M}_E = \mathcal{M}_I = \emptyset$.

Consider the state $B = \{a('a','b','c')\}$ and the query $q = \{(A,C) \mid b(A,C)\}$. The certain answers to q with respect to B are obviously $\{('a','c')\}$, since every definite instance $D \in Sem(\mathcal{S},B)$ has to contain $b('a','c')$ in order to satisfy $a[1] \subseteq b[1]$ and $a(A,\_,C_1), b(A,C_2) \rightarrow C_1 = C_2$. Moreover $b('a','c')$ has already to be contained in every $\mathcal{S}$-admissible state (due to the fact that the constraint $a(A,\_,C_1), b(A,C_2) \rightarrow C_1 = C_2$ cannot be enforced on data not already materialised in an $\mathcal{S}$-admissible state[4].), and therefore in every universal $\mathcal{S}$-solution of B.

4 A formal proof is omitted due to space restrictions

But b('a','c') $\notin$ EI-CHASE($\mathcal{S}, B$): The only possible chase rule that could be applied to B is the EGD-rule: Obviously, B is $\mathcal{S}^I$-consistent. Therefore denote with q_1 the conjunctive query $\{(C_1, C_2) \mid a(A, \_, C_1), b(A, C_2)\}$. B contains no labelled nulls, hence $\sigma_{\mathcal{S}'}(B) = B$, and also no labelled null should be considered as constant. Therefore $certain_{Null}(q_1, \mathcal{S}^I, B) = certain(q_1, \mathcal{S}^I, B)$. Since $\mathcal{S}^I$ contains only the constraints from $\mathcal{C}_I$, $certain(q_1, \mathcal{S}^I, B) = \emptyset$. As a result, the EGD-rule is not applicable, and EI-CHASE$(\mathcal{S}, B) = B$, since B is obviously $\mathcal{S}$-consistent. But since B does not contain b('a','c'), EI-CHASE$(\mathcal{S}, B)$, although returning a value different from FAIL, does not return a universal $\mathcal{S}$-solution, which contradicts Theorem 5.31. ◇

Example 6.5. Let $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ be a PDEI-system where

- $\mathcal{P} = \{P_1, P_2\}$,
- $S(P_1) = \{a_1/2\}, S(P_2) = \{a_2/3, b_2/4\}$,
- $\mathcal{C}_E = \{a_2(A, B, C) \rightarrow b_2(A, B, C, D)\}$,
- $\mathcal{M}_I = \{a_1(B, C) \rightarrow a_2(A, B, C)\}$, and
- $\mathcal{M}_E = \mathcal{C}_I = \emptyset$.

Consider a state $B = \{a_1('b','c')\}$. Then every $D \in Sem(\mathcal{S}, B)$ has to contain some fact $a_2(\_,'b','c')$ with an arbitrary value at the first position (s.t. $Sem(\mathcal{S}, B)$ satisfies $\mathcal{M}_I$) and therefore every $D \in Sem(\mathcal{S}, B)$ further has to contain some fact $b_2(\_,'b','c',\_)$ with arbitrary values at the first and last position (to satisfy $\mathcal{C}_E$). Therefore every universal $\mathcal{S}$-solution must contain such a fact $b_2(\_,'b','c',\_)$.

But EI-CHASE$(\mathcal{S}, B) = B$, hence the result does not contain such a fact: The only chase rule that could be applied during EI-CHASE is $a_2(A, B, C) \rightarrow b_2(A, B, C, D)$ (call this rule ϕ). Denote with q_1 the query $\{(A, B, C) \mid a_2(A, B, C)\}$. The application of ϕ then depends on the results of $certain_{Null}(q_1, \mathcal{S}^I, B)$. Since B contains no labelled nulls, $\sigma_{\mathcal{S}^I}(B) = B$ and no labelled null is considered as constant. Therefore $certain_{Null}(q_1, \mathcal{S}^I, B) = certain(q_1, \mathcal{S}^I, B)$. Obviously the certain answers to q_1 in B are equal to the empty set. Therefore ϕ is not applicable and because obviously $Sem(\mathcal{S}, B) \neq \emptyset$, EI-CHASE$(\mathcal{S}, B) = B$. However, since B contains no fact $b_2(\_,'b','c',\_)$, B is no universal $\mathcal{S}$-solution (it can be also easily verified that B is not even an $\mathcal{S}$-admissible state). This contradicts Theorem 5.31, as EI-CHASE$(\mathcal{S}, B)$, while not returning FAIL does not return a universal $\mathcal{S}$-solution. ◇

In both examples the problems arise from considering only those labelled nulls as constants that are already materialised in the state B. For example in Example 6.5 one would consider the labelled null implied by the TGD $a_1(B, C) \rightarrow a_2(A, B, C)$ from the fact a_1*('b','c')* as constant when computing $certain_{Null}(q_1, \mathcal{S}^I, B)$, ϕ would be applicable and the application of ϕ would materialise a fact b_2*(A,'b','c',D)* (with, A, D being labelled nulls), hence EI-CHASE$(\mathcal{S}, B)$ would yield the correct result. A similar result holds for Example 6.4.

On the other hand, considering all labelled nulls implied by the constraints and dependencies (i.e. those labelled nulls that would be introduced if chasing all dependencies and constraints on B) as constants is not correct as well, since in the case of certain cyclic inclusion dependencies there are infinitely many of them, hence computing EI-CHASE becomes undecidable.

As can be seen from the above examples, problems only occur whenever some constraint from $\mathcal{C}_E$ is based on the result of some virtual constraint or mapping, i.e. when a relational symbol occurring on the right hand side of a constraint in $\mathcal{C}_I \cup \mathcal{M}_I$ also occurs on the left hand side of a constraint in $\mathcal{C}_E$ (it is easy to construct similar examples for not stratified systems).

To overcome these problems, we applied the following rules for implementation: In accordance with the result of the previous subsection, for the evaluation of the i-mappings only "real" constants are considered as constants. On the other hand, for the evaluation of the e-constraints, not only those labelled nulls already materialised in the current state are considered as constants, but also those labelled nulls implied by the i-constraints and i-mappings, that is those labelled nulls that would be introduced by chasing the current state with the i-mappings and i-constraints. To avoid the problems introduced by cyclic foreign key dependencies that may give rise to infinitely many labelled nulls, we use an extended definition for stratified PDEI-systems that also forbids configurations where the head of some foreign key dependency appears on the left hand side of any $\mathtt{EGD} \in \mathcal{C}_E$. That is, our implementation is based on the following definition of stratified PDEI-systems:

Definition 6.6. A PDEI-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ is a *stratified PDEI-system*, if the TGDs in $\mathcal{C}_E$ form a weakly acyclic set, $\mathcal{C}_I$ corresponds to a set of legal key constraints and foreign key dependencies, and further no head of a foreign key dependency in $\mathcal{C}_I$ appears on the left hand side of any TGD or EGD in $\mathcal{C}_E$. ⊣

Unfortunately, there are still some problems left, as shown next.

Problems with Computing $\mathtt{certain}_{\mathtt{Null}}(q, \mathcal{S}^I, B)$

The problem with labelled nulls that are not materialised in an instance to which a chase rule is applied is that they may be materialised within a tuple that is not the same tuple where they originated from (because the tuple in which the labelled null was introduced as skolem term is implied only by a virtual constraint, hence is not materialised). This problem can be illustrated by the following example:

Example 6.7. Let $\mathcal{S}$ be the PDEI-system from Example 6.5, B be the state $\{a_1(\text{'b'},\text{'c'})\}$ for $\mathcal{S}$, denote with q_1 the query $\{(A, B, C) \mid a_2(A, B, C)\}$ and consider the computation of EI-CHASE$(\mathcal{S}, B)$. When applying the TGD-rule to B, $\mathtt{certain}_{\mathtt{Null}}(q_1, \mathcal{S}^I, B)$ returns (according to the interpretation described above) the tuple $(A_1, \text{'}b\text{'}, \text{'}c\text{'})$, where A_1 is a fresh labelled null. Therefore the result of applying the TGD-rule to B is the state $B' = \{a_1(\text{'}b\text{'},\text{'}c\text{'}), b_2(A_1, \text{'}b\text{'}, \text{'}c\text{'}, D_1)\}$. Now the problem is that the fact $a_2(A_1, \text{'}b\text{'}, \text{'}c\text{'})$, where the labelled null A_1 originated, is not part of this instance. Therefore, with respect to B', the

P2P mapping $a_1(B, C) \rightarrow a_2(A, B, C)$ again implies a fresh labelled null not yet present in B'. Hence applying the same chase rule once again to state B' would add another fact, say $b_2(A_2, \text{'}b\text{'}, \text{'}c\text{'}, D_2)$, as in the current framework there is no way to mark A_1 as that labelled null that is implied by this P2P mapping from the fact $a_1(\text{'}b\text{'},\text{'}c\text{'})$. ◇

Since these superfluous facts are all subsumed by the fact first inserted, they do not alter the semantics of the instance. However, they prevent termination of the EI-CHASE (since no fixpoint is reached). Moreover, adding facts to an instance that are completely redundant is no nice solution.

In fact the problem of nontermination of the EI-CHASE can be avoided very easily by remembering the tuples created by the application of some virtual dependency between two chase steps of the same EI-CHASE. This can simply be done by not recalculating E-CHASE $(\tau(\mathcal{S}^I), B_i)$ from scratch for each chase step of the EI-CHASE, but by keeping the result from the last chase step, adding those tuples materialised in the current step, and running the EI-CHASE from this state again. This is exactly how we implemented the EI-CHASE (see Section 6.3.4).

However, the problem remains that for every run of the EI-CHASE (i.e. each time the EI-CHASE is started) redundant facts may be added to the database, hence increasing its size unnecessarily. By now, we have no solution to this problem, that certainly deserves further considerations in the future.

6.3 DESIGN DECISIONS

While the last section describes and justifies our assumptions concerning some ambiguities in the formal definitions of the PDEI-system, this section describes our decisions concerning the concrete implementation of the PDMS, concentrating on critical and important aspects. It therefore offers a description of all the basic components of the system, describes how certain theoretical concepts have been implemented and explains the most important design decisions.

To make it easier to denote the neighbourhood of a peer, we will use the following terms to refer to certain neighbours of a peer: Assume some peer $\mathcal{P}_i$ of the PDMS. Then a *target neighbour* of $\mathcal{P}_i$ is a peer $\mathcal{P}_j$ such that there exists some P2P-mapping $\phi\colon\ q_i \rightarrow q_j$ where q_i is defined over the schema of $\mathcal{P}_i$ and q_j is defined over the schema of $\mathcal{P}_j$. On the other hand, a *source neighbour* of of $\mathcal{P}_i$ is a peer $\mathcal{P}_j$ such that there exists some P2P-mapping $\phi\colon\ q_j \rightarrow q_i$, again where q_i is defined over the schema of $\mathcal{P}_i$ and q_j is defined over the schema of $\mathcal{P}_j$. (Thereby ϕ may be $\in \mathcal{M}_E$, $\mathcal{M}_I$ or $\mathcal{M}_E \cup \mathcal{M}_I$.)

Moreover, when talking about a P2P mapping $q_i \rightarrow q_j$ from peer $\mathcal{P}_i$ to peer $\mathcal{P}_j$, we call $\mathcal{P}_i$ the *source peer* and $\mathcal{P}_j$ the *target peer*.

6.3.1 *Peer Configuration and Neighbourhood Setup*

Every peer is configured using an XML file. The information specified in this file includes the IP-address and port number on which the peer shall set up

its webservice, the connection information to its database, the structure of its peer schema and the set of e-constraints and i-constraints over its schema. Moreover within the configuration file it is defined how the schemas of its source neighbours are mapped to the local peer schema. That is all P2P mappings $q_i \rightarrow q_j$ are specified in the configuration file of the peer over whose schema q_j is defined. Beside the mappings, also contact information (IP-address and port number of the webservice) of the source neighbours are given.

Therefore after startup each peer knows from which peers it should receive data, but not to which peers it should send data during a chase. To exchange this information, after startup each peer informs its source neighbours about the existing P2P mappings. This is done by sending the queries on the left hand sides of the TGDs together with the information whether the query belongs to a virtual mapping or not and its own contact information to the source neighbours. There this information is stored, and queries belonging to virtual mappings are rewritten according to the local virtual inclusion dependencies. Thereby, every peer assigns a unique id to each of its source neighbours, simply by enumerating them. Moreover, within one neighbour, every P2P mapping gets a unique id (again a number) assigned.

This process of announcing the P2P mappings does not start automatically but has to be initiated by the user. Moreover, if sending the information to some source neighbour fails (e.g. because it has not started yet), no automatic retry is performed, but the user has to manually restart the announcement process.

From the above description it follows that it is assumed that both the addresses of all its source peers and their peer-schemas are known at configuration time by any peer. For a prototype implementation this restriction seems plausible. Moreover, the distributed system community provides several solutions to overcome these restriction. However, due to their high implementation costs, they were omitted.

6.3.2 *Database and Database Schema*

As already mentioned, we tested our implementation with HSQLDB as RDBMS, since it provides a very easy and lightweight setup of new instances, such that we were able to easily provide every peer with its own database instance.

Most of the interaction with the database is done through functionality provided by the CoDE system. For manipulations of the database schema (like adding attributes or creating tables), CoDE uses the open source library Apache DDLUtils[5], while for accessing the data stored in relations, the corresponding classes from the spring framework [74] are used. When accessing the database directly, we use the same libraries. Moreover, CoDE abstracts from the used RDBMS by providing a mechanism to support databases using different SQL dialects. However, this requires to provide connectors for database systems to use with CoDE (hence our implementation). CoDE al-

5 http://db.apache.org/ddlutils/

ready provides connectors to Oracle and PostgreSQL, that have not yet been tested in combination with our implementation. Moreover it should be easy to create new connectors for other JDBC-compatible databases.

CoDE uses several relations to store all information needed for efficient core computation. For a description of these tables and views, that can be easily identified by their names starting either with `DX_` (for tables) or `VW_DX_` (for views), we refer to [71], and only describe those tables and views added by our extension in Section 6.3.3.

By application of EGDs, it might happen that duplicate rows are introduced in the database. But, especially for core computation, the CoDE system needs to uniquely identify every row in the database. Therefore an additional column `dx_id` is added to every table of the peer schema that stores a unique id for each row. Since in theory set semantics for relations is assumed, we do not further care about duplicates, but always use `SELECT DISTINCT` in the SQL statements and leave duplicate entries in the database.

The tables, views and columns required by CoDE and those introduced in the next section are created at startup automatically. Therefore it is necessary to provide a specification of the peer schema within the configuration file, using the Turbine XML format. [71] gives a description of how to define the peer schemas.

6.3.3 *Labelled Nulls and Temporarily Materialised Data*

Handling Labelled Nulls

Because none of the common DBMSs currently supports labelled nulls, the handling of labelled nulls has to be implemented as part of the application. In CoDE, this is done by introducing auxiliary columns to the peer relations, one for every original attribute. These columns are used to store the labels of the null values for the corresponding attribute.

For example, the table $\mathit{Authorised}(\mathrm{SID}, \mathrm{FN}, \mathrm{LN})$ would be augmented by the three additional columns `(SID_var, FN_var, LN_var)`, storing the labels of the nulls in `(SID, FN, LN)`.

It was mentioned in [71] that this "simulation" of labelled nulls is probably not efficient enough to be used in real world applications, but that it was very convenient for the development of the prototype because of its flexibility. Therefore we stick to this approach.

Handling Temporarily Materialised Data

For query answering in a PDEI-system it is necessary to temporarily materialise the data implied by the virtual P2P mappings on the one hand and the equalities enforced by the key constraints in $\mathcal{C}_I$ on the other hand (that is the result of E-CHASE$(\tau(\mathcal{S}, B))$, see Section 5.6.2). To mark tuples in the database that are only materialisations of such virtual data, we extend the tables of the peer schema by an extra attribute `pdei_chasetype`. The standard value "0" indicates that the corresponding tuple actually exists materialised in the

instance, while a value of "1" indicates that this tuple exists only virtually in the instance and was only materialised temporarily.

The mappings derived from the application of the EGDs in $\mathcal{C}_I$ are not materialised directly on the data, but are stored in an extra table `PDEI_VCHASE_MAPPING(var_name, target, target_var)`. Every row in this table encodes a mapping for the labelled null stored in `var_name`: if `target NOT IS NULL`, then it is mapped to the constant value stored in `target`, otherwise it is mapped to the labelled null whose label is stored in `target_var`.

To provide easy access to the data under these mappings, for each table <tablename> we create a view `PDEI_V_`<tablename> that contains the content of <tablename> after applying the mappings from `PDEI_VCHASE_MAPPING`. As an example for the definition of such a view, the following is the view definition corresponding to a table `Books(ISBN, ISBN_var)`:

```
CREATE VIEW PDEI_V_BOOKS AS
SELECT
      Books.dx_id,
      COALESCE (Books1.ISBN, M1.target) AS ISBN,
      CASE WHEN (COALESCE (Books1.ISBN, M1.target) IS NULL) THEN
          COALESCE(M1.target_var,Books1.ISBN_var) ELSE NULL END AS
              ISBN_var
FROM Books Books1 LEFT JOIN PDEI_VCHASE_MAPPING M1
      ON Books1.ISBN_var = M1.var_name
```

Thus for the handling of the key constraints we reuse the same mechanism that is already used in CoDE to model homomorphisms.

Figure 10 shows an example of the concepts considered above. Assuming that the peer schema contains a relation $\mathtt{Books}(\mathtt{ISBN}, \mathtt{author})$, it sketches the part of the database representing the information discussed above.

6.3.4 *The Chase*

To be able to distinguish between the chase as procedure and one run of this procedure, we will use *chase instance* to denote one run of the chase. Therefore each chase instance gives rise to a chase sequence.

Like all reasoning and coordination tasks in our PDMS, also the chase is implemented as distributed algorithm.

Thereby the basic idea of this algorithm is as follows: A chase instance is initialised at some peer $\mathcal{P}_i$. This peer first chases its local constraints (i.e. those TGDs and EGDs from $\mathcal{C}_E$ defined over $S(\mathcal{P}_i)$), which means that it applies them until no more dependency is applicable. We refer to this step as *local chase*. Once the local chase has finished, it propagates a request to continue the chase instance to all its target neighbours, and waits until it receives from them the information that they have finished processing the request.

Whenever a peer $\mathcal{P}_j$ receives such a request from a peer $\mathcal{P}_i$, it performs the following steps: First it applies the P2P mappings between $\mathcal{P}_i$ and $\mathcal{P}_j$ (i.e. those TGDs $q_i \rightarrow q_j$ from $\mathcal{M}_E$ where q_i is defined over the schema of $\mathcal{P}_i$ and q_j is defined over the schema of $\mathcal{P}_j$) until none of them is applicable any more.

ISBN	ISBN_var	author	author_var	dx_id	pdei_chasetype
001	NULL	THE author	NULL	1	0
NULL	N_1	THE author	NULL	2	1
NULL	N_2	NULL	N_3	3	0

var_name	target	target_var
N_1	001	NULL
N_2	NULL	N_4

ISBN	ISBN_var	author	author_var	dx_id
001	NULL	THE author	NULL	1
001	NULL	THE author	NULL	2
NULL	N_4	NULL	N_3	3

Figure 10: The tables Books(ISBN, author), `PDEI_VCHASE_MAPPING` and `PDEI_V_Books` as example for the modelling of labelled nulls and materialised virtual data.

Afterwards it runs the local chase. Once it has finished, $\mathcal{P}_j$ decides whether it needs to further propagate the chase to its target neighbours, or whether it can inform $\mathcal{P}_i$ immediately about having finished processing the request. If $\mathcal{P}_j$ further propagates the chase, then the described procedure repeats in all of its target neighbours, and $\mathcal{P}_j$ has to wait until all of its target neighbours have processed the request. Then it can inform $\mathcal{P}_i$ about having finished the request.

Once the initiating peer received from all of its target neighbours that they have finished, the chase terminates.

If an error occurs at some peer or some peer discovers an inconsistency, it immediately returns a corresponding error to the peer from which it has received the chase. Once a peer receives such an error, it does not wait for its other target neighbours to finish, but immediately returns this error. Hence, as stated above and according to the definition that the result of the chase is FAIL if the state is inconsistent, the chase is simply aborted in case of any error.

Note that in the above description, only the dependencies from $\mathcal{C}_E$ and $\mathcal{M}_E$ are applied. This is because both, E-CHASE and EI-CHASE only materialise values implied by dependencies from $\mathcal{C}_E$ and $\mathcal{M}_E$, hence $\mathcal{C}_I$ and $\mathcal{M}_I$ can be neglected for the moment.

The implementation of this algorithm is heavily based on the CoDE-system. For computing a universal solution of a given source instance with respect to a data exchange setting, CoDE provides an implementation of the chase for data exchange as described in Section 3.1.2. Especially the target chase (i.e. chasing TGDs and EGDs over a single schema under the FOL semantics) is of interest for us, since this is exactly the same as the local chase described above. Hence for the moment we regard the chase implementation of the CoDE system

as blackbox (denoted by `localchase()`) that takes a database instance and a set of dependencies as arguments and materialises all values implied by these dependencies on the given instance. For running the local chase, the peer just calls `localchase()` accordingly. Moreover, we apply the technique used in [15] to prove several properties of the alternative semantics for P2P mappings proposed there to also reduce the chase of the P2P mappings (under the alternative semantics) to the local chase, such that it can be done by also using `localchase()`. The idea is to chase a P2P mapping $\phi\colon q_i \to q_j$ by first computing the certain answers to q_i, and sending them to the target peer, where they are stored in a new relation containing exactly one attribute for every free variable of q_i. Then the values implied by ϕ on the instance of the target peer are materialised by chasing a new local TGD $q_n \to q_j$ using `localchase()`, where q_n simply select all entries from the new relation.

We demonstrate the idea on the TGD $\phi\colon \mathit{Authorised}(\mathsf{SID}, \mathsf{FN}, \mathsf{LN}) \to \mathit{Allowed}(\mathsf{ID}, \mathsf{FN}, \mathsf{LN})$ from our example. Denote with q_1 the CQ $q_1 = \{(\mathsf{FN}, \mathsf{LN}) \mid \mathit{Authorised}(\mathsf{FN}, \mathsf{LN})\}$. Then in the first step, $\mathit{certain}(q_1, \mathcal{S}^I, B)$ is computed, where B denotes the current state of the PDE- or PDEI-system $\mathcal{S}$. For a PDE-system, $\mathit{certain}(q_1, \mathcal{S}^I, B) = \mathit{Eval}_{\mathit{Null}\downarrow}(q_1, B)$. For a PDEI-system it means to compute the certain answers to q_1 over B by only taking the virtual dependencies into account. In both cases, it corresponds to query answering every peer is capable of. Hence this step can be performed by the source peer. The result of evaluating the query is then sent to the target peer. There, an additional relation `PDEI_1_1(FN,LN)` is added to the schema, and a new local TGD $\mathsf{PDEI\_1\_1}(\mathsf{FN}, \mathsf{LN}) \to \mathit{Allowed}(\mathsf{ID}, \mathsf{FN}, \mathsf{LN})$ is created. (The name of the relation added is of the form `PDEI_`$<\mathit{neighbourId}>$`_`$<\mathit{dependencyId}>$.) Finally `localchase()` is called with this new TGD as argument, which materialises all values implied by the original P2P-mapping.

Note that the newly created TGD could be simply added to the local dependencies, to be processed during the local chase, without the need of taking any extra care. But for performance reasons it is called separately, as once it is no longer applicable, similar to the source to target TGDs in data exchange, it will not become applicable again within the same run of `localchase()`, hence need not be checked in every iteration of the local chase.

On the other hand, although not being very efficient, each time a P2P mapping $q_i \to q_j$ is checked for being applicable, all certain answers to q_i are exchanged between the peers, regardless of whether they have been already sent in an earlier iteration or not.

Every chase instance is identified by a unique id, which is created by the peer initialising the instance. This id is used to determine whether a chase instance arrives for the first time at a peer or not. Moreover, whenever a peer propagates a chase instance to its target neighbours, it attaches a new, local id to the request sent to its neighbours. When a peer has finished processing such a request, it returns the corresponding id. This local id is necessary since it might happen that, because of cycles in the P2P mappings, a peer forwards a chase instance a second time while it is still waiting for an answer to a previous request. To be able to correctly assign responses of target peers to the sent requests, this local id is used. To guarantee the uniqueness of those

ids (both, global and local ones) each id encodes the id of the peer, the time when the id was generated and a random number.

The basic idea behind our implementation of the chase is summarised in Algorithm 5.

Alg. 5 Basic structure of the chase implementation

```
global callers, oldId, seen, waiting;
procedure CHASE(data, id, localId, callingPeer)
    callers[localId] ← callingPeer;
    fillLocalStubs(data);
    dataAdded ← localchase(P2PMappings);
    dataAdded ← localchase(localConstraints) || dataAdded;
    if propagate_chase_to_target_neighbours(dataAdded, seen) then
        seen ← seen ∪ id;
        waiting ← ∅;
        newLocalId ← createNewLocalId();
        oldId[newLocalId] ← localId;
        for all target neighbour tn do
            data ← getCertainAnswersForMappingsTo(tn);
            waiting ← waiting ∪ tn;
            tn.chase(data, id, newLocalId, this);
    else
        seen ← seen ∪ id;
        if dataAdded < 0 then
            callingPeer.chaseFinished(id, localId, this, "inconsistent");
        else
            callingPeer.chaseFinished(id, localId, this, false);


procedure CHASEFINISHED(id, localId, callingPeer, error)
    if error then
        callers[oldId[localId]].chaseFinished(id, oldId[localId], this, error);
    waiting ← waiting \ callingPeer;
    if waiting = ∅ then
        callers[oldId[localId]].chaseFinished(id, oldId[localId], this, error);
```

We now shortly comment on the implementation of the chase in the CoDE system. It iterates over the set of given dependencies until either none of them is violated any more, or an inconsistency is detected. For each dependency, in a first step it is checked whether there exist facts violating the dependency. This is done by evaluating the query on the left hand side of the dependency over the database instance, and restricting the result according to the type of the dependency: For a TGD, all tuples that are also contained in the result of evaluating the query on the right hand side of the dependency over the instance are removed. For an EGD, all pairs of equal values are removed. If the remaining result still contains some tuples, then the dependency is violated.

Note that this first step can be expressed as a single SQL query. Therefore, every dependency is transformed into an SQL query that selects all violations of this dependency. In a second step, these violations are tried to be repaired, by either unifying the terms violating an EGD or by materialising facts such that a TGD is no longer violated. (A detailed description of this idea and the implementation can be found in [71].) Thereby, the second step is performed as a kind of batch processing for all violations of a dependency found in the first step. While in the definitions of the chase and the chase sequence, in every chase step only the violation of a single tuple is repaired, and then the violating tuples are recalculated, in CoDE all violations of a dependency are (tried to be) repaired at once. Due to the semantics of data exchange, this returns the same results but is far more efficient.

In our implementation we distinguish between three different chase types, referred to as *echase*, *eichase* and *virtual chase* (or *tchase*): The echase and eichase are the implementations of the E-CHASE and EI-CHASE as described in Section 5.4 and and Section 5.5 respectively. Hence they are used for reasoning in PDE- and PDEI-systems (deciding $\mathcal{S}$-consistency, checking states for being $\mathcal{S}$-admissible, computing universal $\mathcal{S}$-solutions). The tchase on the other hand is used to denote the computation of E-CHASE$(\tau(\mathcal{S}), B)$, needed for query answering over a stratified PDEI-system $\mathcal{S}$ and a state B for $\mathcal{S}$ (see Section 5.6).

echase

The echase is the simplest of the three types. Query evaluation for checking dependencies for violations reduces to simply evaluating the queries from the dependency definition over the current state. Also the batch processing as described above is valid, because repairing one violation of a dependency cannot solve another violation of the same dependency: A violation is either a tuple that is contained in the answer of one query, but not in another, or a pair of values that are not equal. Hence unifying two values cannot make any other pair of unequal values equal. Also adding facts to the instance such that a certain tuple is contained in the answer to a conjunctive query is done in such a way that only this tuple is added to the answer of the query. Hence all the other results still remain violations. Therefore `localchase()` from the CoDE system can be used without changes for implementing the local chase.

Also deciding whether to issue a request to the target neighbours to continue the chase or not is easy: Whenever a chase instance arrives for the first time, it is propagated, just to ensure that all peers (transitively) reachable from the initialising peer by P2P mappings check their states and outgoing mappings. If a peer receives a chase instance it has already propagated before, it only forwards the chase again if chasing the incoming TGDs or the local chase changed the database instance of the local peer (in fact, if chasing the P2P mappings did not cause any changes, then the local chase is omitted, as none of its rules can be applicable, since otherwise this rule would have been applied during the previous local chase).

Moreover, since all data implied by the dependencies has to be materialised and because of the monotonicity of conjunctive queries, the same chase instance can be processed by several peers in parallel.

virtual echase (tchase)

As stated above, the tchase is used to compute E-CHASE($\tau(\mathcal{S}), B$), which is needed for query answering over PDEI-systems. With respect to the overall structure (i.e. how the chase is propagated, how termination is reached and how the rules are applied), the tchase is equivalent to the echase. There are only two important differences in the semantics of the rules: First, the facts created and the equalities enforced by rule applications during a run of the tchase shall not be materialised permanently on the instance, but only temporarily. It must be possible to undo the effects of the tchase after query answering. Second, instead of using the TGDs $\in \mathcal{M}_I$ as defined by the user, $\mathtt{Expand}(\mathcal{M}_I, J)$ (where J denotes the inclusion dependencies from $\mathcal{C}_I$) has to be used. (Note that for enforcing the key constraints from $\mathcal{C}_I$ it is not necessary to rewrite the lhs of the dependency because of the separation property.)

It was already shown in Section 6.3.3 how we deal with this materialised virtual data. We extended `localchase()` by a parameter controlling whether the result of `localchase()` is written directly into the database instance of a peer or whether it is stored as virtual data. During a tchase, `localchase()` sets `pdei_chasetype` for the created tuples to "1", and equalities are not inserted in the base tables, but are stored in the `PDEI_VCHASE_MAPPING` table.

By rewriting the P2P mappings, instead of a single conjunctive query, the left hand side of the TGD consists of a UCQ. Each of the subqueries of the UCQ can be rewritten by the same XSLT rule used by CoDE for rewriting the conjunctive queries of TGDs. We therefore extended the XML format used for defining the TGDs such that it is possible to define several subgoals for the lhs, and changed the XSLT template such that each of the goals is rewritten into an SQL query that are then combined using `UNION`.

Whenever a new tchase instance (i.e. an instance with a yet unseen id) arrives at a peer, before its performing, all virtual data currently stored at the peer is dropped. This means all facts with `pdei_chasetype` = "1" are deleted together with all the content of the `PDEI_VCHASE_MAPPING` table. Because of this, there must not be two tchase instances with different ids active at the same time within the whole PDMS.

The virtual data materialised by a tchase can also be deleted explicitly by the user, either only in a single peer or in the complete system (which is implemented by flooding the system with the request to drop all virtual data).

eichase

For computing the eichase (i.e. computing the result of EI-CHASE), three aspects need to be considered: First the computation of $\mathtt{certain}_{\mathtt{Null}}(q, \mathcal{S}, B)$ and $\mathtt{certain}(q, \mathcal{S}, B)$ for a PDEI-system $\mathcal{S}$ with $\mathcal{C}_E = \mathcal{M}_E = \emptyset$. Second, when to stop propagation of the eichase. And third, that in each application of a chase rule, only a single violation may be repaired, meaning that the batch processing of violations as implemented by CoDE is not allowed any more. This is because facts materialised to solve one violation may, by virtual dependencies, imply data that resolves further violations. Hence the list of tuples that belong to the answer of the lhs of a TGD but not to the rhs of a

TGD has to be updated every time a violation has been resolved, to ensure not to materialise data no longer required.

Example 6.8. To clarify the problem, assume the following example:
Consider the PDEI-system $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}_E, \mathcal{M}_E, \mathcal{C}_I, \mathcal{M}_I \rangle$ with

- $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$ with $S(\mathcal{P}_1) = \{a/1\}$, $S(\mathcal{P}_2) = \{b/1\}$, $S(\mathcal{P}_3) = \{c/2, d/1\}$,
- $\mathcal{C}_E = \mathcal{C}_I = \emptyset$,
- $\mathcal{M}_E = \{a(X) \rightarrow b(X)\}$, and
- $\mathcal{M}_I = \{b(X) \rightarrow d(X); b(X), c(X,Y) \rightarrow b(Y)\}$.

and a state $B = \{a(1), a(2), c(1,2)\}$. Evaluating $\{X \mid a(X)\}$ gives $\{1,2\}$, which are both not in the result of $\{X \mid b(X)\}$. Hence $b(1)$ and $b(2)$ could be added to satisfy $\mathcal{M}_E$. But after adding $b(1)$, $b(2)$ is already implied by the mappings in $\mathcal{M}_I$, and hence should not be added any more (according to the definition of EI-CHASE). ◇

We implemented this behaviour by extending the implementation of `localchase()` by another parameter. If called accordingly, instead of selecting all violating tuples during dependency checking, only one tuple is selected (simply by adding `LIMIT 1` to the created SQL query). Moreover, because recomputing $certain_{Null}(q, \mathcal{S}^I, B)$ and $certain(q, \mathcal{S}^I, B)$ is a little bit more complicated, `localchase()` stops and returns not only if no more violations are detected, but also as soon as one rule has been applied.

Also to avoid materialisation of facts already implied by data added elsewhere, in our implementation we do not allow the eichase to run in parallel on different peers, but for every chase instance there must exists a single point in the P2P network where the chase is currently active. This is, there exists a kind of token, and only the peer currently possessing the token is allowed to apply a chase rule. Therefore, a chase instance is not propagated in parallel to all target neighbours of a peer, but only to one peer at a time. Once one target neighbour has finished and returns the control, the chase is propagated to the next target peer.

Under our assumptions made for the semantics of $certain_{Null}(q, \mathcal{S}^I, B)$ justified in Section 6.2, the computation of $certain_{Null}(q, \mathcal{S}^I, B)$ equals the computation of $certain(q, \mathcal{S}^I, B)$, except that in the latter case in an additional last step all tuples containing labelled nulls are removed from the result. The implementation is based on Theorem 5.43 and the fact that for $\mathcal{S}^I$, every consistent state is also admissible. By these results, $certain_{Null}(q, \mathcal{S}^I, B) =$ E-CHASE$(\tau(\mathcal{S}^I), B)^{rewrite(q,\mathcal{J})}$, if E-CHASE$(\tau(\mathcal{S}), B) \neq$ FAIL. I.e. $certain_{Null}$ is computed by evaluating the perfect rewriting of q over an instance of a peer $\mathcal{P}_i$ when the data implied by the i-mappings and the equalities enforced by the key constraints in $\mathcal{C}_I$ are temporarily materialised. Therefore, also for the eichase the `localchase()` implementation of CoDE can be used and needs only a small extension for handling UCQs (i.e. concatenating several conjunctive queries by `UNION`).

To speedup the eichase, the result of the tchase is not removed after each computation of $certain_{Null}(q, \mathcal{S}^I, B)$ ($certain(q, \mathcal{S}^I, B)$) and recomputed for

the next rule application. Because of the monotonicity of conjunctive queries, the results are kept, and only an update of the tchase is performed.

Once no more chase rule of EI-CHASE is applicable, in a final step the resulting state has to be checked for $\mathcal{S}$-consistency. This is again done by computing E-CHASE$(\tau(\mathcal{S}^I), B)$. If the result is FAIL, then the state is not consistent and the result of EI-CHASE is FAIL. Otherwise the resulting state is consistent.

Therefore a typical run of an eichase instance initialised by peer $\mathcal{P}_i$ consists of the following steps: First $\mathcal{P}_i$ asks its source neighbours to initialise a tchase. Once it receives the information that all requested tchases have finished, it starts chasing its local constraints. After each rule application, the local chase is interrupted, and $\mathcal{P}_i$ initialises a tchase with the same id as the tchases it requested at the beginning. This has the effect, that no new tchase is started from scratch, but it is only tested whether some additional data is implied because of the rule application. Once the tchase has finished, $\mathcal{P}_i$ runs the local chase again. This is repeated until no local constraint is applicable any more. Then $\mathcal{P}_i$ propagates the eichase to its first target neighbour ($\mathcal{P}_j$), together with the results of $certain(q, \mathcal{S}^I, B)$ for the P2P mappings between them. $\mathcal{P}_j$ chases the P2P mappings, but also after each rule application updates the results of the tchase (by initialising a tchase with the same id as before). Once the tchase has finished, $\mathcal{P}_j$ requests $\mathcal{P}_i$ to resend the results of the queries of the P2P mappings (as they might have changed because of cycles in P2P topology). After having finished the chase of the P2P mappings, $\mathcal{P}_j$ chases its local constraints just as $\mathcal{P}_i$ and then forwards the chase. Once $\mathcal{P}_i$ is informed by $\mathcal{P}_j$ that it has finished the eichase, $\mathcal{P}_i$ propagates the eichase to its second target neighbour. If it receives from its last target neighbour that the eichase has finished, the eichase terminates.

It now only remains to clarify when the eichase is forwarded and when to stop the propagation. Thereby, it is necessary to propagate the eichase more often than the echase, because stopping at a peer (that is no reached for the first time) just because neither the application of the P2P mappings nor the local chase changed the local instance is no longer correct. As long as there are outgoing i-mappings, it might be the case that new data has been propagated by them, that make some rule applicable in another peer. Therefore propagation of the eichase stops if no rule is applicable when chasing the P2P mappings and local dependencies and there are no outgoing i-mappings, or if in a loop the same peer is reached again without that there has been any applicable rule in this circle. To recognise such a situation, when propagating the eichase, also a list of peers is forwarded that contains the ids of all peers where no rule was applicable since the last rule application. When at a peer some rules are applicable, the content of this list is deleted. Once a peer finds itself on this list, it knows that it does not need to propagate the chase any further. (Obviously, if in a peer no rule was applicable, the eichase is only propagated along i-mappings.)

6.3.5 *Query Answering*

At the moment, only answering of conjunctive queries is supported by our implementation.

For answering a query posed over a peer $\mathcal{P}_i$ in a PDE-system, given some state B, it is necessary to compute a universal $\mathcal{S}$-solution for B. This is done by running the echase. However, it is yet within the responsibility of the user to initialise the echase at the correct peers such that all data implied by some mappings is materialised in the instance of $\mathcal{P}_i$. Given a universal $\mathcal{S}$-solution for B, computing the answers to a query q is simply done by rewriting the given conjunctive query into a corresponding SQL query that selects only tuples not containing labelled nulls. This query is then issued to the database using the `SQLQuery` class provided by the spring framework.

For query answering over PDEI-systems, also in a first step a universal $\mathcal{S}$-solution needs to be computed. This is done by the eichase. Again the user has to take care of initialising an eichase in all peers required to guarantee the correctness of the result. In a second step, the data implied by the virtual mappings is temporarily materialised by the tchase. Unlike for echase and eichase, our implementation allows a peer to request its source neighbours to initialise a tchase instance. The reason for this is that we assume that the peers aim to keep the state of the PDEI-system $\mathcal{S}$-admissible, i.e. that if there is some change in the database instance of a peer, it immediately calls the correct chase to propagate these changes. The virtual data however is only used for query answering, and is therefore only computed when requested. If the requested tchases have finished, first a perfect reformulation of the given query w.r.t. the local inclusion dependencies is computed, and then the query is transformed into SQL and issued to the views providing access to the data including the virtually implied data.

For the implementation of eichase, it is important to note that computing answers over PDEI-systems with $\mathcal{C}_E = \mathcal{M}_E = \emptyset$ can be done just with the use of the echase and query rewriting. Hence computing $certain_{Null}$ and the certain answers during the eichase can be reduced to query answering over a PDE-system, and does not require query answering over a PDEI-system (which is in turn based on the eichase).

Queries are posed in the style of $a_1(A_{1_1}, \ldots, A_{1_n}), \ldots, a_n(A_{n_1}, \ldots, A_{n_n}) \rightarrow result(B_1, \ldots, B_n)$ ($\{B_1, \ldots, B_n\} \subseteq \{A_{1_1}, \ldots, A_{n_n}\}$) to the system. Both, computing their perfect rewriting (if necessary) and their transformation into an SQL query are implemented purely in Java.

6.3.6 *Requesting a Chase*

As mentioned in the previous section, the implementation supports a peer to request its source neighbours to initialise a tchase. Whenever a peer receives such a request, it propagates it to its own source neighbours, until either a peer without source neighbours is reached, or a cycle is detected, i.e. a peer receives a request it has already forwarded. To detect such cycles, the request propagated by the peers contains a list of all peers that have already forwarded

the request. Moreover, to avoid problems of concurrent tchase instances, the id of the tchase instance is defined by the peer originally requesting the tchase and is also propagated together with the request. Hence all tchase instances started due to such a request have the same id and are therefore considered to be one instance.

6.3.7 *Communication Between the Peers*

Listing 6.1: Java Class implementing the Webservice Interface

```
@WebService
public class WebServiceListener {

  ...

  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
  public void registerNeighbour(String xmlData){...}

  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
  public void chase(String xmlData){...}

  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
  public void resendChaseData(String globalChaseId, String localChaseId, String
      recPeerId, String tChaseId){...}

  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
  public void finishedChase(String peerId, String globalChaseId, String
      localChaseId, String success){...}

  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
  public void chaseRequest(String xmlData){...}

  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
  public void finishedChaseRequest(String peerId, String requestId, String
      success){...}

  @SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
  public void propagateResetVirtualData(String requestId){...}
}
```

The communication between the peers is implemented using webservices: Every peer publishes the interface depicted in Listing 6.1 (we omit the WSDL description due to space restrictions) as webservice methods, and invoke the appropriate webservice of other peers to exchange information. We opted for using webservices because for the implementation of a prototype their

great flexibility (especially their platform independence and the existing support for a variety of programming languages and technologies) and low implementation costs outweight the fact that they are certainly not the most efficient solution for this task.

The server side of the webservice is created using `wsgen`[6] from an annotated Java object implementing the interface shown in Listing 6.1, and is deployed directly by the Java runtime environment, that also creates and publishes the WSDL description at runtime.

On the client side we use Apache Axis [24] for invoking the webservices. The code for the client stubs is created using the axis tool `wsdl2java`. Conceptually, except `registerNeighbour`, all invocations of a webservice at peer $\mathcal{P}_i$ from a peer $\mathcal{P}_j$ are one-way calls. If $\mathcal{P}_j$ needs to be informed about some result or just the fact that computation in $\mathcal{P}_i$ has finished, this is done by $\mathcal{P}_i$ invoking a corresponding webservice method of $\mathcal{P}_j$. The reason for this is that it may take quite some time until such an answer is sent, and it would therefore make no sense for $\mathcal{P}_j$ to actively wait for a response. However, technically we use blocking two-way calls for webservice invocation and mimic the behaviour of a one-way call in the webservice method by just starting a new worker thread and returning immediately after. The worker thread then performs all requested operations and, if required, invokes the corresponding service of $\mathcal{P}_j$ when finished. We had to implement it this way because we encountered some problems using real one-way calls as offered by WSDL: There seems to be no way to send one-way calls synchronously w.r.t. to the control flow of the invoking peer. But when sending them asynchronously, it happens that the invoking peer terminates before sending all data has finished. In this case the connection is closed although not all data has been sent to the invoked peer.

6.3.8 *Further Design Decisions*

In the following we describe briefly more details of the implementation.

Error and Inconsistency Handling

As already indicated by the description of the chase, and because for inconsistent states the result of the chase is only defined as FAIL, if an inconsistency is discovered during a chase, the resulting content of the peer instances is not defined: As soon as an inconsistency is discovered, the chase stops, and no more changes on the database instances are made. Hence everything materialised up to this point remains materialised, but which data was actually added is undefined. Only the information that the current state of the system is inconsistent is propagated backwards to the peer that has initialised the chase where the user is informed.

The same holds if an error occurs during a chase that makes it impossible to finish the chase. Also in this case the chase is aborted, but the effects of the chase so far are not discarded (i.e. the tuples inserted are removed and unifications are made undone), but only the user is informed about the error.

6 `https://jax-ws.dev.java.net/nonav/2.1.2/docs/wsgen.html`

No Core Computation

Although, as stated, the implementation is based on the CoDE system of [71], core computation is not considered in the implementation because the results of [71] indicate that although solvable in polynomial time, core computation is a very expensive task. Also the fact that for PDEI-systems the core need not be unique [30] makes core computation for PDEI-system less attractive. Moreover, the core computation as implemented in CoDE assumes the target instance (which corresponds to the peer instance in our implementation) to be empty before the chase. Since this is not the case for the PDMS, appropriate adoptions of core computation would have been necessary.

Therefore, given a PDE- or PDEI-system $\mathcal{S}$ and a state for $\mathcal{S}$, not the (or a) $\mathcal{S}$-core but only a universal $\mathcal{S}$-solution can be computed.

User Interface

Every peer provides a very basic command-line interface, that allows to control the peer, but does not give much feedback or information to the user. For example if the user initiates a chase instance, then it is informed once it has finished, either with success or failure. This just enables the user to access the basic functionality provided by the peer as described in this chapter.

Summary of Restrictions of the Current Implementation

We shortly summarise some simplifying assumptions that have been made for the implementation, because without them the system would have exceeded the scope of this thesis. Most of them concern convenient functionality that affects the handling of a peer, but that does not have any effect on the semantics of the PDMS.

NO INCREMENTAL CHASE OF P2P MAPPINGS: Every time a P2P mapping $q_i \rightarrow q_j$ is chased, then all evaluation results q_i are sent to the target peer, neglecting that tuples that have already been exchanged in a previous iteration step need not be sent again, as they are already assured to not cause a violation.

NO GLOBAL ERROR HANDLING: There exists no global error handling strategy. If the chase fails at some peer, then this information is propagated backwards to the peer initialising the chase instance, but nowhere else in the system. Also the state of the system is undefined after an error, as every peer stops execution as soon as it is informed about the error, and the state is not changed any more.

NO TIMEOUT FOR CHASE INSTANCES: There exists no timeout for a peer waiting for a target neighbour to finish the chase, hence peers may wait infinitely (e.g. if some peer crashes). However, the peer remains responding to requests of other peers. It may only be that some crash of a peer will not be detected.

UNCHECKED PREREQUISITES: The constraints on the mappings (weakly acyclic, stratified, ...) are not checked for being satisfied. This has to be ensured by the user who configures the system.

PEERS MUST NOT GO OFFLINE: The implementation cannot yet handle the situation if a peer leaves the system. Adding peers to the network on the other hand is no problem.

ONLY `varchar` IS SUPPORTED: At the moment, only the datatype `VARCHAR` is supported, just to avoid problems when defining dependencies between different datatypes (e.g. equalities in a conjunctive query). Due to the `VARCHAR` implementation of many database systems, moreover the length of all fields is restricted to some fixed size.

NUMBER OF PARALLEL CHASES: While it is possible to run several echase instances at the same time in the system, this does not hold for tchase and eichase: Within the whole PDMS, only tchases with the same id are allowed to run at the same time, otherwise they will not return the correct results. For eichase, at every time there must be at most one instance in the system. Thereby this has to be guaranteed by the user, since the system will allow for several instances at the same time.

`dx_id`-COLUMN: If a relation of a peer schema does not possess a column `DX_ID` that is a primary key of the relation, this column will be created at startup. However, thereby all data in the relation will be deleted.

PEER IDENTIFICATION: Every peer is assumed to have an unique id (within the whole PDMS), and it is assumed that all information about the neighbours of a peer are known at configuration time.

PROBLEM WITH COMPUTING $certain_{null}$: The problems with the EI-CHASE as described in Section 6.2.2 with computing $certain_{Null}$ are unresolved, and superfluous facts may be added to the peer instances.

6.4 IMPLEMENTATION DETAILS

6.4.1 *Configuration of a Peer*

For TGDs and EGDs definitions, we use the XML format already applied in the CoDE system and described in detail in [71]. There also the internal XML representation of dependencies is described.

Although key constraints and foreign keys are special cases of TGDs and EGDs, we provide an own XML format for the specification of the i-constraints. One the one hand, this makes it easier for the user to define them (instead of having to translate them first to TGDs and EGDs), and on the other hand makes their processing easier as well.

We demonstrate this format on the following two dependencies:

- $BooksDep(ISBN, Title, Author, Status) \subseteq BooksUB(ISBNId, FirstName, LastName, Booktitle)$

- $key(Books_{UB}) = \{ISBN, Title\}$.

They are specified in the configuration files by

```
<pdei:iDep>
  <pdei:lhs relation="BooksDep">
    <pdei:field>ISBN</pdei:field>
    <pdei:field>Title</pdei:field>
  </pdei:lhs>
  <pdei:rhs relation="BooksUB">
    <pdei:field>ISBNId</pdei:field>
    <pdei:field>Booktitle</pdei:field>
  </pdei:rhs>
</pdei:iDep>

<pdei:kDep relation="BooksUB">
  <pdei:keyAttribute>ISBNId</pdei:keyAttribute>
  <pdei:keyAttribute>Booktitle</pdei:keyAttribute>
</pdei:kDep>
```

The key constraints are translated into the format of EGDs (and later further into the intermediate formats) that are then used during the tchase. For example the key constraint from above is rewritten to the egds

```
<dx:dependency>
  <dx:premise>
    <BooksUB ISBId='V1' Booktitle='V2' FirstName='V3'/>
    <BooksUB ISBId='V1' Booktitle='V2' FirstName='V4'/>
  </dx:premise>
  <dx:conclusion>
    <dx:eq dx:a='V3' dx:b='V4'/>
  </dx:conclusion>
</dx:dependency>

<dx:dependency>
  <dx:premise>
    <BooksUB ISBId='V1' Booktitle='V2' LastName='V3'/>
    <BooksUB ISBId='V1' Booktitle='V2' LastName='V4'/>
  </dx:premise>
  <dx:conclusion>
    <dx:eq dx:a='V3' dx:b='V4'/>
  </dx:conclusion>
</dx:dependency>
```

which are then handed over to the CoDE system.

The foreign key dependencies on the other hand are parsed and the relation and attribute names are stored in a Java object (of class `InclusionDependency`) such that they are easily accessible for query rewriting.

Concerning the XML namespaces of the elements, all elements and attributes added for the implementation of our prototype system are marked

by the namespace `http://www.dbai.tuwien.ac.at/PDEI/0.1`, while all elements and attributes taken from the CoDE implementation keep the `http://www.dbai.tuwien.ac.at/DataExchange/0.1` namespace. This convention is used throughout the implementation, not only for configuration.

6.4.2 *Main System Classes*

In the following we introduce the most important classes of the PDEI-system. We omitted most of the important classes from CoDE, since their description can be found in [71]. The class diagram in Figure 11 shows these classes and their relationship.

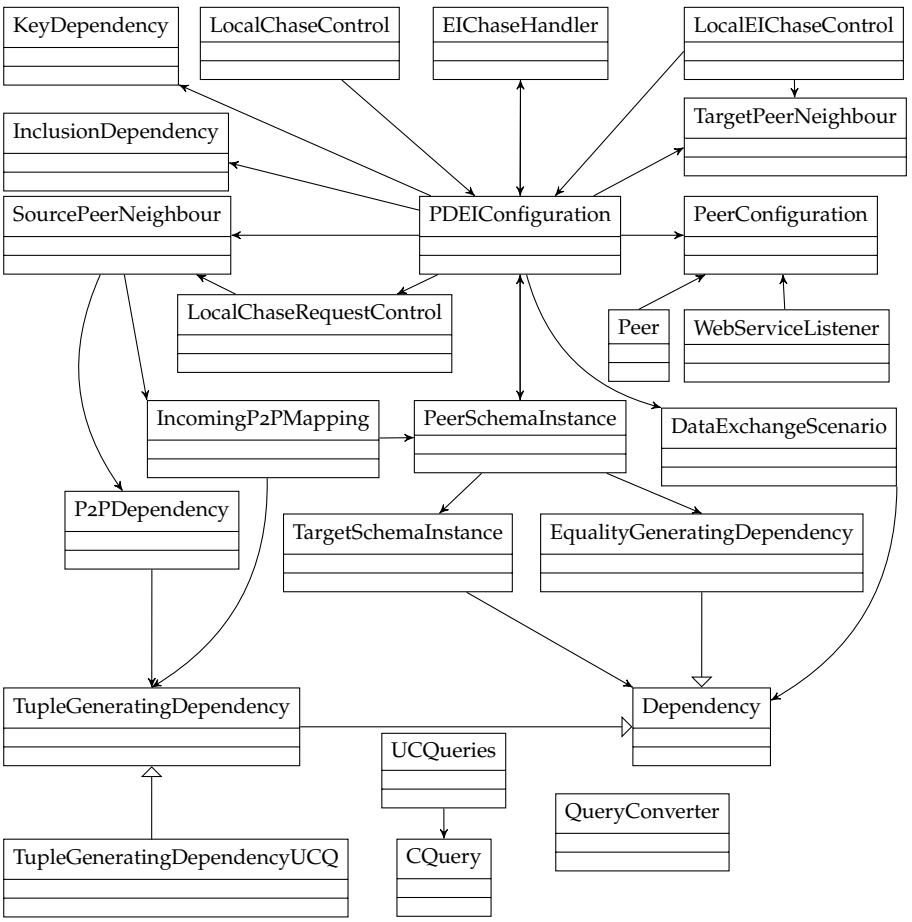

Figure 11: Class diagram of some of the most important classes of our prototype implementation.

- `Peer`: The main class of the program. Starts up the program instance, initialises the webservice and an instance of `PDEIConfiguration` for

coordinating the peer with its neighbours. It also implements the main method.

- `PeerConfiguration`: Stores all configuration information except mappings and constraints. An instance of this class is the connector between the `Peer` object representing the program implementing a peer and the `PDEIConfiguration` keeping track of and coordinating the distributed algorithms.
- `PDEIConfiguration`: Central class for all P2P related tasks. It coordinates a peer with its neighbours. Stores all information about mappings and neighbours. Coordinates the distributed algorithms: All requests received from other peers are dispatched by an object of this class. Hence it is also responsible for coordinating the chase instances.
- `PeerSchemaInstance`: Provides access to the database. Offers an extended functionality of the encapsulated `TargetSchemaInstance` object from CoDE.
- `SourcePeerNeighbour`: Represents a source neighbour of the current peer. The objects are responsible for the communication with the corresponding peer. Managed by the `PDEIConfiguration`.
- `TargetPeerNeighbour`: Represents a target neighbour of the current peer. The objects are responsible for the communication with this peer: They select the data needed for chasing P2P mappings from the database, create the messages to send and also sends them. Managed by the `PDEIConfiguration`, but used also from the different objects responsible for the chase coordination.
- `KeyDependency, InclusionDependency, P2PDependency`: Represent the corresponding dependency and offer functions that provide easy access to the required information about the dependency.
- `IncomingP2PMapping, OutgoingP2PMapping`: Implement all functionality needed to allow using the local chase for chasing P2P mappings. Create the new TGDs and manage the table where the result of the query over the source peer are stored for the local chase.
- `LocalChaseControl, EIChaseControl, LocalEIChaseControl, LocalChaseRequestControl`: Helper classes for keeping track about the state of the chase instances, for coordinating the propagation of chase instances and especially to control the eichase instances (helping to coordinate the request of an initial tchase, the update of the tchase after each rule application, ...). Managed by the `PDEIConfiguration`.
- `UCQueries, CQuery, QueryConverter`: Together with some more helper classes they are responsible for query rewriting and transforming conjunctive queries into SQL queries.

- `WebServiceListener`: Implements the interface provided by each peer for communication with the other peers. Forwards the request via the `PeerConfiguration` to the `PDEIConfiguration`, where the request is processed.

6.4.3 *Communication Formats*

While most of the parameters for the webservice functions introduced in Section 6.3.7 are self explaining, the three arguments consisting of XML strings need further explanations.

For the `registerNeighbour` function, the XML string should be formatted like:

```
<pdei:connect xmlns:dx="http://www.dbai.tuwien.ac.at/DataExchange/0.1"
              xmlns:pdei="http://www.dbai.tuwien.ac.at/PDEI/0.1">
  <pdei:peer>
    <id><!-- ID --></id>
    <ip><!-- IP --></ip>
    <port><!-- PORT --></port>
  </pdei:peer>

  <dx:dependencies>
    <!-- list of e-mappings in the XML format as defined in
        configuration file -->
  </dx:dependencies>

  <pdei:dependencies>
    <!-- list of i-mappings in the XML format as defined in
        configuration file -->
  </pdei:dependencies>
</pdei:connect>
```

When sending a chase request to a source peer, the XML string should look like:

```
<pdei:chaseRequest xmlns:pdei="http://www.dbai.tuwien.ac.at/PDEI/0.1"
        pdei:requestId="!?requestId"
        pdei:peerId="!?localPeerId" <!-- id of the peer the request was
             sent/forwarded from -->
        pdei:chaseType="!?chaseType"
        pdei:virtualChase="!?isVirtual">
  <peer id="!?localPeerId"/>
  ... <!-- list of peers that have already processed the request -->
</pdei:chaseRequest>
```

And when forwarding a chase instance, the correct XML format is :

```
<pdei:chase xmlns:pdei="http://www.dbai.tuwien.ac.at/PDEI/0.1"
        pdei:chaseId="!?chaseId" " +
```

```
        pdei:peerId="!?localId"
        pdei:localChaseId="!?localChaseId"
        pdei:chaseType="!?chaseType"
        pdei:virtualChase="!?isVirtualChase"
        !?freeAdditionalAttributes ">

  <!-- only for eichase -->
  <history>
    <!-- list of peers visited since the last dependency was
        applicable -->
    <peer id="!?peerId1"/>
    <peer id="!?peerId2"/>
    ...
  </history>

  <!-- for all chasetypes -->
  <data>
    <!-- for each outgoing P2P mapping to the target peer -->
    <pdei:mappingResult pdei:id="!?idOfDependency">
      <!-- for each result of the query -->
      <row>
        <!-- for each selected attribute -->
        <!?attributeName>
          <!-- value -->
        </!?attributeName>
        ...
      </row>
      ...
    </pdei:mappingResult>
    ...
  </data>
</pdei:chase>
```

(In the previous examples, "!?" prefixed to an attribute value or element name are used to denote placeholders for the corresponding value.)

6.4.4 *User Interface*

The following list states the commands that can be issued to a peer through the command-line interface. (<*conjunctive query*> is an expression of the form $q_1(A_{1_1},\ldots,A_{1_k}),\ldots,q_n(A_{n_1},\ldots,A_{n_k})$ -> $result(A_{i_1},\ldots,A_{i_j})$ where $q_1,\ldots,q_n$ are relational symbols from the local peer-schema, A_{l_h} are either variable names or constants, where constants are circumscribed by single quotes, and all A_{i_l} either appear also on the left hand side or are constants. The name of the atom on the right hand side — $result$ in the example above — can be freely chosen.)

CONNECT Tries to send the P2P-mappings to the source peers (see Section 6.3.1).

ECHASE Initiates an echase (see Section 6.3.4).

EICHASE Initiates an eichase (see Section 6.3.4).

TAUCHASE Initiates a tchase (see Section 6.3.4).

INITTAUCHASE Requests the source neighbours to initiate a tchase (see Section 6.3.6).

LRW <*conjunctive query*> Rewrites the given conjunctive query according to the local virtual inclusion dependencies and outputs the result.

CA <*conjunctive query*> Rewrites the given conjunctive query according to the local virtual inclusion dependencies and executes the result on the local database instance and outputs the result. Tuples containing nulls are removed from the output. Can be used to compute the certain answers if called on an $\mathcal{S}$-admissible state and if `inittauchase` has been called before.

NCA <*conjunctive query*> The same as `ca`, but without removing tuples containing null values from the output.

RESETVIRTUALDATA Removes all temporarily materialised data and mappings from the local database (see Section 6.3.3).

GLOBALRESETVIRTUALDATA Removes all temporarily materialised data and mappings from the local database and asks all neighbouring peers to do the same (see Section 6.3.3).

SQL <*sql command or query*> Allows to directly execute an SQL command on the local database or to query the local database directly.

6.5 DISCUSSION

6.5.1 *System Evaluation*

We did not perform any systematic evaluation of the performance of our implementation yet and therefore can only report on some preliminary tests. These first results suggest that while the echase (hence also the tchase) may perform reasonable well, the performance of the eichase seems not to be good enough, but it seems to be too slow.

The echase for even small instances of the PDMS with up to 6 peers and only about 100 facts in the initial state takes up to 40 seconds[7]. However, as our focus for the prototype was not on performance and when considering the evaluation results of the CoDE system presented in [71], this still lies within the expected, reasonable scope. In these settings, also the eichase takes

7 Intel Core 2 Duo 6600 with 2.4GHz, 2GB RAM, Ubuntu Linux and HSQLDB, neglectable network latency because all peers are running on the same PC.

only about 50% longer than the echase, which still could be regarded to be acceptable when taking into account all the additional requirements for the eichase. But when increasing the number of facts in the initial state, even for very small settings consisting only of a single peer, two relations and a single TGD, the eichase does not scale reasonable with the number of facts.

But our observations suggest so far that these negative results for the eichase occur only if data exchange and virtual dependencies are combined. For PDEI-systems with $\mathcal{C}_E = \mathcal{M}_E = \emptyset$, i.e. for pure P2P data integration systems, as there is no rule to apply during eichase, query answering behaves just as in PDE-systems. Hence maybe the combination of data exchange and virtual mappings is problematic for the performance in practise, although being still tractable in theory.

Not surprisingly, all types of chases scale very badly w.r.t. the number of peers and the number of mappings in the system (just as one would expect as the chase is only polynomial in the data complexity).

However, more systematic tests are necessary to confirm these first observations and to try to identify relevant flaws that are responsible for the performance issues, either in the implementation or in the semantics. Also the effect of the database system used should be studied.

Comparing our implementation with other prototype systems is difficult, not only because this would require extensive tests of our system, but also because of the differences in the implemented approaches. However, as already stated, from the first results it seems as if (for comparable scenarios) the performance of our implementation is not much below the performance of CoDE. This indicates that building the PDMS on top of CoDE causes only a justifiable overhead and that the performance of our prototype can be regarded to be satisfactory. The only other prototype PDMS implementing a similar semantics as our PDMS is coDB [26]. Also here our first tests suggest that although being not competable, at least for smaller systems (we do not have data for bigger ones) the difference is still within reasonable bounds. Comparisons with other systems (like e.g. Piazza) are questionable, as they are not only based on a different semantics, but for example in Piazza query answering is implemented by a centralised algorithm which dramatically outperforms our implementation.

Concerning semantic properties of the PDMS, the framework of De Giacomo et al. provides a very powerful and well defined tool, whose advantages have been already discussed in the previous chapters. However, using this framework for implementing a real, dynamic PDMS, several issues arise that are beyond the scope of [30].

A major problem w.r.t. the data exchange dependencies are updates of the database instance at a peer. If the original data has been propagated to another peer, the update is often inconsistent with this previously forwarded data. Although being completely correct and also reasonable from a semantical point of view, it seems to be inconvenient in many situations. (From a semantical point of view, once the data has been exchanged, it is no longer within the scope of the peer where it originated, but is just part of the other peer instance. If by some update the original and the exchanged data become

inconsistent, then the two peer schemas simply are no longer consistent. In practise however, this would limit the use of data exchange dependencies to data known to be stable.)

This problem correlates with the question when to perform data exchange actually, i.e. when to apply the chase. Possibilities are: every time a change in one of the peer databases happens or only when a peer explicitly requests to perform a data exchange. This is a critical point w.r.t. the semantics of a dynamic PDMS.

Another aspect worth mentioning is the effect of preferring virtual dependencies over data exchange dependencies. When the same fact is implied by a virtual dependency and a data exchange dependency, then the data is not materialised. Again, when considering one fixed setting and one given state, this semantics is probably the best choice. However, in dynamic settings, where both the system configuration and the state may change, this can lead to some unintended (at least in most of the cases) effects. For example if a fact is implied on an instance of some peer $\mathcal{P}_i$ by both, virtual and data exchange dependencies. When running an eichase, this fact is not materialised. If, because of some change in the PDMS, this fact is no longer implied neither by virtual nor by data exchange dependencies, and a query is posed over $\mathcal{P}_i$, then this fact is not taken into consideration for query answering, as it is not contained in the instance of $\mathcal{P}_i$. Although being exactly correct according to the defined semantics, this might not be the expected effect when defining a data exchange dependency.

Concerning the restrictions required to maintain tractability of the chase, it might be of interest to apply the same constraints to the i-constraints as to the e-constraints, as this may be the more flexible approach. However, further evaluations are necessary to really get a clue about this.

Finally the XML configuration of a peer allows an easy and flexible setup of different scenarios for the evaluation of certain concepts. Moreover, for small size test scenarios although of the restrictions presented in Section 6.3.8 the handling of a single peer and the complete network is convenient.

6.5.2 *Open Issues and Further Improvements*

The only known problem with the current implementation concerns data manually added by a user to a peer instance while the peer is active. For efficient core computation, CoDE stores a lot of additional information to every fact in the database. Although not using the core computation capabilities, also the functionality we use of CoDE in our prototypes assumes these information to exist. Therefore, for data not created by a chase but inserted manually, the PDMS tries to add appropriate information for these facts (although being irrelevant for the result, they still have to satisfy certain conditions for `localchase()` to work correctly). Sometimes there seems to be a problem with setting the correct information, what leads to errors when running a chase.

Since determining and inserting all these additional values during `localchase()` by CoDE slows down the chase, one could consider rewriting `localchase()` without these computations. However, this would at most

give a speedup by some constant factor, but would certainly not solve the problems with the scalability, especially those of the eichase.

The list of simplifying assumptions made for our implementation (see Section 6.3.8) immediately gives rise to a list of open issues for improving our prototype. Especially a practical error and inconsistency handling seems to be of interest, as just leaving some undefined state is of no help. However, this requires first some satisfying definition of what the result should be in these cases. For improving the handling of a peer, also some of the (from the theoretical viewpoint) trivial improvements should be implemented, like allowing a peer to leave the P2P network, or to provide the user some help with the configuration of the peers by checking the local constraints for validity.

During considering solutions that allow more than one eichase or tchase running in parallel in the PDMS, at least some distributed locking algorithm could be implemented to ensure that there is at most one such chase active at the same time.

Other improvements of the theory concern the problems mentioned in Section 6.2 and the discussion about more dynamic settings in the first half of this section. Another open problem is yet why in $\mathtt{Expand}(\phi, \mathcal{J})$ only the query on the left hand side of the P2P mapping is rewritten.

Additional improvements will depend on more detailed evaluation results, and will mainly consider the speedup of the chase.

7 CONCLUSION

In this thesis we presented our implementation of a PDMS prototype system. To the best of our knowledge, this prototype is the first implementation of the semantical framework for PDMSs suggested by De Giacomo et al. [30]. This framework combines the ideas of data exchange, data integration and peer data management, and by using a semantics suggested in [15] it does not require for restrictions on the topology of the P2P network. We described our prototype system and some problems encountered during the implementation.

Further, additionally to a detailed description of the implemented approach, we also summarised the basic idea of PDMSs and we presented a selection of other PDMS prototypes, an overview of different semantics for PDMSs and approaches for modelling relationships between peer schemas that have been suggested in the literature.

Because being the technique used in the framework of De Giacomo et al. to express relationships between peers, we especially concentrated on schema mappings for expressing relationships between different schemas (or within a single schema). As schema mappings are well known from data exchange and data integration, and our implementation generalises both of them, we also summarised the main results of these areas that are also of interest for Peer Data Management. Thereby, apart from the formal description of the semantics of such settings, we mainly concentrated on differentiating between undecidable, decidable and even tractable scenarios.

7.1 FUTURE WORK

There is still work to do on both, the practical side (improving and extending the implementation of the created prototype PDMS) and on the theoretical side (solving problems or yet not considered aspects of the semantics of the implemented framework).

First, an extensive and systematic evaluation of the created prototype system is of interest to justify or refute the results of the first test runs. Especially the performance of the EI-CHASE seems to be of importance. Thereby it is of special interest whether the bad performance observed is due to implementation issues or whether the combination of data exchange and virtual mappings is generally problematic in practice.

Beside applying the insights gained from the evaluation of the system, also some trivial convenience functionality may be added to the prototype to make handling of the peers easier.

Concerning theoretical aspects, one future goal is to overcome the open problems described in Section 6.2. Moreover it would be interesting to study satisfactory semantics for more dynamic settings of the PDMS not covered by the current framework, for example the questions discussed in Section 6.5.

Towards a practical use it would be interesting to find a convenient way to handle inconsistencies, as the current approach cannot be applied in practice. However, for most of the well defined semantics for handling PDMSs inconsistencies elegantly (e.g. [16]) query answering becomes intractable.

Finally general extensions of PDMSs, for example as proposed in [18] seem to open research topics of great interest.

BIBLIOGRAPHY

[1] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1–3, 1998, Seattle, Washington*, pages 254–263. ACM Press, 1998. (Cited on pages 33 and 34.)

[2] Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, and John Mylopoulos. The hyperion project: from data integration to data coordination. *SIGMOD Record*, 32(3):53–58, 2003. (Cited on pages 1 and 56.)

[3] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J.ACM*, 31(4):718–741, 1984. (Cited on pages 9 and 11.)

[4] Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and Ilya Zaihrayeu. Data management for peer-to-peer computing : A vision. In *ACM SIGMOD WebDB Workshop*, pages 89–94, 2002. (Cited on pages 43 and 44.)

[5] Angela Bonifati, Elaine Qing Chang, Terence Ho, and Laks V. S. Lakshmanan. HepToX: Heterogeneous peer to peer XML databases. *CoRR*, abs/cs/0506002, 2005. (Cited on pages 1 and 57.)

[6] Angela Bonifati, Elaine Qing Chang, Terence Ho, Laks V. S. Lakshmanan, and Rachel Pottinger. HePToX: Marrying XML and heterogeneity in your P2P databases. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 – September 2, 2005*, pages 1267–1270. ACM, 2005. (Cited on page 57.)

[7] Angela Bonifati, Panos K. Chrysanthis, Aris M. Ouksel, and Kai-Uwe Sattler. Distributed databases and peer-to-peer databases: past and present. *SIGMOD Record*, 37(1):5–11, 2008. (Cited on pages 39, 40, and 42.)

[8] Athman Bouguettava, Boualem Benatallah, and Ahmed Elmagarmid. An overview of multidatabase systems: Past and present. In Ahmed K. Elmagarmid, Marek Rusinkiewicz, and Amit Sheth, editors, *Management of heterogeneous and autonomous database systems*, pages 1–32. 1999. (Cited on pages 1 and 39.)

[9] Andrea Cali, Giuseppe De Giacomo, and Maurizio Lenzerini. Models for information integration: Turning local-as-view into global-as-view. In *In Proc. of Int. Workshop on Foundations of Models for Information Integration (10th Workshop in the series Foundations of Models and Languages for Data and Objects)*, 2001. Online Available: citeseer.ist.psu.edu/495550.html. Accessed 1. August 2009. (Cited on page 26.)

[10] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the role of integrity constraints in data integration. *IEEE Data Eng. Bull.*, 25(3):39–45, 2002. (Cited on page 27.)

[11] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the expressive power of data integration systems. In *Conceptual Modeling - ER 2002, 21st International Conference on Conceptual Modeling, Tampere, Finland, October 7–11, 2002, Proceedings*, volume 2503 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 2002. (Cited on pages 26 and 27.)

[12] Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9–15, 2003*, pages 16–21. Morgan Kaufmann, 2003. (Cited on pages 30, 91, 92, 93, 94, and 95.)

[13] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9–12, 2003, San Diego, CA, USA*, pages 260–271. ACM, 2003. (Cited on pages 29 and 30.)

[14] Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic data integration in p2p systems. In *Databases, Information Systems, and Peer-to-Peer Computing, First International Workshop, DBISP2P, Berlin Germany, September 7–8, 2003, Revised Papers*, volume 2944 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2003. (Cited on pages 48, 50, 51, and 53.)

[15] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical foundations of peer-to-peer data integration. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14–16, 2004, Paris, France*, pages 241–251. ACM, 2004. (Cited on pages 1, 2, 41, 51, 70, 99, 111, and 131.)

[16] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Inconsistency tolerance in p2p data integration: An epistemic logic approach. *Inf. Syst.*, 33(4–5):360–384, 2008. (Cited on pages 52, 53, and 132.)

[17] O. Duschka. *Query Planning and Optimization in Information Integration*. PhD thesis, Stanford University, 1997. (Cited on page 25.)

[18] Schahram Dustdar, Reinhard Pichler, Vadim Savenkov, and Hong-Linh Truong. Service-oriented data integration: an envisaged architecture and research challenges. submitted, 2009. (Cited on page 132.)

[19] Ronald Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4): 952–985, 1982. (Cited on page 9.)

[20] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8–10, 2003, Proceedings*, volume 2572 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2003. (Cited on pages 1, 6, 7, 8, 11, 13, and 17.)

[21] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 90–101. ACM, 2003. (Cited on pages 1 and 6.)

[22] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336 (1):89–124, 2005. (Cited on pages 6, 11, 15, 18, and 19.)

[23] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005. (Cited on pages 6, 19, and 20.)

[24] Apache Foundation. Apache axis (ws.apache.org/axis). `http://ws.apache.org/axis`, 2009. Accessed 01-August-2009. (Cited on page 119.)

[25] Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Databases, Information Systems, and Peer-to-Peer Computing, First International Workshop, DBISP2P, Berlin Germany, September 7–8, 2003, Revised Papers*, volume 2944 of *Lecture Notes in Computer Science*, pages 64–76. Springer, 2003. (Cited on pages 1, 48, 51, 52, and 58.)

[26] Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. The coDB robust Peer-to-Peer database system. In *Proceedings of the Twelfth Italian Symposium on Advanced Database Systems, SEBD 2004, S. Margherita di Pula, Cagliari, Italy, June 21–23, 2004*, pages 382–393, 2004. (Cited on pages 1, 42, 58, and 128.)

[27] Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. A distributed algorithm for robust data sharing and updates in p2p database networks. In *Current Trends in Database Technology - EDBT 2004 Workshops, EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14–18, 2004, Revised Selected Papers*, volume 3268 of *Lecture Notes in Computer Science*, pages 446–455. Springer, 2004. (Cited on page 58.)

[28] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In *AAAI/IAAI*, pages 67–73, 1999. (Cited on page 25.)

[29] Ariel Fuxman, Phokion G. Kolaitis, Renée J. Miller, and Wang Chiew Tan. Peer data exchange. *ACM Trans. Database Syst.*, 31(4):1454–1498, 2006. (Cited on page 9.)

[30] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11–13, 2007, Beijing, China*, pages 133–142. ACM, 2007. (Cited on pages 1, 2, 3, 8, 11, 42, 48, 69, 70, 71, 72, 73, 74, 75, 76, 77, 79, 80, 81, 83, 84, 86, 87, 89, 90, 96, 97, 100, 102, 120, 128, and 131.)

[31] Georg Gottlob. Computing cores for data exchange: new algorithms and practical solutions. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13–15, 2005, Baltimore, Maryland, USA*, pages 148–159. ACM, 2005. (Cited on page 20.)

[32] Georg Gottlob and Alan Nash. Data exchange: computing cores in polynomial time. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26–28, 2006, Chicago, Illinois, USA*, pages 40–49. ACM, 2006. (Cited on page 20.)

[33] Todd J. Green, Gregory Karvounarakis, Nicholas E. Taylor, Olivier Biton, Zachary G. Ives, and Val Tannen. ORCHESTRA: facilitating collaborative data sharing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12–14, 2007*, pages 1131–1133. ACM, 2007. (Cited on page 58.)

[34] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23–27, 2007*, pages 675–686. ACM, 2007. (Cited on pages 58, 59, and 60.)

[35] Steven D. Gribble, Alon Y. Halevy, Zachary G. Ives, Maya Rodrig, and Dan Suciu. What can databases do for peer-to-peer? In *WebDB Workshop on Databases and the Web*, pages 31–36, 2001. (Cited on pages 1, 38, 54, and 61.)

[36] Jarek Gryz. Query folding with inclusion dependencies. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23–27, 1998, Orlando, Florida, USA*, pages 126–133. IEEE Computer Society, 1998. (Cited on page 25.)

[37] Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. Piazza: data management infrastructure for semantic web applications. In *Proceedings of the 12th international conference on World Wide Web*, pages 556–567. ACM, 2003. (Cited on pages 54, 61, and 68.)

[38] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *Proceedings of the 19th International Conference on Data Engineering, March 5–8, 2003, Bangalore, India*, pages 505–516. IEEE Computer Society, 2003. (Cited on pages 1, 2, 41, 61, 62, 63, 64, 65, 67, and 68.)

[39] Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004. (Cited on pages 61, 67, and 68.)

[40] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation for large-scale semantic data sharing. *VLDB J.*, 14(1):68–83, 2005. (Cited on pages 61, 64, 67, and 68.)

[41] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12–15, 2006*, pages 9–16. ACM, 2006. (Cited on pages 2, 21, 25, and 39.)

[42] Pavol Hell and Jaroslav Nesetril. The core of a graph. *Discrete Mathematics*, 109(1–3):117–126, 1992. (Cited on pages 19 and 20.)

[43] Katja Hose, Armin Roth, Andre Zeitz, Kai-Uwe Sattler, and Felix Naumann. A research agenda for query processing in large-scale peer data management systems. *Inf. Syst.*, 33(7–8):597–610, 2008. (Cited on pages 39 and 42.)

[44] Zachary G. Ives, Nitin Khandelwal, Aneesh Kapur, and Murat Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *CIDR*, pages 107–118, 2005. (Cited on pages 1, 58, and 60.)

[45] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28(1):167–189, 1984. (Cited on page 30.)

[46] Vasiliki Kantere, Iluju Kiringa, John Mylopoulos, Anastasios Kementsietsidis, and Marcelo Arenas. Coordinating peer databases using ECA rules. In *Databases, Information Systems, and Peer-to-Peer Computing, First International Workshop, DBISP2P, Berlin Germany, September 7–8, 2003, Revised Papers*, volume 2944 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003. (Cited on pages 1, 46, and 47.)

[47] Vasiliki Kantere, John Mylopoulos, and Iluju Kiringa. A distributed rule mechanism for multidatabase systems. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3–7, 2003*, volume 2888 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2003. (Cited on pages 46 and 47.)

[48] Anastasios Kementsietsidis. Data sharing and querying for Peer-to-Peer data management systems. In *Current Trends in Database Technology - EDBT 2004 Workshops, EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14–18, 2004, Revised Selected Papers*, volume 3268 of *Lecture Notes in Computer Science*, pages 177–186. Springer, 2004. (Cited on page 56.)

[49] Anastasios Kementsietsidis and Marcelo Arenas. Data sharing through query translation in autonomous sources. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 – September 3 2004*, pages 468–479. Morgan Kaufmann, 2004. (Cited on pages 44 and 46.)

[50] Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9–12, 2003*, pages 325–336. ACM, 2003. (Cited on pages 1, 44, 45, and 46.)

[51] Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Managing data mappings in the hyperion project. In *Proceedings of the 19th International Conference on Data Engineering, March 5–8, 2003, Bangalore, India*, pages 732–734. IEEE Computer Society, 2003. (Cited on page 56.)

[52] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13–15, 2005, Baltimore, Maryland, USA*, pages 61–75. ACM, 2005. (Cited on pages 6, 7, 9, 11, 17, and 35.)

[53] Phokion G. Kolaitis, Jonathan Panttaja, and Wang Chiew Tan. The complexity of data exchange. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26–28, 2006, Chicago, Illinois, USA*, pages 30–39. ACM, 2006. (Cited on page 17.)

[54] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3–5, Madison, Wisconsin, USA*, pages 233–246. ACM, 2002. (Cited on pages 2, 21, 22, and 33.)

[55] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *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. (Cited on page 24.)

[56] Alexander Löser, Wolf Siberski, Martin Wolpers, and Wolfgang Nejdl. Information integration in Schema-Based Peer-To-Peer networks. In *Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, Klagenfurt, Austria, June 16–18, 2003, Proceedings*, volume 2681 of *Lecture Notes in Computer Science*, pages 258–272. Springer, 2003. (Cited on pages 55 and 56.)

[57] Aleksander Madry. Data exchange: On the complexity of answering queries with inequalities. *Inf. Process. Lett.*, 94(6):253–257, 2005. (Cited on page 19.)

[58] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979. (Cited on page 11.)

[59] Mehedi Masud, Iluju Kiringa, and Anastasios Kementsietsidis. Don't mind your vocabulary: Data sharing across heterogeneous peers. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 – November 4, 2005, Proceedings, Part I*, volume 3760 of *Lecture Notes in Computer Science*, pages 292–309. Springer, 2005. (Cited on pages 44 and 46.)

[60] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *VLDB*, pages 898–909, 2003. (Cited on page 68.)

[61] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *Proceedings of the 11th international conference on World Wide Web*, pages 604–615. ACM, 2002. (Cited on pages 1 and 55.)

[62] Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. PeerDB: A P2P-based system for distributed data sharing. In *Proceedings of the 19th International Conference on Data Engineering, March 5–8, 2003, Bangalore, India*, pages 633–644. IEEE Computer Society, 2003. (Cited on pages 1 and 54.)

[63] University of Toronto Database Group. Hyperion project website (www.cs.toronto.edu/db/hyperion/index.html. `http://www.cs.toronto.edu/db/hyperion/index.html`, 2009. Accessed 1. August 2009. (Cited on page 56.)

[64] Daniel Olmedilla. Working with edutella. Technical report, L3S Research Center and Hanover University. URL `http://www.l3s.de/~olmedilla/projects/edutella/edutella.pdf`. Accessed 1. August 2009. (Cited on pages 55 and 56.)

[65] Beng Chin Ooi, Kian-Lee Tan, Aoying Zhou, Chin Hong Goh, Yingguang Li, Chu Yee Liau, Bo Ling, Wee Siong Ng, Yanfeng Shu, Xiaoyu Wang, and Ming Zhang. PeerDB: Peering into personal databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9–12, 2003*, page 659. ACM, 2003. (Cited on page 54.)

[66] M. Tamer Ozsu and P. Valduriez. *Principles of distributed database systems*. Prentice-Hall, Inc. (Cited on page 40.)

[67] Md. Anisur Rahman, Illuju Kiringa, and Abdulmotaleb El Saddik. Generalization of an algorithm for checking consistency of mapping tables in a p2p system. In *Proceedings of the 7th International Conference on Computer and Information Technology (ICCIT 2004), Dhaka, Bangladesh, Dec. 2004*, 2004. (Cited on pages 44 and 46.)

[68] Raymond Reiter. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling (Intervale)*, pages 191–233. Springer, 1984. (Cited on page 43.)

[69] Patricia Rodríguez-Gianolli, Maddalena Garzetti, Lei Jiang, Anastasios Kementsietsidis, Iluju Kiringa, Mehedi Masud, Renée J. Miller, and John Mylopoulos. Data sharing in the hyperion peer database system. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 – September 2, 2005*, pages 1291–1294. ACM, 2005. (Cited on page 56.)

[70] Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26–28, 2006, Chicago, Illinois, USA*, pages 356–365. ACM, 2006. (Cited on pages 31 and 32.)

[71] Vadim Savenkov. Implementing core computation for data exchange. Master thesis, Technische Universität Wien, October 2007. (Cited on pages 2, 20, 98, 108, 113, 120, 121, 123, and 127.)

[72] Luciano Serafini, Fausto Giunchiglia, John Mylopoulos, and Philip A. Bernstein. Local relational model: A logical formalization of database coordination. In *Modeling and Using Context, 4th International and Interdisciplinary Conference, CONTEXT 2003, Stanford, CA, USA, June 23–25, 2003, Proceedings*, volume 2680 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2003. (Cited on pages 1, 43, and 44.)

[73] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990. (Cited on pages 1 and 39.)

[74] Springsource. Spring framework (www.springsource.org). `http://www.springsource.org`, 2009. Accessed 01-August-2009. (Cited on pages 98 and 107.)

[75] Igor Tatarinov, Zachary G. Ives, Jayant Madhavan, Alon Y. Halevy, Dan Suciu, Nilesh N. Dalvi, Xin Dong, Yana Kadiyska, Gerome Miklau, and Peter Mork. The piazza peer data management project. *SIGMOD Record*, 32(3):47–52, 2003. (Cited on pages 61 and 68.)

[76] Nicholas E. Taylor and Zachary G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27–29, 2006*, pages 13–24. ACM, 2006. (Cited on pages 42, 58, and 60.)

[77] Dan Zhao, John Mylopoulos, Iluju Kiringa, and Verena Kantere. An ECA rule rewriting mechanism for peer data management systems. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26–31, 2006,*

Proceedings, volume 3896 of *Lecture Notes in Computer Science*, pages 1069–1078. Springer, 2006. (Cited on pages 46 and 47.)