Peer Data Management *

Armin Roth[1] and Sebastian Skritek[2]

1 **Cartesiusstrasse 47, 89075 Ulm, Germany**
`armin@arminroth.de`
2 **Technische Universität Wien**
Favoritenstraße 9, 1040 Wien, Austria
`skritek@dbai.tuwien.ac.at`

Abstract

Peer Data Management (PDM) deals with the management of structured data in unstructured peer-to-peer (P2P) networks. Each peer can store data locally and define relationships between its data and the data provided by other peers. Queries posed to any of the peers are then answered by also considering the information implied by those mappings.

The overall goal of PDM is to provide semantically well-founded integration and exchange of heterogeneous and distributed data sources. Unlike traditional data integration systems, peer data management systems (PDMSs) thereby allow for full autonomy of each member and need no central coordinator. The promise of such systems is to provide flexible data integration and exchange at low setup and maintenance costs.

However, building such systems raises many challenges. Beside the obvious scalability problem, choosing an appropriate semantics that can deal with arbitrary, even cyclic topologies, data inconsistencies, or updates while at the same time allowing for tractable reasoning has been an area of active research in the last decade. In this survey we provide an overview of the different approaches suggested in the literature to tackle these problems, focusing on appropriate semantics for query answering and data exchange rather than on implementation specific problems.

1998 ACM Subject Classification H.2.5 [Database Management]: Heterogeneous Databases, H.2.4 [Database Management]: Systems

Keywords and phrases Peer Data Management, PDM, Peer Data Management Systems, PDMS, Survey, P2P

Digital Object Identifier 10.4230/DFU.Vol5.10452.185

1 Introduction

Peer Data Management (PDM) on the one hand describes the complete area of data management in peer-to-peer (P2P) systems, while on the other hand it is also used to denote a very specific type of data management systems. In this survey, we follow the second interpretation, referring to management of structured data in unstructured P2P networks only. Concentrating on *Peer Data Management Systems (PDMSs)*, we provide a summary of different approaches introduced in the literature to design and create such systems, and consider both theoretical aspects as well as actual implementations:

PDMSs consist of a set of *peers*, where each peer offers some data through a so called *peer schema*. If a peer is interested in enhancing the information published through its peer schema with the data provided by some other peer, mappings between these two peers,

* S. Skritek was supported by the Vienna Science and Technology Fund (WWTF), projects ICT08-032 and ICT12-15, and by the Austrian Science Fund (FWF): P25207-N23.

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt; pp. 185–215

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

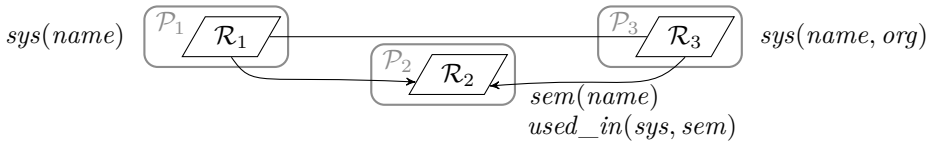

Figure 1 Example of a PDMS. Rectangles with rounded borders represent the peers P_1, P_2, P_3 and the trapezoids the peer schemas. Their schema is depicted next to the peers.

defined either on schema or instance level, are used to express the relationship between the data offered by these peers. Queries are posed against a single peer schema. Their answers do not only include the information stored locally at that peer, but also contain all the information implied by these mappings.

▶ **Example 1.** Figure 1 shows an example of a small PDMS. Consider first only the peers P_1 and P_2, both collecting and providing information about PDMSs. P_1 offers a list of prototype systems ($sys(name)$) and P_2 information about different semantics for PDMSs ($sem(name)$), and which prototype implements which semantics ($used\_in(sys, sem)$). Also, P_2 retrieves the information about available prototypes from P_1. Now assume another peer P_3 joins the group. It also offers a list of prototype systems, which contains beside the name of the system also the organization that built it ($sys(name, org)$). P_3 enhances its list by the data provided from P_1, and P_2 tries to complete its data by defining a mapping from P_3 to itself. ◀

Example 1 already illustrates some of the main advantages of PDMSs: They can be easily set up, members can join and leave the network at will, they support heterogeneous schemas and domains, and there is no need for global coordination, as e.g. required in typical data integration or multi-database systems. This allows each peer to take care of the mappings it is interested in only and no global coordination mechanism is required. However, the example also raises some questions. The probably most interesting one among them is how such mappings between two peers really look like and, related to this, how data sharing along them works. For example, P_3 could either import all relevant data from P_1, or retrieve the required information only at query time by answering queries not only on its local data, but also by forwarding them to P_1.

In fact, while PDMSs possess very promising properties, building such a system is a challenging task. One of the main problems is to find an appropriate formalism for defining the mappings between peers that is powerful enough to be useful, but at the same time allows for decidable (or preferably: efficient) reasoning (e.g., query answering). As a result, several different formalisms and semantics have been suggested in the literature for the specification of such mappings. Obviously, this leads to several different semantics that can be applied to a PDMS. Beside these problems of creating a suitable theory for PDM (like the relational model for single databases), due to the distribution and autonomy of the peers, also implementing such systems is no easy task. In combination with the lack of a clear semantics, several prototype systems have been created in the last years, addressing specific problems of building such systems.

In this survey we provide an overview of the different approaches taken so far to overcome these problems: We describe the most important and influential suggestions for useful (i.e. powerful, yet efficiently decidable) semantics for mappings in PDMSs, concentrating on relational systems, but also considering systems using other data models. One focus of this discussion is how schema mappings from data exchange or data integration have been applied to PDM. Beside those theoretical frameworks, we also provide an overview

on existing prototype systems and point out specific and interesting properties of them. The organization of the paper is as follows: After some preliminary definitions in Section 2, we discuss general properties and characteristics of PDMSs in Section 3 and point out the differences between PDMSs, other P2P systems, and other kinds of distributed database systems. Next (Section 4), we describe the Local Relational Model, a concrete formalism for modeling PDMSs. Section 5 contains the discussion of the application of schema mappings to PDM, followed by alternative mapping strategies in Section 6. An overview on system prototypes is started in Section 7 which is completed in Section 8 with a discussion of PDMSs not applying the relational data model. We summarize and conclude in Section 9.

2 Preliminaries and System Model

Schemas and instances. A *relational schema* $\mathcal{R} = \{R_1, \dots, R_n\}$ is a set of relation symbols R_i each of a fixed arity k_i and with an assigned sequence of k_i attributes $(A_1, \dots, A_{k_i})$. Unless defined otherwise, an *instance* (or *interpretation*) I over a schema $\mathcal{R}$ consists of a k_i-ary relation R_i^I for each relation symbol $R_i \in \mathcal{R}$. We write $\vec{x}$ for a tuple $(x_1, \dots, x_n)$, but may also use $R_i(\vec{s}) \in I$ to denote a tuple $\vec{s} \in R_i^I$. By slight abuse of notation, we also refer to the set $\{x_1, \dots, x_n\}$ as $\vec{x}$. Hence, we may use expressions like $x_i \in \vec{x}$ or $\vec{x} \subseteq X$, etc. Tuples of the relations may contain two types of *terms*: *constants* and *labeled nulls*, taken from the sets *consts* and *null* respectively. Although (unless stated otherwise) the *domain* (or *universe*) $dom = consts \cup null$ is considered to be a countable infinite set, we only consider finite instances here, and denote with $dom(I) = consts(I) \cup null(I)$ the *active domain* of I.

Homomorphisms and conjunctive queries. Let I, J be instances. A *homomorphism* $h\colon I \to J$ is a mapping $dom(I) \to dom(J)$ s.t. (1) $h(c) = c$ for all $c \in consts(I)$ and (2) whenever $R(\vec{x}) \in I$, then $R(h(\vec{x})) \in J$, where by slight abuse of notation, for a tuple $\vec{x} = (x_1, \dots, x_n)$ we write $h(\vec{x})$ for $(h(x_1), \dots, h(x_n))$.

A *conjunctive query (CQ)* Q on a database schema $\mathcal{R}$ is of the form $Q\colon ans(\vec{x}) \leftarrow \exists \vec{y} \phi(\vec{x}, \vec{y})$, where $\phi(\vec{x}, \vec{y}) = \bigwedge_{i=1}^{n} r_i$ is a conjunction of atoms $r_i = R_j(\vec{z})$, s.t. $R_j \in \mathcal{R}$ with arity k, and $\vec{z} \subseteq \vec{x} \cup \vec{y}$ with $|\vec{z}| = k$. A tuple $\vec{s}$ is called an *answer* or *solution* to a CQ Q on an instance I if $\vec{s} = \mu(\vec{x})$, where $\mu\colon \vec{x} \cup \vec{y} \to dom(I)$ is a variable assignment on $\vec{x} \cup \vec{y}$, s.t. for every atom $R_i(\vec{z})$ occurring in Q it holds that $R_i(\mu(\vec{z})) \in I$.

Constraints, Mappings, and Theories. Given some logic L and a relational schema $\mathcal{R}$, a *(L-)theory over* $\mathcal{R}$ is a set of formulas of L over the relation symbols in $\mathcal{R}$. A formula that contains no free variables is called a *closed formula* or *sentence*. A *relational theory* over $\mathcal{R}$ consists of function free FO formulas over $\mathcal{R}$. Two special kinds of FO constraints are very well studied: A *tuple generating dependency (tgd)* is a FO formula $\forall \vec{x}\big(\exists \vec{z} \phi(\vec{x}, \vec{z}) \to \exists \vec{y} \psi(\vec{x}, \vec{y})\big)$, where ϕ and ψ are conjunctions of atoms. An *equality generating dependency (egd)* is a FO formula $\forall \vec{x}\big(\phi(\vec{x}) \to x_1 = x_2\big)$ where $x_1, x_2 \in \vec{x}$ and ϕ again is a conjunction of atoms.

Due to the structure of tgds, it is natural to identify a CQ with the lhs and rhs of a tgd each, i.e. to consider tgds as mappings $Q_1(\vec{x}) \to Q_2(\vec{x})$ for CQs Q_1, Q_2. Intuitively, a tgd then requests all answers to Q_1 also to be answers of Q_2. Further (based on this characterization) note that tgds can define GAV, LAV, and GLAV mappings, well known from data integration [54]. We therefore consider those mappings as special cases of tgds.

The *chase* is a well known procedure to repair instances that do not satisfy a set of tgds and egds by inserting tuples or unifying labeled nulls. For information on the chase, we refer to Chapter 1 of this book.

The *cyclicity* of a set of tgds (and mappings/constraints in general) is characterized via a graph representation of such sets, i.e. they are acyclic iff some corresponding directed graph is. Typically, for a set of mappings, the nodes of the graph are the relation symbols occurring in the mappings (or pairs of relation symbols and variables). Directed edges are added according to the structure of the mappings; for example for tgds from occurrences at the lhs of a tgd to occurrences on its rhs. Weaker notions of acyclicity do only forbid certain types of cycles. A prominent such example is *weak acyclicity* [20].

Peer Data Management Systems. We focus on the class of *Peer Data Management Systems* (PDMSs) that offer semantically well founded sharing of structured data. Following the terminology in [19], we consider a Peer Data Management System (PDMS) as a triple $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, where $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a set of autonomous *peers*, $\mathcal{R}$ is a set of *peer schemas* (one schema $\mathcal{R}_i$ for each $P_i \in \mathcal{P}$), and $\mathcal{M}$ is a set of *mappings*. We say that $\mathcal{R}_i$ is the schema of a local database if data is actually stored under $\mathcal{R}_i$. Alternatively, $\mathcal{R}_i$ can also be the global schema of some local data integration system at P_i (in this case, we refer to the source relations as $\mathcal{L}_i$ and assume the mappings between $\mathcal{L}_i$ and $\mathcal{R}_i$ for each $P_i \in \mathcal{P}$ to be contained in $\mathcal{M}$ as well), or a mediator schema (or view) defined by mappings from other peer schemas. Hence a peer may either bring new data into the system or just act as a mediator, restructuring (through corresponding mappings) data already present in the system. For the semantics of a peer, this does not make any difference, as in any case each peer offers some well defined data through its peer schema. Finally, the mappings in $\mathcal{M}$ may be either defined on schema or instance level, or both. As we will see, this depends on the concrete formalism used for defining the mappings. We use dom_i to denote the domain of P_i.

Instances, consistent global instances, and queries for PDMSs. Given a PDMS $\mathcal{S}$, an instance I for $\mathcal{S}$ is either just an instance for $\mathcal{R}$ (if each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database) or an instance for $\bigcup_{P_i \in \mathcal{P}} \mathcal{L}_i$ if the peers consist of local data integration systems. For an instance I for $\mathcal{S}$, let $I|P_i$ denote the restriction of I to the schema $\mathcal{R}_i$ (resp. $\mathcal{L}_i$). Given an instance I for $\mathcal{S}$, a *consistent global instance I' for $\mathcal{S}$ w.r.t. I* is in general an instance of $\mathcal{R}$, s.t. (i) I' satisfies $\mathcal{M}$ and (ii) if I is already an instance for $\mathcal{R}$, then there exists a homomorphism $h\colon I \to I'$. Thereby the notion "I' satisfies $\mathcal{M}$" depends on the concrete semantics applied, and will be a main focus of this survey. We denote with $\mathcal{I}_I$ the set of all consistent global instances w.r.t. I, and drop the I if clear from the context. However, some systems do not define a global instance for $\mathcal{S}$ as an instance for $\mathcal{R}$, but as a tuple $(\mathcal{I}_I[\mathcal{R}_i])_{\mathcal{R}_i \in \mathcal{R}}$, where each $\mathcal{I}_I[\mathcal{R}_i]$ is a *set* of instances for $\mathcal{R}_i$.

A query to a PDMS $\mathcal{S}$ is formulated over a single peer schema $\mathcal{R}_i \in \mathcal{R}$. Given an instance I for $\mathcal{S}$, the result of a query Q usually is defined as the *certain answers* $Q^c(I) = \bigcap_{I' \in \mathcal{I}_I} Q(I')$, where $Q(I')$ denotes the evaluation of Q over I'. If only $\mathcal{I}_I[\mathcal{R}_i]$ is defined, then $Q^c(I) = \bigcap_{I' \in \mathcal{I}_I[\mathcal{R}_i]} Q(I')$. Unless stated otherwise, we always assume Q to be a CQ.

3 PDMS: Characterization and Properties

In this section, we start with a quick overview over a classification of P2P systems and discuss how PDM fits into these concepts. We then relate PDMSs to traditional database systems, and finally take a look onto properties specific for PDMSs.

P2P systems are nowadays wide spread and used for a variety of applications in many different areas. Tasks solved by P2P systems cover, among others, sharing of data (e.g., Gnutella, BitTorrent) and other resources, communication (e.g. Skype), or the implementation

of fail-safe systems. The reasons for the success of P2P systems include their scalability, low setup costs, the lack of need for central coordination, or a high reliability (since replicating and shifting resources and tasks between different nodes allows to compensate node failures).

A prominent way to classify P2P systems is according to the logical structure of the resulting P2P network (overlay network). In *pure* P2P systems, all peers are conceptually equal and have the same role, while in *hybrid* systems so called super-peers (which have more knowledge about the current state of the system) act as servers and control and coordinate the system. In the extreme case of *centralized P2P systems*, all requests are issued against a single central server who dispatches them to an appropriate network node. Pure P2P systems are further divided into *unstructured* and *structured* systems. In unstructured systems, each peer is free to choose which data to store, which peers to communicate with, or which requests to accept. In *structured* P2P systems, data placement or message routing follows strict rules determined by the system, which are enforced by the peers in a distributed manner. A prominent example for structured P2P systems are distributed hash tables (DHTs), where the placement of a data item is determined by a key or hash value assigned to each item: Each peer gets assigned a (not necessarily distinct) part of the keyspace, and items are stored at exactly those peers that cover their key values. The reason that we will concentrate on the management of structured data in unstructured P2P systems is that while most of the aspects and applications of P2P systems described above are already covered by surveys or books, to the best of our knowledge, a summary of Peer Data Management Systems is missing: We refer to [73] for a general introduction to – and overview on – P2P systems, including characterizations, architectures, applications and systems as well as aspects like routing, load balancing, security or trust. [69] provides an extensive summary of applications for P2P systems and general P2P techniques, including three chapters on DHTs. A specific overview over data management P2P systems can be found in [5, Chapter 16] with a focus on query evaluation and replica management. A short classification of P2P data management systems based on their structure is presented in [11]. Finally [68] provides an introduction into the combination of P2P and Semantic Web techniques and applications. However, the only overview paper in the area of PDMSs is [41], reviewing design and implementation aspects and challenges for PDMSs, but not providing a comprehensive summary of existing approaches. It can therefore be considered as a complement to the present paper.

One of the initial goals of PDMSs was to extend the idea of unstructured P2P file sharing systems to structured data [32], i.e. to allow for data sharing between a large number of highly autonomous participants, but supporting a rich semantics and expressive query languages (see [32, 6] for early visions of this idea). Note that PDMSs are not primarily intended to provide a distributed, fail safe storage system (like DHTs) or to provide load distribution, but just to provide an easy way to allow participants to share their data with others. Each peer can freely select its neighbors and define mappings to them. Data is shared according to the semantics of the mappings only, such that each peer has full control over the data it stores. Following these ideas, PDMSs obviously belong to the group of unstructured P2P-systems.

Compared to other distributed database systems like multi-databases [12], which allow for a similar distribution and heterogeneity of their members, PDMSs provide a higher autonomy for each peer. A classification and description of PDMSs w.r.t. traditional database approaches is given in [11]. Higher autonomy of each member in the network also distinguishes PDMSs from traditional data integration systems [54]. There, the data provided by each peer is integrated in terms of a global schema instead of pairwise mappings.

While the above discussion holds for all PDMSs, we next take a look onto properties that characterize and distinguish different approaches to PDM and shortly summarize the most

important aspects of PDM that influence those properties. We start with considering design choices that influence the semantics of a PDMS, continue with characteristics resulting from these choices and finish with choices that deal with the concrete implementation but do not influence the semantics. Obviously a fundamental property of a PDMS with heavy influence on its semantics is the supported data model (e.g., relational, XML, RDF). Another important aspect are the P2P mappings, as they define what data is exchanged and how. The exchange could be either done by enforcing the constraints defined through the mappings on the data, i.e. by indeed materializing tuples in the different peers such that the mappings are satisfied over the stored instances (we refer to corresponding systems as *exchange systems*). In general there exist several (up to infinitely many) possible instances that could be materialized to satisfy the mappings. In this case, the goal is to identify some "best" instance to materialize. Most of the time, this means to materialize some most general instance that only contains information indeed implied by the mappings. Another desired property of the chosen instance is to allow to compute the certain answers for some query w.r.t. all these possible instances from the information in the instance only. Another possibility is not to materialize these extra information, but to use the mappings to infer additional information only for query answering at query time (*integration systems*), or to consider mappings only as rules for how to translate updates made at one peer to an update on another peer and to exchange only updates. Further aspects of mappings are the mapping language (e.g., FOL or restrictions thereof, coordination formulas — cf. Section 4, mapping tables — cf. Section 6.1), the semantics under which they are evaluated (e.g., local reasoning or global reasoning for FOL mappings — cf. Section 5.1), whether they are defined e.g. on schema or instance level, and whether they support different domains for each peer or assume a shared domain among all peers. The supported query language is another interesting property of a PDMS, just like the technique used for query answering. For example, the latter could be based on query rewriting, answering queries using views, temporarily materialization of data (universal solutions), or using logic programs (e.g., under the stable model semantics). Further aspects are the incorporation of trust, the capability to deal with inconsistencies in the data in a meaningful way, or the maintenance and use of data quality metrics.

Choices on the above properties have a direct influence on properties like the concrete degree of the autonomy, modularity or heterogeneity of the peers or the decidability and complexity of query answering. Finally, further properties of PDMSs arise from aspects related directly with the implementation of such systems. Those include the query planning algorithm (which can be centralized or distributed), the incorporation of query optimization (including relaxations on the correctness or completeness of answers), the maintenance of indices and/or replica, or optimizations of the inter peer mappings.

For a throughout discussion of these aspects related to PDM as well as pointers to different solutions for them, we refer to the recent survey in [41]. Further discussions of these topics can be found e.g. in [73, Chapter 4] and [5, Chapter 16]. In this paper, we provide a summary of different approaches to PDM and PDMSs. Thereby the main focus will be on the way the P2P mappings are defined and formalized, as they are the central component of PDMSs. Beside describing these approaches, we will use (some of) the properties listed above to discuss the effects of different approaches and to characterize them, but the main focus lies on their descriptions. We will further give an overview of prototype implementations of PDMSs, pointing out interesting or notable design decisions or specifics of these systems.

4 A Model for PDM: The Local Relational Model

After the rather general last section, we now start to take a look onto concrete approaches taken to formalize and implement PDMSs. One such proposal was the *Local Relational Model (LRM)* [6, 67], one of the first models proposed for PDMSs. Because it addresses several problems of PDM we use it to illustrate several different key concepts for PDMSs: In the LRM, dependencies between the different databases can be expressed by mappings on schema level. Peers are allowed to use different domains by providing a domain translation mechanism with a meaningful, well-defined semantic. Finally, queries can exploit all these mechanisms to incorporate information stored at several peers into the answer.

Because the LRM is a powerful mechanism addressing several problems of PDM, we will use it as a reference model and compare other approaches with it, or – if appropriate – will even describe other approaches in terms of the LRM.

▶ **Definition 2.** A LRM-PDMS is a PDMS $\mathcal{S}_{LRM} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database and $\mathcal{M} = \mathcal{M}_T \cup \mathcal{M}_{CF} \cup \mathcal{M}_R$, s.t. $\mathcal{M}_T$ contains for each $P_i \in \mathcal{P}$ a relational theory T_i over $\mathcal{R}_i \in \mathcal{R}$, $\mathcal{M}_{CF}$ is a set of *coordination formulas*, and $\mathcal{M}_R$ is a set of *domain relations*.

Domain relations $r_{ij} \subseteq dom_i \times dom_j$ provide domain translations between pairs (P_i, P_j) of peers in $\mathcal{P}$. They allow to support different domains dom_i (which, unlike in most other formalisms, are assumed to be finite) at each $P_i \in \mathcal{P}$. Domain relations need not be symmetric. In the following let $\mathcal{C} = \{i \mid P_i \in \mathcal{P}\}$. A *coordination formula (CF)* is an expression of the form $CF ::= i:\phi \mid CF \to CF \mid CF \wedge CF \mid CF \vee CF \mid \exists i:x(CF) \mid \forall i:x(CF)$, where ϕ is a function free FO formula over some $\mathcal{R}_i \in \mathcal{R}$ and $i \in \mathcal{C}$. Their intuition is as follows: $\forall i:x(CF)$ denotes that the universal quantification of x is over the domain dom_i (similar for $\exists$), while $i:\phi$ indicates that variables in ϕ shall be evaluated w.r.t. P_i (i.e. under dom_i). For variables occurring free in ϕ this may require to incorporate domain translation: If such a free variable is bound outside $i:\phi$ by a quantifier under a different context P_j (i.e. bound under domain dom_j), the translation from dom_j to dom_i must be considered for evaluation (see below for a formal definition of the semantic of such a translation). Although not required in [67], here we consider all CFs in $\mathcal{M}$ to be closed. Recall that a relational theory T_i over $\mathcal{R}_i$ consists of function free FO sentences over $\mathcal{R}_i$ (i.e. function free FO constraints over $\mathcal{R}_i$) with constants from dom_i. Obviously, each theory T_i can be easily encoded as a set $\{i:\phi \mid \phi \in T_i\}$ of CFs. Further, in [67] it was shown that also domain relations may be expressed as CFs. Hence $\mathcal{M}$ may be considered to contain coordination formulas only.

▶ **Example 3.** The following coordination formula adapted from [67] creates a record in the *Person* relation of peer *Hospital* based on the data in *Patient* at peer *Doc*:

$$\forall(Doc : fName, lName, gender).(Doc : Patient(SSN, fName, lName, gender) \quad \to$$
$$Hospital : \exists(persID, name, age).Person(persID, SSN, name, gender, age, Doc) \wedge$$
$$name = concat(fName, lName))).$$

A unique value for *persID* has to be generated and the unknown age has to be filled with a so-called Skolem constant. ◀

In the LRM, an instance $\mathcal{I}$ for $\mathcal{S}$ is not defined as an instance for $\mathcal{R}$, but $\mathcal{I} = (\mathcal{I}[\mathcal{R}_i])_{i \in \mathcal{C}}$ is a tuple of databases $\mathcal{I}[\mathcal{R}_i]$ for each $\mathcal{R}_i \in \mathcal{R}$. Each such database $\mathcal{I}[R_i]$ in turn is considered to consist of a set of instances I' of $\mathcal{R}_i$ that satisfy T_i while interpreting constants as themselves. Thereby the idea is that while $|\mathcal{I}[\mathcal{R}_i]| = 1$ describes the case of a traditional database instance,

$|\mathcal{I}[\mathcal{R}_i]| = 0$ indicates an inconsistent database at peer P_i and $|\mathcal{I}[\mathcal{R}_i]| > 1$ models incomplete databases. Next we define satisfiability of CFs. To be able to deal with domain translations, in the LRM a variable assignment μ for a set $\vec{x}$ of variables is a *set* of mappings $\mu = \{\mu_i\}_{i \in \mathcal{C}}$ with $\mu_i \colon \vec{x} \to dom_i$. For $i \in \mathcal{C}$ and $J \subset \mathcal{C}$, μ is an i-to-J-assignment (i-from-J-assignment) of a variable x if for all $j \in J$ with $j \neq i$, $(\mu_i(x), \mu_j(x)) \in r_{ij}$ ($(\mu_j(x), \mu_i(x)) \in r_{ji}$, resp.), i.e. if μ sticks to the domain translations defined by the domain relations between dom_i and dom_j for all $j \in J$. Now given $\mathcal{I}$, $\mathcal{M}_R$, and an assignment μ, a CF $i : \phi$ is satisfied by $(\mathcal{I}, \mathcal{M}_R)$ under μ (denoted $(\mathcal{I}, \mathcal{M}_R) \models i : \phi[\mu]$) if for each $I' \in \mathcal{I}[\mathcal{R}_i]$ it holds $I' \models \phi[\mu_i]$. I.e. $i : \phi$ is satisfied if ϕ, interpreted under the scope of P_i (in terms of the assignment μ_i), is satisfied over all instances in $\mathcal{I}[\mathcal{R}_i]$. For CFs $\forall i : x(A)[\mu]$ ($\exists i : x(A)[\mu]$) on the other hand let $J \subset \mathcal{C}$ contain all $j \in \mathcal{C}$ s.t. x occurs free in a subformula $j : \phi$ of A. The idea is, that while x is quantified over dom_i, it is evaluated in A under the scopes of the peers P_j ($j \in J$), i.e. under the domains dom_j. Hence when evaluating A, for every possible value for x over dom_i, the corresponding domain translations must be taken into account. Therefore $(\mathcal{I}, \mathcal{M}_R) \models \forall i : x(A)[\mu]$ if $(\mathcal{I}, \mathcal{M}_R) \models A[\mu']$ for all i-to-J-assignments μ' on x that differ from μ only on x. Further, $(\mathcal{I}, \mathcal{M}_R) \models \exists i : x(A)[\mu]$ if $(\mathcal{I}, \mathcal{M}_R) \models A[\mu']$ for some i-from-J-assignments μ' on x that differ from μ only on x. Intuitively, in case of universal quantification, for each possible assignment μ_i on x, each subformula $j : \phi$ must be satisfied under each translation of $\mu_i(x)$ to dom_j. The case for the existential quantification is similar, but a little bit more involved (note that in i-from-J-assignments we have r_{ji} instead of r_{ij}). Due to space restrictions, for a discussion of this we have to refer to [67]. Satisfaction of the connectives $\rightarrow, \wedge, \vee$ is defined as usual.

Queries Q against some $P_i \in \mathcal{P}$ are of the form $(i : q(\vec{x})) \leftarrow A(\vec{x})$, where $A(\vec{x})$ is a coordination formula with free variables $\vec{x}$, $|\vec{x}| = n$, and q is a new n-ary relation symbol. Given $(\mathcal{I}, \mathcal{M}_R)$, the answer to such a query is defined as $\{\vec{d} \in dom_i^n \mid (\mathcal{I}, \mathcal{M}_R) \models \exists i : \vec{x}(A(\vec{x}) \wedge i : (\vec{x} = \vec{d}))\}$. Queries can be defined recursively, i.e. $A(\vec{x})$ may use the result q' of another query Q' (these recursions may be cyclic).

Note that while this defines how a query is evaluated over $(\mathcal{I}, \mathcal{M}_R)$, $\mathcal{M}_{CF}$ is not taken into account for query answering. Originally, in the LRM P2P mappings were not considered for information exchange or reasoning, but just to express constraints between peers. For a query to also include data stored at different peers, the corresponding CFs need to be specified explicitly as part of the (recursive) query. As a result, there is no concept like consistent global instances (w.r.t. some instance $\mathcal{I}$ for $\mathcal{S}$). Also, while all $I' \in \mathcal{I}[\mathcal{R}_i]$ must satisfy T_i, it is not required that $\mathcal{I}[\mathcal{R}_i]$ contains indeed all models of T_i. So, given a pair $(\mathcal{I}, \mathcal{M}_R)$ as input, the LRM only defines if a CF is satisfied by $(\mathcal{I}, \mathcal{M}_R)$ and the result of a query over this instance. If $\mathcal{I}$ contains incomplete databases, then this answer is however certain w.r.t. this incompleteness (but not w.r.t. to possible repairs). Restricting to a certain class of CFs, [22] extends the LRM by a notion of certain answers also taking $\mathcal{M}_{CF}$ into account. Therefore it is assumed that some input instance I is already encoded into $\bigcup_{i \in C} T_i$, i.e. the tuples in I are expressed as part of the theory T_i. Basically [22] then defines the notion of global consistent instances by considering $\mathcal{I}[\mathcal{R}_i]$ to consist of all interpretations of the theory T_i (i.e. instances of $\mathcal{R}_i$ that satisfy T_i). This is then used to define the set of certain answers w.r.t. I as those answers that occur in each such instance $I' \in \mathcal{I}[\mathcal{R}_i]$. Obviously, this idea could be extended to the LRM in general. I.e. given I encoded into $\mathcal{M}$, the certain answers to a query would be $\bigcap_{\{\mathcal{I} \mid (\mathcal{I}, \mathcal{M}_R) \models \mathcal{M}\}} \{\vec{d} \in dom_i^n \mid (\mathcal{I}, \mathcal{M}_R) \models \exists i : \vec{x}(A(\vec{x}) \wedge i : (\vec{x} = \vec{d}))\}$.

5 Schema Mappings for PDMSs

We have seen that the LRM provides mappings on schema and instance level and supports different domains at each peer. Many approaches to PDM however assume a unique domain shared by all peers, and consider schema level mappings only. They can be generally described as PDMSs $\mathcal{S}_S = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where $\mathcal{M}$ is a set of formulas of some logic $\mathcal{L}$ over $\mathcal{R}$. In this section, we will show how such settings can be used to express various semantics for PDMSs.

The characterization of $\mathcal{S}_S$ is reminiscent of *schema mappings* as considered for example in data exchange [46]. And in fact, like for data exchange, most schema level mapping based PDM settings use tgds (or slight variations thereof) to define P2P mappings, and sets of tgds and egds on single peer schemas $\mathcal{R}_i \in \mathcal{R}$ to define local constraints. Unfortunately, just applying the typical semantics of schema mappings to PDMSs is no satisfying solution, as reasoning becomes undecidable and the structure of the system cannot be modeled appropriately, leading to a loss of peer autonomy. As this raised a lot of work on identifying suitable semantics that on the one hand resolve these problems and on the other hand support a wider range of applications, we devote this section to the discussion of these suggestions.

5.1 Global and Local Reasoning

A major distinction between such PDMSs $\mathcal{S}_S$ is made according to whether $\mathcal{M}$ is interpreted under *global* or *local* reasoning[1]. Global reasoning means that $\mathcal{M}$ is interpreted as a single (global) FO theory. This is the semantics obtained by extending data exchange [46] and data integration [54] scenarios to P2P settings. Under local reasoning, each peer is modeled as a distinct (local) theory, and inter-peer mappings are interpreted as to exchange certain facts between such theories only. It was explicitly suggested in [14, 16], and occurs implicitly in the LRM on implicational coordination formulas.

It is easy to see that under global reasoning, deriving all certain answers may become an undecidable problem, due to the network topology (e.g., P2P mappings could form not weakly acyclic sets of tgds [20], or in combination with local egds could form sets of 1-key conflicting inclusion dependencies [13]). Global reasoning further does not completely reflect the modularity of PDMSs [16]. On the other hand, it allows to derive more information than local reasoning. The latter two properties are illustrated in the following example.

▶ **Example 4** (cf. [22]). Assume a PDMS $\mathcal{S}_S = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, with $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$, $\mathcal{R}_1 = \{C\}$, $\mathcal{R}_2 = \{M, F\}$, $\mathcal{R}_3 = \{TP\}$, (let C stand for *Citizen*, F for *Female*, M for *Male*, TP for *TaxPayer*, and let all be unary) and $\mathcal{M} = \{C(x) \to M(x) \vee F(x), M(x) \to TP(x), F(x) \to TP(x)\}$[2], and queries $Q_1 \colon q(x) \leftarrow TP(x)$, $Q_2 \colon q(x) \leftarrow M(x)$, and $Q_3 \colon q(x) \leftarrow F(x)$.

For an instance $I = \{C(alice)\}$, $Q_2^c(I) = Q_3^c(I) = \emptyset$ (under both kinds of reasoning), as $\mathcal{I}_I$ contains two instances not containing $F(alice)$ and $M(alice)$ respectively. Under global reasoning however, $Q_1^c(I) = \{q(alice)\}$, as $TP(alice)$ is derivable in any $I' \in \mathcal{I}_I$. While entailing a maximal amount of information, this contradicts modularity in a way as mappings not only transfer the "visible" content of a peer, but the information exchanged depends on the complete structure of the network (hence it is to some extend unpredictable for a single peer). Under local reasoning on the other hand $Q_1^c(I) = \emptyset$, as mappings only exchange information present in every $I' \in \mathcal{I}_I$, which is neither the case for $F(alice)$ nor $M(alice)$. ◀

[1] Do not confuse the notions of global and local reasoning here with the global and local semantics in [22]. The (equivalent) latter two notions are concrete formalizations of local reasoning.

[2] We use disjunctive tgds for the sake of illustration only. Similar effects occur with ordinary tgds as well.

Note that mappings defined over a single peer schema behave equivalent under both kinds of reasoning. We next take a closer look onto these two formalisms and two concrete examples.

5.1.1 Global Reasoning & $\mathcal{PPL}$

If we consider a PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where $\mathcal{M}$ consists of FO formulas, the typical semantic of $\mathcal{S}$ under global reasoning is defined in terms of a FO theory T that contains all formulas in $\mathcal{M}$. Given an instance I for $\mathcal{S}$ (i.e. either for $\mathcal{R}$ or for the source relations $\mathcal{L}_i$), $\mathcal{I}_I$ is defined by all models of T that agree with I. As already noted, considering inter-peer tgds and local tgds and egds leads to settings where query answering becomes undecidable (cf. [20, 16, 38]) as it requires to derive all information implied by T. The only ways to overcome this are either to drastically restrict the allowed mapping language or to restrict the structure of T, and therefore the topology of the inter-peer mappings. While the first choice often requires weak mapping languages that render the complete system useless, the latter reduces the autonomy of the peers, as they are no longer free to define mappings to neighbors arbitrarily. Hence, one has to find a trade-off between these two possibilities.

One of the first and most prominent examples for a PDMSs dealing with this trade-off is the Piazza PDMS [35, 36, 37, 38, 71]. Mappings in Piazza are defined using the *Peer Programming Language ($\mathcal{PPL}$)*, introduced for this purpose in [37, 38].

▶ **Definition 5.** A $\mathcal{PPL}$-PDMS $\mathcal{S}_{\mathcal{PPL}} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local data integration system (where possibly $\mathcal{L}_i = \emptyset$), and $\mathcal{M} = \mathcal{M}_L \cup \mathcal{M}_P$ is a set of mappings defined in $\mathcal{PPL}$. Thereby $\mathcal{M}_L$ contains the mappings between all $\mathcal{L}_i$ and $\mathcal{R}_i$, while $\mathcal{M}_P$ contains the mappings and constraints on $\mathcal{R}$.

Due to space restrictions, we can only give a short overview on $\mathcal{PPL}$. $\mathcal{PPL}$ distinguishes two general kinds of mappings, *storage descriptions* (used to define $\mathcal{M}_L$) and *peer mappings* (for $\mathcal{M}_P$). Storage descriptions are either exact or sound LAV mappings between $\mathcal{L}_i \in \mathcal{L}$ and $\mathcal{R}_i \in \mathcal{R}$. Peer mappings are either exact or sound GLAV mappings or certain GAV-style mappings (*definitional mappings*; defined as datalog rules with a single atom in the head and a CQ in the body) between two (not necessarily distinct) peer schemas $\mathcal{R}_i, \mathcal{R}_j \in \mathcal{R}$. ([37, 38] allow all mappings in $\mathcal{M}$ to refer to arbitrary relations in $\mathcal{R}$. However, by introducing additional relations and mediator peers, the above definition is equally expressive, but more modular.) Given an instance I for $\mathcal{L}$, an instance I' for $\mathcal{R}$ satisfies all LAV and GLAV mappings according to the usual semantics (cf. [54]). The definitional mappings are satisfied by I' if for each relation R occurring in the head of such a mapping, $R^{I'} = \bigcup_{i=1}^{n} Q_i(I')$, where the Q_i are the bodies of the n definitional mappings where R is the head predicate symbol. Given I, the goal is not to materialize any data under $\mathcal{R}$, but at query time to return the certain answers w.r.t. all instances I' satisfying $\mathcal{M}$ that are equal to I on $\mathcal{L}$.

The expressive power of $\mathcal{PPL}$ requires to constrain the topology of mappings in $\mathcal{M}$ in order to allow for decidable query answering. Following the notion of acyclicity defined in Section 2, we obtain the following results.

▶ **Theorem 6** ([38]). *Given a $\mathcal{PPL}$-PDMS $\mathcal{S}_{\mathcal{PPL}}$, a CQ Q, and an instance I for $\mathcal{S}_{\mathcal{PPL}}$, computing $Q^c(I)$ is undecidable. If $\mathcal{M}$ contains only sound LAV and sound GLAV mappings, and $\mathcal{M}$ is acyclic, then computing $Q^c(I)$ is in polynomial time (data complexity).*

[38] also presented an algorithm that runs in polynomial time (data complexity) and computes certain answers w.r.t. a $\mathcal{PPL}$-mapping. While it is guaranteed to always return only certain answers, it also returns all of them for acyclic mappings. One key observation for this algorithm is that each GLAV mappings can be split into one LAV and one GAV mapping.

The algorithm then creates a rule-goal tree, by interleaving steps of query unfolding (for GAV style mappings) and answering queries using views [34] for LAV style mappings. At the end, the query is rewritten in terms of the local schemas $\mathcal{L}_i$.

Although the border of decidability and tractability can be pushed further by resorting to a little bit less restrictive constraints on $\mathcal{M}$ that still allow for decidable (tractable) query answering (see [38]), the general problem of using global constraints remains.

5.1.2 Local Reasoning: Exchanging Certain Answers

Local reasoning applies an interpretation to inter-peer tgds that allows for decidable query answering whenever query answering over each isolated peer can be decided. The basic idea is that P2P mappings are not satisfied by a single consistent global instance, but by the set of all such instances. Intuitively, they only exchange certain answers. This idea has been formalized in several (equivalent) ways: It was explicitly introduced in [14] by using the modal logic KT45 (by modeling a tgd as sentence $\forall\vec{x}\big(\mathbf{K}(\exists\vec{z}\phi(\vec{x},\vec{z})) \rightarrow \exists\vec{y}\psi(\vec{x},\vec{y})\big)$, where $\mathbf{K}$ is a modal operator expressing certainty) together with a distributed algorithm for query answering, and shown to support modularity better than global reasoning in [16]. In [22], two further formalisms for defining the semantics of a PDMS were introduced. Based on ideas of the LRM they consider a restricted form of coordination formulas and were shown to be equivalent with the formalism in [14]. Finally, in [27] local reasoning was formalized directly via sets of global consistent instances. Here, we follow this approach: Assume a PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ with $\mathcal{M} = \mathcal{C}_M \cup \mathcal{M}_M$, where $\mathcal{C}_M = \{\mathcal{C}_i \mid P_i \in \mathcal{P}\}$ are formulas of some logic $\mathcal{L}$ defined over single peer schemas (including, if present, the mappings between $\mathcal{L}_i$ and $\mathcal{R}_i$), and $\mathcal{M}_M$ is a set of inter-peer tgds. Satisfiability of $\mathcal{C}_M$ is still defined for single instances: Given an instance I for $\mathcal{R}$, some instance I' satisfies $\mathcal{C}_M$ w.r.t. I if, for every $\mathcal{R}_i \in \mathcal{R}$, it satisfies the logical theory containing $\mathcal{C}_i$ and the facts in I. For $\mathcal{M}_M$ on the other hand, being satisfied is defined for *sets* of instances: Given an instance I, a set $\hat{\mathcal{I}}$ of ground instances I' for $\mathcal{R}$ satisfies $\mathcal{M}$ w.r.t. I if (i) each $I' \in \hat{\mathcal{I}}$ satisfies $\mathcal{C}_M$ w.r.t. I, and (ii) for each tgd $\tau \in \mathcal{M}_M$ with $\tau = \forall\vec{x}(\exists\vec{z}\phi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}\psi(\vec{x},\vec{y}))$, it holds that $\bigcap_{I'\in\hat{\mathcal{I}}} Q_\phi(I') \subseteq \bigcap_{I'\in\hat{\mathcal{I}}} Q_\psi(I')$, where Q_ϕ, Q_ψ are the CQs associated to the lhs and rhs of τ, respectively. I.e., instead of testing for each instance $I' \in \hat{\mathcal{I}}$ if all answers to Q_ϕ are also answers to Q_ψ, under local reasoning the certain answers to Q_ϕ w.r.t. $\hat{\mathcal{I}}$ must be contained in the certain answers to Q_ψ.

Hence, given some instance I the semantics of $\mathcal{S}$ (i.e. the information implied by $\mathcal{M}$) depends on the set $\hat{\mathcal{I}}$. Since there may exist more than one possible set to choose from, the question is which is the "right" one. However, note that if two distinct sets $\hat{\mathcal{I}}$ and $\hat{\mathcal{I}}'$ satisfy $\mathcal{M}$, so does $\hat{\mathcal{I}} \cup \hat{\mathcal{I}}'$. Hence there exists a unique maximal set that satisfies $\mathcal{M}$. The set $\mathcal{I}_I$ is thus defined as this maximal set, and mappings in $\mathcal{M}_M$ are interpreted w.r.t. $\mathcal{I}_I$. I.e., the data implied by $\mathcal{M}_M$ is defined as those information shared by all instances $I' \in \mathcal{I}_I$. Given some instance I, all information implied by $\mathcal{M}_M$ can be efficiently (data complexity) materialized using a variant of the chase. Recall that the chase "repairs" violations of tgds (witnessed by tuples $\vec{t} \in Q_\phi(I) \setminus Q_\psi(I)$) by adding tuples into I s.t. $\vec{t}$ is also an answer to Q_ψ in the resulting instance. Now while the traditional chase considers all tuples $\vec{t}$, for local reasoning it suffices to only consider tuples that contain no labeled nulls. Beside efficient reasoning, this procedure thus provides also an alternative, procedural description of the semantics of local reasoning: tgds (an thus, peers) exchange only ground tuples.

One of the most elaborated frameworks based on this semantics is the PDEI-system also defined in [27], which we discuss next and use to illustrate the above definitions.

▶ **Definition 7.** A PDEI-System (Peer Data Exchange and Integration) $\mathcal{S}_{PDEI} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance, and $\mathcal{M} = \mathcal{C}_E \cup \mathcal{C}_I \cup \mathcal{M}_E \cup \mathcal{M}_I$, where $\mathcal{C}_E$ and $\mathcal{C}_I$ are sets of tgds and egds defined over single peer schemas $\mathcal{R}_i \in \mathcal{R}$, and $\mathcal{M}_E$ and $\mathcal{M}_I$ are sets of inter-peer tgds.

According to the semantics described above, $\mathcal{C}_E$ and $\mathcal{C}_I$ are interpreted as FO-theories over $\mathcal{R}$, while $\mathcal{M}_E$ and $\mathcal{M}_I$ only need to be satisfied w.r.t. tuples that are present in all $I' \in \mathcal{I}_I$. In addition, PDEI-systems allow for both materialization of data and "virtual" exchange: The mappings in $\mathcal{C}_E \cup \mathcal{M}_E$ are enforced on the instance on $\mathcal{R}$, i.e. tuples are materialized to satisfy them, while those in $\mathcal{C}_I \cup \mathcal{M}_I$ are only considered for query answering (if some tuple can be derived by both kinds of mappings, it needs not to be materialized). Hence reasoning over a PDEI-system is twofold: Given some instance I for $\mathcal{R}$, the goal is on the one hand to materialize an instance over $\mathcal{R}$ that satisfies $\mathcal{C}_E \cup \mathcal{M}_E$ ("admissible instance"), and on the other hand to answer queries over such an instance by returning certain answers also w.r.t. $\mathcal{C}_I \cup \mathcal{M}_I$. For these tasks to be decidable, a PDEI-system $\mathcal{S}$ has to be *stratified*, i.e. $\mathcal{C}_E$ is weakly acyclic, $\mathcal{C}_I$ consists of legal key constraints and foreign key dependencies (FK) only, and no head of a FK appears in the lhs of any tgd in $\mathcal{C}_E$. Note that these are local constraints only, verifiable by each peer $P_i \in \mathcal{P}$ in separation. For such systems, computing admissible instances and query answering can be done by combining a variant of the chase that considers the special semantics of inter-peer tgds (to retrieve information implied by $\mathcal{C}_E \cup \mathcal{M}_E \cup \mathcal{M}_I$) with query rewriting (to access information implied by $\mathcal{C}_I$).

▶ **Theorem 8** ([27]). *Let $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ be a PDEI-System, and I an instance for $\mathcal{R}$. If $\mathcal{S}$ is stratified, then an admissible state I' for $\mathcal{R}$ and $Q^c(I')$ for a CQ Q can be computed in polynomial time (data complexity).*

5.1.3 Schema Mappings and the LRM

We shortly comment on the relationship between global and local reasoning over schema mappings and the LRM by discussing the translation of some schema mapping based PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ into a LRM-PDMS. Assuming a shared domain *dom* between all peers (i.e. $r_{i,j} = \{(a, a) \mid a \in dom\}$ for all $P_i, P_j \in \mathcal{P}$), we can neglect the effect of the domain relations.

For global reasoning, assume that $\mathcal{M}$ is a set of function free FO formulas over $\mathcal{R}$. This corresponds to a LRM-PDMS $\hat{\mathcal{S}} = (\{\hat{P}_1\}, \{\hat{\mathcal{R}}_1\}, \hat{\mathcal{M}})$, where $\hat{\mathcal{M}} = \{1 : \phi \mid \phi \in \mathcal{M}\}$. This also illustrates nicely how the structure of the PDMS is lost under global reasoning. Given an instance I for $\mathcal{S}$, $\mathcal{I}_I$ w.r.t. $\mathcal{S}$ contains exactly those instances I' for $\hat{\mathcal{S}}$ that satisfy $\hat{\mathcal{M}}$ and s.t. there is a homomorphism from I into I'.

We already mentioned that when considering certain implicational coordination formulas, the LRM exhibits exactly the semantics of local reasoning. To see this, assume in accordance with the last subsection, that $\mathcal{M}$ contains function free FO formulas over single peer schemas and inter-peer tgds only. This translates to a LRM-PDMS $\hat{\mathcal{S}} = (\mathcal{P}, \mathcal{R}, \hat{\mathcal{M}})$, where $\hat{\mathcal{M}}$ contains one coordination formula $i : \phi$ for each formula $\phi \in \mathcal{M}$ over $\mathcal{R}_i$, and for each tgd $\forall \vec{x}\big(\exists \vec{z} \phi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y} \psi(\vec{x}, \vec{y})\big)$ between peers P_i, P_j one coordination formula $\forall i : \vec{x}\big(i : (\exists \vec{z} \phi(\vec{x}, \vec{z})) \rightarrow j : (\exists \vec{y} \psi(\vec{x}, \vec{y}))\big)$. Again, for an instance I for $\mathcal{S}$, $\mathcal{I}_I$ w.r.t. $\mathcal{S}$ contains exactly those instances I' for $\hat{\mathcal{S}}$ that satisfy $\hat{\mathcal{M}}$ and agree with I.

5.2 Inconsistency Handling

Given a PDMS $\mathcal{S}$, we say an instance I for $\mathcal{S}$ is inconsistent (w.r.t. $\mathcal{S}$) if $\mathcal{I}_I = \emptyset$. One drawback of the semantics and approaches seen so far is that they cannot deal with such a situation.

They do not have sensible notions for query answering nor materialization in this case, and therefore their algorithms will either fail or return useless results. This is unsatisfactory, as it is very unlikely that a complete PDMS is consistent. In general, two kinds of inconsistencies are distinguished. *Local inconsistency* occurs if the data stored at a single peer is already inconsistent. In case of *P2P inconsistency* each peer is locally consistent, but the local data contradicts data implied by inter-peer mappings, or a peer imports contradicting information from different sources. In both cases, inconsistency occurring at a single peer immediately renders the complete system useless (as it leads to global inconsistency).

Local inconsistency is in general addressed by "excluding" locally inconsistent peers, i.e. by defining semantics that behave as if these peers were not part of the network. P2P inconsistency on the other hand is tackled by defining semantics that consider suitable repairs of the data, either by not importing contradicting facts or by ignoring the local data.

In [15], the first approach has been taken: In case of P2P inconsistencies, the local data at each peer is preferred, and a maximal amount of consistent data is imported. Applying local reasoning and considering an integration system, this semantics is formalized as an extension of [16] using the nonmonotonic, multi-modal epistemic logic $K45_n^A$.

▶ **Definition 9** ([15]). A P2PDIS is a PDMS $\mathcal{S}_{P2PDIS} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local data integration system and $\mathcal{M} = \mathcal{M}_L \cup \mathcal{M}_R \cup \mathcal{M}_P$. $\mathcal{M}_R$ is a set of constraints $\mathbf{K}_i\phi$ where ϕ is a function free FO formula over a single peer schema $\mathcal{R}_i \in \mathcal{R}$. $\mathcal{M}_L$ is a set of mappings $\mathbf{K}_i(\forall\vec{x}(\exists\vec{z}\phi(\vec{x},\vec{z})) \rightarrow (\exists\vec{y}\psi(\vec{x},\vec{y})))$ between some $\mathcal{L}_i$ and $\mathcal{R}_i$, and $\mathcal{M}_P$ contains inter peer mappings $\forall\vec{x}(\neg\mathbf{A}_i\bot_i \wedge \mathbf{K}_i(\exists\vec{z}\phi(\vec{x},\vec{z})) \wedge \neg\mathbf{A}_j(\neg\exists\vec{y}\psi(\vec{x},\vec{y})) \rightarrow \mathbf{K}_j(\exists\vec{y}\psi(\vec{x},\vec{y})))$ from P_i to P_j, where ϕ and ψ are conjunctions of atoms.

Again, $\mathbf{K}_i$ and $\mathbf{A}_i$ are modal operators from $K45_n^A$. Let I be an instance for $\mathcal{S}$. Intuitively, $\mathcal{M}_R$ and $\mathcal{M}_L$ express that the local mappings and constraints must be satisfied in each $I' \in \mathcal{I}_I$. The intuitive reading of the P2P mappings is as follows: If peer P_i is not locally inconsistent, and $\exists\vec{z}\phi(\vec{x},\vec{z})$ holds in every $I' \in \mathcal{I}_I$, and $\exists\vec{y}\psi(\vec{x},\vec{y})$ is consistent with the data at P_j, then $\exists\vec{y}\psi(\vec{x},\vec{y})$ should hold at P_j. Being a proper extension of local reasoning as described before, the drawback of the approach is the high complexity of query answering.

▶ **Theorem 10** ([15]). *Let $\mathcal{S}_{P2PDIS}$ be a P2PDIS where $\mathcal{M}_L$ is a set of GAV mappings and $\mathcal{M}_R$ contains key constraints only, and Q a CQ over some $\mathcal{R}_i \in \mathcal{R}$. Given an instance I for $\mathcal{L}$, and tuple t, deciding if $t \in Q^c(I)$ is coNP-complete (data complexity).*

Using repairs, [7, 8] considered both, omitting imported data and local data to resolve inconsistencies. This approach is able to deal with inconsistencies between imported and local data, but not with inconsistencies between data imported from different sources.

▶ **Definition 11** ([7]). A CPDE-System is a PDMS $\mathcal{S}_{CPDE} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance and $\mathcal{M} = \bigcup_{P_i,P_j\in\mathcal{P}} \mathcal{M}_{(P_i,P_j)} \cup \bigcup_{P_i\in\mathcal{P}} \mathcal{M}_{P_i}$. Each $\mathcal{M}_{(P_i,P_j)}$ and $\mathcal{M}_{P_i}$ may be empty or contain full disjunctive tgds and tgds with one atom in the lhs and rhs, defined over a single $\mathcal{R}_i \in \mathcal{R}$ ($\mathcal{M}_{P_i}$) or two $\mathcal{R}_i, \mathcal{R}_j \in \mathcal{R}$ ($\mathcal{M}_{(P_i,P_j)}$). In addition, each $\mathcal{M}_{(P_i,P_j)} \neq \emptyset$ is annotated with a trust relation $(P_i, [< \mid =], P_j)$.

P_j is a neighbor of P_i if $\mathcal{M}_{(P_i,P_j)} \neq \emptyset$, and $\mathcal{M}_{(P_i,P_j)} \neq \mathcal{M}_{(P_j,P_i)}$ is allowed. Based on the assumption that the transitive closure of the neighbor relation is acyclic, the semantics is inductively defined based on *neighborhood solutions* and *solution instances*: Given an instance I for $\mathcal{S}$, for a peer P_i with $\bigcup_{P_j\in\mathcal{P}} \mathcal{M}_{(P_i,P_j)} = \emptyset$, the set of *solution instances* $\mathcal{I}_I[P_i]$ for P_i is defined as the set of minimal repairs of $I|P_i$, and a *solution* $S(P_i) = \bigcap_{I'\in\mathcal{I}_I[P_i]} I'$. For a peer with neighbors $P^1, \ldots, P^n$, the *neighborhood solutions* are defined via repairs of the instance

$\hat{I} = I|P_i \cup \bigcup_{P^j} S(P^j)$ over the combined schemas of these peers, satisfying $\mathcal{M}_{P_i}$ and all P2P mappings $\mathcal{M}_{P_i,P^j}$. Thereby only those repairs are considered that are "closest" (cf. [7]) to $\hat{I}$ and agree with $\hat{I}$ on all $\mathcal{R}_j$ s.t. $(P_i, <, P_j)$, i.e. on data from neighbors trusted more than the local data. $\mathcal{I}_I[P_i]$ is then defined as the set of all neighborhood solutions restricted to $\mathcal{R}_i$.

Being an integration system, no data is materialized, but the goal is, given a query over some peer P_i and an instance I to compute the certain answers w.r.t. $\mathcal{I}_I[P_i]$. As shown in [7, 8], the problem can be encoded as answer set program. However, due to the disjunctive tgds and the repair semantics, it is intractable in general.

▶ **Theorem 12** ([7]). *Let $\mathcal{S}$ be an acyclic CPDE-System and Q a FO query over an $\mathcal{R}_i \in \mathcal{R}$. Given instance I for $\mathcal{R}$ and a tuple t, deciding if $t \in Q^c(I)$ is Π_2^P-complete (data complexity).*

In [22], the formalization of local reasoning presented there was also extended to address local inconsistency by redefining the semantics of mappings from locally inconsistent peers.

5.3 Update Exchange

Another problem not considered by the approaches presented so far are updates. For integration systems, in fact nothing changes in case of an update, as a query is answered on the data present at query time. In exchange systems however, updates pose several problems, based on the fact that later updates may revise and contradict earlier ones. Think e.g. of changing a non-key value. Standard methods like the chase would lead to inconsistencies. Another problem are e.g. deletions that lead to a mapping violation. Again, the chase would just undo this deletion, which is not satisfactory. Hence the methods discussed previously may not be appropriate in such settings, where data is both materialized and continuously changed (often referred to as *Collaborative Data Sharing (CDS)* or *Collaborative Data Integration*). The main idea behind approaches addressing these specific problems is not to exchange the information directly, but to exchange information about the updates. We will next take a look onto the most prominent examples of this approach.

5.3.1 Orchestra

One of the first and most cited approaches to realize CDS was formulated in the Orchestra project [43, 72, 31, 28]. Most notably its semantics can be defined almost completely in terms of schema mappings, although it differs a lot from the semantics seen so far.

▶ **Definition 13.** An O-PDMS is a PDMS $\mathcal{S}_{Orchestra} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instances, and $\mathcal{M}$ is a weakly acyclic set of tgds τ_i and key constraints. Each $\tau_i \in \mathcal{M}$ may contain relations from several peer schemas in both, its lhs and rhs and has a *trust condition* θ_i attached.

Trust conditions assign to each update propagated by a tgd a numerical *priority*, based on the content and the provenance of the update. The restriction to weak acyclicity is only made to guarantee the termination of the chase, so every set of tgds on which the chase terminates can be used.

So far, the setting looks similar to the previous ones. However, a different scenario is assumed, resulting in a completely different semantics. Here, every user works on its local database instance, i.e. queries are answered only locally and updates only affect the local instance. Further, updates done by users on their local databases are not immediately visible to the other peers in the network, but recorded in a local update log. At any time, a peer can decide to either publish its updates, which means that they are copied into some global

update store. Or he can decide to import updates done at other peers into his local database. In this case, all updates published to the global update store by any peer since the last import are first translated according to the mappings $\mathcal{M}$, then filtered according to the trust conditions, and finally checked for mutual conflicts between these updates as well as for conflicts with the local updates. Thereby the system is explicitly designed to handle such conflicting updates: Local updates are always preferred to updates done at other peers, and conflicts between imported updates are either resolved using the trust conditions, and if this is not possible, the updates are deferred and a user has to select which to apply.

Formally, assume peer P_u chooses to import updates published by the others. Then the content of its peer relations is defined by the following PDMS $\mathcal{S}' = (\mathcal{P}, \mathcal{R}', \mathcal{M}')$, where $\mathcal{R}'$ contains for each peer relation $R \in \bigcup_{\mathcal{R}_i \in \mathcal{R}, R_j \in \mathcal{R}_i} R_j$ five relations: R^ℓ (*local contributions* table), R^r (*rejections* table), R^i (*input* table), R^t (*trusted input* table), and R^o (*output table*). Further, $\mathcal{M}'$ contains the following mappings: For each $\tau \in \mathcal{M}$, $\mathcal{M}'$ contains a mapping τ' obtained from τ by replacing any relation symbol in the lhs by the corresponding R^o, and each relation symbol on the rhs by R^i. Further, for each R, $\mathcal{M}'$ contains the tgds $R^i(\vec{x}) \wedge \mathit{trusted}(\vec{x}) \to R^t(\vec{x})$, $R^t(\vec{x}) \wedge \neg R^r(\vec{x}) \to R^o(\vec{x})$, and $R^\ell(\vec{x}) \to R^o(\vec{x})$. Thereby *trusted* is no real relation, but just denotes the filtering of updates according to the trust conditions and resolving of conflicts using priorities (see below). In fact, in [28], $R^t(\vec{x})$ was defined as $R^t(\vec{x}) = \mathit{trusted}(R^i(\vec{x}))$.

The content of the relations in $\mathcal{R}'$ is defined based on the information retrieved from the global update store: First, the sequence of updates in the store is *flattened* [43], i.e. dependencies between updates are removed (if for example a tuple t is first inserted and then changed to t', these updates are replaced by just inserting t'. Also, if a tuple is first inserted and then deleted, both updates are removed), such that no update depends on another one. Based on this flattened update sequence, an instance I for the relations in $\mathcal{R}'$ is defined as follows: Each R^ℓ contains all tuples locally inserted into R. R^r contains all tuples that were not inserted locally into R, but were delete from R according to the log (this means, the tuple was imported during an earlier update, and then deleted. It should therefore not be reinserted). R^i, R^t, and R^o are left empty. For each $R \in \mathcal{R}_u$, the content of R after the update is now defined as the content of R^o after chasing I with $\mathcal{M}'$.

Inconsistencies occur whenever for the same key values, different updates show up (e.g., if two different values are assigned to the non-key values). Due to space restrictions, we only sketch the basic idea of the conflict resolution algorithm (for details see [72, 28]): Updates in R^i are considered as candidate updates. If such a candidate update conflicts with local data, always the local data is preferred. Conflicts between candidate updates are resolved by the trust mappings: The update with the higher priority is chosen, the other discarded. In case that several conflicting updates have the same priority, the update is deferred until the user selected one to apply. As the priority value of an update depends heavily on its provenance, using an appropriate provenance model is important. In Orchestra, provenance semirings are used, as on the one hand they are powerful enough to provide all required information, and on the other hand their provenance expressions can be nicely transformed into trust expressions. A description of the provenance model of provenance semirings can be found in [29], a detailed discussion of how trust (trust or distrust an update) can be derived from provenance expression is given in [28]. The reconciliation algorithm for conflicting updates based on priorities was presented in [72], and recently developed further in [26].

Note that the description above was meant to define *what* tuples should be contained in the local instance after the update, and not *how* these values are indeed computed, as this would require to recompute the complete content. Instead, the instances can be computed

incrementally. [28] presents algorithms for both, computing the effect of insertions and deletions, where the main problem is to determine the effect of a deletion, as propagating this deletion means to find all tuples that are consequences from the deleted one and can no longer be derived from other local insertions.

5.3.2 Youtopia

Yet another approach was chosen more recently in the Youtopia system [52].

▶ **Definition 14.** A Y-PDMS is a PDMS $\mathcal{S}_{Youtopia} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance, and $\mathcal{M}$ is a set of tgds.

Note that except inter-peer tgds – which<f can always be repaired by inserting tuples – no further constraints exist on $\mathcal{R}$. Violations of some $\tau_i \in \mathcal{M}$ introduced by an update are repaired using a variant of the chase, which, in the presence of tgds only, cannot fail. As usual, a violation occurs if in an instance I for $\mathcal{S}$, there is a set of tuples (called *witness*) matching the lhs of a tgd τ, without appropriate tuples matching the rhs of τ. Youtopia now distinguishes two kinds of violations: A *lhs-violation* occurs if the tuple affected by an update is part of the witness (e.g., after tuple insertion or replacing all occurrences of a labeled null by the same constant). It is corrected by a *forward chase*, which means that new tuples are added to I just like in the traditional chase. If, however, the tuple affected by the update is not part of the witness (e.g., if the violation is due to a tuple deletion), this is called a *rhs-violation*. Repairing such a violation by a forward chase would just undo the update, which is probably undesired. Therefore, Youtopia resolves this via a *backward chase*, that deletes tuples in the witness from I such that the tgd is no longer violated. If this causes again a violation, it is again a rhs-violation, and therefore resolved the same way.

In this setting, there are now two unresolved problems: First, the forward chase may not terminate (the backward chase terminates after deleting all tuples the latest), and second, there might be several possibilities for deleting tuples to satisfy a violated tgd. Both issues are resolved by asking the user for input: For the backward chase, whenever there is more than one possibility for deletion, the user is prompted to select one of them. On the other hand, whenever the forward chase produces a tuple t for some relation $R^I \in I$ that can be mapped via a homomorphism h onto a tuple $t' \in R^I$, the chase is stopped, and the user has to decide whether to add t to R^I, or to apply h to I (thus indicating that t contains no new information). Note that both actions satisfy τ. Then the following was shown.

▶ **Theorem 15** ([52]). *Any forward chase will either stop along all paths and ask for user input or terminate after finitely many steps.*

This does still not guarantee the termination of the forward chase, and even if it eventually terminates, it might be running (or waiting for user interaction) for quite a long time. To not freeze the system while waiting for termination, Youtopia allows different chase sequences to run in parallel. A well-defined semantics for these concurrent chases is provided by defining useful notions of serialization (e.g., extending final-state serializability) and safety (of executing and interleaving chase steps). A discussion of the concurrency related notions and results is out of the scope of this survey. We thus have to refer to [52] for any details.

5.3.3 ECA Rules

Event-Condition-Action (ECA) rules are a general, often used technique to coordinate distributed systems by triggering actions based on the occurrence of certain events. Not

completely fitting to our definition of schema mapping based PDMSs, their use in PDMSs has been considered in [44, 45, 74], especially as a possibility for implementing schema mappings in exchange systems. We therefore close this section by a short review of them.

▶ **Definition 16.** A PDMS $\mathcal{S}_{ECA} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance, and $\mathcal{M}$ is a set of ECA-rules.

Following [74], we consider ECA rules in a generic way as rules of the form **WHEN** $<$ *event* $>$, **IF** $<$ *condition* $>$, **THEN** $<$ *action* $>$, where the "IF" part is optional. ECA-rules easily allow to react to changes and updates in the system. On the other hand, the event based character makes it hard to formally define $\mathcal{I}_I$ given an instance I for $\mathcal{S}$. The idea is therefore to see them as an implementation of schema mappings, that ensure that in case of updates a materialized instance remains consistent w.r.t. a set of schema mappings.

Work done in this area addresses two main topics: Creation and evaluation of ECA-rules. [45] proposed a distributed evaluation mechanism for ECA-rules: Given one rule including several peers, the idea is to split it into several subrules that are then distributed among the peers involved and allow for a distributed evaluation of the rule. [45] further introduces a powerful event language and algebra. [44, 74] both consider the problem of semi-automatically creating ECA-rules: Given default rules between standard schemas for a certain domain and mappings between these schemas and concrete peer schemas, the goal is to translates the default rules into rules between the peer schemas.

6 Alternative Semantics

In the previous section, we concentrated on approaches based on schema level mappings between different peers. While they represent an important part of the discussion on PDMSs, those systems do not represent the complete range of possible semantics. In this section, we discuss semantics and approaches to PDMSs that are not based on mappings defined on the schema level. As an example of data mappings, we will take a closer look onto *mapping tables*. After this we discuss the idea of not just mapping schemas to schemas and data to data, but to also map between data and schemas on the example of *data-schema interplay*.

6.1 Instance based mappings: Mapping Tables

One assumption common to all approaches in the previous section was that all peers share the same domain, i.e. that they use the same domain elements to represent the real world. However, this assumption might be too strong for certain applications and, in addition, restricts peer autonomy. Recall that the LRM uses domain relations to extend the semantics of schema mappings to also support domain translations. This idea was developed further by dropping the schema mapping and defining P2P mappings solely on the data level by specifying value correspondences only. One such approach, that can be considered as an extension of the LRMs domain relations, are *mapping tables* [49, 47].

▶ **Definition 17.** A MT-PDMS $\mathcal{S}_{MT} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is either the schema of a local database instance or of a local data integration system, and $\mathcal{M}$ is a set of *mapping tables*.

Note that in [49, 47], the data is assumed to be stored directly under each $\mathcal{R}_i$ only. However, as we will see, mapping tables are only used to translate queries between peer schemas, so this does not make any difference. Next we will first describe mapping tables and their

semantics as introduced in [49], and then, following [47], discuss how they can be used for query answering in PDMSs.

Mapping tables describe a relationship between data stored under two sets of attributes, extending domain relations in two ways: First, they need not be binary relations but may have bigger arities, and second, they may also contain variables and domain restrictions for these variables instead of just constants from the domain. Also, several mapping tables between two peers may exist.

In the following, assume that each attribute A_i appearing in $\mathcal{R}$ has its own domain $dom(A_i)$, and let $\mathcal{V}$ be a set of variables. An *attribute mapping* over a set $\vec{A}$ of attributes is a tuple t that contains for each $A_i \in \vec{A}$ either some $c \in dom(A_i)$, some $v \in \mathcal{V}$, or an expression $v - D$ where $v \in \mathcal{V}$ and D is a finite subset of $dom(A_i)$, i.e. $t = (t_1, \ldots, t_n)$ where $t_i \in (dom(A_i) \cup \mathcal{V} \cup \mathcal{D})$ and $\mathcal{D}$ is the set of expressions $v - D$. A *mapping table* T_M between two sets $\vec{A}, \vec{B}$ of attributes is a set of attribute mappings t^i over $\vec{A} \cup \vec{B}$ s.t. no variable occurs in two different attribute mappings $t^i, t^j \in T_M$ $(i \neq j)$. Further, a *mapping table constraint* τ is a triple $(T_M, \vec{A}, \vec{B})$. Intuitively, each mapping table T_M associates values for $\vec{B}$ to given values for $\vec{A}$. Formally, this is defined by means of *valuations*: For a variable v in an attribute mapping, let $\vec{A}_v$ be the set of attributes under which v appears. A valuation μ is a function mapping all constants in T_M onto themselves, each variable v in T_M into $\bigcap_{A_i \in \vec{A}_v} dom(A_i)$, and satisfies $\mu(v) \notin D$ for all expressions $v - D$ in T_M. Given a mapping table T_M and values $\vec{a}$ for $\vec{A}$, the set of values associated to $\vec{a}$ by T_M is $\vec{B}_{T_M}(\vec{a}) = \{\vec{b} \mid$ there exists a valuation μ s.t. $t[\vec{A}] = \vec{a}$ and $t[\vec{B}] = \vec{b}$ for some $t \in \mu(T_M)\}$. Finally a tuple t for attributes $\vec{C}$ with $\vec{A} \cup \vec{B} \subseteq \vec{C}$ satisfies $\tau = (T_M, \vec{A}, \vec{B})$ if $t[\vec{B}] \in \vec{B}_{T_M}(t[\vec{A}])$, and a relation R^I satisfies τ if each $t \in R^I$ satisfies τ. Even more expressive mappings can be constructed by composing mapping table constraints to *mapping table formulas (MTF)* as $MTF = \tau | (MTF \wedge MTF) | (MTF \vee MTF) | \neg MTF$, where τ is a single mapping table constraint, and the definition of whether a tuple satisfies a MTF is a straight forward extension of the corresponding notion for mapping table constraints.

According to the above definitions, values for attributes $\vec{A}$ not appearing in the mapping table have no translation to $\vec{B}$. Assuming that a mapping table contains only partial information, another interpretation considered in [49] is that such values map to any values for $\vec{B}$. However, this behavior can be explicitly defined under the above semantics using the $v - D$ construct. We therefore omit its discussion here.

Next, following [47], we discuss how to use mapping tables to define P2P mappings. The general idea is to use mapping tables to rewrite a query on one peer to a query on another peer, to forward this query, and to repeat these steps. Unlike similar procedures seen in the previous sections, in this case answers of the resulting queries are not merged into a single answer, but returned separately. One reason for this is that in a mapping table constraint it need not be that $|\vec{A}| = |\vec{B}|$, hence the arities of the rewritten queries may not match.

Towards this goal it is first necessary to define when such a rewriting is correct, defined as *sound rewritings* in [47]. Informally, the idea is that a rewritten query should return only such tuples t' that are translations of correct answers t to the original query (but t may not be derived directly due to missing data). The formal definition for a sound rewriting from $\mathcal{R}_i$ to $\mathcal{R}_j$ is based on a mapping table T_M covering all attributes in $\mathcal{R}_i \cup \mathcal{R}_j$. Such a mapping table constraint can be composed from several mapping tables (basically as conjunctive mapping table formula matching uncovered attributes to all values — see [47] for details). Let Q_1 be a CQ. For the ease of notation, assume for the moment that equalities in Q_1 are not expressed by reusing variables but explicitly, and let $\psi(\vec{x}, \vec{y})$ be the conjunction of these equalities, i.e. $Q_1\colon ans(\vec{x}_1) \leftarrow \exists \vec{y}_1 \phi(\vec{x}_1, \vec{y}_1) \wedge \psi(\vec{x}_1, \vec{y}_1)$. As now every variable occurs only once in ϕ, we

can identify variables with the attribute for the position where they occur, and therefore $\vec{x}_1$ defines a set of attribute names. A query $Q_2\colon ans(\vec{x}_2) \leftarrow \exists\vec{y}_2\phi'(\vec{x}_2,\vec{y}_2)\wedge\psi(\vec{x}_2,\vec{y}_2)$ is a a *sound rewriting* of Q_1 over $\mathcal{R}_j$ w.r.t. T_M if for every instance I for $\mathcal{R}_j$ and every $t' \in Q_2(I)$ there exists a valuation μ s.t. $t[\vec{x}_2] = t'$ holds for some $t \in \mu(T_M)$. A rewriting is further *complete* if it is sound and for every sound rewriting Q_2' and instance I it holds that $Q_2'(I) \subseteq Q_2(I)$.

The problem of testing if a CQ is a sound rewriting of another CQ was shown to be Π_2P-complete in [47]. There, also an algorithm for computing a complete rewriting of a CQ w.r.t. a mapping table T_M was presented, that we cannot discuss here due to space restrictions. Within a PDMSs, termination of the query forwarding is achieved in two ways. On the one hand, a maximal number of forwards and rewritings for each query can be specified. On the other hand for each query its path through the system is recorded. This record is used for cycle detection. Once a cycle is detected, forwarding on this path stops.

Returning our attention to mapping table formulas in general, there are two interesting problems that can be studied for sets of MTFs: The consistency and the inference problem. Given a mapping table formula τ and an attribute set $\vec{A}$, the consistency problem asks if there exists a relation for $R(\vec{A})$ that satisfies ϕ. The inference problem asks, given a set Σ of MTFs and a single MTF τ, if every relation satisfying Σ also satisfies τ. While the consistency problem is of obvious interest, the main interest in the inference problem stems from the wish to create mapping tables automatically. Given a set of mapping tables, the goal is to automatically derive new explicit mapping tables from them, as this may allow for more and more exact query rewritings. Unfortunately, both problems (being interreducible to each other) were shown to be NP-complete for the general case in [49].

Mapping tables have been combined with other data translation mechanisms (e.g., merge and conversion rules in [56]) or schema mappings (in form of ECA rules in [44]). They are also the main P2P mapping language in the Hyperion project [48, 4, 63] (see next section).

6.2 Data-Schema Interplay

Approaches for so-called data-schema-interplay extend the coordination formulas introduced in Section 4. In such formalisms, the domain dom_i also contains the names of the relations and attributes of the peer schema $\mathcal{R}_i$. Consequently, the formulas ϕ occurring in the coordination formulas can refer to both data and metadata. Pioneering work in this area was presented in [53]. An important example from the peer data management domain is HepToX [10].

7 PDMS prototype systems

So far, we discussed semantic approaches for PDMSs. We will now give a short overview on existing prototype systems, pointing out specifics or notable ideas. Due to space restrictions, we cannot give a general or detailed discussion on the implementation of PDMSs. For a deeper discussion of implementation related aspects, see [41].

A first suggestion for the general structure of a peer in a PDMS was already presented in the vision paper [6] that also first suggested the LRM. Not surprisingly, a peer consists of three main layers: The local data is managed by the storage layer. A P2P layer on the one hand is responsible for establishing and managing mappings and connections to other peers. On the other hand it also handles query rewritings, update exchange, domain translations, or any other kind of information exchange supported by the system. Basically all algorithms of interest for PDM are part of the P2P layer. Finally each peer is controlled through the interaction layer, containing e.g. the user interface. Most of the systems presented in the literature follow this schema.

Hyper. The *Hyper* framework [17] considered the implementation of local reasoning as introduced in [16] on a Data Grid architecture, with the main focus on how query answering under local semantics can be deployed on Grid infrastructure. We are not going to discuss the implementation on Grids here, but use this opportunity to shortly sketch the general idea for query answering under local reasoning with tgds $\tau\colon \forall\vec{x}(\exists\vec{z}\phi(\vec{x},\vec{z}) \to \exists\vec{y}\psi(\vec{x},\vec{y}))$ as inter-peer mappings (cf. Section 5.1.2). Recall that we can identify a CQ $Q_\tau\colon ans(\vec{x}) \leftarrow \exists\vec{z}\phi(\vec{x},\vec{z})$ with the lhs of each τ, and consider a PDEI-system $\mathcal{S} = (\mathcal{P},\mathcal{R},\mathcal{M})$ with $\mathcal{C}_E = \mathcal{M}_E = \emptyset$ but let $\mathcal{C}_I$ contain arbitrary local constraints, and an instance I for $\mathcal{S}$. The basic idea of the query answering algorithm is as follows: For each τ from P_i to P_j, add a new relation symbol R_τ of arity $|\vec{x}|$ to $\mathcal{R}_j$ and a new local constraint $\forall\vec{x}R_\tau(\vec{x}) \to \exists\vec{y}\psi(\vec{x},\vec{y})$ to $\mathcal{C}_I$. Now given a (arbitrary) query Q over some $\mathcal{R}_i$, first compute a *perfect reformulation* Q' of Q w.r.t. I and $\mathcal{C}_I$. (A perfect reformulation of a query w.r.t. an instance and a set of constraints is a rewriting of the query that, evaluated over the instance, returns exactly the certain answers w.r.t. the set of constraints.) Q' may contain original relation symbols from $\mathcal{R}_i$ (these parts of Q' can be evaluated over I immediately) as well as some of the new R_τ. To evaluate Q' on those, their content must be retrieved first. This is done by posing Q_τ on the peer the lhs of τ is defined on, where this procedure is repeated unless a cycle was detected. Finally Q' can be evaluated by iteratively evaluating the (reformulations of the) queries Q_τ and adding the results to R_τ until a fixpoint is reached. This general idea can be implemented e.g. using the chase (cf. [27]), or as datalog program (cf. [16]), and works for local reasoning in general.

coDB. In Section 5.1.2, we pointed out that [22] proposed a formalization of inter peer mappings that resolves to local reasoning in terms of (restricted) coordination formulas. This approach was implemented in the *coDB* PDMS [23, 21, 24]. Inter peer mappings are modelled as coordination formulas $i_1\!:\!\phi_1(\vec{x},\vec{y}_1) \wedge \cdots \wedge i_k\!:\!\phi_k(\vec{x},\vec{y}_k) \to i\!:\!h(\vec{x})$ (where $\vec{x} = \bigcup_{i=1}^{k}\vec{x}_i$), and domain relations are not considered. The very basic idea of the query answering algorithm is similar to that presented for Hyper above. Therefore we do not discuss it here. A detailed description of its distributed implementation can be found in [23, 24]. Notably, the authors consider the problem of changes in the mappings while the algorithm is running. They show that their algorithm is sound and complete w.r.t. those mappings that remain stable during the runtime of the algorithm. Evaluation results of the coDB system are presented in [23].

PeerDB. *PeerDB* [59, 62, 61] takes a completely different approach than those seen so far. Instead of defining semantic mappings between the peers, for each relation and attribute name a set of metadata (basically a list of keywords) is maintained. If a query is posed against a peer, the system identifies relations at other peers that might be worth also querying by finding matches between the keywords attached to the relations and attributes used in the query and those stored at the other peers. If a match is found, the corresponding relation is added to a list of candidate relations. This search is done by sending software agents to all neighbors of a peer, where they search for matches and are again forwarded. Forwarding is stopped after a certain number of times. The list of possible matches is sent back to the initiating peer, where it is ranked and presented to the user, who selects those relations to use. The query is then rewritten accordingly and sent to the corresponding peers that return the answer. Besides this completely different kind of mappings, the system further adapts the topology of the network such that peers that contribute a lot of answers and matches become a direct neighbor. (Being a neighbor just means to send agents directly to this peer.) PeerDB further supports caching of answers to reduce the required bandwidth.

Orchestra. We already presented the *Orchestra* system in Section 5.3.1. As mentioned there, the ideas of update translation and exchange as well as conflict reconciliation based on trust mappings and provenance have been all implemented in the Orchestra prototype system [30, 43]. One focus of the implementation is how to perform the update exchange incrementally, i.e. how to identify which tuples to add or to delete, without recomputing all instances from scratch (for a detailed discussion of the corresponding algorithms see [28, 43]). Towards these goals, Orchestra always tries to reuse existing relational database management systems (RDBMSs) as much as possible and to find efficient implementations of all these aspects on top of traditional RDBMSs. One example for this is the encoding of provenance information in an RDBMS. Another important aspect is the implementation of the global update store. [72] provides a comparison between a centralized and a distributed solution for the storage of the published update logs.

Youtopia. Another system we already discussed earlier is *Youtopia* [52] (Section 5.3.2). The distinguishing property of its implementation is probably its capability to support several concurrent chases. Although providing algorithms to identify conditions under which it is safe for a chase to proceed, Youtopia does not block chases until their execution is guaranteed to be save, but applies an optimistic strategy that allows further chase steps to be scheduled even if they are not safe. This may lead to conflicts that are detected and resolved by aborting the corresponding chase. As this in turn may lead to cascading aborts, [52] considers three different scheduling algorithms and provides an experimental comparison of them. It was show that the number of aborts can be reduced to what seems to be an acceptable number.

Hyperion. The *Hyperion* project [60] was a large project on PDMSs. Part of it was the development of the *Hyperion* PDMS [48, 4, 63]. The main focus of this system was the use of mapping tables (see Section 6.1) for P2P mappings. The prototype provides algorithms for checking consistency of mapping table formulas, deriving new mapping expressions from existing ones, and computing rewritings of queries according to mapping tables. Further, in addition to mapping tables, Hyperion also uses ECA-rules (see Section 5.3.3) to support update exchange between peers, which is used to keep different peer instances consistent. Obviously those updates are also translated according to the information provided by the mappings tables. Consisting of three main components, the Hyperion PDMS basically follows the general structure of PDMSs presented at the beginning of this section.

Humboldt Peers. Humboldt Peers is a full-fledged relational PDMS that offers different strategies for completeness-driven query answering [64, 65]. As such, it follows a best-effort approach. To preserve peer autonomy as much as possible, Humboldt Peers resorts completely to local reasoning both in query answering and in building statistics of the data distribution accessible through neighboring peers. For that purpose, query answers are exploited to maintain multi-dimensional histograms on the potential cardinality of query answers received from neighboring peers. Building on that statistics, each peer performs local optimization in that it cuts off less promising peers from further query processing. To limit resource consumption for the highly redundant problem of query answering in large PDMS, Humboldt Peers sends a time budget along with each query. Using these pruning strategies, it was shown in [64, 65] that response time for query answering can be cut down by one or more orders of magnitude while still yielding a high completeness of the query result that can be satisfying for many applications.

8 Non-relational PDM

So far we concentrated solely on systems based on the relational data model. In this section, we will discuss approaches based on other data models like XML or RDF. We will also loosen the restriction only to consider unstructured P2P systems a little bit and will include some hybrid systems as, unlike the relational case, they are very common in this area.

Piazza. It might be surprising to find the *Piazza* PDMS here, as we already discussed it in Section 5.1.1 related to schema mappings. This was because the mapping language $\mathcal{PPL}$ and the algorithm for query answering were introduced and formalized in terms of relational schemas [37, 38]. However, the Piazza system was designed and implemented to work on XML [35] and even to support both XML and RDF, including mappings between both data models [36] by resorting to the XML representation of RDF. Nevertheless all basic ideas described in Section 5.1.1 remain unchanged: Mappings are still either storage descriptions or directed peer mappings (both can either be inclusion or equality mappings), but expressed in an (adapted) fragment of XQuery instead of relational CQs. Using XQuery allows the definition of more complex mappings that account to the nesting structure of XML. The main focus of the prototype system lies on the algorithm for query answering and its optimization. Implemented as a centralized algorithm, the reformulation step is computed at that peer the query is posed on. A global system catalog allows to retrieve all mappings currently present in the system. Recall that the idea of the algorithm is to compute rewritings of the query via a rule-goal tree. The resulting queries over the source relations are then sent to the corresponding peers to be executed. As the size of this tree grows quickly w.r.t. the number of mappings, optimizing query rewriting is crucial. Several strategies have been considered in [70], including pruning (i.e. identifying subtrees whose expansion will not create any new answers), finding good strategies for which nodes to expand next, or the computation of "shortcuts" by mapping compositions. Further, instead of returning the complete answers at the end, whenever the rewriting produces a query over some source relations, it is immediately executed and the result is presented to the user. Concerning the implementation of PDMSs, these optimization methods are specific to Piazza.

AXML. Another XML-based approach that gained a lot of attraction is *Active XML (AXML)* [42, 2]. First of all, AXML is an extension of XML that allows XML-documents not to contain all information explicitly, but to embed web service calls. Executing those calls then retrieves new data that is appended to the document. In general these may be arbitrary web services just returning AXML documents, not giving rise to any PDMS. However, as part of the AXML project so called *AXML peers* were created (cf. [2]). Each of these peers contains a set of AXML documents, is capable of performing webservice calls, and may publish its own services. These services, defined as (parameterized) queries over the stored documents, give rise to the following AXML-PDMS[3]: Each peer stores a set of AXML documents and may provide access to parts of these information through a web service, defined as parameterized query over its documents and returning AXML documents. Information offered by other peers can be accessed by including calls to their web services into the own AXML documents. I.e., P2P mappings are defined in terms of queries on other peers, a concept similar to tgds, that can be considered to describe the data to import also in terms of queries. Within an AXML document, service calls are encoded as special

[3] Note that our definition of a PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ only makes sense for schema based systems.

XML elements whose children represent the call parameters. When activated, the root node of the result-AXML document is appended to the document as sibling of the call element. Several policies can be applied to define how long the result remains valid or what happens with the result in case the service is called again. One difficulty with this materialization of data is that since the result of a web service call can be an arbitrary AXML document, it may contain further call elements, hence materializing all implicit information might not be possible. As a result, when computing the answer to a query an important goal is to call only those web services necessary to answer the query, but maybe even return service calls (instead of their return values) in the answer (lazy evaluation). However, as web services in general are just black boxes for the caller, reasoning about these questions (termination, was already enough data materialized for query answering) is not possible. But as in AXML-PDMSs the service definitions are known (at least from some global point of view), these problems have been investigated in [1] for AXML-PDMSs where web services are defined using a monotone conjunctive fragment of XQuery. Still undecidable in the general case, several decidable fragments of these problems have been identified. We close the discussion by pointing to two (out of many) extensions of AXML: [18] suggested a trust model for services, and [3] proposed an algebra for evaluating AXML expressions. For a general overview on AXML see [2].

SmurfPDMS. *SmurfPDMS* focuses on skyline-querying in a PDMS setting [40, 39]. Similarly to *Humboldt Peers*, this prototype system employs so-called data summaries to route queries through the network of peers. This means, that paths are pruned if they do not promise a certain size of contribution to the query answer. The statistics in the data summaries are maintained by exchanging updates between peers. To limit this update traffic in the network, the updates are only propagated over a certain number of peer mappings. So the statistics at each peers have a limited horizon.

HePToX. We already mentioned *HePToX* [9, 10] shortly in Section 6.2. Unlike most of the other systems, that require the user to provide a complete specification of the mappings, HePToX heavily supports the mapping creation. The data stored at each peer must be structured according to a DTD. All the user has to do is to draw some arrows between elements of the schemas that relate to each other, and to visually group different elements that match to the same element in the other schema. The system then translates these mappings into HePToX mapping language, which are datalog-style rules, adapted to be able to deal with the nesting structure of XML documents. They are somewhat similar to nested tgds [25], using Skolem functions to identify nodes (instead of using existential variables like FO-tgds). Using these mappings, queries posed against the local schema of a peer and formulated in a fragment of XQuery are translated to match the schemas of the neighbors. Thereby a rewriting is considered to be correct if evaluating the rewritten query over the data of the other peer returns the same result as applying the transformation to the data and evaluating the original query there.

XPeer. XPeer [66] is the first hybrid P2P system that we consider. Peers, clustered around super-peers, publish a schema of their locally stored data to their super peer in terms of so called *tree-guides* that are automatically generated from the data. Basically these tree-guides represent the structure of the XML data but omitting the actual data. The super-peer network forms a tree and super-peers exchange information such that each super-peer knows about the schema information stored at its children. Queries, issued against single peers,

are handed over to the corresponding super-peer, from where they are routed through the super-peer network by trying to find matches between the query and the peer schemas stored at the super-peers. To improve this search for peer clusters that might be relevant for answering a query, the systems aims at assigning peers with similar schemas to the same super-peer, i.e. into the same cluster. Further, peers within the same cluster may replicate data from each other to speedup query answering. Also, the super-peer network adapts the number and distribution of super-peers automatically to the system load to avoid bottlenecks.

A survey of XML data management in P2P systems can be found in [51], focusing on the use of indices, clustering, replication and query processing in such systems.

The idea of PDM has been also picked up by the Semantic Web community, resulting in a variety of RDF based PDMSs. Interestingly, most of these systems differ greatly from the systems we have seen so far, introducing several new ideas for defining mappings. In the remainder of this section we will introduce some of those systems and discuss a selection of those approaches. Although arguable not all of them are covered by our definition of a PDMS we gave at the beginning, we think it is worth to mention them nevertheless. Probably influenced by the idea of the web, many of these systems do not define explicit mappings between pairs of peers: all peers in the network are considered as possible candidates for query forwarding, and the task is to identify those peers that indeed contain relevant data.

Edutella. One of the most prominent RDF based PDMSs is Edutella [55, 57, 58], that was originally designed for sharing educational resources by publishing RDF metadata describing these resources and to provide a querying service on this metadata. The main goal of Edutella is to provide an efficient mechanism for query routing, that forwards the query quickly to all peers in the network that may contribute to its answer, but avoiding forwarding the query to peers that do not. To reach this goal, a strong focus was laid on indices for query routing, which is specific to Edutella. Although Edutella offers several other functionalities, we will only shortly sketch the idea of using indices to guide query answering in a PDMS. Edutella is a *hybrid* P2P system where each peer connects to exactly one super-peer. Those super-peers, arranged in some predefined topology are then responsible for efficient routing. Queries posed against a peer are handed to the super-peer, from where they are routed through the super-peer network to peers that may contribute to the answer. Routing is based on several indices maintained by the super-peers: First of all, each super-peer stores information about the data provided by the peers connected to it that allows to determine if a peer can understand the query. This includes information on the peer schema and for which parts of this schema the peer actually contains data, but also the ranges of present values or other value summaries. If a query arrives at a super-peer, these indices are used to identify suitable peers for the query. In addition to the information about the peers connected to it, each super-peer also stores indices about its neighboring super-peers. These are summaries of the peer indices hold by the neighbors, describing the overall data offered by all peers connected to a neighbor. Routing within the super-peer network is directed by these indices.

To ensure that this routing strategy does not lead to basically broadcasting the query through the network, it is necessary that peers that may contribute to the answer are not distributed randomly in the network. Edutella therefore tries to cluster peers based on the data they offer. To do so, each super-peer can define certain constraints (like e.g. the supported schemas) peers have to satisfy in order to be allowed to connect to this super peer.

SQPeer and Bibster. We have just seen how index information are used for guiding query answering. Another possible use of data descriptions offered by a peer are advertisements,

which gives rise to a different way of defining mappings between peers. The general idea of advertisements is that peers announce description of their data. If another peer decides to store such an advertisement, a mapping between these peers is established: Every time a peer receives a query, it will not only try to answer it over its own data, but will also check all of its stored advertisements whether they are relevant for this query. Two systems following this approach in different ways are *Bibster* [33] and *SQPeer* [50].

Bibster was designed for the very limited scope of sharing bibliographic information stored in Bibtex files at each peer. Advertising is done referring to some global ontology (for the bibliographic domain, this was the ACM classification hierarchy). Each peer describes its *expertise*, i.e. that parts of the ontology for which it actually provides data, as subsets of this ontology. When a query is posed against a peer, it is first answered locally, and then the stored expertises are used to identify peers that may be worth forwarding the query to. This decision is based on a similarity measure computed between the query and advertisements. Note that due to the need of a global ontology, Bibster is not a PDMS according to our definition, but still an interesting approach as we think.

SQPeer takes a little bit different approach than Bibster. Storing again data under RDF schemas, similar to the situation in Edutella (and considered in several RDF based systems), a peer may not contain data for all parts of this schema. The advertisements of a peer consist of descriptions of the so called *active schema*, i.e. of those parts of the schema a peer actually stores data for. Queries are again sent through the network based on the published and stored advertisements. But instead of immediately answering the query, each peer uses its stored advertisements to identify which parts of the query could be answered by which peer and annotates the query with the corresponding information. At the end, the annotated query is sent back the peer where the query was originally issued. This peer then uses the annotated information from the query to contact all relevant peers and collects their answers.

9 Conclusion

In this paper, we provided a survey of Peer Data Management Systems (PDMSs), concentrating on the management of structured data in unstructured peer-to-peer (P2P) systems. The promise of these systems is the combination of a strong semantics (like for relational data integration or exchange) with the high flexibility and autonomy offered by P2P systems. As the design of a PDMS raises many questions, several suggestions have been made in the literature to overcome the different design problems. Providing a summary of these approaches, we laid a strong focus on the formalisms and semantics of the P2P mappings, that more or less determine the semantics of the overall system.

P2P mappings can be designed for different purposes. They may be used for integrating data at query time, for data exchange or update propagation. Mappings may be able to deal with inconsistencies or support translations between different domains. In summary, several characteristics of inter-peer mappings can be identified. They may be defined on schema or instance level or even between instances and schemas. Further, while some offer a well-defined semantics and allow for reasoning along them, others just define explicit rewriting or translation rules for queries or data. Another distinction can be made on whether they are explicitly defined or created on the fly at runtime, e.g. based on a query issued by a user and some additional metadata.

Beside the description of the main theoretical concepts, we also discussed some of the problems arising from the implementation of such systems, pointing out specifics of several prototype systems published in the literature.

Table 1 An overview on the most important systems and approaches presented in this survey. (p) in the "special mentions" column indicates that a prototype implementation exists, (e) and (i) in the "semantics" column denote exchange and integration systems, respectively, and $\subset L$ is used to denote a fragment of L. (* syntactically restricted FO formulas)

	data model	mapping level	mapping language	query language	semantics	special mentions	Ref.
trad. data exchange/ integration	relational	schema	tgds	UCQs	global		[46, 54]
Piazza	relational/ XML	schema	$\mathcal{PPL}$	UCQs $\subset$XQuery	global (i)	(p), restricted topology	[38]
PDEI-Framework	relational	schema	tgds	UCQs	local (e/i)	eff. decidable	[27]
LRM	relational	schema & instance	CFs/dom. relations	CFs	local (i)	domain translation	[67]
Orchestra	relational	schema	tgds	UCQs	global (e)	(p), trust, update exchange	[28]
Hyperion	relational	schema & instance	ECA & mapping tables	CQs	- (e/i)	(p)	[4]
[7],[8]	relational	schema	UDECs* RDECs*	FO	repair (i)	inconsistency handling	[7, 8]
Hyper	relational	schema	tgds	UCQs	local (i)	(p)	[17]
AXML	XML	schema	WS calls	various	- (i)	(p)	[2]
Youtopia	relational	schema	tgds	keyword & structured	global (e)	backward chase concurrency,(p)	[52]
coDB	relational	schema	$\subset$CFs	UCQs	local (i)	$\exists$ ext. for local inconsistency,(p)	[22]
HepToX	XML	data/schema interplay	datalog style rules	$\subset$XQuery	- (i)	graphical mapping generation,(p)	[10]
Humboldt Peers	relational	schema	GLaV	CQs w/o projections	local (i)	(p), compl. driven, data statistics	[65]
Smurf-PDMS	relational, XML	schema	GLaV	CQs	local (i)	(p), data statistics	[40]
PeerDB	relational	schema	keywords	CQs	- (i)	(p)	[59]

A summary of the more important systems and approaches discussed in this survey is presented in Table 1. It lists for each of them the main characterizing properties: The data model used, whether the P2P mappings are on schema or instance level (or a mixture of that), the formalism used to define the inter-peer mappings, how these mappings are interpreted (i.e. the semantics applied to the mappings), and whether the system is an exchange or integration system. The table further shows the query language that is either supported by the approach or discussed in detail within the context of the specific proposal. Any special characteristics of a system are stated in the "special mentions" column, where (p) indicates that a prototype implementation exists. Note that the table only contains a selection of systems and is not complete: even some systems mentioned in this survey do not show up.

Despite all these different approaches, none of them seems to have been established yet as *the* model for PDM. This may indicate that not all problems have been resolved in a satisfactory way yet. One of these problems might be data inconsistency. Although very elegant ways have been suggested for how to deal with contradicting data, most of the time they come for the price of a high computational complexity. In general, performance is

another critical aspect, and how to further speed up query answering or the data exchange also is an interesting research direction. Altogether, although the semantics of PDMSs is already understood quite well and the interest in those systems was decreasing the last two years, there is still research potential in PDM.

References

1 Serge Abiteboul, Omar Benjelloun, and Tova Milo. Positive active xml. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 35–45, 2004.

2 Serge Abiteboul, Omar Benjelloun, and Tova Milo. The active xml project: an overview. *VLDB J.*, 17(5):1019–1040, 2008.

3 Serge Abiteboul, Ioana Manolescu, and Emanuel Taropa. A framework for distributed xml data management. In *Proc. of the Int. Conf. on Extending Database Technology (EDBT)*, LNCS, pages 1049–1058. Springer, 2006.

4 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.

5 M.T. Ăzsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 2011.

6 Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and Ilya Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proc. of the ACM SIGMOD Workshop on The Web and Databases (WebDB)*, pages 89–94, 2002.

7 Leopoldo E. Bertossi and Loreto Bravo. The semantics of consistency and trust in peer data exchange systems. In *Proc. of the Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4790 of *LNCS*, pages 107–122. Springer, 2007.

8 Leopoldo E. Bertossi and Loreto Bravo. Information sharing agents in a peer data exchange system. In *Proc. of the Int. Conf. on Data Management in Grid and Peer-to-Peer Systems*, pages 70–81, 2008.

9 Angela Bonifati, Elaine Qing Chang, Terence Ho, and Laks V. S. Lakshmanan. HepToX: Heterogeneous peer to peer XML databases. *CoRR*, abs/cs/0506002, 2005.

10 Angela Bonifati, Elaine Qing Chang, Terence Ho, Laks V. S. Lakshmanan, and Rachel Pottinger. HePToX: Marrying XML and heterogeneity in your P2P databases. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 1267–1270. ACM, 2005.

11 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.

12 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.

13 Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 260–271. ACM, 2003.

14 Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic data integration in p2p systems. In *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, volume 2944 of *LNCS*, pages 77–90. Springer, 2003.

15 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.

16 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical foundations of peer-to-peer data integration. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 241–251. ACM, 2004.

17 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo Rosati, and Guido Vetere. Hyper: A framework for peer-to-peer data integration on grids. In *Semantics for Grid Databases, First Int. IFIP Conf. on Semantics of a Networked World (ICSNW). Revised Selected Papers*, volume 3226 of *LNCS*, pages 144–157. Springer, 2004.

18 Etienne Canaud, Salima Benbernou, and Mohand-Said Hacid. Managing trust in active xml. In *Proc. of the Int. Conf. on Services Computing (SCC 2004), 15-18 September 2004, Shanghai, China*, pages 41–48, 2004.

19 Philippe Cudre-Mauroux. Peer data management system. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 2055–2056. Springer US, 2009.

20 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.

21 Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. Queries and updates in the codb peer to peer database system. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, VLDB '04, pages 1277–1280. VLDB Endowment, 2004.

22 Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, volume 2944 of *LNCS*, pages 64–76. Springer, 2003.

23 Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. The coDB robust Peer-to-Peer database system. In *Proc. of the Twelfth Italian Symposium on Advanced Database Systems (SEBD)*, pages 382–393, 2004.

24 Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. A distributed algorithm for robust data sharing and updates in p2p database networks. In *EDBT 2004 Workshops, Revised Selected Papers*, volume 3268 of *LNCS*, pages 446–455. Springer, 2004.

25 Ariel Fuxman, Mauricio A. Hernández, C. T. Howard Ho, Renée J. Miller, Paolo Papotti, and Lucian Popa. Nested mappings: Schema mapping reloaded. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 67–78, 2006.

26 Wolfgang Gatterbauer and Dan Suciu. Data conflict resolution using trust mappings. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 219–230. ACM, 2010.

27 Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 133–142. ACM, 2007.

28 Todd Green, Grigoris Karvounarakis, Zachary Ives, and Val Tannen. Update exchange with mappings and provenance. Technical report, University of Pennsylvania, 2007. Department of Computer and Information Science; Technical Report No. MS-CIS-07-26.

29 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 31–40. ACM, 2007.

30 Todd J. Green, Gregory Karvounarakis, Nicholas E. Taylor, Olivier Biton, Zachary G. Ives, and Val Tannen. ORCHESTRA: facilitating collaborative data sharing. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 1131–1133. ACM, 2007.

31 Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 675–686. ACM, 2007.

32 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.

33 Peter Haase, Jeen Broekstra, Marc Ehrig, Maarten Menken, Peter Mika, Mariusz Olko, Michal Plechawski, Pawel Pyszlak, Björn Schnizler, Ronny Siebes, Steffen Staab, and Christoph Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proc. ISWC*, volume 3298 of *LNCS*, pages 122–136. Springer, 2004.

34 Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

35 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.

36 Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. Piazza: data management infrastructure for semantic web applications. In *Proc. of the Int. World Wide Web Conf. (WWW)*, pages 556–567. ACM, 2003.

37 Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 505–516. IEEE Computer Society, 2003.

38 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.

39 Katja Hose. *Processing Rank-Aware Queries in Schema-Based P2P Systems*. PhD thesis, Technische Universität Ilmenau, 2009.

40 Katja Hose, Christian Lemke, and Kai-Uwe Sattler. Processing relaxed skylines in PDMS using distributed data summaries. In *Proc. of the Conf. on Information and Data Management (CIKM)*, 2006.

41 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.

42 INRIA. Active xml website (www.activexml.net/). `http://www.activexml.net/`, 2009. Accessed 22. April 2011.

43 Zachary G. Ives, Nitin Khandelwal, Aneesh Kapur, and Murat Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *CIDR*, pages 107–118, 2005.

44 Vasiliki Kantere, Iluju Kiringa, John Mylopoulos, Anastasios Kementsietsidis, and Marcelo Arenas. Coordinating peer databases using ECA rules. In *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, volume 2944 of *LNCS*, pages 108–122. Springer, 2003.

45 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 *LNCS*, pages 56–73. Springer, 2003.

46 Anastasios Kementsietsidis. Data sharing and querying for Peer-to-Peer data management systems. In *EDBT 2004 Workshops, Revised Selected Papers*, volume 3268 of *LNCS*, pages 177–186. Springer, 2004.

47 Anastasios Kementsietsidis and Marcelo Arenas. Data sharing through query translation in autonomous sources. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 468–479. Morgan Kaufmann, 2004.

48 Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Managing data mappings in the hyperion project. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 732–734. IEEE Computer Society, 2003.

49 Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 325–336. ACM, 2003.

50 Giorgos Kokkinidis and Vassilis Christophides. Semantic query routing and processing in p2p database systems: The ics-forth sqpeer middleware. In *EDBT 2004 Workshops, Revised Selected Papers*, volume 3268 of *LNCS*, pages 486–495. Springer, 2004.

51 Georgia Koloniari and Evaggelia Pitoura. Peer-to-peer management of xml data: issues and research challenges. *SIGMOD Record*, 34(2):6–17, 2005.

52 Lucja Kot and Christoph Koch. Cooperative update exchange in the youtopia system. *PVLDB*, 2(1):193–204, 2009.

53 L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. Schemasql: A language for interoperability in relational multidatabase systems. In *22nd Conference on Very Large Databases, Bombay,India, 1996*, pages 239–250. Morgan Kaufman Publishers, 1996.

54 Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

55 Alexander Löser, Wolf Siberski, Martin Wolpers, and Wolfgang Nejdl. Information integration in Schema-Based Peer-To-Peer networks. In *Proc. of the Conf. on Advanced Information Systems Engineering (CAiSE)*, volume 2681 of *LNCS*, pages 258–272. Springer, 2003.

56 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 *LNCS*, pages 292–309. Springer, 2005.

57 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 *Proc. of the Int. World Wide Web Conf. (WWW)*, pages 604–615. ACM, 2002.

58 Wolfgang Nejdl, Boris Wolf, Wolf Siberski, Changtao Qu, Stefan Decker, Michael Sintek, Ambjorn Naeve, Mikael Nilsson, Matthias Palmer, and Tore Risch. Edutella: P2p networking for the semantic web. Technical report, Hannover University: Distributed Systems Institute - Knowledge Based Systems, 2003. Accessed 1. August 2009.

59 Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. PeerDB: A P2P-based system for distributed data sharing. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 633–644. IEEE Computer Society, 2003.

60 University of Toronto Database Group. Hyperion project website (www.cs.toronto.edu/db/hyperion/index.html. `http://dblab.cs.toronto.edu/project/hyperion/`, 2009. Accessed 19. May 2011.

61 Beng Chin Ooi, Yanfeng Shu, and Kian-Lee Tan. Relational data sharing in peer-based data management systems. *SIGMOD Record*, 32(3):59–64, 2003.

62 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 *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, page 659. ACM, 2003.

63 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 *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 1291–1294. ACM, 2005.

64 Armin Roth. Completeness-driven query answering in peer data management systems. In *Proc. of the VLDB 2007 PhD Workshop*, 2007.

65 Armin Roth. *Efficient Query Answering in Peer Data Management Systems.* PhD thesis, Humboldt Universität zu Berlin, 2012.

66 Carlo Sartiani, Paolo Manghi, Giorgio Ghelli, and Giovanni Conforti. Xpeer: A self-organizing xml p2p database system. In *EDBT Workshops, Revised Selected Papers*, LNCS, pages 456–465. Springer, 2004.

67 Luciano Serafini, Fausto Giunchiglia, John Mylopoulos, and Philip A. Bernstein. Local relational model: A logical formalization of database coordination. In *Proc. of the Int. and Interdisciplinary Conf. on Modeling and Using Context*, volume 2680 of *LNCS*, pages 286–299. Springer, 2003.

68 Steffen Staab and Heiner Stuckenschmidt, editors. *Semantic Web and Peer-to-Peer – Decentralized Management and Exchange of Knowledge and Information*. Springer, 2006.

69 Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*. Number 3485 in LNCS. Springer, 2005.

70 Igor Tatarinov and Alon Y. Halevy. Efficient query reformulation in peer-data management systems. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 539–550. ACM, 2004.

71 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.

72 Nicholas E. Taylor and Zachary G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 13–24. ACM, 2006.

73 Quang Hieu Vu, Mihai Lupu, and Beng Chin Ooi. *Peer-to-Peer Computing - Principles and Applications*. Springer, 2010.

74 Dan Zhao, John Mylopoulos, Iluju Kiringa, and Verena Kantere. An ECA rule rewriting mechanism for peer data management systems. In *Proc. of the Int. Conf. on Extending Database Technology (EDBT)*, volume 3896 of *LNCS*, pages 1069–1078. Springer, 2006.