Chapter 11
Federated Data Integration in Data Spaces

Matthias Jarke and Christoph Quix

Abstract Data Spaces form a network for sovereign data sharing. In this chapter, we explore the implications that the IDS reference architecture will have on typical scenarios of federated data integration and question answering processes. After a classification of data integration scenarios and their special requirements, we first present a workflow-based solution for integrated data materialization that has been used in several IDS use cases. We then discuss some limitations of such approaches and propose an additional approach based on logic formalisms and machine learning methods that promise to reduce data traffic, security, and privacy risks while helping users to select more meaningful data sources.

11.1 Introduction

Data of all kinds and media formats constitute evidence about the state and events in the world. Analogous to legal procedures, these evidences must be checked for quality and synthesized in a semantically meaningful and purpose-oriented manner in order to assist human-machine learning and evidence-driven decision making. This is the task of data integration.

Transactional databases (ERP systems), huge social networks, and most recently billions of sensors in the Internet of Things produce an ever-growing volume and variety of such evidences in ever-growing velocity but sometimes doubtful veracity. It is therefore not surprising that, despite almost 40 years of research, data integration

M. Jarke (✉)
Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, Germany

Informatik 5 (Information Systems), RWTH Aachen University, Aachen, Germany
e-mail: jarke@dbis.rwth-aachen.de

C. Quix
Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, Germany

Hochschule Niederrhein, University of Applied Sciences, Krefeld, Germany
e-mail: christoph.quix@fit.fraunhofer.de

B. Otto et al. (eds.), *Designing Data Spaces*,
https://doi.org/10.1007/978-3-030-93975-5_11

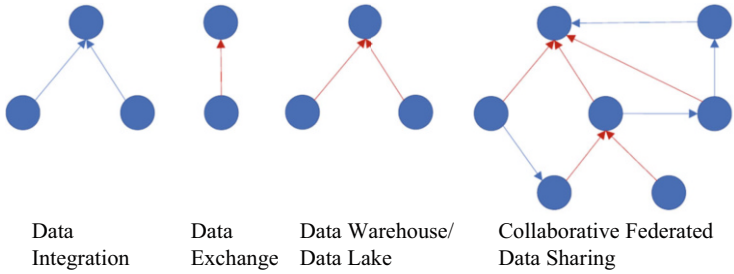

Fig. 11.1 Data integration scenarios (adapted from [1])

remains a key challenge in data science which still requires about 80% of all data analytics work.

Maurizio Lenzerini, arguably the internationally best-known data integration researcher, has recently proposed a useful classification of data integration scenarios that have shaped requirements in the field (cf. Fig. 11.1).

In the figure, the blue nodes represent datasets. They can play the role of data sources or integration results in the data integration tasks indicated by the arrows; arrows are formally expressed as query specifications, also called *view definitions*. The blue arrows indicate *virtual data integration* by query specifications that selectively produce integrated data only when actually activated by a user or application program. In contrast, red arrows indicate *materialized data integration*, i.e., the complete set of specific query results from the view definition is pre-computed and stored as a physical dataset and needs to be maintained when source data change. Briefly, materialized data integration moves the data from the source to the query site, whereas virtual data integration moves the query algorithm partially to the sources. Thus, materialization increases query performance by pre-computation of views, whereas the specific user query in virtual data integration can reduce data transfer, thus reducing communication network load and increasing data sovereignty.

Applying these observations to the four scenarios in Fig. 11.1, early *Data Integration* research laid the formal foundations for query answering among the growing number of distributed transactional, usually relational, databases.

In the 1990s, the novel concept of decision support by data analytics required a separation of data warehouses (DW) [2] from transactional databases: firstly, the need to keep historical data over long period of times for, e.g., time series analyses; secondly, problems in transaction management concurrency control caused by interference of short update transactions and analytics queries spanning large datasets; and thirdly, the need for more user-friendly multidimensional online-analytical processing (OLAP) query facilities in so-called data marts. In practice, the view materialization in DW is usually supported by a so-called *ETL workflow* which coordinates various commercial or open-source tools for *Extraction* from data sources, *Transformations* for cleaning the extracted data and integrating data from

different sources into a uniform DW data format, and finally *Loading* the transformed into the DW storage.

While the DW approach resulted in high-quality integrated data, the ETL effort turned out to be a very high upfront investment, often well before or even without a return on investment which came only with successful analytic queries.

Reacting to the growing need for real-time decision support based on high-frequency stream data inputs – first in finance, later on in web shops, social media, and recently technical and science applications – this upfront effort appeared to be no longer doable. In addition, novel algorithms such as MapReduce enabled highly parallel initial loading and structuring of incoming data and analytics queries on the sources, and the concurrency control issue was partially resolved by novel main memory techniques such as the column store of SAP/HANA. As a consequence, data lakes introduced since the mid-2010s changed the workflow structure from ETL to an ELT format: after Extraction, the data is loaded to the data lake in its original form; Transformations are applied only to the data in the lake if required for a specific application, to avoid the complexity of ETL process design for big data. Service-oriented architectures enable not only data integration but also the integration of complex workflows in which several application systems and services such as Hadoop, Apache Spark, or KAFKA are involved.

On the data lake usage side, data scientists explore various datasets to find rules or functions that confirm and formalize correlations. In these settings, each scenario requires a different set of data sources to be integrated, with different requirements for granularity and data quality. Thus, materialized data integration as applied in data warehouses is less applicable for big data use cases; instead, rich structural and semantic metadata are typically created on top of the raw data, to permit multiple topical perspectives on the data lake. Further details on data lake functions and metadata are elaborated in a recent survey paper [3].

However, data management does not only include data sources within an organization; increasingly, external data sources have to be considered. These external sources come in different forms, e.g., datasets provided by some marketplace. Therefore, data exchange across organizations and usage of data provided by external partners, as studied by the International Data Space Association, is becoming highly relevant [4]. As we shall discuss more deeply in the remainder of this chapter, such workflow solutions also dominate practice in the tasks that are typical for the *Data Space* scenarios of individual *Data Exchange* and trusted *Collaborative Data Sharing* in federations of partners within a Data Space. They do not just combine virtual and materialized data integration and querying, but also include additional Data Space management tasks such as data source discovery, negotiated access, and usage control components in data integration workflow.

In the following section, we use a mobility data space scenario to illustrate a solution which supports a view materialization workflow in the International Data Space setting. Such a setting has been implemented in similar form in several IDS use cases described in later chapters of this book.

Less attention has been paid so far to transferring the virtual, query-driven data integration strategy to the data exchange and federated data sharing of Data Spaces.

In Sect. 11.3, we review such technologies in a more formal, logic-based framework and present a perspective how logic-based query optimization could further improve federated data integration and exchange in Data Spaces.

11.2 Federated Data Integration Workflows in Data Spaces

In this section, we first sketch the challenges of federated data integration in a case study setting of mobility engineering, and then use this case study to present extensions to data integration workflows for a data space support beyond similar efforts in data warehouses and data lakes.

11.2.1 A Simple Demonstrator Scenario

Although data exchange scenarios have also been considered in data integration research [5], constraints in using external data have not been considered. For example, there might be usage constraints on the data, i.e., data may not be stored in local repositories and can be processed only once. Another requirement might be to protect the privacy of data providers, i.e., data can only be accessed to perform aggregate computations which do not reveal detailed information. To illustrate these requirements, Fig. 11.2 shows a mobility scenario extracted from a large collaborative project with the automotive and communication industries on Car2X communication [7].

In the scenario, cars equipped with Car2X communication devices [8] expose detailed vehicle data to a data aggregation service (e.g., a specific platform of the car manufacturer such as BMW ConnectedDrive). This data is made available in a mobility data marketplace (e.g., Mobilitätsdatenmarktplatz (MDM) in Germany, https://www.mdm-portal.de/). Furthermore, additional external data sources could be made available in this marketplace, such as weather data, information about construction sites, or information about local events. This information could be used by service providers that offer information about the current traffic state, compute fastest routes, or give real-time warnings about road hazards.

However, these service providers should not have full access to detailed information of data providers, e.g., their locations, speed, etc. For example, detailed location data might reveal sensitive information as it has been shown with taxi data from New York City, even though the data has been anonymized.[1] To address this challenge, a data exchange platform should enable data providers to specify usage policies for their data. For example, a usage policy could state that data may not be

[1] https://www.fastcompany.com/3036573/nyc-taxi-data-blunder-reveals-which-celebs-dont-tip-and-who-frequents-strip-clubs

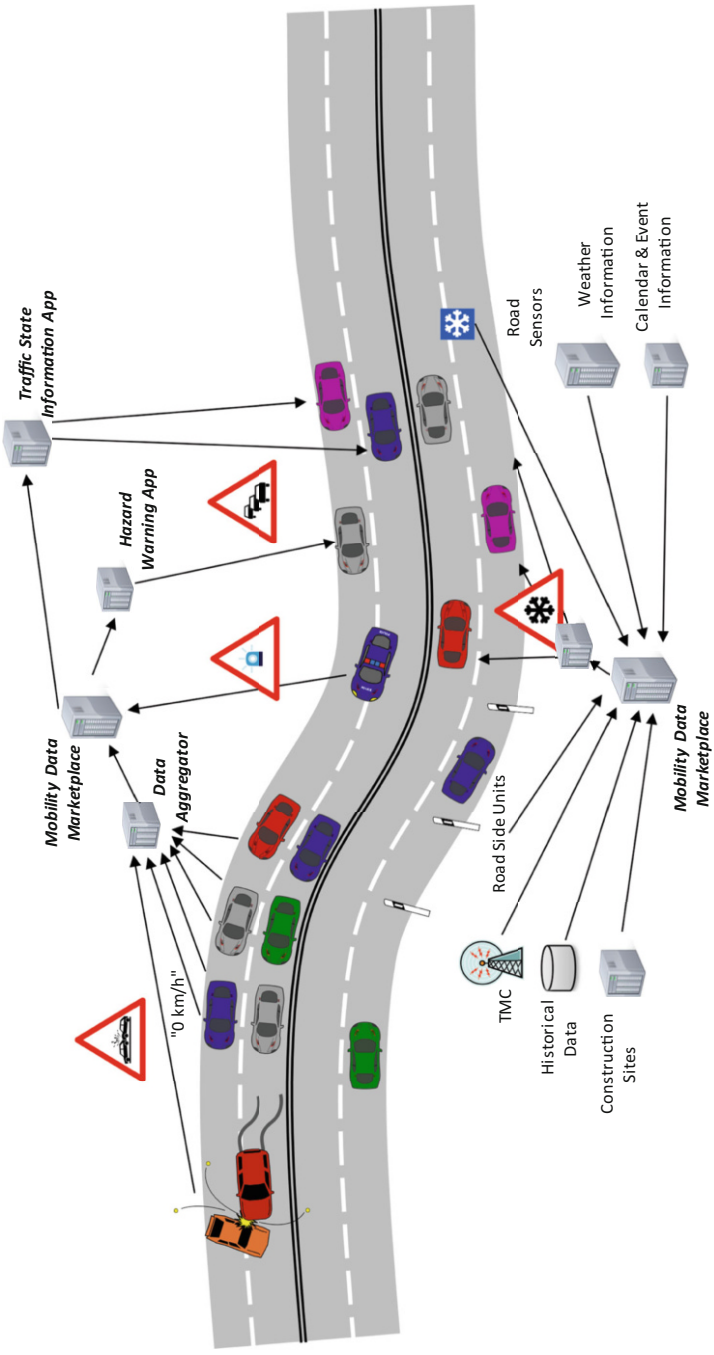

Fig. 11.2 Data exchange in a mobility scenario (adapted from [6])

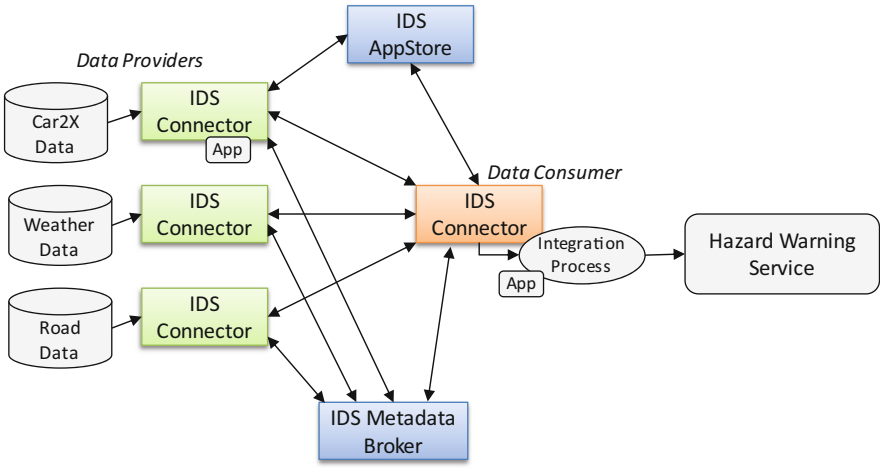

Fig. 11.3 IDS integration architecture for traffic scenario

stored and can be processed only once as in data stream management systems [9], or data can only be processed if it is anonymized and aggregated with other data items.

11.2.2 A Data Integration Workflow Solution for Data Spaces

Current IDS implementations follow a pragmatic, workflow-oriented approach that has been applied also successfully in data warehousing and many other integration approaches. In the IDS, data integration is done within a connector by using integration workflows that are implemented as routes in the Apache Camel framework. Figure 11.3 illustrates an integration architecture for the traffic scenario from Fig. 11.2.

In the architecture, three connectors are used to provide data about vehicles, their location, speed, etc. weather data, and data about roads. These connectors could be run by three different data providers that aim at selling their data in the IDS. To advertise the data and to make it available in the IDS, they need to publish descriptions of their datasets, including technical information of their connectors (e.g., URLs and protocols to query the data), at the IDS Metadata Broker. The IDS Metadata Broker is a central component in the IDS architecture [10] that manages metadata according to the IDS Information Model [11]. The IDS Information Model uses the Web Ontology Language (OWL) to define the basic vocabulary in which data resources in the IDS can be described.

Another central component of the IDS architecture is the App Store. It can provide "Data Apps," i.e., applications that can be deployed inside a connector to perform certain data operations. For example, a data app could transform data items between different standard formats (e.g., XML to JSON), join datasets from different

sources, and perform a complex aggregation or machine learning task. Data apps can be part of the integration processes that are defined within a connector.

Within such a process, a data app might have an input, an output, or both. In case the app has only input or output, it is considered to be a system adapter, i.e., it has an interface to a backend system in which data is written or from which data is read. However, this backend system is not part of the IDS infrastructure; in the viewpoint of the IDS connector, the data app might have only input or output.

As stated, data apps might perform any kind of data operation. In the case of the connector for Car2X data, the data app is responsible for aggregating, anonymizing, and noising the data in order to guarantee the privacy of the original data providers, e.g., car drivers.

To integrate data from different connectors, as required in this scenario, the connector of the data consumer has to employ an integration process. An integration process could be implemented in many different ways, e.g., manually implemented in any programming language or using one of the many (open-source) data integration frameworks. In the IDS, some connector implementations have decided to use Apache Camel. Apache Camel is an integration framework that allows to define data integration and transformation as *routes*. A route is a sequence of operations that are applied to datasets. Several routes can be combined by using a join-like operation. A benefit of Apache Camel is its broad support for different source systems and operations. It can also be extended with customized operations. Apache Camel uses a domain-specific language for the specification of routes that can be defined in various languages (e.g., XML, Java).

In the scenario of Fig. 11.3, we can use a Camel route to integrate the data from the three data providers. A data app could implement more intelligent behavior than just joining and aggregating data, e.g., analyze the data stream with current traffic information and apply a machine learning model to predict traffic states in the near future or to detect road hazard (e.g., icy road, queue end) [7].

The additional functionality to guarantee data sovereignty can be easily integrated into the Camel routes by using the extensibility features of Camel. When data is exchanged between two IDS connectors, the route needs to be enriched with an interceptor pattern. The interceptor will check the usage policies which are defined for the current dataset, and enforce the corresponding usage restrictions. Interceptors can be defined in different ways and depend on the language that is used to define usage policies. MY DATA[2] and LUCON [12] are two examples for usage control systems which implement the interceptor pattern.

[2] https://www.dataspaces.fraunhofer.de/de/software/usage-control/mydata.html

11.3 Toward Formalisms for Virtual Data Space Integration

The materialization-oriented workflow satisfies the requirements but does have a few limitations. If the user asks a specific query, the integrated answer will usually be much smaller than the sum of the data needed to produce it. This will result in many more data to be transported to the integration site than necessary, creating network overload and unnecessary access risks and related IDS controls for data that perhaps would not even need to be exported by the data provider if the query could be adequately optimized.

However, there is an additional issue that we did not mention before. In the Introduction, we interpreted data as evidence about reality; this would argue for an understanding that data sources are not a "ground truth" but themselves limited-quality views on a reality that could be structured according to different integrative worldview (or ontology) by the provider and consumer organizations. In database jargon, this perspective is called "local-as-view" (LAV). In contrast, most materialization-based workflows like the one shown in Sect. 11.2 do consider the data sources as ground truth and the global integrated data as an integrative view on these sources ("global as view – GAV"). The same holds where data are only accessible through parameterized reusable software services [13] or apps. Thus, it is impossible to make statements, e.g., about the completeness of data sources with respect to a broader ontology and about the credibility of conflicting evidences among different sources.

Modern logic-based theories of data integration therefore adopt a combined GLAV ("global-local-as-view") perspective, cf. the combination of red and blue arrows in the federated collaborative data sharing scenario of Fig. 11.1.

In the remainder of this section, we demonstrate the necessary logic formalisms, reusing the example of Fig. 11.2, and discuss how the recent approaches for data integration in data lakes could be extended to the Data Space setting.

11.3.1 Logical Foundations for Data Integration

To illustrate the concepts of data integration in this section, Fig. 11.4 extracts a simplified heterogeneous schema integration setting from the scenario in Fig. 11.2. An XML web service *S* provides information about vehicles and their location. A relational data source *R* delivers information about roads, their locations, and other geographical data. A service provider needs integration of these two sources into a web service T, providing current traffic state information in a JSON format which is frequently used in web or mobile applications.

Research in data integration has developed several key technologies and fundamental concepts to address such data integration problems.

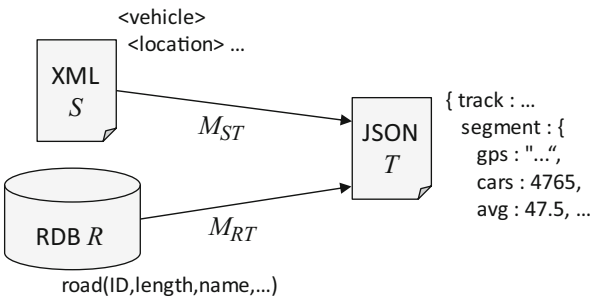

Fig. 11.4 Simplified heterogeneous integration scenario for traffic data

Model and Mapping Management Early approaches to data integration mainly focused on efficient data transformation and scheduling the integration and cleaning jobs in such a way that the operational systems were not affected [14, 15]. Management of schemas or formal specifications of the transformation procedures was a side activity in optimizing the ETL workflows. Only the vision of model management was to consider schemas, mappings, and other metadata of information systems as "first-class citizens" of data integration systems [16]. The goal was to develop an algebra with basic operations to match, integrate, and compose models and mappings. Although the vision of an algebra has not been achieved completely, the idea of model management initiated research on these fundamental operations for data integration [17], such as schema matching and mappings, as discussed in the following paragraphs.

Heterogeneity An immanent problem of data integration is heterogeneity. An obvious challenge in data integration is the wide variety of data formats that are used in data sources. Once the data formats have been identified, appropriate transformation functions can be applied to deal with this problem. A more challenging problem is heterogeneity at the model level and at the semantic level. In the example of Fig. 11.4, three different modeling languages are used. A modeling language is a formalism that is used to describe the data structures, e.g., XML Schema, "CREATE TABLE" statements in SQL, or JSON Schema. In order to map and link the models in these different languages, one has to translate these models into a common formalism that uses a generic representation [18]. More sophisticated is the problem of semantic heterogeneity. In the example, different terminology is used (e.g., "track" vs. "road", "vehicle" vs. "car"). In order to link data sources, a data integration engineer has to understand the different entities that are encoded in the data sources. While advanced algorithms for schema matching have been developed in the last two decades (see paragraph on matching below), a fully automatic approach that can resolve all heterogeneities is most likely not possible. Human intervention will be always required to verify and to complete schema alignments.

Schema Matching Schema matching is the task of identifying correspondences in two schemas. This field has been very active in the last two decades and can be subdivided into the following categories [19, 20]:

- *Algorithms:* Various algorithms detect similarities in schemas, based on labels, structure, or auxiliary information. Usually, not a single approach is sufficient; thus, a combination of several algorithms is important, for which different strategies for combining individual results can be developed [21].
- *Schema representation:* for schema matching, it is often sufficient to represent schemas as directed graphs. Although this representation loses some details about schema constraints, algorithms for graph alignment and matching can be applied in this case. Other approaches transform the schemas into a generic representation with detailed representations for constraints and datatypes, which can be leveraged to find correspondences of schema elements [22].
- *Semantic matching:* especially in the field of ontology alignment [20], additional sources with semantic information are exploited. This includes, for example, dictionaries and thesauri to identify terms with similar meaning, or repositories with existing alignments. In addition, reasoning about the relationships and rules expressed in the ontologies can be used to detect related elements. More information on the IDS metalevel ontologies and linked metadata structures is presented in a later chapter of this book.
- *Matching systems:* As stated above, algorithms have to be combined in order to achieve a satisfying matching result. Schema matching systems have focused on the integration of different algorithms, such that the output of one algorithm can be used as input for another algorithm to compute an extended alignment [23, 24]. Another important point in the development of matching systems is performance. A complexity of $O(n^2)$ seems to be unavoidable as all elements of two given schemas should be compared, but this would not scale to large schemas. Thus, schema matching systems have used different optimization techniques to reduce complexity and to compute alignments efficiently, e.g., by exploiting hashing or parallel computations.

11.3.2 Data Integration Tool Extensions for Data Spaces

Some of the technologies have been integrated into commercial data integration products, but due to the complexity of matching algorithms, these technologies are limited to simple label similarity (or even equality) and structural comparisons.

However, there exist algorithms and large-scale prototypes in scientific and industrial research that are promising for extension to the Data Space setting:

Schema Integration Given the information about similarities of two schemas, a new schema should be derived that integrates the information of both schemas [25]. In the example of Fig. 11.4, the schema of the JSON document *T* that integrates *R* and *S* needs to be defined. In this scenario, application requirements for the traffic state information will mainly determine the outline of the *T*; however, in more information-centric applications, e.g., data warehouse systems, the integrated

schema will be defined as the "union" of the source schemas, taking into account computed alignments. While intuitively the problem of schema integration is clear, the difficulty is to formalize the schema integration process: which requirements should be satisfied by the integrated schema, how can we verify that the integrated schema is correct, and what is the meaning of "correctness" and "completeness" in schema integration? One approach could be to follow a query-oriented approach: queries that can be answered on the sources should deliver the same result on the integrated schema [26].

Mappings The alignments that have been computed by schema matching algorithms only state that there is a similarity between two schema elements. However, to create integrated datasets, one needs to transform the data instances from the sources into the desired target structure. In the example from Fig. 11.4, we need to have an XQuery and an SQL query that extract the required information from the XML document and the relational database. Furthermore, we need to have a function that creates the required JSON document. All these queries and transformations need to be transformed within the same data transformation tool, such that the data can be integrated and aggregated into one dataset. In the example, we need to aggregate the number of cars per road and compute their average speed, which requires to combine the data from both sources. Mapping frameworks for heterogeneous data integration scenarios as in this example have been proposed in research [27, 28] several open-source projects have adopted these approaches and provide similar functionality, e.g., Apache Spark and Apache Drill.

Related Dataset Discovery In today's huge distributed data lake or Data Space environment with hundreds or thousands of possible data sources, consumers often face the problem that they do not even know which data source (and data provider) contains useful relevant information for their intended query. In the example of Fig. 11.3, this has simply been delegated to a separate IDS Agent called the Information Broker, but in reality, where no fully integrated schemas exist, also this "Related Dataset Discovery" problem needs to be addressed in a query-specific manner to maintain relevance of the answer and avoid unnecessary network traffic. A promising recent approach pursued, e.g., in the Aurum [29], JOSIE [30], and D3L [31] projects, is that the user specifies the format of their desired answer table together with some relationship aspects that are important to them (value overlaps, numerical data distributions, textual attribute semantics, etc.), and fast similarity algorithms propose possible extensions of the query table in the sense of "what else is relevant beyond what you have explicitly asked for"; the algorithm then adds the corresponding data sources for those extensions considered relevant by the user to the query rewriting process.

Query Rewriting The benefit of the aforementioned open-source projects is especially the ability of query rewriting (or query translation): queries are defined in a generic query language (e.g., Spark SQL in the case of Apache Spark) and translated into the query language of the data sources. Without such mappings, even the simplest, but often extremely efficient, data optimization heuristic such as "execute

selection before join" cannot be propagated backwards from the query to the data sources. However, the query translation is usually done only with simple one-to-one mappings between an integrated schema and the source schemas, which usually implies that the source data has first to be transformed into the required structure.

More complex query rewritings, which involve join and union operations or distributed data and heterogeneous polystore source databases, are still subject to research [32–34]. An additional demand in query rewriting for sovereign data exchange in Data Spaces is the inclusion of privacy, access, and usage control rules into the query evaluation. Fortunately, we already know since Mike Stonebraker's 1975 Ph.D. thesis on how to do this.

In summary, even though operational solutions for federated data integration as the basis for sovereign data exchange and collaborative data sharing in Data Spaces exist, there remains ample room for further research and promising industrial uptake in the field.

Acknowledgments This research was supported by the German BMBF (IndaSpacePlus, Fkz 01IS17031), by DFG (national excellence cluster EXC-2023 "Internet of Production"—390621612), and by the Fraunhofer cluster of excellence "Cognitive Internet Technologies".

References

1. Lenzerini, M. (2019). Direct and reverse rewriting in data interoperability. In *Proceedings CAiSE* (pp. 3–13). Springer.
2. Jarke, M., Lenzerini, M., Vassiliou, Y., & Vassiliadis, P. (2003). *Foundations of data warehouses* (2nd ed.). Springer.
3. Hai, R., Quix, C., & Jarke, M. (2021). *Data lake concept and systems: A survey*. CoRR abs/2106.09592.
4. Otto, B., & Jarke, M. (2019). Designing a multi-sided data platform: Findings from the international data spaces case. *Electronic Markets, 29*(4), 561–580.
5. Fagin, R., Kolaitis, P., Miller, R. J., & Popa, L. (2005). Data exchange: Semantics and query answering. *Theoretical Computer Science, 336*, 89–124.
6. Jarke, M., Jeusfeld, M., & Quix, C. (2014). Data-centric intelligent information integration - from concepts to automation. *Journal of Intelligent Information Systems, 43*(3), 437–462.
7. Geisler, S., Quix, C., Schiffer, S., & Jarke, M. (2012). An evaluation framework for traffic information systems based on data streams. *Transportation Research Part C, 23*, 29–55.
8. Fuchs, H., Hofmann, F., Löhr, H., & Schaaf, G. (2015). Car-2-X. In H. Winner, S. Hakuli, F. Lotz, & C. Singer (Eds.), *Handbuch Fahrerassistenzsysteme* (p. 28). Springer Vieweg. https://doi.org/10.1007/978-3-658-05734-3
9. Geisler, S. (2013). Data stream management systems. In P. G. Kolaitis, M. Lenzerini, & N. Schweikardt (Eds.), *Data exchange, integration, and streams* (Vol. 5, pp. 275–304). Dagstuhl follow-Ups. https://doi.org/10.4230/DFU.Vol5.10452.275
10. Otto, B., & et al. (2019). *Reference architecture model*. IDSA. https://www.internationaldataspaces.org/ressource-hub/publications-ids/, version 3.0.
11. Bader, S. R., Pullmann, J., Mader, C., Tramp, S., Quix, C., Müller, A. W., Akyürek, H., Böckmann, M., Imbusch, B. T., Lipp, J., Geisler, S., & Lange, C. (2020). The international data spaces information model - an ontology for sovereign exchange of digital content. In

Proceedings of International Semantic Web Conference (ISWC), Athens, Greece, 176–192 (part II), LNCS 12507. Springer.
12. Schütte, J., & Brost, G. S. (2018). LUCON: Data flow control for message-based IoT systems. In *17th IEEE International Conference on trust, security and privacy in computing and communications/12th IEEE International Conference on big data science and engineering* (pp. 289–299). https://doi.org/10.1109/TrustCom/BigDataSE.2018.00052.
13. Chen, P. S., Hennicker, R., & Jarke, M. (1993). On the retrieval of reusable software components. In *Proceedings of 2nd International Workshop Software Reusability, Lucca Italy* (pp. 99–108). IEEE.
14. Simitsis, A., & Vassiliadis, P. (2018). Extraction, transformation, and loading. In *Encyclopedia of database systems* (2nd ed.). Springer.
15. Inmon, W. H. (1996). *Building the data warehouse* (2nd ed.). John Wiley & Sons.
16. Bernstein, P. A., Halevy, A. Y., & Pottinger, R. (2000). A vision of management of complex models. *SIGMOD Record, 29*(4), 55–63.
17. Bernstein, P. A., & Melnik, S. (2007). Model management 2.0: Manipulating richer mappings. In L. Zhou, T. W. Ling, & B. C. Ooi (Eds.), *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 1–12). ACM Press.
18. Kensche, D., Quix, C., Chatti, M. A., & Jarke, M. (2007). GeRoMe: A generic role based metamodel for model management. *Journal on Data Semantics, 8*, 82–117., Springer, LNCS 4380.
19. Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *VLDB Journal, 10*(4), 334–350.
20. Shvaiko, P., & Euzenat, J. (2013). Ontology matching: State of the art and future challenges. *IEEE Trans. Knowledge and Data Engineering, 25*(1), 158–176.
21. Marie, A., & Gal, A. (2008). Boosting schema matchers. In R. Meersman & Z. Tari (Eds.), *On the move to meaningful internet systems, (OTM 2008). LNCS 5331*. Springer. https://doi.org/10.1007/978-3-540-88871-0_20
22. Quix, C., Kensche, D., & Li, X. (2007). Matching of Ontologies with XML Schema using a Generic Metamodel. In *Proceedings of the 6th International Conference Ontologies, DataBases, and Applications of Semantics (ODBASE), Vilamoura, Portugal* (pp. 1081–1098). Springer.
23. Aumüller, D., Do, H. H., Massmann, S., & Rahm, E. (2005). Schema and ontology matching with COMA++. In *Proceedings SIGMOD Conference* (pp. 906–908). ACM Press.
24. Faria, D., Pesquita, C., Santos, E., Palmonari, M., Cruz, I. F., & Couto, F. M. (2013). The AgreementMakerLight ontology matching system. In R. Meersman et al. (Eds.), *On the move to meaningful internet systems (OTM 2013). LNCS* (Vol. 8185). Springer. https://doi.org/10.1007/978-3-642-41030-7_38
25. Batini, C., Lenzerini, M., & Navathe, S. B. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys, 18*(4), 323–364.
26. Li, X., Quix, C. (2011). Merging relational views: A minimization approach. In *Proceedings 30th International Conference Conceptual Modeling (ER 2011),* Brussels, Belgium.
27. Haas, L. M., Hernández, M. A., Ho, H., Popa, L., & Roth, M. (2005). Clio grows up: From research prototype to industrial tool. In F. Özcan (Ed.), *Proceedings ACM SIGMOD Conference* (pp. 805–810). ACM.
28. Kensche, D., Quix, C., Li, X., Li, Y., & Jarke, M. (2009). Generic schema mappings for composition and query answering. *Data & Knowledge Engineering, 68*(7), 599–621.
29. Fernandez, R. C., Abedjan, Z., Koko, F., Yuan, G., Madden, S., & Stonebraker, M. (2018). Aurum: A data discovery system. In *ICDE 2018* (pp. 1001–1012). IEEE.
30. Zhu, E., Deng, D., Nargesian, F., & Miller, R. J. (2019). JOSIE: Overlap set similarity search for finding joinable tables in Data Lakes. In *Proceedings ACM-SIGMOD* (pp. 847–864). ACM.

31. Bogatu, A., Fernandes, A. A. A., Paton, N. W., & Konstantinou, N. (2020). Dataset discovery in data lakes. In *ICDE 2020* (pp. 709–720). IEEE.
32. Halevy, A. Y., Rajaraman, A., & Ordille, J. J. (2006). Data integration: The teenage years. *Proceedings VLDB, 5*(2), 9–16.
33. Hai, R., Quix, C., & Zhou, C. (2018). Query rewriting for heterogeneous data lakes. In *ADBIS 2018* (pp. 35–49). Springer.
34. Lenzerini, M. (2002). Data integration: A theoretical perspective. In *Proceedings ACM-PODS* (pp. 233–246). ACM.