# Contents

# Chapter 6

# Scalable End-user Access to Big Data

**Martin Giese,[1] Diego Calvanese,[2] Peter Haase,[3] Ian Horrocks,[4] Yannis Ioannidis,[5] Herald Kllapi,[5] Manolis Koubarakis,[5] Maurizio Lenzerini,[6] Ralf Möller,[7] Özgür Özçep,[7] Mariano Rodriguez Muro,[2] Riccardo Rosati,[6] Rudolf Schlatte,[1] Michael Schmidt,[3] Ahmet Soylu,[1] Arild Waaler[1]**

This chapter proposes steps towards the solution to the *data access problem* that end-users typically face when dealing with Big Data:

- They need to pose ad hoc queries to a collection of data sources, possibly including streaming sources;
- They are unable to query those sources on their own, but are dependent on assistance from IT experts;
- The turnaround time for information requests is in the range of days, possibly weeks, due to the involvement of the IT personnel;

[1] University of Oslo
[2] Free University of Bozen-Bolzano
[3] fluid Operations AG
[4] Oxford University
[5] National and Kapodistrian University of Athens
[6] Sapienza University of Rome
[7] TU Hamburg-Harburg

The work on this chapter was partially funded by the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, “Optique.”

- The volume, complexity, variety and velocity of the underlying data sources put very high demands on the scalability of the solution.

We propose to approach this problem using *ontology-based data access* (OBDA), the idea being to capture end-user conceptualizations in an ontology and use declarative mappings to connect the ontology to the underlying data sources. End user queries posed are posed in terms of concepts of the ontology and are then rewritten to queries against the sources.

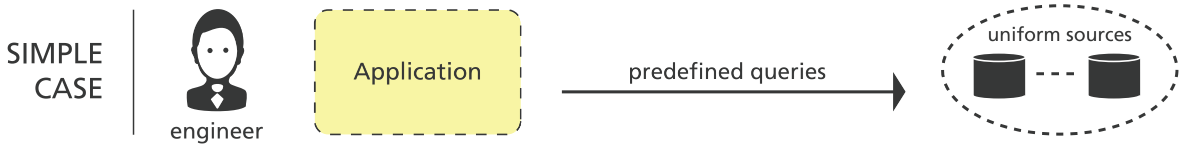
The chapter is structured in the following way. First, in Section 6.1, we situate the problem within the more general discussion about Big Data. Then, in Section 6.2, we review the state of the art in OBDA, explain why we believe OBDA is the superiour approach to the data access challenge posed by Big Data, and explain why the field of OBDA is currently not yet sufficiently mature to deal satisfactory with these problems. The rest of the chapter contains concepts for lifting OBDA to a level where it can be successfully deployed to Big Data.

The ideas proposed in this chapter are investigated and implemented in the FP7 Integrating Project *Optique – Scalable End-user Access to Big Data*, which runs until the end of year 2016. The *Optique* solutions are evaluated on two comprehensive use cases from the energy sector with a variety of data access challenges related to Big Data.[8]

## 6.1 The Data Access Problem of Big Data

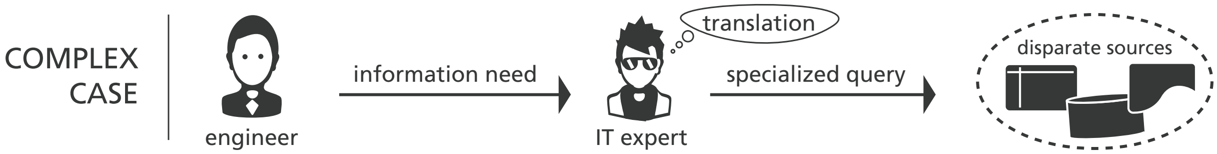
The situation in knowledge- and data-intensive enterprises is typically as follows. Massive amounts of data, accumulated in real time and over decades, are spread over a wide variety of formats and sources. End users operate on these collections of data using specialized applications, the operation of which requires expert skills and domain knowledge. Relevant data is extracted from the data sources using predefined queries that are built into the applications. Moreover, these queries typically access just some specific sources with identical structure. The situation can be illustrated like this:

[8] See `http://www.optique-project.eu/`.



In these situations the turnaround time, by which we mean the time from when the end-user delivers an information need until the data are there, will typically be in the range of minutes, maybe even seconds, and Big Data technologies can be deployed to dramatically reduce the execution time for queries.

Situations where users need to explore the data using ad hoc queries are considerably more challenging, since accessing relevant parts of the data typically requires in depth knowledge of the domain *and* of the organisation of data repositories. It is very rare that the end-users possess such skills themselves. The situation is rather that the end-user needs to collaborate with an IT skilled person in order to jointly develop the query that solves the problem at hand, illustrated in the figure below:

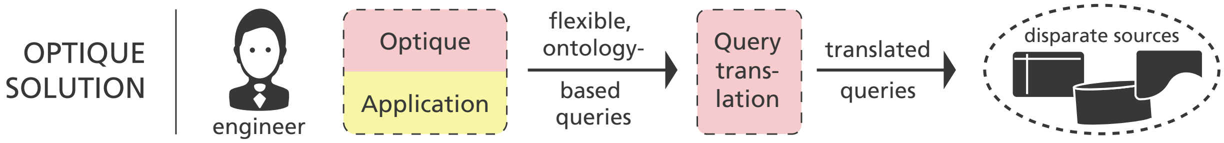


The turnaround time is then mostly dependent on human factors and is in the range of days, if not worse. Note that the typical Big Data technologies are of limited help in this case, as they do not in themselves eliminate the need for the IT expert.

The problem of end-user data access is ultimately about being able to put the enterprise data in the hands of the expert end-users. Important aspects of the problem are volume, variety, velocity, and complexity (Beyer et al., 2011), where by volume we mean the sheer size of the data, by variety we mean the number of different data types and data sources, by velocity we mean the rate at which data streams in and how fast it needs to be processed, and by complexity we mean factors such as standards, domain rules and size of database schemas that in normal circumstances are manageable, but quickly complicate data access considerably when they escalate.

Factors such as variety, velocity and complexity can make data access challenging even with fairly small amounts of data. When, in addition to these factors, data volumes are extreme, the problem becomes seemingly intractable; one must then not only deal with large data sets, but at the same time one also has to cope with dimensions that to some extent are complementary. In Big Data scenarios, one or more of these dimensions go to the extreme, at the same time interacting with other dimensions.

Based on the ideas presented in this chapter, the *Optique* project implements a solution to the data access problem for Big Data in which all the above mentioned dimensions of the problem are addressed. The goal is to enable expert end-users access the data themselves, without the help of the IT experts, as illustrated in this figure:



## 6.2 Ontology-Based Data Access

We have seen that the bottleneck in end-user access to Big Data is to be found in the process of *translating* end-users' information needs into executable, optimized queries over the data sources. An approach known as "Ontology-Based Data Access" (OBDA) has the potential to avoid this bottleneck by automating this query translation process. Fig. 6.1 shows the essential components in an OBDA setup.

The main idea is to use an *ontology*, or domain model, that is a formalisation of the vocabulary employed by the end-users to talk about the problem domain. This ontology is constructed entirely independently of how the data is actually stored. End users can

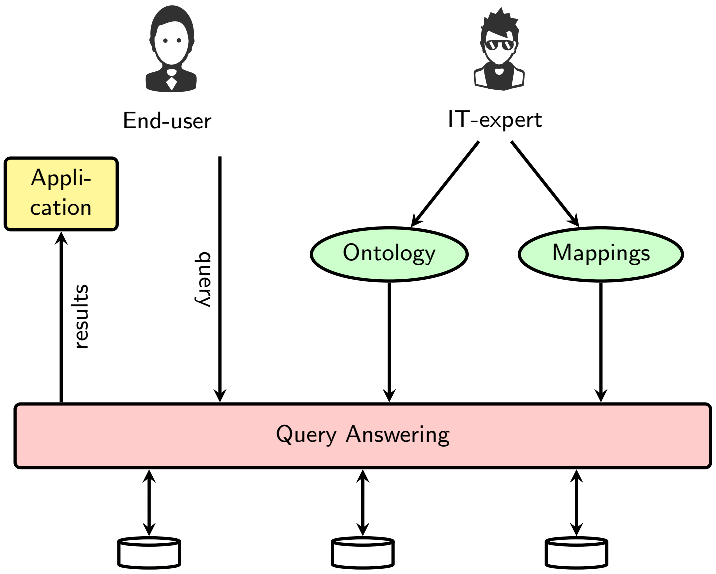


Figure 6.1: The basic set-up for Ontology-Based Data Access

formulate queries using the terms defined by the ontology, using some formal query language. In other words, queries are formulated according to the end-users' view of the problem domain.

To execute such queries, a set of *mappings* is maintained which describe the relationship between the terms in the ontology and their representation(s) in the data sources. This set of mappings is typically produced by the IT-expert, who previously translated end-users' queries manually.

It is now possible to give an algorithm that takes an end-user query, the ontology, and a set of mappings as inputs, and computes a query that can be executed over the data sources, and which produces the set of results expected for the end-user query. As Fig. 6.1 illustrates, the result set can then be fed into some existing domain-specific visualisation or browsing application which presents it to the end-user.

In the next section, we will see an example of such a query translation process, which illustrates the point that including additional information about the problem domain in the ontology can be very useful for end-users. In general, this process of query translation becomes much more complex than just substituting pieces of queries using the mappings.

The generated query can also in some cases become dramatically larger than the original ontology-based query formulated by the end-user.

The theoretical foundations of OBDA have been thoroughly investigated in recent years (Möller et al., 2006; Calvanese et al., 2007b,a; Poggi et al., 2008). There is a very good understanding of the basic mechanisms for query rewriting, and the extent to which expressivity of ontologies can be increased while maintaining the same theoretical complexity as is exhibited by standard relational database systems.

Also, prototypical implementations exist (Acciarri et al., 2005; Calvanese et al., 2011) which have been applied to minor industrial case studies (e.g. Amoroso et al., 2008). They have demonstrated the conceptual viability of the OBDA approach for industrial purposes.

There are several features of a successful OBDA implementation that lead us to believe that it is the right basic approach to the challenges of end-user access to Big Data:

- It is declarative, i.e. there is no need for end-users, nor for IT-experts, to write special purpose program code.
- Data can be left in existing relational databases. In many cases, moving large and complex data sets is impractical, even if the data owners were to allow it. Moreover, for scalability it is essential to exploit existing optimised data structures (tables), and to avoid increasing query complexity by fragmenting data. This is in contrast to, e.g., data warehousing approaches which copy data: OBDA is more flexible and offers an infrastructure which is simpler to set up and maintain.
- It provides a flexible query language that corresponds to the end-user conceptualisation of the data.
- The ontology can be used to hide details and introduce abstractions. This is significant in cases where there is a source schema which is too complex for the end-user.
- The relationship between the ontology concepts and the relational data is made explicit in the mappings. This provides a means for the DB experts to make their knowledge available to the end-user independently of specific queries.

Table 6.1: Example ontology and data for turbine faults

| *Human readable* | *Logic* |
|---|---|
| *Ontology* | |
| Condenser is a CoolingDevice that is part of a Turbine | $\mathsf{Condenser} \sqsubseteq \mathsf{CoolingDevice} \sqcap \exists\mathsf{isPartOf}.\mathsf{Turbine}$ |
| Condenser Fault is a Fault that affects a Condenser | $\mathsf{CondenserFault} \equiv \mathsf{Fault} \sqcap \exists\mathsf{affects}.\mathsf{Condenser}$ |
| Turbine Fault is a Fault that affects part of a Turbine | $\mathsf{TurbineFault} \equiv \mathsf{Fault} \sqcap \exists\mathsf{affects}.(\exists\mathsf{isPartOf}.\mathsf{Turbine})$ |
| *Data* | |
| g1 is a Generator | $\mathsf{Generator}(\mathsf{g1})$ |
| g1 has fault f1 | $\mathsf{hasFault}(\mathsf{g1}, \mathsf{f1})$ |
| f1 is a CondenserFault | $\mathsf{CondenserFault}(\mathsf{f1})$ |

### 6.2.1 Example

We will now present a (highly) simplified example that illustrates some of the benefits of OBDA, and explains how the technique works. Imagine that an engineer working in the power generation industry wants to retrieve data about generators that have a turbine fault. The engineer is able to formalise this information need, possibly with the aid of a suitable tool, as a query of the form:

$$Q1(g) \leftarrow \mathsf{Generator}(g) \wedge \mathsf{hasFault}(g, f) \wedge \mathsf{TurbineFault}(f)$$

which can be read as "return all $g$ such that $g$ is a generator, $g$ has a fault $f$, and $f$ is a turbine fault".

Now consider a database that includes the tuples given in the lower part of Table 6.1. If $Q1$ is evaluated over this data, then $g1$ is not returned in the answer, because $f1$ is a condenser fault, but not a turbine fault. However, this is not what the engineer would want or expect, because the engineer knows that the condenser is a part of the turbine, and that a condenser fault is thus a kind of turbine fault.

The problem is caused by the fact that the query answering system is not able to use the engineer's expert knowledge of the domain. In an OBDA system, (some of) this knowledge is captured in an *ontology*, which can then be exploited in order to answer queries "more intelligently". The ontology provides a conceptual model that is more intuitive for users: it introduces familiar vocabulary terms, and captures declaratively the relationships between terms.

In our running example, the ontology might include the declarative statements shown in the upper part of Table 6.1. These introduce relevant vocabulary, such as condenser, cooling device, affects, etc., and establish relationships between terms. The first axiom, for example, states that "every condenser is a cooling device that is part of a turbine". If we formalise these statements as axioms in a suitable logic, as shown on the right hand side of Table 6.1, we can then use automated reasoning techniques to derive facts that must hold, but are not explicitly given by the data, such as $TurbineFault(g1)$. This in turn means that $g1$ is recognized as a correct answer to the example query. Using an ontology and automated reasoning techniques, query answering can relate to the whole set of implied information, instead of only that which is explicitly stated.

Automated reasoning can, in general, be computationally very expensive. Moreover, most standard reasoning techniques would need to interleave operations on the ontology and the data, which may not be practically feasible if the data is stored in a relational database. OBDA addresses both these issues by answering queries using a two-stage process, first using the ontology to rewrite the query, and then evaluating the rewritten query over the data (without any reference to the ontology). The rewriting step generates additional queries, each of which can produce extra answers that follow from a combination of existing data and statements in the ontology. Ensuring that this is possible for all possible combinations of data and ontology statements requires some restrictions on the kinds of statement that can be included in the ontology. The OWL 2 QL ontology language profile has been designed as a maximal subset of OWL 2 that enjoys this property.

Coming back to our example, we can easily derive from the ontology that a condenser

Table 6.2: Database Tables

**Generator**

| id | serial |
|---|---|
| $g1$ | 1234 |
| $g2$ | 5678 |
| ⋮ | ⋮ |

**Fault**

| id | type |
|---|---|
| $f1$ | C |
| $f2$ | T |
| ⋮ | ⋮ |

**hasFault**

| g-id | f-id |
|---|---|
| $g1$ | $f1$ |
| $g2$ | $f2$ |
| ⋮ | ⋮ |

fault is a kind of turbine fault, and we can use this to rewrite the query as:

$$Q2(g) \leftarrow \mathsf{Generator}(g) \wedge \mathsf{hasFault}(g, f) \wedge \mathsf{CondenserFault}(f)$$

Note that there are many other possible rewritings, including, e.g.:

$$Q3(g) \leftarrow \mathsf{Generator}(g) \wedge \mathsf{hasFault}(g, f) \wedge \mathsf{Fault}(f) \wedge \mathsf{affects}(f, c) \wedge \mathsf{Condenser}(c)$$

all of which need to be considered if we want to guarantee that the answer to the query will be complete for any data set, and this can result in the rewritten query becoming very large (in the worst case, exponential in the size of the input ontology and query).

One final issue that needs to be considered is how these queries will be evaluated if the data is stored in a data store such as a relational database. So far we have assumed that the data is just a set of ground tuples that use the same vocabulary as the ontology. In practice, however, we want to access data in some kind of data store, typically a relational database management system (RDBMS), and typically one whose schema vocabulary does not correspond with the ontology vocabulary. In OBDA we use *mappings* to declaratively capture the relationships between ontology vocabulary and database queries. A mapping typically takes the form of a single ontology vocabulary term (e.g., Generator ) and a query over the data sources that retrieves the instances of this term (e.g., "SELECT id FROM Generator"). Technically, this kind of mapping is known as global as view (GAV).

In our example, the data might be stored in an RDBMS using tables for generators and faults, and using a hasFault table to capture the one to many relationship between generators and faults, as shown in Figure 6.2. Mappings from the ontology vocabulary to RDBMS queries can then be defined as follows:

```
      Generator  ↦  SELECT id FROM Generator
 CondenserFault  ↦  SELECT id FROM Fault WHERE type=‘C’
   TurbineFault  ↦  SELECT id FROM Fault WHERE type=‘T’
       hasFault  ↦  SELECT g-id,f-id FROM hasFault
```

When combined with $Q2$, these mappings produce the following query over the RDBMS:

```
SELECT Generator.id FROM Generator, Fault, hasFault
WHERE Generator.id=g-id AND f-id=Fault.id AND type=‘C’
```

The answer to this query will include $g1$. However, in order to ensure that all valid answers are returned we also need to include the results of $Q1$ (the original query), and all other possible rewritings. In an SQL setting this leads to a UNION query of the form:

```
SELECT Generator.id FROM Generator, Fault, hasFault
WHERE Generator.id=g-id AND f-id=Fault.id AND type=‘T’
UNION
SELECT Generator.id FROM Generator, Fault, hasFault
WHERE Generator.id=g-id AND f-id=Fault.id AND type=‘C’
UNION
...
```

### 6.2.2 Limitations of the State of the Art in OBDA

As mentioned above, OBDA has been successfully applied to first industrial case studies. Still, realistic applications, where non-technical end-users require access to large corporate datastores, lie beyond the reach of current technology in several respects:

(i) The *usability* is hampered by the need to use a formal query language that makes it difficult for end-users to formulate queries, even if the vocabulary is familiar.

(ii) The *prerequisites* of OBDA, namely ontology and mappings, are in practice expensive to obtain.

(iii) The *scope* of existing systems is too narrow: they lack many features that are vital for applications.

(iv) The *efficiency* of both the translation process and the execution of the resulting queries is too low.

In the remainder of this chapter, we discuss possible approaches to overcome these shortcomings, and how the state of the art will have to be advanced in order to realise them. Figure 6.2 shows a proposed architecture supporting this approach. In short terms, the ideas are as follows:

End-user acceptance depends on the **usability** for non-technical users, e.g. by providing a user-friendly *Query Formulation* front-end (see Fig. 6.2) that lets the end-user navigate the vocabulary and presents a menu of possible refinements of a query (see Sect. 6.3). Advanced users must have the possibility to switch back and forth as required between the navigational view and a more technical view where the query can be edited directly. This will make it possible for a non-technical user to author large parts of a query, but receive help from a technical expert when required.

The second problem that needs to be addressed is providing and maintaining the **prerequisites**: ontology and mappings. In practice, these will have to be treated as evolving, dynamic entities which are updated as required for formalising end-users' information requirements. An industrial-scale front-end needs to support both the semi-automatic derivation of an initial ontology and mappings in new deployments, and the extension of the ontology during query formulation, e.g., by adding new technical terms or relationships that were not previously captured. In the architecture of Fig. 6.2, this is accomplished by the *Query Formulation* and *Ontology and Mapping Management* front-end components. This mechanism of bootstrapping and query-driven ontology construction can enable the creation of an ontology that fits the end-users' needs at a moderate cost. The same *Ontology and Mapping Management* component can then also support the IT-expert in maintaining a set of mappings that is consistent with the evolving ontology. The sections on "Query Formulation" (see Sect. 6.3) and "Ontology and Mapping Management" (see Sect. 6.4) expand on the requirements for such a management component.

Providing a robust answer to the **scope** problem is difficult, because there is a trade-off between expressivity and efficiency: very expressive mechanisms in the ontology and mapping languages, that would guarantee applicability to virtually any problem that might occur in industrial applications, are known to preclude efficient query rewriting and execution (Brachman and Levesque, 1984; Artale et al., 2009; Calvanese et al., 2012). To ensure efficiency, a restricted set of features must be carefully chosen for ontology and mapping languages, with the aim of covering as many potential applications as possible.

Still, concrete applications will come with their own specific difficulties that cannot be covered by a general-purpose tool. This expressivity problem needs to be resolved by plugging application-specific modules into the query answering engine. These domain-specific plugins must take care of query translation and optimisation in those cases where a generic declarative mechanism is not powerful enough for an application. A wide range of special purpose vocabulary and reasoning could be covered by such domain-specific modules, such as, to name just a few,

- geological vocabulary in a petroleum industry application
- protein interactions and pathways in molecular biology
- elementary particle interactions for particle physics

On the other hand, important features that occur in many applications need to be built into the core of any OBDA system. Notably, temporal aspects and the possibility of progressively processing data as it is generated (stream processing) are vital to many industrial applications. Fortunately, existing research on temporal databases, as well as time and streams in semantic technologies can be integrated into a unified OBDA framework (see Sect. 6.6). Another important domain that occurs in many applications is that of geo-spatial information, spatial proximity, containment, etc. Again, we expect that existing research about geo-spatial data storage, querying, and semantics can be integrated into the OBDA framework.

Other examples are aggregation (summation, averaging, etc.) and epistemic negation (questions about missing data) that have received little theoretical or practical attention,

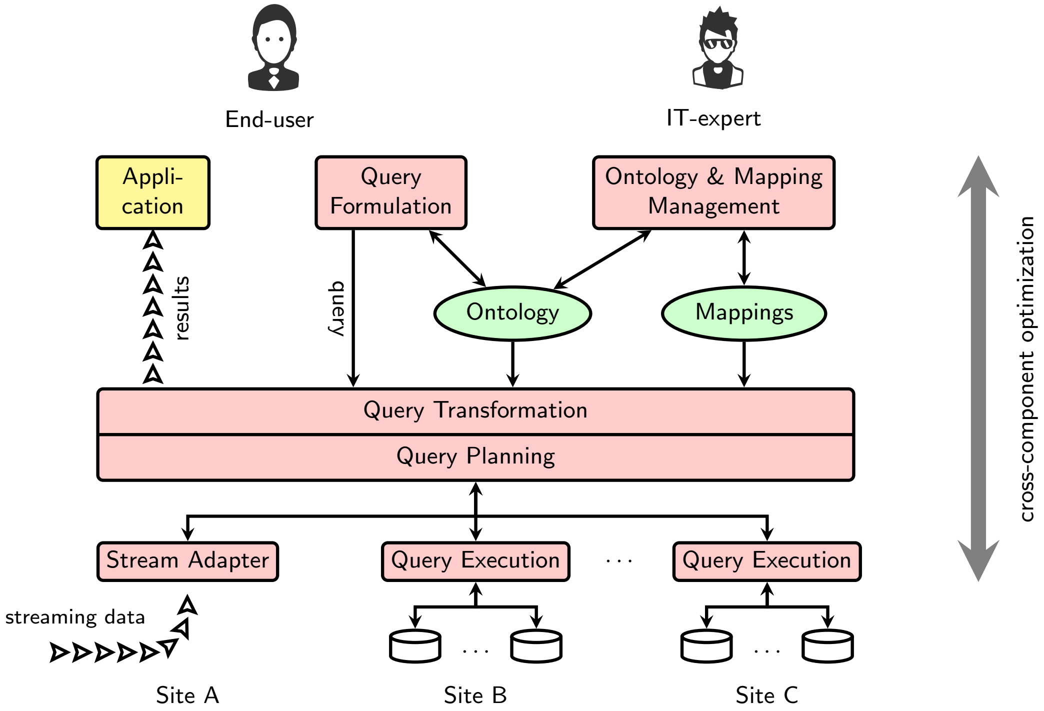


Figure 6.2: Platform architecture for scalable OBDA

but which are important in many practical applications.

To address **efficiency**, we propose to decompose the "Query Answering" component into several layers, as shown in Fig. 6.2:

1. *Query Transformation* using the ontology and mappings,
2. *Query Planning* to distribute queries to individual servers,
3. *Query Execution* using existing scalable data stores, or a massively parallelised (cloud) architecture.

The implementation of the query transformation layer can take recent theoretical advances in query rewriting into account, which can lead to significantly improved performance (see Sect. 6.5). The same holds for query execution, which can take advantage of research on massive parallelisation of query execution, with the possibility of scaling orders of magnitude beyond a conventional RDBMS architecture (see Sect. 6.7).

We surmise however, that to gain a *real* impact on efficiency, a holistic, cross-component view on query answering is needed: current OBDA implementations leave query planning and execution to off-the-shelf database products, often leading to suboptimal performance on the kinds of queries produced by a rewriting component. The complete query answering stack needs to be optimised as a whole, so that the rewritten queries can capitalise on the strengths of the query execution machinery, and the query execution machinery is optimised for the queries produced by the rewriting component.

In the following sections, we give a detailed discussion of the state of the art in the mentioned aspects, and the necessary expansions for an industrial-scale OBDA tool.

## 6.3 Query Formulation Support

Traditional database query languages, such as SQL, require some technical skills and knowledge about language syntax and domain schema. More precisely, they require users to recall relevant domain concepts and syntax elements and communicate their information need in a programmatic way. Such an approach makes information systems almost, if not completely, inaccessible to the end-users. Direct manipulation (Schneiderman, 1983) languages, which employ recognition (rather than recall) and direct manipulation objects (rather than a command language syntax), have emerged as a response to provide easy to use and intuitive interactive systems. In the database domain, *Visual Query Systems* (VQS, Catarci et al., 1997) follow the direct manipulation approach in which the domain of interest and the information need are represented by visual elements. Various interaction mechanisms and visualization paradigms – such as diagrams, forms etc. – have been employed (Catarci et al., 1997; Epstein, 1991) to enable end-users to easily formulate and construct their query requests. However, early approaches mostly missed a key notion, that is usability (Catarci, 2000), whose concern is the quality of the interaction between the user and the software system rather than the functionality or the technology of the software product. Increasing awareness on the usability in database domain is visible through a growing amount of research addressing end-user database access (e.g. Popov et al., 2011; Barzdins et al., 2008;

Uren et al., 2007)).

One of the key points for the success of a system, from the usability perspective, is its ability to clearly communicate the provided affordances for user interaction and the domain information that the user is expected to operate on. This concerns the representation and interaction paradigm employed by the system and the organization of the underlying domain knowledge. Concerning the former, researchers mostly try to identify the correlation between task (e.g., simple, complex etc.) and user type (e.g., novice, expert etc.) and the visual representation and interaction paradigm used (Catarci et al., 1997; Catarci, 2000; Popov et al., 2011). Regarding the latter, ontologies are considered as a key paradigm for capturing and communicating domain knowledge with the end-users (Uren et al., 2007; Barzdins et al., 2008; Tran et al., 2011).

A key feature of any OBDA system is that the ontology needs to provide a user-oriented conceptual model of the domain against which queries can be posed. This allows the user to formulate "natural" queries using familiar terms and without having to understand the structure of the underlying data sources. However, in order to provide the necessary power and flexibility, the underlying query language will inevitably be rather complex. It would be unrealistic to expect all domain experts to formulate queries directly in such a query language, and even expert users may benefit from tool support that exploits the ontology in order to help them to formulate coherent queries. Moreover, the ontology may not include all the vocabulary expected or needed by a given user. Ideally, it should be possible for users with differing levels of expertise to cooperate *on the same query*, by allowing them to switch between more or less technical representations as required, and to extend the ontology *on the fly* as needed for the query being formulated.

Many applications existing today use navigation of simple taxonomic ontologies in order to search for information; a user of eBay, for example, can navigate from "electronics" to "cameras & photography" to "camcorders" in order to find items of interest. In some cases, additional attributes may also be specified; in the above example, attributes such as "brand", "model" and "price" can also be specified. Sometimes called *faceted search* (Schneiderman, 1983; D.Tunkelang, 2009; Suominen et al., 2007; Lim et al., 2009), but the structure of the ontology is very simple, as is the form of the query—effectively just retrieving the instances of a given concept/class. The faceted search is based on series of orthogonal categories that can be applied in combination to filter the information space. Facets are derived from the properties of the information elements. In an ontology-based system, identification of these properties is straightforward. An important benefit of faceted search is that it frees users from the burden of dealing with complex form-based interfaces and from the possibility of reaching empty result sets. This faceted search approach, however, in its most common form breaks down as soon as a "join" between information about several objects is required. Consider e.g. searching for camcorders available from sellers who also have a digital camera with $\geq$ 12MP resolution on offer.

Similarly, ontology development tools such as Protégé may allow for the formulation of query concepts using terms from the ontology, but the query is again restricted to a single concept term. Specialised applications have sometimes used GUIs or form-based interfaces for concept formulation, e.g., the Pen & Pad data entry tool developed in the GALEN project (Nowlan et al., 1990), but if used for querying this would again provide only for concept/class instance retrieval queries.

An essential part of any practically usable system must be an interface that supports technically less advanced users by some kind of "query by navigation" interface, where the user gradually refines the query by selecting more specific concepts and adding relationships to other concepts, with the ontology being used to suggest relevant refinements and relationships (Nowlan et al., 1990; Catarci et al., 2004). Work on ontology-supported faceted search (Suominen et al., 2007; Lim et al., 2009) is also relevant in this context. Due to the rising familiarity of users with faceted search interfaces, a promising direction seems to be to extend faceted search with, amongst others,

- the ability to select several pieces of information for output (querying instead of search)
- a possibility for adding restrictions on several objects connected through roles, in order to allow joins
- a possibility to specify aggregation, like summation or averaging

- a possibility to specify the absence of information (e.g. that a vendor has no negative reviews)

The amalgamation of faceted search and navigational search, so-called *query by navigation* (ter Hofstede et al., 1996), is of importance for the realization of the aforementioned objectives. The navigational approach exploits the graph-based organization of the information to allow users to browse the information space by iteratively narrowing the scope. Stratified hypermedia (Bruza and van der Weide, 1992), a well known example of the navigational approach, is an architecture in which information is organized via several layers of abstraction. The base layer contains the actual information while other layers contain the abstraction of this information and enable access to the base layer. In a document retrieval system, the abstraction layer is composed of hierarchically organized keywords. An indexing process is required to characterize the documents and to construct the abstraction layer. However, the characterization of information instances in an ontology-based system is simple and provided by the reference ontology (ter Hofstede et al., 1996). The query by navigation approach is particularly supportive at the exploration phase of the query formulation (Marchionini and White, 2007). Recent applications of query by navigation are available in the Semantic Web domain in the form of textual semantic data browsers (e.g. Soylu et al., 2012; Popov et al., 2011).

A particular approach which combines faceted search and a diagrammatic form of query by navigation is presented in (Heim and Ziegler, 2011). The approach is based on the hierarchical organization of facets, and hence allows joins between several information collections. The main problem with such diagrammatic approaches and with textual data browsers is that they do not support dealing with large complex ontologies and schemata well, mainly lacking balance between overview and focus. For instance, a diagrammatic approach is good at providing an overview of the domain; however, it has its limits in terms of information visualization and users' cognitive bandwidths. A textual navigation approach is good at splitting the task into several steps; however, it can easily cause users to lose the overview. Therefore it is not enough to provide navigation along the taxonomy and relations captured in the ontology. In many cases, it turns out that accessing data is difficult even for end-users who are very knowledgeable in their domain, for two reasons: a) the complexity of the data model – which can be hidden using an ontology and mappings, but also b) the complexity of an accurate description of the domain. Often an ontology that accurately describes all relevant details of the domain will be more complicated than even experienced domain experts usually think about it in their daily work. This means that they approach the task of query construction without having complete knowledge of all the details of the domain model. It is therefore necessary to develop novel techniques to support users in formulating coherent queries that correctly capture their requirements. In addition to navigation, a query formulation tool should allow searching by name for properties and concepts the expert knows must be available in the ontology. The system should help users understand the ontology by showing how the concepts and properties relevant for a query are interconnected.

For instance, assume that the user would like to list all digital cameras with $\geq$ 12MP resolution. This sounds like a reasonable question that should have a unique interpretation. But the ontology might not actually assign a "resolution" to a camera. Rather, it might say that a camera has at least one image sensor, possibly several,[9] each of which has an effective and a total resolution. The camera also may or may not support a variety of video resolutions, independently of sensor's resolution. The system should let the users search for "resolution," help them find chains of properties from "Camera" to the different notions of "Resolution" and help them find out whether all sensors need to have $\geq$ 12MP, or at least one of them, etc., and which kind of resolution is meant.

For complex queries, any intuitive user interface for non-technical users will eventually reach its limits. It is therefore important to also provide a textual query interface for technically versed users that allows direct editing of a query using a formal syntax like the W3C SPARQL language. Ideally, both interfaces provide views on an underlying partially constructed query, and users can switch between views at will. Even in the textual interface, there should be more support than present-day interfaces provide, in the form of context-sensitive completion (taking account of the ontology), navigation support, etc. (as is done, e.g., in the input fields of the Protégé ontology editor (Knublauch et al., 2005)).

Finally, no ontology can be expected to cover a domain's vocabulary completely. The vocabulary is to a certain extent specific to individuals, projects, departments, etc. and

[9] For instance front-facing and rear-facing on a mobile phone, two sensors in a 3D camcorder, etc.

subject to change. To adapt to changing vocabularies, cater for omissions in the ontology, and to allow a light weight process for ontology development, the query formulation component should also support "on the fly" extension of the ontology during query formulation. This can be achieved by adapting techniques from ontology learning (Cimiano et al., 2005; Cimiano, 2006) in order to identify relevant concepts and relations, and adapting techniques from ontology alignment (aka matching) in order to relate this new vocabulary to existing ontology terms. In case such on-the-fly extensions are insufficient, users should also have access to the range of advanced tools and methodologies discussed in Sect. 6.4, although they may require assistance from an IT expert in order to use such tools.

## 6.4 Ontology and Mapping Management

The OBDA architecture proposed in this chapter depends crucially on the existence of suitable ontologies and mappings. In this context, the ontology provides a user-oriented conceptual model of the domain that makes it easier for users to formulate queries and understand answers. At the same time, the ontology acts as a "global schema" onto which the schemas of various data sources can be mapped.

Developing suitable ontologies from scratch is likely to be expensive. A more cost-effective approach is to develop tools and methodologies for semi-automatically "bootstrapping" the system with a suitable initial ontology and for extending the ontology "on the fly" as needed by a given application. This means that in this scenario ontologies are dynamic entities that evolve (i) to incorporate new vocabulary required in user queries, and (ii) to accommodate new data sources. In both cases, some way is needed to ensure that vocabulary and axioms are added to the ontology in a coherent way.

Regarding the ontology/data-source mappings, many of these will, like the ontology, be generated automatically from either database schemata and other available metadata or formal installation models. However, these initial mappings are unlikely to be sufficient in all cases, and they will certainly need to evolve along with the ontology. Moreover, new data sources may be added, and this again requires extension and adjustment of the mappings.

The management of large, evolving sets of mappings must be seen as an engineering problem on the same level as that of ontology management.

Apart from an initial translation from structured sources like, e.g., a relational database schema, present-day ontology management amounts to using interactive ontology editors like Protégé, NeOn, or TopBraid Composer.[10] These tools support the construction and maintenance of complex ontologies, but they offer little support for the kind of ontology evolution described above.

The issue of representing and reasoning about schema mappings has been widely investigated in recent years. In particular, a large body of work has been devoted to studying operators on schema mappings relevant to model management, notably, composition, merge, and inverse (Bernstein and Ho, 2007; Kolaitis, 2005; Madhavan and Halevy, 2003; Fagin et al., 2005c; Fagin, 2007; Fagin et al., 2008b, 2009b; Arenas et al., 2009; Arocena et al., 2010; Arenas et al., 2010a,b). In (Fagin et al., 2005a,b; Arenas et al., 2004; Fuxman et al., 2005; Libkin and Sirangelo, 2008) the emphasis is on providing foundations for data interoperability systems based on schema mappings. Other works deal with answering queries posed to the target schema on the basis of both the data at the sources, and a set of source-to-target mapping assertions (e.g. Abiteboul and Duschka, 1998; Arenas et al., 2004; Calì et al., 2004) and the surveys in (Ullman, 1997; Halevy, 2001; Halevy et al., 2006)).

Another active area of research is principles and tools for comparing both schema mapping languages, and schema mappings expressed in a certain language. Comparing schema mapping languages aims at characterizing such languages in terms of both expressive power and complexity of mapping-based computational tasks (ten Cate and Kolaitis, 2009; Alexe et al., 2010). In particular, (ten Cate and Kolaitis, 2009) studies various relational schema mapping languages with the goal of characterizing them in terms of structural properties possessed by the schema mappings specified in these languages. Methods for comparing schema mappings have been proposed in (Fagin et al., 2008a; Gottlob et al., 2009; Fagin et al., 2009b; Arenas et al., 2010a), especially in the light of the need of a theory of schema mapping optimisation. In (Fagin et al., 2009b; Arenas et al., 2010a), schema mappings are compared with respect to their ability to transfer source data and avoid redundancy in the

[10] http://protege.stanford.edu/, http://neon-toolkit.org/, http://www.topbraidcomposer.com/

target databases, as well as their ability to cover target data. In (Fagin et al., 2008a), three notions of equivalence are introduced. The first one is the usual notion based on logic: two schema mappings are logically equivalent if they are indistinguishable by the semantics, i.e., if they are satisfied by the same set of database pairs. The other two notions, called data exchange and conjunctive query equivalence, respectively, are relaxations of logical equivalence, capturing indistinguishability for different purposes.

Most of the research mentioned above aims at methods and techniques for analyzing schema mappings. However, mapping management is a broader area, which includes methods for supporting the development of schema mappings, debugging such mappings, or maintaining schema mappings when some part of the specification (for example, one of the schemas) changes. Although some tools are already available, (e.g., CLIO Fagin et al., 2009a), and some recent papers propose interesting approaches (e.g. Glavic et al., 2010), this problem is largely unexplored, especially in the realm of OBDA. Specifically, the following problems are so far unsolved in the area, but are crucial in dealing with complex scenarios:

*(i)* Once a set of mappings has been defined, the designer often needs to analyze them, in order to verify interesting properties (for example, minimality).

*(ii)* Mappings in OBDA systems relate the elements of the ontology to the data structures of the underlying sources. When the ontology changes, some of the mappings may become obsolete. Similarly, when the sources changes, either because new sources are added, or because they undergo modifications of various types, the mappings may become obsolete.

*(iii)* Different types of mappings (LAV, GAV, etc.) have been studied in the literature. It is well known that the different types have different properties from the point of view of expressive power of the mapping language, and computational complexity of mapping-based tasks. The ideal situation would be to use rich mapping languages during the design phase, and then transforming the mappings in such a way that efficient query answering is possible with them. This kind of transformation is called *mapping simplification.* Given a set of mappings $M$, the goal of simplification is to come up with a set of mappings that are expressed in a tractable class $C$, and approximate

at best $M$, i.e., such that no set $M'$ of mappings in $C$ exists which is "closer" to $M$ than $M'$.

Regarding ontologies, the required management and evolution described above could be reached by a combination of different techniques, including ontology alignment (Shvaiko and Euzenat, 2005; Jiménez-Ruiz et al., 2009) and ontology approximation (Brandt et al., 2001; Pan and Thomas, 2007). Both the addition of user-defined vocabulary from a query formulation process and the incorporation of the domain model for a new data source are instances of an ontology alignment problem. The results of aligning new user-requested vocabulary or new knowledge coming from new data sources with the existing ontology do not necessarily fall within the constrained fragments required for efficient OBDA (like, e.g., OWL 2 QL, Calvanese et al., 2007b). This problem can be dealt with by an *approximation* approach, i.e., transforming the ontology into one that is as expressive as possible while still falling within the required profile. In general, finding an optimal approximation may be costly or even undecidable, but effective techniques are known for producing good approximations (Brandt et al., 2001; Pan and Thomas, 2007).

Concerning mapping management, in order to be able to freely analyze schema mappings, one possibility is to define a specific language for querying schema mappings. The goal of the language is to support queries of the following types: return all mappings that map concepts that are subsets of concept $C$; or, return all mappings that access table $T$ in the data source $S$. The basic step is to define a formal meta-model for mapping specification, so that queries over schema mappings will be expressions over this meta-model. A query language can thus be defined over such a meta-model: the general idea is to design the language in such a way that important properties (e.g., scalability of query answering) will be satisfied.

Based on this meta-model, reasoning techniques could be designed that support the evolution of schema mappings. The meta-model could also be used in order to address the issue of monitoring changes and reacting to them. Indeed, every change to the elements involved in the schema mappings may be represented as specific updates on the instance level of the meta-level. The goal of such a reasoning system is to specify the actions to perform, or the actions to suggest to the designer, when these update operations change the instances of the meta-model.

## 6.5 Query Transformation

The OBDA architecture proposed in this chapter relies heavily on query rewriting techniques. The motivation for this is the ability of such techniques to separate ontology reasoning from data reasoning, which can be very costly in the presence of Big Data. However, although these techniques have been studied for several years, applying this technology to Big Data introduces performance requirements that go far beyond what can be obtained with simple approaches. In particular, emphasis must be put both on the performance of the rewriting process and on the performance of the evaluation of the queries generated by it. At the same time, meeting these performance requirements can be achieved by building on top of the experiences in the area of OBDA optimisation which we now briefly mention.

In the context of query rewriting with respect to the ontology only (i.e., mappings and query execution aside), recent results (Kikot et al., 2011; Rosati and Almatelli, 2010) have shown that performing query rewriting by means of succinct query expressions, e.g., non-recursive Datalog programs, can be orders of magnitude faster than the approaches that produce UCQs (unions of conjunctive queries) apriori (Calvanese et al., 2007b; Pérez-Urbina et al., 2008; Chortaras et al., 2011; Calì et al., 2009). Moreover, these succinct query representations are in general cheaper to deal with during optimisation, since the structure that needs to be optimised is smaller. Complementary to these results are optimisation results for OBDA systems in which the data is in control of the query answering engine, where dramatic improvements can be achieved when load-time pre-computation of inferences is allowed. In particular, it has been shown in (Rodríguez-Muro and Calvanese, 2011) that full materialisation of inferences is not always necessary to obtain these benefits, that it is possible to capture most of the semantics of DL-Lite ontologies by means of simple and inexpensive indexing structures in the data-storage layer of the query answering system. These pre-computations allow to further optimise the rewriting process and the queries returned by this process.

In the context of query rewriting in the presence of mappings and where the data sources cannot be modified by the query answering system, a point of departure are recent approaches that focus on the analysis of the data sources and the mappings of the OBDA system (Rodríguez-Muro, 2010). Existing approaches (Rodríguez-Muro and Calvanese, 2011) focus on detecting the state of completeness of the sources w.r.t. the semantics of the ontology. The result of this analysis can be used for at least two types of optimisation, namely (i) optimisation of the ontology and mappings used during query rewriting (offline optimisation), and (ii) optimisation of the rewritten queries (online optimisation). For the former, initial work can be found in the semantic preserving transformations explored in (Rodríguez-Muro and Calvanese, 2011). For the latter, early experiences (Rodríguez-Muro, 2010; Rodriguez-Muro and Calvanese, 2012; Calvanese et al., 2011; Pérez-Urbina et al., 2008) suggest that traditional theory of Semantic Query Optimisation (SQO, Grant et al., 1997) can be applied in the OBDA context as long as the chosen rewriting techniques generate queries that are cheap to optimise using SQO techniques, for example (Kikot et al., 2011; Rosati and Almatelli, 2010). Complementary, a first-order logics based approach to Semantic Query Optimization in the context of semantic data formats and reasoning has been proposed in (Schmidt et al., 2010). Finally, previous experiences also suggest that in the context of query rewriting into SQL, obtaining high performance is not guaranteed by using an optimised DBMS system; instead, it has been shown (Rodríguez-Muro, 2010) that the form of the SQL query (e.g., use of sub-queries, views, nested expressions, etc.) plays a critical role, and that even in commercial DBMS engines, care must be taken to guarantee that the SQL queries are in a form that the DBMS can plan and execute efficiently.

Regarding systems, a lot of experience has been accumulated in the last years and can be used to build the next generation OBDA systems. In particular, most of the aforementioned rewriting techniques, as well as optimisation techniques, have been accompanied by prototypes that were used to benchmark and study the applicability of these techniques empirically. The first example of these systems is QuOnto (Acciarri et al., 2005), a system that implements the core algorithms presented in (Calvanese et al., 2007b) and that seeded the idea of query answering through query rewriting in the context of DL ontologies. QuOnto has also served as a platform for the implementation of the epistemic-query answering techniques proposed in (Calvanese et al., 2007a) and served as a basis for the Mastro system (Calvanese et al., 2011), which implements OBDA-specific functionality. While these systems allowed for query answering over actual databases, initially they put little attention to the performance issue. Because of this, following prototypes focused strongly on the performance of the query rewriting algorithms; examples are Requiem (Pérez-Urbina et al., 2010), which implemented the resolution-based query rewriting techniques from (Pérez-Urbina et al., 2008), and Presto (Rosati and Almatelli, 2010), which implemented a succinct query translation based on non-recursive Datalog programs. Finally, the latest generation OBDA systems such as Quest (Rodriguez-Muro and Calvanese, 2012) and Prexto (Rosati, 2012) have focused on the exploitation of efficient rewriting techniques, SQO optimisation as well as the generation of efficient SQL queries.

At the same time, while these initial steps towards performance are promising, there are many challenges that arise in the context of industrial applications and Big Data that are not covered by current techniques. For example, optimisations of query rewriting techniques have only been studied in the context of rather inexpressive ontology and query languages such as OWL 2 QL/DL-Lite and UCQs; however, empirical evidence indicates that none of these languages is enough to satisfy industrial needs. Also, current proposals for optimisation using constraints have considered only the use of few classes of constraints, in particular, only simple inclusion dependencies, and little attention has been given to the use of functional dependencies and other forms of constraints that allow to represent important features of the sources and that are relevant for query answering optimisation.

Likewise, optimisation of OBDA systems has so far only been considered either in a pure 'on-the-fly' query rewriting context, in which sources are out of the scope of the query answering system, or in a context in which the data has been removed from the original source and transformed into an ABox. However, the experience that has been obtained experimenting with the current technology indicates that in practice, a middle ground could give rise to a higher degree of optimisation of the query answering process. It also appears that in the context of Big Data and the complex analytical queries that are often used in this context, good performance cannot be achieved otherwise and these hybrid approaches might be the only viable alternative. It has also become clear that declarative OBDA might not be the best choice to handle all tasks over Big Data. In some cases, domain specific

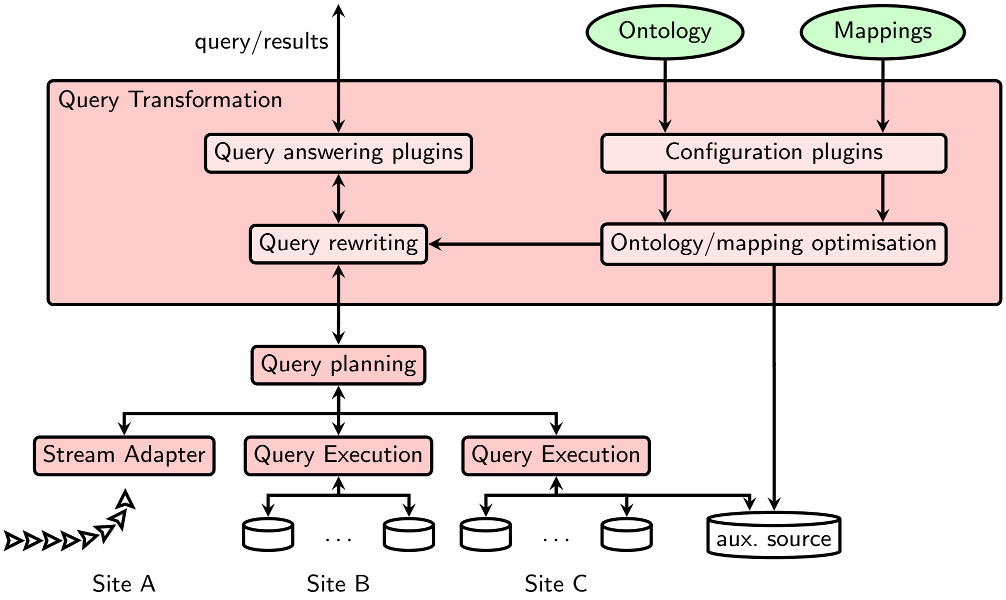


Figure 6.3: Fine structure of the Query Transformation component.

procedures can be more efficient, and hence, an OBDA system should provide the means to define such procedures (e.g., by means of domain specific plugins).

To conclude, an optimal system for query answering through query rewriting in the context of Big Data must be approached in an integral way, including modules that handle and optimise each of the aspects of the query answering process, while trying to maximize the benefits that are obtained by separating reasoning w.r.t. the Ontology vs. reasoning w.r.t. the data. The resulting architecture of such a system may look like the one proposed in this chapter and depicted in Fig. 6.3, where all optimisation techniques previously mentioned are combined into a framework that is expressive enough to capture industrial requirements, can understand the data sources (in the formal sense), and is able to identify the best way to achieve performance, being able to go from pure on-the-fly query answering to (partially) materialised query answering as needed.

## 6.6 Time and Streams

Time plays an important role in many industrial applications. Hence, OBDA based solutions for such applications have to provide means for efficiently storing and querying timed-stamped data. If we recast these user requirements to technical ones on the OBDA components involved in the life cycle of a query, we come to the conclusion that, first, the user query language should allow the reference to time (instances, intervals) and allow for adequate combinations with concepts of the ontology; that, second, the mapping language should allow the handling of time; and that, third, the back-end database should provide means for efficiently storing and retrieving temporal data, in particular, it should provide a temporal query language into which the user query will be transformed. One might also want to add a requirement for the ontology language such that it becomes possible to build temporal-thematic concepts in the user ontology; but regarding well-known unfeasibility results on temporal description logics (Artale et al., 2010), we will refrain from discussing any aspect concerning temporal constructors for ontology languages and rather focus on temporal query languages and temporal DBs.

While SQL provides built-in data types for times and dates, which can be used, for instance, for representing birthday data, representing validity of facts using, say, two attributes *Start* and *End* imposes severe problems for formulating queries in SQL. For instance, in a Big Data scenario involving possibly mobile sensors of one or more power plants, measurement values might be stored in a table *Sensor* with schema

$$Sensor(ID, Location, Value, Start, End),$$

and it might happen that the location changes while the value remains the same.

| ID | Location | Value | Start | End |
|---|---|---|---|---|
| ... | ... | ... | ... | ... |
| S_42 | Loc_1 | 16 | 15 | 20 |
| S_42 | Loc_1 | 17 | 20 | 25 |
| S_42 | Loc_2 | 17 | 25 | 30 |
| S_42 | Loc_2 | 18 | 30 | 35 |
| ... | ... | ... | ... | ... |

Now, querying for the (maximum) duration of a measurement with a particular value 17 (and neglecting the location of the sensor) should return a relation

$$\{(S\_42, 17, 20, 30)\}.$$

Although in principle one could specify an SQL query that maximizes the interval length to be specified in result tuples (see Zaniolo et al. (1997) for examples and for pointers to the original literature in which solutions were developed), the query is very complex (Zaniolo et al., 1997, p. 104) and will hardly be optimized appropriately by standard SQL query engines. Even worse, if only (irregular) time points are stored for measurements, one has to find the next measurement of a particular sensor and timepoint by a minimization query, and the problem of maximizing validity intervals in output relations as described above remains. In addition, an attribute *Timepoint* might also refer to the insertion time (transaction time) of the tuple rather than to the valid time as we have assumed in the discussion above.

In order to support users in formulating simpler queries for accessing temporal information appropriately, extensions to relational database technology and query languages such as SQL have been developed (e.g., TSQL2, see (Zaniolo et al., 1997) for an overview). The time ontology usually is defined by a linear time structure, a discrete representation of the real time line, and proposals for language standards as well as implementations provide data types for intervals or timestamps. A useful distinction adapted from constraint databases (Kuper et al., 2000) is the one between abstract and concrete temporal databases (Chomicki and Toman, 2005). The representation-independent definitions of temporal databases relying on the infinite structures of the time ontology are called abstract temporal databases; these are the objects relevant for describing the intended semantics for query answering. Finite representations of abstract temporal databases are termed concrete temporal databases (Chomicki and Toman, 2005); these rely on compact representations by (time) intervals.

Temporal databases provide for means of distinguishing between valid time and transaction time. Valid time captures the idea of denoting the time period during which a fact is considered to be true (or to hold w.r.t. the real world). With transaction time the time point (or time period) during which a fact is stored in the database is denoted. It might be the case that valid time is to be derived from transaction time (due to a sampling interval). In case of a transaction time point often valid time is to be derived by retrieving the "next" entry, assuming an assertion is valid until the next one appears. It might also be the case that both types of time aspects are stored in the database leading to so called bitemporal databases (Jensen et al., 1993). Using a temporal database, a query for checking which values the sensors indicate between time units should be as easy as in the following example:

```
SELECT ID, Value FROM Sensor WHERE Start >= 23 and End <= 27;
```

with the intended result being a single tuple

$$\{(S\_42, 17)\}.$$

The reason for expecting this result can be explained by the abstract vs. concrete distinction. The table with the mobile sensor from the beginning of this section is considered to be part of a concrete temporal database that represents an abstract temporal database. The abstract temporal database holds relations of the form

$$Sensor(ID, Location, Value, T)$$

meaning that sensor $ID$, located in *Location*, has a value $Value$ measured/valid in time $T$ for all $T$ such that there is an entry in the concrete temporal database with *Start* and *End* values in between which $T$ lies.

Note that the resulting answer to the last query above is the empty set if the mobile sensor data are understood as being part of a pure (non-temporal) SQL DB.

One could extend the simple SQL query from above to a temporal SQL query that also retrieves the locations of the sensors.

```
SELECT ID, Location, Value FROM Sensor WHERE Start >= 20 and End <= 27;
```

The expected result w.r.t. the semantics of abstract temporal databases is

$$\{(S\_42, Loc\_1, 17), (S\_42, Loc\_2, 17)\}.$$

Again note that the resulting answer set would have been different for non-temporal SQL, namely

$$\{(S\_42, Loc\_1, 17)\}.$$

As a third example for querying temporal databases think of a query retrieving temporal information. For the query

```
SELECT ID, Value, Start, End FROM Sensor WHERE Start <= 23 and End >= 27;
```

the expected result is

$$\{(S\_42, 17, 20, 30)\}.$$

Index structures for supporting these kinds of queries have been developed, and add-ons to commercial products offering secondary-memory query answering services such as those sketched above are on the market. Despite the fact that standards have been proposed (e.g., ATSQL) no agreement has been achieved yet, however. Open source implementations for mapping ATSQL to SQL have been provided as well (e.g., TimeDB, Steiner, 1997; Tang et al., 2003).

For many application scenarios, however, only small "windows" of data are required, and thus, storing temporal data in a database (and in external memory) as shown above might cause a lot of unnecessay overhead in some application scenarios. This insight gave rise to the idea of stream-based query answering. In addition to window-based temporal data access, stream-based query answering adopts the view that multiple queries are registered and assumed to be answered "continuously". For this kind of continuous query answering, appropriate index structures and join algorithms have been developed in the database community (see, e.g., Cammert et al. (2003, 2005) for an overview). Data might be supplied incrementally by multiple sources. Combining these sources defines a fused stream of data over which a set of registered queries is continuously answered. In stream-based query answering scenarios, an algebraically specified query (over multiple combined streams set up for a specific application) might be implemented by several query plans that are optimized w.r.t. all registered queries. An expressive software library for setting up stream-based processing scenarios is described in (Cammert et al., 2003).

Rather than by accessing the whole stream, continuous queries refer to only a subset of all assertions, which is defined by a sliding time window. Interestingly, the semantics of sliding windows for continuous queries over data streams is not easily defined appropriately, and multiple proposals exist in the literature (e.g. Krämer and Seeger, 2009; Zhang et al., 2001).

For event recognition, temporal aggregation operators are useful extensions to query languages, and range predicates have to be supported in a special way to compute temporal aggregates (Zhang et al., 2001). In addition, expectation on trajectories help to answer continuous queries in a faster way (Schmiegelt and Seeger, 2010). Moreover, it is apparent that the "best" query plan might depend on the data rates of various sources, and dynamic replanning might be required to achieve best performance over time (Krämer et al., 2006; Heinz et al., 2008).

While temporal query answering and stream-based processing has been discussed for a long time in the database community (e.g. Law et al., 2004), recently L data representation formats and query answering languages have become more and more popular. Besides XML, e.g., the Resource Description Format (RDF) has been investigated in temporal or stream-based application contexts. Various extensions to the RDF query language SPARQL have been proposed for stream-based access scenarios in the RDF context (Bolles et al., 2008; Barbieri et al., 2010b; Calbimonte et al., 2010). With the advent of SPARQL 1.1, aggregate functions are investigated in this context as well.

Streaming SPARQL (Bolles et al., 2008) was one of the first approaches based on a specific algebra for streaming data. However, data must be provided "manually" in RDF in this approach. On the other hand, mappings for relating source data to RDF ontologies in an automated way have been investigated in stream-based query answering scenarios as well (e.g. Calbimonte et al., 2010). In contrast to ontology-based data access (OBDA) methods, nowadays these approaches require the materialization of structures at the ontology level

(RDF) in order to provide the input data for stream-based query systems. For instance, C-SPARQL queries are compiled to SPARQL queries over RDF data that was produced with specific mappings. C-SPARQL deals with entailments for RDFS or OWL 2 RL by relying on incremental materialization (Barbieri et al., 2010a). See also (Ren and Pan, 2011) for an approach based on EL++.

SPARQLstream (Calbimonte et al., 2010) provides for mappings to ontology notions and translates to stream-based queries to SNEEQL (Brenninkmeijer et al., 2008), which is the query language for SNEE, a query processor for wireless sensor networks. Stream-based continuous query answering is often used in monitoring applications for detecting events, possibly in real time. EP-SPARQL (Anicic et al., 2011a), which is tailored for complex event processing, is translated to ETALIS (Event TrAnsaction Logic Inference System (Anicic et al., 2011b)), a Prolog-based real time event recognition system based on logical inferences.

While translation to SQL, SPARQL or other languages is attractive w.r.t. reusing existing components in a black box approach, some information might be lost, and the best query execution plan might not be found. Therefore, direct implementations of stream-based query languages based on RDF are also investigated in the literature. CQELS (Phuoc et al., 2011) is a much faster "native" implementation (and does not rely on transformation to underlying non-stream-based query languages). In addition, in the latter stream-based querying approach, queries can also refer to static RDF data (e.g., linked open data). In addition, a direct implementation of temporal and static reasoning with ontologies has been investigated for media data interpretation in (Möller and Neumann, 2008; Peraldi et al., 2011). Event recognition w.r.t. expressive ontologies has been investigated recently (Luther et al., 2008; Wessel et al., 2007, 2009; Baader et al., 2009).

As we have seen, it is important to distinguish between temporal queries and window-based continuous queries for streams. Often the latter are executed in main memory before data is stored in a database, and much work has been carried out for RDF. However, temporal queries are still important in the RDF context as well. T-SPARQL (Grandi, 2010) applies techniques from temporal DBs (TSQL2, SQL/Temporal, TimeDB) to RDF querying (possibly also with mappings to plain SQL) to define a query language for temporal RDF. For data represented using the W3C standard RDF, an approach for temporal query answering has been developed in an industrial project (Motik, 2010). It is shown that ontology-based answering of queries with specific temporal operators can indeed be realized using a translation to SQL.

In summary, it can be concluded that there is no unique semantics for the kind of queries discussed above, i.e., neither for temporal nor for stream-based queries. A combination of stream-based (or window-based), temporal (history-based), and static querying is useful in applications, but is not provided at a time by most approaches. Early work on deductive event recognition (Neumann and Novak, 1983; Neumann, 1985; Neumann and Novak, 1986; Kockskämper et al., 1994; André et al., 1988) already contains many ideas of recently published efforts, and in principle a semantically well-founded combination of quantitative temporal reasoning w.r.t. valid time has been developed. However, scalability was not the design goal of these works.

While database-based temporal querying approaches or RDF-based temporal and stream-querying approaches as discussed above offer fast performance for large data and massive streams with high data rates, query answering w.r.t. (weakly expressive) ontologies is supported only with brute-force approaches like materialization. It is very unlikely that this approach results in scalable query answering for large real-world ontologies of the future due to the enormous blowup (be the materialization managed incrementally or not). Furthermore, reasoning support is quite limited, i.e., the expressivity of the ontology languages that queries can refer to is quite limited. Fortunately, it has been shown that brute-force approaches involving materialization for ontology-based query answering are not required for efficiently accessing large amounts of data if recently developed ontology based data access (OBDA) techniques are applied.

A promising idea for scalable stream-based answering of continuous queries is to apply the idea of query transformation w.r.t. ontologies also for queries with temporal semantics. Using an ontology and mapping rules to the nomenclature used in particular relational database schemas, query formulation is much easier. Scalability can, e.g., be achieved by a translation to an SQL engine with temporal extensions and native index structures and processing algorithms (e.g., as offered by Oracle).

As opposed to what current systems offer, stream-based processing of data usually does not give rise to the instantiation of events with absolute certainty. Rather, data acquired (observations) can be seen as cues that have to be aggregated or accumulated in order to be able to safely infer that a certain event has occurred. These events might be made explicit in order to be able to refer to them directly in subsequent queries (rather than recomputing them from scratch all the time). The central idea of (Gries et al., 2010) is to use aggregation operators for data interpretation. Note that interpretation is more than mere materialization of the deductive closure: with interpretation new and relevant data is generated to better focus temporal and stream-based query answering algorithms.

## 6.7 Distributed Query Execution

In the past, OBDA approaches simply assumed centralized query execution using a well-known relational database system, e.g., PostgreSQL (Savo et al., 2010). However, this assumption does not usually hold in the real world where data is distributed over many autonomous, heterogeneous sources. In addition, existing relational database systems, such as PostgreSQL, cannot scale when faced with TBs of data and the kinds of complex queries to be generated by a typical OBDA query translation component (Savo et al., 2010).

Relevant research in this area includes previous work on query processing in parallel, distributed, and federated database systems, which has been studied for a long time by the database community (Sheth, 1991; DeWitt and Gray, 1992; Kossmann, 2000; Özsu and Valduriez, 1999). Based on principles established in these pioneering works, recently also a variety of approaches for federated query processing in the context of semantic data processing have been proposed (see Görlitz and Staab (2011) for a recent survey). Falling into this category, our own work on FedX (Schwarte et al., 2011) presents a federated query processing engine operating on top of autononous semantic databases. The FedX engine enables the virtual integration of heterogeneous sources and implements efficient query evaluation strategies, driven by novel join processing and grouping techniques to minimize the number of requests sent to the federation members. These techniques are based on innovative source selection strategies, pursuing the goal to identify minimal sets of federation members that can contribute answers to the respective subqueries. Coming with all these features, FedX can easily be leveraged to OBDA scenarios whenever the source systems scale with the amounts of data and queries to be processed in the concrete ODBA scenario.

For truly large-scale, heterogeneous datastores, efficient evaluation of queries produced by the query translation component discussed in Section 6.5 requires massively parallel and distributed query execution. To cover such scenarios, cloud computing has attracted much attention in the research community and software industry. Thanks to virtualization, cloud computing has evolved over the years from a paradigm of basic IT infrastructures used for a specific purpose (clusters), to grid computing, and recently to several paradigms of resource provisioning services: depending on the particular needs, infrastructures (IaaS — Infrastructure as a Service), platforms (PaaS — Platform as a Service), and software (SaaS — Software as a Service) can be provided as services (Gonzalez et al., 2009). One of the important advantages of these newest incarnations of cloud computing is the cost model of resources. Clusters represent a fixed capital investment made up-front and a relatively small operational cost paid over time. In contrast, IaaS, PaaS, and SaaS clouds are characterized by elasticity (Kllapi et al., 2011), and offer their users the ability to lease resources only for as long as needed, based on a per quantum pricing scheme, e.g., one hour on Amazon EC2.[11] Together with the lack of any up-front cost, this represents a major benefit of clouds over earlier approaches.

The ability to use computational resources that are available on demand challenges the way that algorithms, systems, and applications are implemented. Thus, new computing paradigms that fit closely the elastic computation model of cloud computing were proposed. The most popular of these paradigms today is MapReduce (Dean and Ghemawat, 2008). The intuitive appeal of MapReduce, and the availability of platforms such as Hadoop (Apache, 2011), has recently fueled the development of Big Data platforms that aim to support the query language SQL on top of MapReduce (e.g., Hive, Thusoo et al. (2010), and HadoopDB, Bajda-Pawlikowski et al. (2011)).

[11] http://aws.amazon.com/ec2

Our own work on massively parallel, elastic query execution for Big Data takes place in the framework of the *Athena Distributed Processing (ADP)* system (Tsangaris et al., 2009; Kllapi et al., 2011). Massively parallel query execution is the ability to run queries with the maximum amount of parallelism at each stage of execution. Elasticity means that query execution is flexible; the same query can be executed with more or less resources, given the availability of resources for this query and the execution time goals. Making sure that these two properties are satisfied is a very hard problem in a federated data sources environment such as the ones discussed in this chapter. The current version of ADP, and its extensions planned for the near future, provide a framework with the right high-level abstractions and an efficient implementation for offering these two properties.

ADP utilizes state-of-the-art database techniques: (i) a declarative query language based on data flows, (ii) the use of sophisticated optimization techniques for executing queries efficiently, (iii) operator extensibility to bring domain specific computations into the database processing, and (iv) execution platform independence to insulate applications from the idiosynchracies of execution environments such as local clusters, private clouds, or public clouds.

Figure 6.4 shows the current architecture of ADP. The queries are expressed in a data flow language allowing complex graphs with operators as nodes and with edges representing producer-consumer relationships. Queries are optimized and transformed into execution plans that are executed in ART, the ADP Run Time. The resources needed to execute the queries (machines, network, etc.) are reserved or allocated by ARM, the ADP Resource Mediator. Those resources are wrapped into containers. Containers are used to abstract from the details of a physical machine in a cluster or a virtual machine in a cloud. The information about the operators and the state of the system is stored in the Registry. ADP uses state of the art technology and well proven solutions inspired by years of research in parallel and distributed databases (e.g., parallelism, partitioning, various optimizations, recovery).

Several services that are useful to the OBDA paradigm discussed in this chapter have already been developed on top of ADP: an SQL engine (AdpDB), a MapReduce engine (AdpMR), and a data mining library (AdpDM). Some core research problems have also

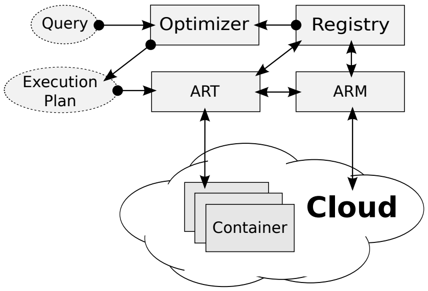


Figure 6.4: The architecture of the ADP system

been studied in depth. For example, in (Kllapi et al., 2011), we have studied the problem of scheduling dataflows that involve arbitrary data processing operators in the context of three different optimization problems: 1) minimize completion time given a fixed budget, 2) minimize monetary cost given a deadline, and 3) find trade-offs between completion time and monetary cost without any a-priori constraints. We formulated these problems and presented an approximate optimization framework to address them that uses resource elasticity in the cloud. To investigate the effectiveness of our approach, we incorporate the devised framework into ADP and instantiate it with several greedy, probabilistic, and exhaustive search algorithms. Finally, through several experiments that we conducted with the prototype elastic optimizer on numerous scientific and synthetic dataflows, we identified several interesting general characteristics of the space of alternative schedules as well as the advantages and disadvantages of the various search algorithms. The overall results are very promising and show the effectiveness of our approach.

To maximize the impact of ADP to the OBDA Big Data paradigm discussed in this chapter, ADP will be extended as follows:

- **Tight integration with query transformation modules:** We will develop query planning and execution techniques for queries produced by query translators such as the ones of Section 6.5 by integrating the ADP system tightly with Quest (Rodriguez-Muro and Calvanese, 2012). The integration will start by interfacing with SQL using the AdpDB service, and continue using lower level dataflow languages and providing hints (e.g., the degree of parallelism) to the ADP engine in order to increase its scalability.

- **Federation:** Building on the input of the query transformation modules, federation will be supported in ADP by scheduling the operators of the query to the different sites so that appropriate cost metrics are minimized. Among others, the techniques include scheduling of operators close to the appropriate data sources and moving data (when possible) to sites with more compute power.

- **Continuous and Temporal Query Support:** Continuous queries such as the ones discussed in Section 6.6 will be supported natively by data streaming operators. Similarly, temporal queries will be supported by special operators that can handle temporal semantics.

## 6.8 Conclusion

Giving end-users with limited IT expertise flexible access to large corporate data stores is a major bottleneck in data-intensive industries. Typically, standard domain specific tools only allow users to access data using a predefined set of queries. Any information need that goes beyond these predefined queries will require the help of a highly skilled IT expert, who knows the data storage intimately, and who know the application domain sufficiently well to communicate with the end users.

This is costly, not only because such IT experts are a scarce resource, but also because the time of the expert end-users are not free for core tasks. We have argued how ontology-based data access (OBDA) can provide a solution: by capturing the end-users' vocabulary in a formal model (ontology), and maintaining a set of mappings from this vocabulary to the data sources, we can automate the translation work previously done by the IT-experts.

OBDA has in recent years received a large amount of theoretical attention, and there are also several prototypical implementations. But in order to apply the idea to actual industry data, a number of limitations still needs to be overcome. In Sect. 6.2.2, we have identified the specific problems of usability, prerequisites, scope, and efficiency.

We have argued that these problems can be overcome by a novel combination of techniques, encompassing an end-user oriented query interface, query-driven ontology construction, new ideas for scalable query rewriting, temporal and streaming data processing, and query execution on elastic clouds.

The ideas proposed in this chapter are now investigated and implemented in the FP7 Integrating Project *Optique – Scalable End-user Access to Big Data*.[12]

[12] See `http://www.optique-project.eu/`.

# Bibliography

Abiteboul, Serge and Oliver Duschka (1998), Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, 254–265.

Acciarri, Andrea, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati (2005), QuOnto: Querying ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 1670–1671.

Alexe, Bogdan, Phokion Kolaitis, and Wang-Chiew Tan (2010), Characterizing schema mappings via data examples. In *Proc. of the 29th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2010)*, 261–271.

Amoroso, Alfonso, Gennaro Esposito, Domenico Lembo, Paolo Urbano, and Raffaele Vertucci (2008), Ontology-based data integration with Mastro-i for configuration and data management at SELEX Sistemi Integrati. In *Proc. of the 16th Ital. Conf. on Database Systems (SEBD 2008)*, 81–92.

André, Elisabeth, Gerd Herzog, and Thomas Rist (1988), On the simultaneous interpretation of real world image sequences and their natural language description: The system Soccer. In *Proc. of the European Conference on Artificial Intelligence (ECAI 1988)*, 449–454.

Anicic, Darko, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic (2011a), EP-SPARQL: A unified language for event processing and stream reasoning. In *Proc. of the 20th Int. World Wide Web Conference (WWW 2011)*.

Anicic, Darko, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic, and Rudi Studer (2011b), ETALIS: Rule-based reasoning in event processing. In *Reasoning in Event-Based Distributed Systems* (Sven Helmer, Alex Poulovassilis, and Fatos Xhafa, eds.), volume 347 of *Studies in Computational Intelligence*, 99–124, Springer.

Apache (2011), Apache Hadoop, `http://hadoop.apache.org/`. URL `http://hadoop.apache.org/`.

Arenas, Marcelo, Pablo Barcelo, Ronald Fagin, and Leonid Libkin (2004), Locally consistent transformations and query answering in data exchange. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2004)*, 229–240.

Arenas, Marcelo, Ronald Fagin, and Alan Nash (2010a), Composition with target constraints. In *Proc. of the 13th Int. Conf. on Database Theory (ICDT 2010)*, 129–142.

Arenas, Marcelo, Jorge Pérez, Juan L. Reutter, and Cristian Riveros (2010b), Foundations of schema mapping management. In *Proc. of the 29th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2010)*, 227–238.

Arenas, Marcelo, Jorge Pérez, and Cristian Riveros (2009), The recovery of a schema mapping: Bringing exchanged data back. *ACM Trans. on Database Systems*, 34.

Arocena, Patricia C., Ariel Fuxman, and Renée J. Miller (2010), Composing local-as-view mappings: Closure and applications. In *Proc. of the 13th Int. Conf. on Database Theory (ICDT 2010)*, 209–218.

Artale, A., D. Calvanese, R. Kontchakov, and M. Zakharyaschev (2009), The DL-Lite family and relatives. *J. of Artificial Intelligence Research*, 36, 1–69.

Artale, Alessandro, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev (2010), Past and future of DL-Lite. In *AAAI Conference on Artificial Intelligence*.

Baader, Franz, Andreas Bauer, Peter Baumgartner, Anne Cregan, Alfredo Gabaldon, Krystian Ji, Kevin Lee, David Rajaratnam, and Rolf Schwitter (2009), A novel architecture for situation awareness systems. In *Proc. of the 18th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux 2009)*, volume 5607 of *Lecture Notes in Computer Science*, 77–92, Springer-Verlag.

Bajda-Pawlikowski, Kamil, Daniel J. Abadi, Avi Silberschatz, and Erik Paulson (2011), Efficient processing of data warehousing queries in a split execution environment. In *Proc. of SIGMOD*.

Barbieri, Davide Francesco, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus (2010a), Incremental reasoning on streams and rich background knowledge. In *Proc. of the 7th Extended Semantic Web Conference (ESWC 2010)*, volume 1, 1–15.

Barbieri, Davide Francesco, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus (2010b), Querying RDF streams with C-SPARQL. *SIGMOD Record*, 39, 20–26.

Barzdins, Guntis, Edgars Liepins, Marta Veilande, and Martins Zviedris (2008), Ontology enabled graphical database query tool for end-users. In *Databases and Information Systems V - Selected Papers from the Eighth International Baltic Conference, DB&IS 2008, June 2-5, 2008, Tallinn, Estonia* (Hele-Mai Haav and Ahto Kalja, eds.), volume 187 of *Frontiers in Artificial Intelligence and Applications*, 105–116, IOS Press.

Bernstein, Philip A. and Howard Ho (2007), Model management and schema mappings: Theory and practices. In *Proc. of the 33rd Int. Conf. on Very Large Data Bases (VLDB 2007)*, 1439–1440.

Beyer, Mark A., Anne Lapkin, Nicholas Gall, Donald Feinberg, and Valentin T. Sribar (2011), 'Big Data' is only the beginning of extreme information management. Gartner report G00211490.

Bolles, Andre, Marco Grawunder, and Jonas Jacobi (2008), Streaming SPARQL - Extending SPARQL to process data streams. In *Proc. of the 5th European Semantic Web Conference (ESWC 2008)*, 448–462, URL `http://data.semanticweb.org/conference/eswc/2008/paper/3`.

Brachman, Ronald J. and Hector J. Levesque (1984), The tractability of subsumption in frame-based description languages. In *AAAI*, 34–37.

Brandt, Sebastian, Ralf Küsters, and Anni-Yasmin Turhan (2001), Approximation in description logics. LTCS-Report 01-06, LuFG Theoretical Computer Science, RWTH Aachen, Germany. Available at `http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html`.

Brenninkmeijer, Christian, Ixent Galpin, Alvaro Fernandes, and Norman Paton (2008), A semantics for a query language over sensors, streams and relations. In *Sharing Data, Information and Knowledge* (Alex Gray, Keith Jeffery, and Jianhua Shao, eds.), volume 5071 of *Lecture Notes in Computer Science*, 87–99, Springer.

Bruza, P.D. and T.P. van der Weide (1992), Stratified hypermedia structures for information disclosure. *Computer Journal*, 35, 208–220.

Calbimonte, Jean-Paul, Óscar Corcho, and Alasdair J. G. Gray (2010), Enabling ontology-based access to streaming data sources. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC 2010)*, 96–111.

Calì, A., G. Gottlob, and T. Lukasiewicz (2009), A general datalog-based framework for tractable query answering over ontologies. In *Proc. of the 28th ACM Symposium on Principles of Database Systems (PODS 2009)*, 77–86.

Calì, Andrea, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini (2004), Data integration under integrity constraints. *Information Systems*, 29, 147–163.

Calvanese, Diego, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo (2011), The MASTRO system for ontology-based data access. *Semantic Web Journal*, 2, 43–53.

Calvanese, Diego, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati (2007a), EQL-Lite: Effective first-order query processing in description logics. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, 274–279.

Calvanese, Diego, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati (2007b), Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39, 385–429.

Calvanese, Diego, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati (2012), Data complexity of query answering in description logics. *Artificial Intelligence*. To appear.

Cammert, Michael, Christoph Heinz, Jürgen Krämer, Martin Schneider, and Bernhard Seeger (2003), A status report on XXL - a software infrastructure for efficient query processing. *IEEE Data Eng. Bull.*, 26, 12–18.

Cammert, Michael, Christoph Heinz, Jürgen Krämer, and Bernhard Seeger (2005), Sortierbasierte Joins über Datenströmen. In *BTW*, volume 65 of *LNI*, 365–384, GI, URL `http://dblp.uni-trier.de/db/conf/btw/btw2005.html#CammertHKS05`.

Catarci, Tiziana (2000), What happened when database researchers met usability. *Information Systems*, 25, 177 – 212, URL `http://www.sciencedirect.com/science/article/pii/S0306437900000156`.

Catarci, Tiziana, Maria F. Costabile, Stefano Levialdi, and Carlo Batini (1997), Visual query systems for databases: A survey. *Journal of Visual Languages & Computing*, 8, 215 – 260, URL `http://www.sciencedirect.com/science/article/pii/S1045926X97900379`.

Catarci, Tiziana, Paolo Dongilli, Tania Di Mascio, Enrico Franconi, Giuseppe Santucci, and Sergio Tessaris (2004), An ontology based visual tool for query formulation support. In *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004)* (Ramon López de Mántaras and Lorenza Saitta, eds.), 308–312, IOS Press.

Chomicki, Jan and David Toman (2005), Temporal databases. In *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1, 429–467, Elsevier.

Chortaras, Alexandros, Despoina Trivela, and Giorgos B. Stamou (2011), Optimized query rewriting for OWL 2 QL. In *Proc. of the 23st Int. Conf. on Automated Deduction (CADE 2011)*, 192–206.

Cimiano, Philipp (2006), *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer.

Cimiano, Philipp, Andreas Hotho, and Steffen Staab (2005), Learning concept hierarchies from text corpora using formal concept analysis. *J. of Artificial Intelligence Research*, 24, 305–339.

Dean, Jeffrey and Sanjay Ghemawat (2008), MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51, 107–113, URL `http://doi.acm.org/10.1145/1327452.1327492`.

DeWitt, David J. and Jim Gray (1992), Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 35, 85–98.

D.Tunkelang (2009), *Faceted Search*. Morgan and Claypool.

Epstein, Richard G. (1991), The tabletalk query language. *Journal of Visual Languages & Computing*, 2, 115 – 141, URL `http://www.sciencedirect.com/science/article/pii/S1045926X05800266`.

Fagin, Ronald (2007), Inverting schema mappings. *ACM Trans. on Database Systems*, 32.

Fagin, Ronald, Laura M. Haas, Mauricio A. Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis (2009a), Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos*, 198–236, Springer.

Fagin, Ronald, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa (2005a), Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336, 89–124.

Fagin, Ronald, Phokion G. Kolaitis, Alan Nash, and Lucian Popa (2008a), Towards a theory of schema-mapping optimization. In *Proc. of the 27th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2008)*, 33–42.

Fagin, Ronald, Phokion G. Kolaitis, and Lucian Popa (2005b), Data exchange: Getting to the core. *ACM Trans. on Database Systems*, 30, 174–210.

Fagin, Ronald, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan (2005c), Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. on Database Systems*, 30, 994–1055.

Fagin, Ronald, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan (2008b), Quasi-inverses of schema mappings. *ACM Trans. on Database Systems*, 33, 1–52.

Fagin, Ronald, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan (2009b), Reverse data exchange: Coping with nulls. In *Proc. of the 28th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2009)*, 23–32.

Fuxman, A., P. G. Kolaitis, R. Miller, and W. C. Tan (2005), Peer data exchange. In *Proc. of the 24rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2005)*, 160–171.

Glavic, Boris, Gustavo Alonso, Renée J. Miller, and Laura M. Haas (2010), TRAMP: Understanding the behavior of schema mappings through provenance. *PVLDB*, 3, 1314–1325.

Gonzalez, Luis Miguel Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik A. Lindner (2009), A break in the clouds: towards a cloud definition. *Computer Communication Review*, 39, 50–55.

Görlitz, Olaf and Steffen Staab (2011), Federated data management and query optimization for linked open data. In *New Directions in Web Data Management 1*, 109–137, Springer.

Gottlob, Georg, Reinhard Pichler, and Vadim Savenkov (2009), Normalization and optimization of schema mappings. *Proc. of the VLDB Endowment*, 2, 1102–1113.

Grandi, Fabio (2010), T-SPARQL: a TSQL2-like temporal query language for RDF. In *Proc. of the ADBIS 2010 Int. Workshop on Querying Graph Structured Data (GraphQ 2010)*, 21–30.

Grant, John, Jarek Gryz, Jack Minker, and Louiqa Raschid (1997), Semantic query optimization for object databases. In *Proc. of the 13th IEEE Int. Conf. on Data Engineering (ICDE'97)*, 444–453.

Gries, Oliver, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel (2010), A probabilistic abduction engine for media interpretation based on ontologies. In *Proc. of the 4th Int. Conf. on Web Reasoning and Rule Systems (RR 2010)* (J. Alferes, P. Hitzler, and Th. Lukasiewicz, eds.).

Halevy, Alon Y. (2001), Answering queries using views: A survey. *Very Large Database J.*, 10, 270–294.

Halevy, Alon Y., Anand Rajaraman, and Joann Ordille (2006), Data integration: The teenage years. In *Proc. of the 32nd Int. Conf. on Very Large Data Bases (VLDB 2006)*, 9–16.

Heim, Philipp and Jürgen Ziegler (2011), Faceted visual exploration of semantic data. In *Proceedings of the Second IFIP WG 13.7 conference on Human-computer interaction and visualization*, HCIV'09, 58–75, Springer-Verlag, Berlin, Heidelberg, URL `http://dl.acm.org/citation.cfm?id=1987029.1987035`.

Heinz, Christoph, Jürgen Krämer, Tobias Riemenschneider, and Bernhard Seeger (2008), Toward simulation-based optimization in data stream management systems. In *Proc. of the 24th Int. Conf. on Data Engineering (ICDE 2008)*, 1580–1583.

Jensen, C.S., M.D. Soo, and R.T. Snodgrass (1993), Unification of temporal data models. In *Proceedings of IEEE International Conference on Data Engineering*, 262–271.

Jiménez-Ruiz, Ernesto, Bernardo Cuenca Grau, Ian Horrocks, and Rafael Berlanga Llavori (2009), Logic-based ontology integration using ContentMap. In *Proc. of XIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2009)* (Antonio Vallecillo and Goiuria Sagardui, eds.), 316–319, URL `download/2009/JCHB09c.pdf`.

Kikot, Stanislav, Roman Kontchakov, and Michael Zakharyaschev (2011), On (in)tractability of OBDA with OWL 2 QL. In *Proc. of the 24th Int. Workshop on Description Logic (DL 2011)*.

Kllapi, Herald, Eva Sitaridi, Manolis M. Tsangaris, and Yannis E. Ioannidis (2011), Schedule optimization for data processing flows on the cloud. In *Proc. of SIGMOD*, 289–300.

Knublauch, Holger, Matthew Horridge, Mark A. Musen, Alan L. Rector, Robert Stevens, Nick Drummond, Phillip W. Lord, Natalya Fridman Noy, Julian Seidenberg, and Hai Wang (2005), The Protégé OWL experience. In *Proc. of the OWL: Experience and Directions Workshop*, volume 188 of *CEUR (http://ceur-ws.org/)*.

Kockskämper, S., B. Neumann, and M. Schick (1994), Extending process monitoring by event recognition. In *Proc. of the 2nd Int. Conf. on Intelligent System Engineering (ISE'94)*, 455–460.

Kolaitis, Phokion G. (2005), Schema mappings, data exchange, and metadata management. In *Proc. of the 24rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2005)*, 61–75.

Kossmann, Donald (2000), The state of the art in distributed query processing. *ACM Computing Surveys*, 32, 422–469.

Krämer, Jürgen and Bernhard Seeger (2009), Semantics and implementation of continuous sliding window queries over data streams. *ACM Trans. on Database Systems*, 34.

Krämer, Jürgen, Yin Yang, Michael Cammert, Bernhard Seeger, and Dimitris Papadias (2006), Dynamic plan migration for snapshot-equivalent continuous queries in data stream systems. In *Proc. of EDBT 2006 Workshops*, volume 4254 of *Lecture Notes in Computer Science*, 497–516, Springer.

Kuper, Gabriel M., Leonid Libkin, and Jan Paredaens, eds. (2000), *Constraint Databases*. Springer.

Law, Yan-Nei, Haixun Wang, and Carlo Zaniolo (2004), Query languages and data models for database sequences and data streams. In *Proc. of the 30th Int. Conf. on Very Large Data Bases*, 492–503, VLDB Endowment, URL `http://dl.acm.org/citation.cfm?id=1316689.1316733`.

Libkin, Leonid and Cristina Sirangelo (2008), Data exchange and schema mappings in open and closed worlds. In *Proc. of the 27th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2008)*, 139–148.

Lim, Soon Chong Johnson, Ying Liu, and Wing Bun Lee (2009), Faceted search and retrieval based on semantically annotated product family ontology. In *Proc. of the WSDM 2009 Workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR 2009)*, 15–24, URL `http://doi.acm.org/10.1145/1506250.1506254`.

Luther, Marko, Yusuke Fukazawa, Matthias Wagner, and Shoji Kurakake (2008), Situational reasoning for task-oriented mobile service recommendation. *Knowledge Engineering Review*, 23, 7–19, URL `http://dl.acm.org/citation.cfm?id=1362078.1362080`.

Madhavan, Jayant and Alon Y. Halevy (2003), Composing mappings among data sources. In *Proc. of the 29th Int. Conf. on Very Large Data Bases (VLDB 2003)*, 572–583.

Marchionini, Gary and Ryen White (2007), Find what you need, understand what you find. *International Journal Of Human-Computer Interaction*, 23, 205–237.

Möller, R. and B. Neumann (2008), Ontology-based reasoning techniques for multimedia interpretation and retrieval. In *Semantic Multimedia and Ontologies : Theory and Applications*, Springer.

Möller, Ralf, Volker Haarslev, and Michael Wessel (2006), On the scalability of description logic instance retrieval. In *29. Deutsche Jahrestagung für Künstliche Intelligenz* (C. Freksa and M. Kohlhase, eds.), Lecture Notes in Artificial Intelligence, Springer.

Motik, Boris (2010), Representing and querying validity time in RDF and OWL: A logic-based approach. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC 2010)*, volume 1, 550–565.

Neumann, B. and H.-J. Novak (1986), NAOS: Ein System zur natürlichsprachlichen Beschreibung zeitveränderlicher Szenen. *Informatik Forschung und Entwicklung*, 1, 83–92.

Neumann, Bernd (1985), Retrieving events from geometrical descriptions of time-varying scenes. In *Foundations of Knowledge Base Management – Contributions from Logic, Databases, and Artificial Intelligence* (J.W. Schmidt and Costantino Thanos, eds.), 443, Springer Verlag.

Neumann, Bernd and Hans-Joachim Novak (1983), Event models for recognition and natural language description of events in real-world image sequences. In *Proc. of the 8th Int. Joint Conference on Artificial Intelligence (IJCAI'83)*, 724–726.

Nowlan, W. A., A. L. Rector, S. Kay, C. A. Goble, B. Horan, T. J. Howkins, and A. Wilson (1990), PEN&PAD: A doctors' workstation with intelligent data entry and summaries. In *Proceedings of the 14th Annual Symposium on Computer Applications in Medical Care (SCAMC'90)* (R. A. Miller, ed.), 941–942, IEEE Computer Society Press, Los Alamitos, California.

Özsu, M. Tamer and Patrick Valduriez (1999), *Principles of Distributed Database Systems*, 2 edition. Prentice-Hall.

Pan, Jeff Z. and Edward Thomas (2007), Approximating OWL-DL ontologies. In *Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI-07)*, 1434—1439.

Peraldi, Irma Sofia Espinosa, Atila Kaya, and Ralf Möller (2011), Logical formalization of multimedia interpretation. In *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, volume 6050 of *Lecture Notes in Computer Science*, 110–133, Springer.

Pérez-Urbina, Héctor, Boris Motik, and Ian Horrocks (2008), Rewriting conjunctive queries over description logic knowledge bases. In *Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)* (Klaus-Dieter Schewe and Bernhard Thalheim, eds.), volume 4925 of *Lecture Notes in Computer Science*, 199–214, Springer.

Pérez-Urbina, Héctor, Boris Motik, and Ian Horrocks (2010), Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic*, 8, 186–209.

Phuoc, Danh Le, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth (2011), A native and adaptive approach for unified processing of linked streams and linked data. In *Proc. of the 10th Int. Semantic Web Conf. (ISWC 2011)*, volume 1, 370–388.

Poggi, Antonella, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati (2008), Linking data to ontologies. *J. on Data Semantics*, X, 133–173.

Popov, Igor O., M. C. Schraefel, Wendy Hall, and Nigel Shadbolt (2011), Connecting the dots: a multi-pivot approach to data exploration. In *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, 553–568, Springer-Verlag, Berlin, Heidelberg, URL `http://dl.acm.org/citation.cfm?id=2063016.2063052`.

Ren, Yuan and Jeff Z. Pan (2011), Optimising ontology stream reasoning with truth maintenance system. In *Proc. of the ACM Conference on Information and Knowledge Management (CIKM 2011)*.

Rodríguez-Muro, Mariano (2010), *Tools and Techniques for Ontology Based Data Access in Lightweight Description Logics*. Ph.D. thesis, KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano.

Rodríguez-Muro, Mariano and Diego Calvanese (2011), Dependencies to optimize ontology based data access. In *Proc. of the 24th Int. Workshop on Description Logic (DL 2011)*, volume 745 of *CEUR (`http://ceur-ws.org/`)*.

Rodriguez-Muro, Mariano and Diego Calvanese (2012), Quest, an owl 2 ql reasoner for ontology-based data access. In *Proc. of the 9th Int. Workshop on OWL: Experiences and Directions (OWLED 2012)*, volume 849 of *CEUR Electronic Workshop Proceedings, http://ceur-ws.org/*.

Rosati, Riccardo (2012), Prexto: Query rewriting under extensional constraints in dl - lite. In *Proc. of the 9th Extended Semantic Web Conference (ESWC 2012)*, volume 7295 of *LNCS*, 360–374, Springer.

Rosati, Riccardo and Alessandro Almatelli (2010), Improving query answering over *DL-Lite* ontologies. In *Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*, 290–300.

Savo, Domenico Fabio, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodríguez-Muro, Vittorio Romagnoli, Marco Ruzzi, and Gabriele Stella (2010), Mastro at work: Experiences on ontology-based data access. In *Proc. of the 23rd Int. Workshop on Description Logic (DL 2010)*, volume 573 of *CEUR (`http://ceur-ws.org/`)*, 20–31.

Schmidt, Michael, Michael Meier, and Georg Lausen (2010), Foundations of sparql query optimization. In *ICDT*, 4–33.

Schmiegelt, Philip and Bernhard Seeger (2010), Querying the future of spatio-temporal objects. In *Proc. of the 18th SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems (GIS 2010)*, 486–489, ACM, URL `http://doi.acm.org/10.1145/1869790.1869868`.

Schneiderman, B. (1983), Direct manipulation: A step beyond programming languages. *Computer*, 16, 57 –69.

Schwarte, Andreas, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt (2011), Fedx: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference*, 601–616.

Sheth, Amit P. (1991), Federated database systems for managing distributed, heterogeneous, and autonomous databases. In *VLDB*, 489.

Shvaiko, Pavel and Jérôme Euzenat (2005), A survey of schema-based matching approaches. *J. on Data Semantics*, IV, 146–171.

Soylu, Ahmet, Felix Modritscher, and Patrick De Causmaecker (2012), Ubiquitous web navigation through harvesting embedded semantic data: A mobile scenario. *Integrated Computer-Aided Engineering*, 19, 93–109.

Steiner, Andreas (1997), *A Generalisation Approach to Temporal Data Models and their Implementations*. Ph.D. thesis, Departement Informatik, ETH Zurich, Switzerland.

Suominen, Osma, Kim Viljanen, and Eero Hyvänen (2007), User-centric faceted search for semantic portals. In *Proc. of the 4th European Semantic Web Conf. (ESWC 2007)*, 356–370, URL `http://dx.doi.org/10.1007/978-3-540-72667-8_26`.

Tang, Yong, Lu Liang, Rushou Huang, and Yang Yu (2003), Bitemporal extensions to non-temporal rdbms in distributed environment. In *Proc. of the 8th Int. Conf. on Computer Supported Cooperative Work in Design*.

ten Cate, Balder and Phokion G. Kolaitis (2009), Structural characterizations of schema-mapping languages. In *Proc. of the 12th Int. Conf. on Database Theory (ICDT 2009)*, 63–72.

ter Hofstede, A. H. M., H. A. Proper, and Th. P. van der Weide (1996), Query formulation as an information retrieval problem. *The Computer Journal*, 39, 255–274, URL `http://comjnl.oxfordjournals.org/content/39/4/255.abstract`.

Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy (2010), Hive - a petabyte scale data warehouse using Hadoop. In *Proc. of the 26th IEEE Int. Conf. on Data Engineering (ICDE 2010)*, 996–1005.

Tran, Thanh, Daniel M. Herzig, and Gnter Ladwig (2011), Semsearchpro – using semantics throughout the search process. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9, 349 – 364, URL `http://www.sciencedirect.com/science/article/pii/S1570826811000758`. JWS special issue on Semantic Search.

Tsangaris, Manolis M., George Kakaletris, Herald Kllapi, Giorgos Papanikos, Fragkiskos Pentaris, Paul Polydoras, Eva Sitaridi, Vassilis Stoumpos, and Yannis E. Ioannidis (2009), Dataflow processing and optimization on grid and cloud infrastructures. *IEEE Data Eng. Bull.*, 32, 67–74.

Ullman, Jeffrey D. (1997), Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, 19–40, Springer.

Uren, Victoria, Yuangui Lei, Vanessa Lopez, Haiming Liu, Enrico Motta, and Marina Giordanino (2007), The usability of semantic search tools: A review. *Knowl. Eng. Rev.*, 22, 361–377, URL `http://dx.doi.org/10.1017/S0269888907001233`.

Wessel, M., M. Luther, and R. Möller (2009), What happened to Bob? Semantic data mining of context histories. In *Proc. of the 2009 Int. Workshop on Description Logics (DL 2009)*. CEUR Workshop Proceedings, Vol. 477.

Wessel, Michael, Marko Luther, and Matthias Wagner (2007), The difference a day makes - Recognizing important events in daily context logs. In *Proc. of the Int. Workshop on Contexts and Ontologies: Representation and Reasoning*. CEUR Workshop Proceedings Vol. 298.

Zaniolo, C., S. Ceri, Chr. Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, and R. Zicari (1997), *Advanced Database Systems*, chapter Overview of Temporal Databases. Morgan Kaufmann.

Zhang, Donghui, Alexander Markowetz, Vassilis J. Tsotras, Dimitrios Gunopulos, and Bernhard Seeger (2001), Efficient computation of temporal aggregates with range predicates. In *Proc. of the 25th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2006)*.