Optique: Towards OBDA Systems for Industry

Evgeny Kharlamov[1], Ernesto Jiménez-Ruiz[1], Dmitriy Zheleznyakov[1], Dimitris Bilidas[6], Martin Giese[2], Peter Haase[5], Ian Horrocks[1], Herald Kllapi[6], Manolis Koubarakis[6], Özgür Özçep[4], Mariano Rodríguez-Muro[3], Riccardo Rosati[7], Michael Schmidt[5], Rudolf Schlatte[2], Ahmet Soylu[2], and Arild Waaler[2]

[1] University of Oxford, UK `fname.lname@cs.ox.ac.uk`
[2] University of Oslo, Norway `{martingi, rudi, ahmets, arild}@ifi.uio.no`
[3] Free University of Bozen-Bolzano, Italy `rodriguez@inf.unibz.it`
[4] Hamburg University of Technology, Germany `oezguer.oezcep@tu-harburg.de`
[5] fluid Operations AG, Walldorf, Germany `fname.lame@fluidops.com`
[6] University of Athens, Greece `{d.bilidas, herald, koubarak}@di.uoa.gr`
[7] Sapienza Università di Roma, Italy `lname@dis.uniroma1.it`

Abstract. The recently started EU FP7-funded project Optique will develop an end-to-end OBDA system providing scalable end-user access to industrial Big Data stores. This paper presents an initial architectural specification for the Optique system along with the individual system components.

Keywords: OBDA, ontologies, OWL 2, Big Data, System Architecture

1 Introduction

A typical problem that end-users face when dealing with Big Data is that of data access, which arises due to the three dimensions of Big Data: *volume*, since massive amounts of data have been accumulated over the decades, *velocity*, since the amounts may be rapidly increasing, and *variety*, since the data are spread over different formats. In this context accessing the *relevant* information is an increasingly difficult task. The Optique project[8] [1] on 'Scalable End-user Access to Big Data' advocates a next generation of the well known *Ontology-Based Data Access* (OBDA) approach to address the Big Data dimensions and in particular the data access problem.

The project is focused around two demanding use cases that provide it with motivation, guidance, and realistic evaluation settings. The first use case is provided by Siemens (`http://www.siemens.com`) and encompasses several terabytes of temporal data coming from sensors, with a growth rate of about 30 gigabytes per day. Users need to query this data in combination with many gigabytes of other relational data that describe events. The second use case is provided by Statoil (`http://www.statoil.com`) and concerns more than one petabyte of geological data. The data are stored in multiple databases which have different schemata, and—for the time being—the users have to manually combine information from many databases, including temporal, in order to get the results for a single query.

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

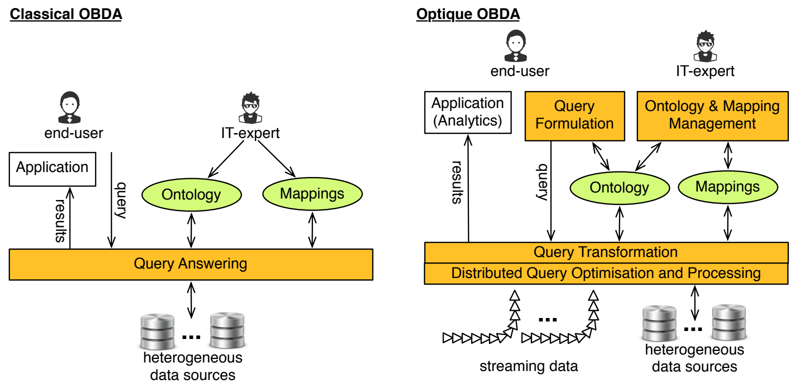

Fig. 1. Left: classical OBDA approach. Right: the Optique OBDA system

Accessing the *relevant* data in this context is becoming increasingly difficult for end-users. E.g., in large enterprises, such as Statoil, end-users work with applications that allow accessing data through a limited set of predefined queries. If an end-user needs data that these predefined queries do not provide, then the help of IT-experts (e.g., database managers) is required to translate the information need of end-users to specialised queries and optimise them for efficient execution. This process is the main bottleneck in the Optique use cases since it may require several iterations and may take several days. In particular in the oil and gas industry, IT-experts spend 30–70% of their time gathering and assessing the quality of data [2]. This is clearly very expensive in terms of both time and money.

The Optique has the potential of reducing the cost of data access dramatically, by automating the process of going from an information requirement to the retrieval of the relevant data, and to reduce the time needed for this process from days to hours, or even to minutes. A bigger goal of the project is to provide a platform with a generic architecture that can be adapted to any domain that requires scalable data access and efficient query execution.

The key to this automated translation is the "Ontology-Based Data Access" (OBDA) [3,4] approach The idea is to use an ontology, which presents to users a semantically rich conceptual model of the problem domain. The users formulate their information requirements (that is, queries) in terms of the ontology, and then receive the answers in the same intelligible form. These requests should be executed over the data automatically, without an IT-expert's intervention. To this end, a set of mappings is maintained which describes the relationship between the terms in the ontology and the corresponding terminology in the data source specifications, e.g., table and column names in relational database schemas.

State-of-the-art OBDA systems that are based on classical OBDA architecture (see Figure 1), however, have shown among others the following limitations.

1. The *usability* of OBDA systems regarding the user interface is still an open issue. Even if the vocabulary provided by the ontology is familiar to end-users, the user may find difficult to formulate complex queries when several concepts and relationships are involved.

2. OBDA systems critically depend on a *suitable ontology* and the corresponding *set of mappings*, which are in practice expensive to obtain. Even if we assume that the ontology and mappings are given, they are not static artefacts and should evolve according to the new end-users' information requirements.
3. Treatment of query answering is usually limited to query rewriting and there is little support of *distributed* query optimisation and processing in OBDA systems. Moreover, even in the scenarios without data distribution current state of the art implementations of rewriting-based query answering have shown serious limitations in *scalability* [3].
4. *Temporal* data, *streaming*, e.g., sensor, data, and corresponding *analytical tools* are generally ignored by OBDA systems, which seriously limits their applicability in enterprises such as Siemens where one has to deal with large amounts of streaming data from many turbines and diagnostic centres, in combination with historical, that is, temporal relational data sources.

In the Optique project, we aim at developing a next generation OBDA system (cf. Figure 1, right) that overcomes these limitations. More precisely, the project aims at a cost-effective approach that requires to revise existing OBDA components and develop new ones, in particular, to support *(i)* novel ontology and mapping management components, *(ii)* user-friendly query formulation interface(s), *(iii)* automated query translation, *(iv)* distributed query optimisation and execution exploiting private and public cloud resources for scale-out, and *(v)* rigorous treatment of temporal and streaming data.

In this work we present an initial specification of the architecture of the Optique system describing the individual system components, their interplay and interfaces, and establishes agreement on system-wide conventions and standards. This architecture will serve as a guideline for the modules and components developed in the technical work packages, and will evolve according to new requirements. The final architecture remains for future work.

2 Optique Architecture

This section presents the designed Optique OBDA solution [5] which aims at overcoming the limitations of current OBDA systems. Figure 2 shows an overview of the Optique's solution architecture and its components. The architecture is developed using the three-tier approach and has three layers:

- The *presentation layer* consists of four main user interfaces, mainly Web based: *(i)* to configure and log-in into the system, *(ii)* to compose queries, *(iii)* to visualise answers to queries, and *(iv)* to maintain the system by managing ontologies and mappings. The first three interfaces are for both end-users and IT-experts, while the fourth one is meant for IT-experts only.
- The *application layer* consists of several main components of the Optique's system, supports its machinery, and provides the following functionality: *(i) query formulation*, *(ii) ontology and mapping management*, *(iii) query answering*, and *(iv) processing and analytics of streaming and temporal data.*

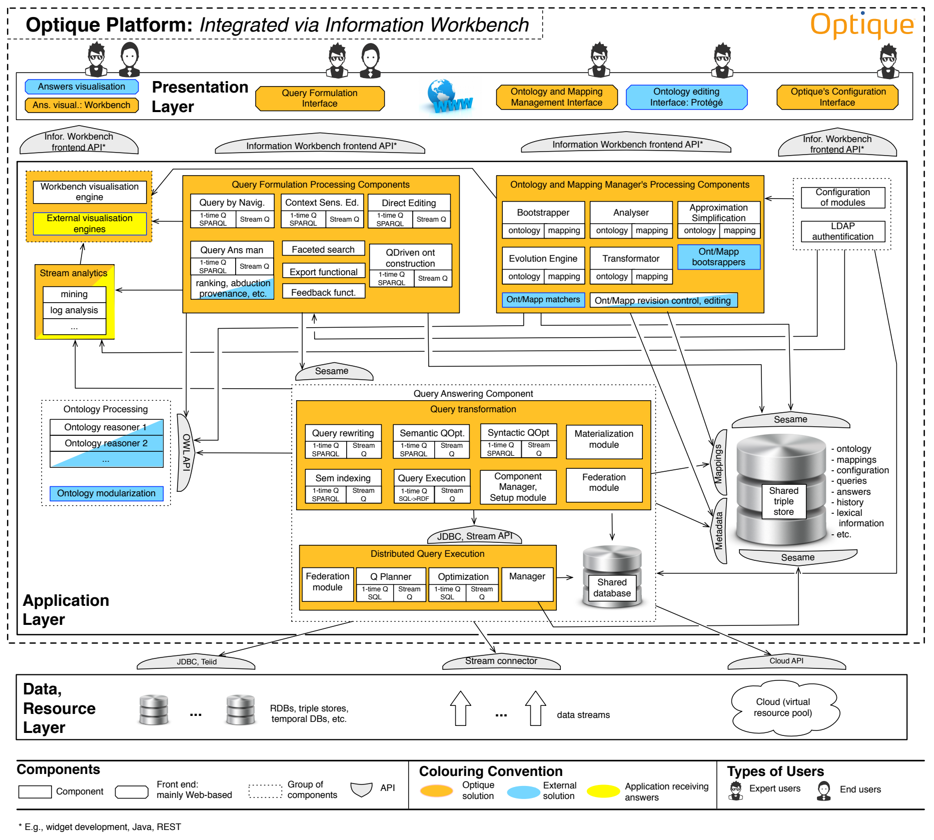

Fig. 2. The general architecture of the Optique OBDA system

Additionally, the Optique system will include an *LDAP authentication* module, to assign different roles to Optique users, and a *Configuration module*, to provide a custom initial set-up to the Optique components.

– The *data and resource layer* consists of the data sources that the system provides access to, i.e., relational, semistructured, temporal databases and data streams. It also includes capabilities to exploit virtual resources such as storage and compute infrastructure available through cloud offerings.

The entire Optique system will be integrated via the Information Workbench (IWB) platform[9] [6]. The IWB is a generic platform for semantic data management, which provides a *shared triple store* for managing the OBDA system assets (such as ontologies, mappings, query logs, (excerpts of) query answers, database metadata, etc.), generic interfaces and APIs for semantic data management (e.g. *ontology processing* APIs). Moreover, the IWB provides general functionality for user management, fine-grained access control and configuration file management.

In addition to these backend data management capabilities, the Information Workbench comes with a flexible user interface that will be used for implement-

[9] `www.fluidops.com/information-workbench/`

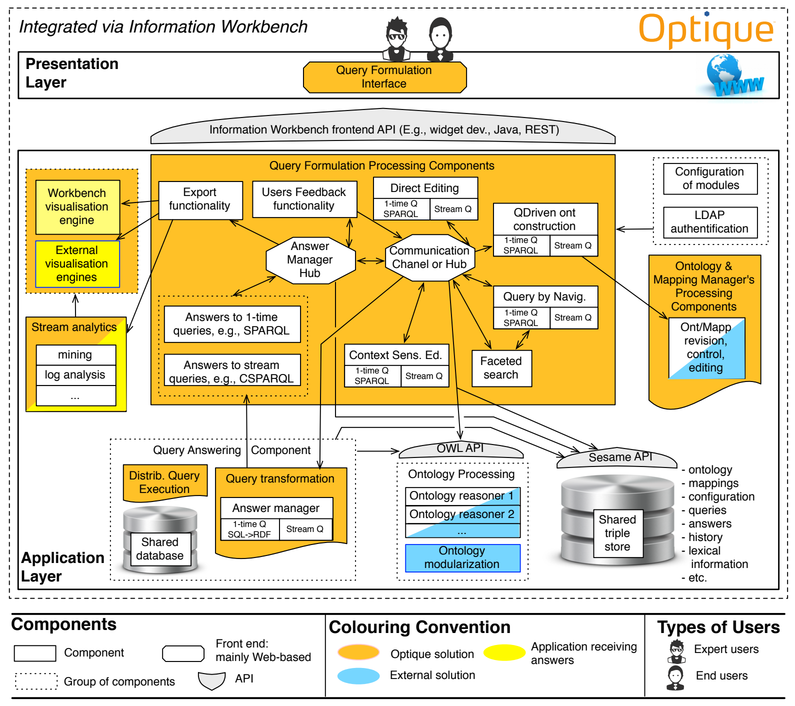

Fig. 3. Query Formulation components of the Optique OBDA system

ing the query formulation components. The user interface follows a semantic wiki approach, based on a rich, extensible pool of widgets for visualization, interaction, mashup, and collaboration, which can be flexibly integrated into semantic wiki pages, allowing developers to compose comprehensive, actionable user interfaces without any programming efforts.

In the following sections we describe in detail the main four application layer components together with their respective sub-architectures.

2.1 Query Formulation Component

The query formulation component aims at providing a user-friendly interface for non-technical users combining multiple representation paradigms [7]. We intend to design and implement novel techniques to exploit the semantics of the underlying ontology in order to formulate both complex and valid queries, as well as to use already existing techniques, e.g., [8,9]. Furthermore, this component will also integrate a query-driven ontology extension subcomponent to insert new end-users' information requirements in the ontology. Figure 3 shows the main query formulation subcomponents for the Optique OBDA solution and their interaction with other components of the system. Next we give a brief overview of each of them. Note that many subcomponents deal with both one-time queries, e.g., SPARQL queries, and continuous queries, e.g., CSPARQL queries.

1. *Editing components.* Different users may cooperate on the same query or

set of queries, thus, the Optique solution aims at providing (at least) three kinds of interfaces to formulate the query: *direct editing*, *context sensitive editing*, and *query by navigation* exploiting *faceted search* and other navigation paradigms. IT-experts may prefer the direct editing of the query using a formal language (e.g. SPARQL), while other users will be provided with a less technical interface, e.g., query by navigation. Additionally, direct editing should also allow the possibility of exploiting the ontology, and provide context sensitive completion. All interfaces should provide views on the partially constructed query, and users should be able to switch between views at will.

2. *Query-driven ontology construction component.* The ontology may not include all vocabulary needed by end-users. Moreover, the vocabulary is to a certain extent specific to individuals, projects, departments, etc., and subject to change. We consider it crucial to keep the ontology up-to-date w.r.t. end-user needs. Thus, the Query-driven ontology construction component will deal with four different changing scenarios driven by end-user requirements:
   (a) Adding new synonyms. Concept synonyms (e.g. annotation labels) do not represent new logical extension of the ontology, and hence end-users will be able to add them to the ontology with no (logical) harm. For example, the concept $\mathsf{WellBore}$ can be extended with the labels "drill hole" or "borehole". To avoid an overloading of the ontology with synonyms, we advocate a separation between the ontology (e.g. logical axioms) and the terminological information (e.g. synonyms, descriptions, related terms, etc.) as proposed in [10].
   (b) Adding basic extensions. End-user queries may also require basic extension of the ontology hierarchy, such as adding a new concept, say, $\mathsf{GeologicalWellBore}$, under $\mathsf{WellBore}$ (i.e. $\mathsf{GeologicalWellBore} \sqsubseteq \mathsf{WellBore}$). These types of additions can be considered *safe* [11] since they represent a *conservative extension* of the ontology. However, other additions to the ontology may require further analysis by the IT-expert if they are not conservative extensions (e.g. reclassifying the concept $\mathsf{WellBore}$ under the new concept $\mathsf{PlannedSideTrack}$).
   (c) "On the fly" extensions. This represents the more challenging scenario where we intend to exploit ontology learning techniques in order to mine formulated queries and identify new relevant concepts and relations (e.g., [12,13]). Ontology alignment techniques (e.g. LogMap [14]) will also be required in order to relate the new vocabulary to the existing ontology concepts.
   (d) IT-expert assistance. In the cases where the manual or on-the-fly extensions are insufficient, the assistance of the IT-expert will be required to extend the ontology accordingly.
3. *The Answer Manager component.* This component should deal with the (basic) visualization of the query results and their transformation into the required output formats (e.g. input formats of external tools).
4. *The User Feedback component.* This component is intended to allow the user to semi-automatically refine a query if the (partially) obtained results are not the expected ones. Furthermore, similar or related queries to the partially constructed query will also be suggested in order to help end-users in the refinement.

This component interacts with other components of the Optique OBDA system:

1. *The Ontology Revision Control system.* Different versions of the ontology may exist concurrently (e.g. extensions driven by different formulated queries or query requirements). These versions will be managed by the IT-experts through a revision control system (from the Ontology and Mapping management system) in order to detect logical defects (e.g. unsatisfiabilities), logical conflicts among versions as in [15], and OWL 2 profile violations (e.g. a new version is outside the OWL 2 QL profile).
2. *The Ontology Processing component.* The ontology will be a key element for the query formulation component and thus, the ontology processing component (e.g. OWL API) will also play an important role. Furthermore, logic-based ontology modularization techniques [16] will also be exploited to achieve a good balance between overview and focus when dealing with large ontologies. The properties of such modules guarantee that the semantics of the concepts of interest is preserved while providing (in general) a much smaller fragment of the ontology.
3. *Shared triple store.* The Query formulation component accesses the Shared triple store where, among others, the ontology, query logs, (excerpts of) query answers and the lexical information are physically stored.
4. *The Query Answering component* will transform the formulated queries into executable and optimized queries with respect to the data sources (e.g. streaming data, relational databases).
5. *Stream Analytics.* By exporting answers to this component component one can do, e.g., data mining in answers for continuous queries.
6. The *Visualisation Engines* allow to visualise both queries and query answers.

2.2 Ontology and Mapping Management Component

The ontology and mapping management component [17] will provide tools and methodologies to *(i)* semi-automatically bootstrap an initial ontology and mappings and *(ii)* maintain the consistency between the evolving mappings and the evolving ontology.

In Figure 4 we present the *Ontology and Mapping Management* component (the O&M manager) of the Optique OBDA system. The O&M manager has a Web interface at the presentation layer which will also include the well known ontology editor Protégé for sophisticated extensions over the ontology. Functionalities of the O&M manager are intended for IT-experts rather than end-users. The manager includes five subcomponents:

1. The *O&M Bootstrapper.* OBDA systems crucially depend on the existence of suitable ontologies and mappings. Developing them from scratch is likely to be expensive and a practical OBDA system should support a (semi-)automatic bootstrapping of an initial ontology and set of mappings. Thus, the Optique system will be equipped with a *bootstrapper*, a routine that takes a set of database schemata and possibly instances over these schemata as an input, and returns an ontology and a set of mappings connecting the ontology entities to the elements of the input schemata.

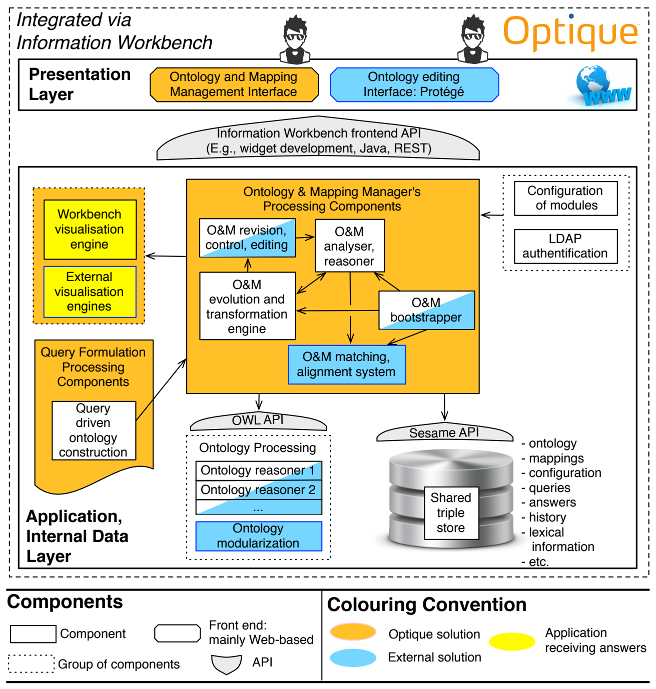
Integrated via Information Workbench
Optique
Presentation Layer
Ontology and Mapping Management Interface
Ontology editing Interface: Protégé
Information Workbench frontend API (E.g., widget development, Java, REST)
Workbench visualisation engine
External visualisation engines
Ontology & Mapping Manager's Processing Components
O&M revision, control, editing
O&M analyser, reasoner
O&M evolution and transformation engine
O&M bootstrapper
O&M matching, alignment system
Configuration of modules
LDAP authentification
Query Formulation Processing Components
Query driven ontology construction
OWL API
Ontology Processing
Ontology reasoner 1
Ontology reasoner 2
...
Ontology modularization
Sesame API
Shared triple store
- ontology
- mappings
- configuration
- queries
- answers
- history
- lexical information
- etc.
Application, Internal Data Layer
Components
Component
Front end: mainly Web-based
Group of components
API
Colouring Convention
Optique solution
External solution
Application receiving answers

Fig. 4. Ontology and Mapping Management component of the Optique OBDA system

2. The *O&M Matching and alignment system*. The bootstrapped ontologies will be aligned with domain ontologies using state of the art ontology alignment techniques[10] (e.g. LogMap [14]).
3. The *O&M Analyser and reasoner*. The ontologies and the mappings are not static objects and are subject to frequent changes. The O&M analyser will check ontologies and mappings for defects caused by these changes. We distinguish between logical and modelling defects. The three types of *logical* defects that are usually discussed in the literature (see, for example, [18,19]) are *inconsistency*, *unsatisfiability*, and *incoherency*. In OBDA scenarios *empty mappings* will also be an important logical defect. A mapping of an OBDA setting is *empty* if it does not propagate any individuals (resp. pairs of individuals) into any concept (resp. property) in the ontology. *Modelling defects* are less intuitive and less formally defined than logical ones. Typical modelling defects are *redundancy* [20] (e.g. redundant axioms, concepts, and mappings) and *unexpected entailments* (e.g. those entailments that do not correspond to the expected by domain experts [21]).
4. The *O&M Evolution and transformation engine*. The functionality of this component is twofold. On the one hand, it performs debugging on defects found by the analyser, and returns a debugged version of the ontology and the mappings. In the development of the Optique OBDA system we plan to exploit existing techniques from ontology evolution, e.g. [22,23]. On the

[10] `http://www.ontologymatching.org/`

other hand, it may perform several kinds of transformations of the ontology and the mappings: for instance, it should transform an input ontology which is in a language not supported by the OBDA systems (e.g., OWL 2) and return a version of the ontology into the supported language (e.g., OWL 2 QL profile). Also, it may perform transformations of the mappings related to formal properties of the mappings or to optimisation strategies [24]. This may involve changes in the syntax and/or semantics of the ontology.
5. The *O&M Revision, control and editing system* will support versioning and editorial processes for both ontologies and mappings. It will also act as a hub, coordinating interoperation between the analyser and evolution engine.

The O&M manager interacts with other components of the Optique system: 1. It accesses the *Shared triple store*, where the ontology and mappings are stored. It both reads and updates the ontology and mappings. 2. The analyser, alignment system, and evolution engine access to reasoning capabilities, e.g., ontology reasoners, ontology modularisation engines, etc.3. The *Query Formulation Component* can call the O&M manager when a user decides to extend the ontology. We refer to this as *query-driven ontology construction*.4. The O&M manager is connected to a *Visualisation engine* to visualise ontology and mappings.

2.3 Query Answering Component

Covering the backend functionalities (query planning, optimization, execution, and use of cloud resources), the query answering component is compound of two large subcomponents: *(i) query transformation* and *(ii) distributed query processing* components. The query transformation component is responsible for translating, also known as *rewriting*, queries received from the query formulation component, e.g., SPARQL queries, into an optimised executable code that is evaluated over the data sources and streams in the data layer. The Quest system [25] implements sophisticated rewriting and optimization techniques and is going to be the core part of the Optique's query transformation.

The distributed query processing subcomponent provides query planning and execution. It distributes queries to individual servers and uses massively parallelised (cloud) computing. The ADP [26] for complex dataflow processing in the cloud is going to be the core part of Optique's distributed query processing.

Query Transformation Component Figure 5 presents the architecture of the *Query transformation* (QT) component of the Optique system [27]. The QT component interacts with other components of the Optique OBDA system:

1. The *query formulation component* provides a query interface for end-users. This component receives queries from an end-user and send them to the QT component, e.g., via Sesame API.
2. The *configuration* module provides the configuration for the QT module that is required for query transformation performance.
3. The *ontology processing* module (a group of components such as ontology reasoners, modularisation systems, etc.) is called by the QT module to perform semantic optimisation.

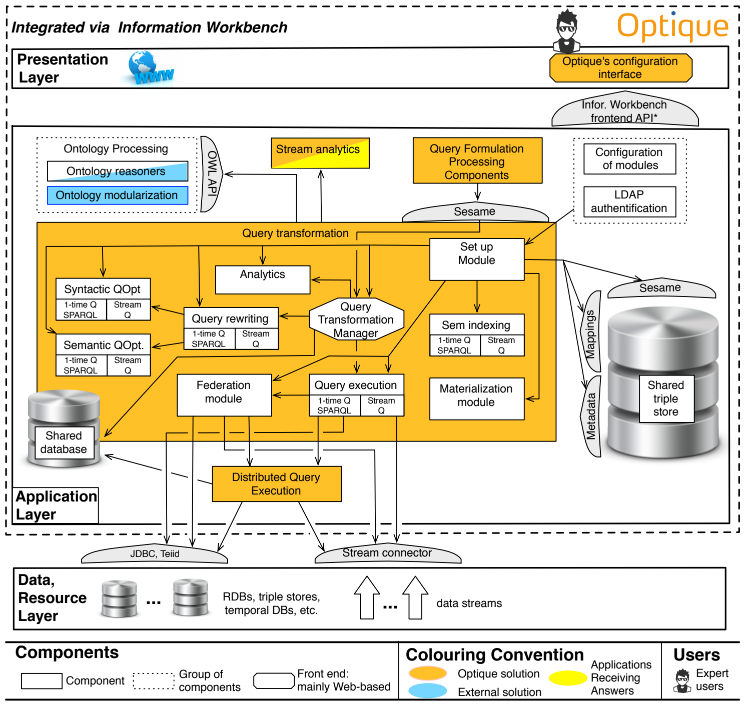

Fig. 5. Query transformation component of the Optique OBDA system

4. The *distributed query processing* component receives rewritten queries from the QT module and performs their evaluation over data sources.
5. The *(internal) shared database* contains the technical data required for data answering process such as semantic indices, answers to queries, etc. This database will be only shared by the QT component and the distributed query processing component.
6. The *shared triple store* contains the data that can be used by (the most of) the components of the Optique OBDA system. E.g., it contains the ontology, the mappings, the query history, etc.
7. The *stream analytics module* analyzes answers to the stream queries.
8. *Data sources* (RDBs, triple stores, data streams) can be also accessed by the QT module during the query execution.

The Query Transformation Manager (QTM) is the principal component of the QT component and will orchestrate the QT process. The QT process is triggered when a query is received from the query formulation component. Next we describe the role of each QT subcomponent and their interplay in the query transformation process. The process can be divided into several stages:

1. *Initialisation.* At this stage the *Configuration* module sends the configuration to the *Set-up module*, which in turn configure the other QT submodules. The initialisation includes several steps in which the input ontology and mappings

get analysed and optimised so as to allow the rewriting and optimisation algorithms to be fast, and the query evaluation over the data sources to be more efficient. The *semantic indexing* and *materialisation* modules are in charge of creation and maintenance of so-called semantic index.

2. *Query rewriting.* The QTM sends the (SPARQL) query Q received from the query formulation component to the *query rewriting* module, which is in charge of rewriting the query in the required format; for example, SQL if it is supposed to be evaluated over RDBs or Streaming SPARQL for querying data streams. Further, for the sake of simplicity, we will assume that the target query format is SQL. Along with the rewriting, this module optimises the rewritten query both *syntactically* and *semantically* as described below:
   – During the transformation process, the initial query (e.g., SPARQL) might be turned into a number of SQL queries $Q_1, \dots, Q_n$ such that their union is equal to Q. In the Syntactic and Semantic optimisation stages, these queries get optimised to improve the performance, e.g., some joints, conditions, filters within this SQL queries are deleted. In the former stage, the methods that are used to detect what parts of the queries have to be optimised are syntactical, that is, they are based only on the shape of a query and do not involve any reasoning. In the latter stage, the methods take into account semantic information such as query containment, integrity constraints of the data sources, etc.
3. *Query execution.* After rewriting and optimization, the queries $Q'_{i_1}, \dots, Q'_{i_m}$ are sent to the *query execution* module. This module decides which of these queries, if any, need to be evaluated using distributing query execution, and which can be evaluated by the standard query answering facilities. In the former case, the corresponding queries are sent to the *distributed query processing* component. In the latter case, the corresponding queries are evaluated directly over the *data sources* by standard means. If needed, some of the queries can be evaluated over a federated database system, by the *Federation module.*
4. *Query answer management.* After the query evaluation process, the answers to the queries that have been sent directly to the data sources are returned to the QTM module. The manager transforms them into the required format and sends them back to the query formulation component, which represents the answers to end-users. The queries that has been sent to the *distributed query processing* component do not necessarily go directly to the query answer manager, but to a *shared database.* The reason is that the answers can be up to several GBs, so sending them directly to the QTM may hang the system. The QTM receives the signal that the answers are in the shared database and metadata about the answers. Then, together with the *analytics module*, it decides how to proceed further. The answers to one-time queries, e.g. SQL queries over RDBs, eventually go to the query formulation component, while the answers to stream queries go to the *stream analytics module.*

Distributed Query Processing Component The Distributed query processing component [28] aims at achieving the efficiency in Big Data scenarios by both *massive parallelism*, i.e., running queries with the maximum amount of

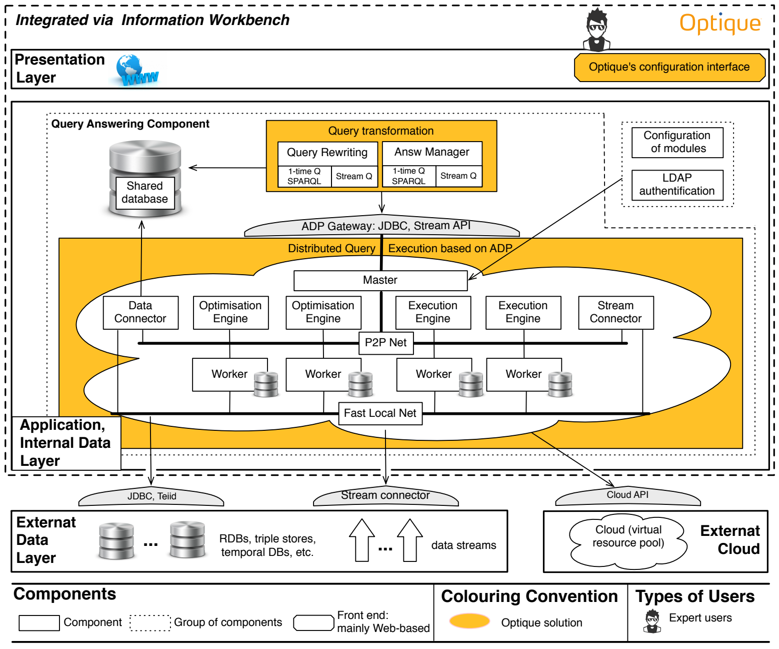

Fig. 6. General architecture of the ADP component within the Optique System

parallelism at each stage of execution, and *elasticity*, i.e., by allowing a flexibility to execute the same query with the use of resources that depends on the the resource availability for this particular query, and the execution time goals. The distributed query execution is based on the ADP (Athena Distributed Processing) [26] a system for complex dataflow processing in the cloud. ADP has been developed and used successfully in several European projects such as Health-e-Child [29].

The general architecture of the distributed query processing component within the Optique platform is shown in Figure 6. The query is received through the gateway using JDBC. This communication mainly involves interaction with the QT component. The *Master* node is responsible for initialization and coordination of the process. The *optimization engine* produces the execution plan for the query. Next, the execution plan is given to the *execution engine* which is responsible for reserving the necessary resources, sending the operators of the graph to the appropriate *workers* and monitor the execution. Next we describe in more detail the distribution process:

– *Language and Optimization:* A query Q, expressed in SQL, are issued to the system through the gateway. Q is transformed to a data flow language allowing complex graphs with operators as nodes and with edges representing producer-consumer relationships. The first level of optimization is planning, which results in an SQL query script. We enhanced SQL by adding the table partition as a first class citizen of the language; it is defined as a set of tuples having a particular property (e.g., the tuples in the same partition share the value of a hash function applied on one column). A table is defined as a set of partitions. The optimizer produces an execution plan in the form of a directed acyclic graph with all the information needed to execute Q.

- *Execution Engine:* ADP relies on an asynchronous execution engine. As soon as a worker node completes one job, it is sending a corresponding signal to the execution engine. The execution engine uses an asynchronous event based execution manager, which records the jobs that have been executed and assigns new jobs when all the prerequisite jobs have finished.
- *Worker Pool:* The resources needed to execute Q (machines, network, etc.) are allocated automatically. Those resources are wrapped into containers, which are used to abstract from the details of a physical machine in a cluster or a virtual machine in a cloud. Workers run Q using a python wrapper of SQLite (`http://www.sqlite.org`). This part of the system can also be used as a standalone single node DB (`https://code.google.com/p/madis/`). Queries are expressed in a declarative language which is an extension of SQL. This language facilitates considerably the use of user-defined functions (UDFs). The system supports row, aggregate, and virtual table functions.
- *Data/Stream Connector:* The Data and Stream Connectors (DC and SC) are responsible for handling and dispatching the relational and stream data through the network respectively. They are used when the system receives a request for collecting the results of executed queries. SC uses an asynchronous stream event listener to be notified of incoming stream data, whereas DC utilizes a table transfer scheduler to receive partitions of relational tables from the worker nodes.

2.4 Streaming and Temporal Data

Processing and Analytics of Streaming and Temporal Data is primarily motivated by the need of large industries. For example, Siemens encompasses several terabytes of temporal data coming from sensors, with an increase rate of about 30 gigabytes per day [30]. Addressing this challenge requires a number of techniques and tools which should be integrated in several modules of the Optique OBDA solution. For example, the query formulation module should support window queries and the query answering module should support rewriting and optimised execution of such queries. Additionally, the ontology and mapping management component is also required to design appropriate formalisms to support ontological modelling of streaming and temporal data.

The query language that the system should provide to end-users should combine *(i) temporal operators*, that address the time dimension of data and allow to retrieve data which was true “always” in the past or “sometimes” in the last X months, etc., *(ii) time series analysis operators*, such as mean, variance, confidence intervals, standard deviation, as well as trends, regression, correlation, etc., and *(iii) stream oriented operators*, such as sliding windows.

Given a query, mapping languages, and ontology, the Optique system should be able to translate queries into highly optimised executable code over the underlying temporal and streaming data. This requires techniques for automated query translation continuous and temporal queries. Existing translation techniques are limited and they do not address query optimisation and distributed query processing. Thus, novel approaches should be developed.

3 Conclusions

The Optique system will provide an end-to-end OBDA solution for Big Data access addressing a number of important industry requirements. The technology and system will be developed in a close cooperation of six universities, two industrial partners, and two use cases: Statoil and Siemens. The system will be deployed and evaluated in our use cases. It will provide valuable insights for the application of semantic technologies to Big Data integration problems in industry.

Acknowledgements. We are thankful to D. Calvanese, T. Hubauer, M. Meier, R. Möller, M. Roshchin, and D. Fabio Savo for insightful discussions. The authors are supported by the EU project Optique (FP7-IP-318338).

References

1. Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Möller, R., Özçep, O., Rodriguez Muro, M., Rosati, R., Schlatte, R., Schmidt, M., Soylu, A., Waaler, A.: Scalable End-user Access to Big Data. In: Rajendra Akerkar: Big Data Computing. Florida : Chapman and Hall/CRC. To appear. (2013)
2. Crompton, J.: Keynote talk, the W3C Workshop on Semantic Web in Oil & Gas Industry: Houston, TX, USA, 9–10 December (2008) available from `http://www.w3.org/2008/12/ogws-slides/Crompton.pdf`.
3. Rodriguez-Muro, M., Calvanese, D.: High Performance Query Answering over DL-Lite Ontologies. In: Knowledge Representation and Reasoning (KR). (2012)
4. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web **2**(1) (2011) 43–53
5. Calvanese, D., Giese, M., Haase, P., Horrocks, I., Hubauer, T., Ioannidis, Y., Jiménez-Ruiz, E., Kharlamov, E., Kllapi, H., Klüwer, J., Koubarakis, M., Lamparter, S., Möller, R., Neuenstadt, C., Nordtveit, T., Özcep, O., Rodríguez-Muro, M., Roshchin, M., Ruzzi, M., Savo, F., Schmidt, M., Soylu, A., Waaler, A., Zheleznyakov, D.: The Optique Project: Towards OBDA Systems for Industry. In: OWLED. (2013)
6. Haase, P., Hütter, C., Schmidt, M., Schwarte, A.: The Information Workbench as a Self-Service Platform for Linked Data Applications. In: Proc. of WWW. (2012)
7. Cuenca Grau, B., Giese, M., Horrocks, I., Hubauer, T., Jiménez-Ruiz, E., Kharlamov, E., Schmidt, M., Soylu, A., Zheleznyakov, D.: Towards Query Formulation and Query-Driven Ontology Extensions in OBDA. In: OWLED. (2013)
8. Bechhofer, S., Horrocks, I.: Driving User Interfaces from FaCT. In: Proceedings of the 2000 International Workshop on Description Logics. (2000) 45–54
9. Catarci, T., Dongilli, P., Mascio, T.D., Franconi, E., Santucci, G., Tessaris, S.: An ontology based visual tool for query formulation support. In: European Conf. on Artif. Intell. (ECAI). (2004) 308–312
10. Jimeno-Yepes, A., Jiménez-Ruiz, E., Llavori, R.B., Rebholz-Schuhmann, D.: Reuse of terminological resources for efficient ontological engineering in life sciences. BMC Bioinformatics **10**(S-10) (2009)

11. Jiménez-Ruiz, E., Cuenca Grau, B., Sattler, U., Schneider, T., Berlanga, R.: Safe and economic re-use of ontologies: A logic-based methodology and tool support. In: European Sem. Web Conf. (ESWC). Volume 5021. (2008) 185–199
12. Zhang, J., Xiong, M., Yu, Y.: Mining query log to assist ontology learning from relational database. In: Frontiers of WWW Research and Development. (2006)
13. Kotis, K., Papasalouros, A., Maragoudakis, M.: Mining query-logs towards learning useful kick-off ontologies: an incentive to semantic web content creation. IJKEDM **1**(4) (2011)
14. Jiménez-Ruiz, E., Cuenca Grau, B.: LogMap: Logic-based and Scalable Ontology Matching. In: Int'l Sem. Web Conf. (ISWC). (2011) 273–288
15. Jiménez-Ruiz, E., Cuenca Grau, B., Horrocks, I., Berlanga, R.: Supporting concurrent ontology development: Framework, algorithms and tool. Data Knowl. Eng. **70**(1) (2011) 146–164
16. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. Artif. Intell. Res. **31** (2008) 273–318
17. Haase, P., Horrocks, I., Hovland, D., Hubauer, T., Jiménez-Ruiz, E., Kharlamov, E., Klüwer, J., Pinkel, C., Rosati, R., Santarelli, V., Soylu, A., Zheleznyakov, D.: Optique System: Towards Ontology and Mapping Management in OBDA Solutions. In: WoDOOM. (2013)
18. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.A.: Debugging unsatisfiable classes in OWL ontologies. J. Web Sem. **3**(4) (2005) 268–293
19. Shchekotykhin, K., Friedrich, G., Fleiss, P., Rodler, P.: Interactive Ontology Debugging: Two Query Strategies for Efficient Fault Localization. Web Semantics: Science, Services and Agents on the World Wide Web **12**(0) (2012)
20. Grimm, S., Wissmann, J.: Elimination of Redundancy in Ontologies. In: European Sem. Web Conf. (ESWC). (2011) 260–274
21. Horrocks, I.: Tool Support for Ontology Engineering. In: Foundations for the Web of Information and Services. (2011) 103–112
22. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of DL-Lite Knowledge Bases. In: International Semantic Web Conference (1). (2010) 112–128
23. Cuenca Grau, B., Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D.: Ontology Evolution Under Semantic Constraints. In: Knowledge Representation and Reasoning (KR). (2012)
24. Pinto, F.D., Lembo, D., Lenzerini, M., Mancini, R., Poggi, A., Rosati, R., Ruzzi, M., Savo, D.F.: Optimizing query rewriting in ontology-based data access. In: Int'l Conference on Extending Database Technology (EDBT). (2013) 561–572
25. Rodriguez-Muro, M., Calvanese, D.: Quest, an OWL 2 QL Reasoner for Ontology-based Data Access. In: OWLED. (2012)
26. Kllapi, H., Sitaridi, E., Tsangaris, M.M., Ioannidis, Y.E.: Schedule optimization for data processing flows on the cloud. In: Proc. of SIGMOD. (2011) 289–300
27. Calvanese, D., Horrocks, I., Jiménez-Ruiz, E., Kharlamov, E., Meier, M., Rodríguez-Muro, M., Zheleznyakov, D.: On Rewriting and Answering Queries in OBDA Systems for Big Data. In: OWLED. (2013)
28. Kllapi, H., Bilidas, D., Horrocks, I., Ioannidis, Y., Jiménez-Ruiz, E., Kharlamov, E., Koubarakis, M., Zheleznyakov, D.: Distributed Query Processing on the Cloud: the Optique Point of View. In: OWLED. (2013)
29. Health-e-Child: Integrated healthcare platform for european paediatrics (2006) `http://www.health-e-child.org/`.
30. Horrocks, I., Hubauer, T., Jiménez-Ruiz, E., Kharlamov, E., Koubarakis, M., Möller, R., Bereta, K., Neuenstadt, C., Özçep, Özgür., Roshchin, M., Smeros, P., Zheleznyakov, D.: Addressing Streaming and Historical Data in OBDA Systems: Optique's Approach. In: Know@LOD. (2013)