Optique System: Towards Ontology and Mapping Management in OBDA Solutions

Peter Haase[2], Ian Horrocks[3], Dag Hovland[6], Thomas Hubauer[5], Ernesto Jimenez-Ruiz[3], Evgeny Kharlamov[3], Johan Klüwer[1] Christoph Pinkel[2], Riccardo Rosati[4], Valerio Santarelli[4], Ahmet Soylu[6], Dmitriy Zheleznyakov[3]

[1] Det Norske Veritas, Norway
[2] fluid Operations AG, Germany
[3] Oxford University, UK
[4] Sapienza University of Rome, Italy
[5] Siemens Corporate Technology, Germany
[6] University of Oslo, Norway

Abstract. The Optique project aims at providing an end-to-end solution for scalable Ontology-Based Data Access to Big Data integration, where end-users will formulate queries based on a familiar conceptualization of the underlying domain, that is, over an ontology. From user queries the Optique platform will automatically generate appropriate queries over the underlying integrated data, optimize and execute them. The key components in the Optique platform are the ontology and mappings that provide the relationships between the ontology and the underlying data. In this paper we discuss the problem of bootstrapping and maintenance of ontologies and mappings. The important challenge in both tasks is debugging errors in ontologies and mappings. We will present examples of different kinds of error, and give our preliminary view on their debugging.

1 Introduction

A typical problem that end-users face when dealing with Big Data is the data access problem, which arises due to the three dimensions (the so called "3V") 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 a huge variety of formats and sources. In the context of Big Data, accessing the *relevant* information is an increasingly difficult problem. The Optique project[7] [5] aims at overcoming this 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,[8] 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

[7] http://www.optique-project.eu
[8] http://www.siemens.com

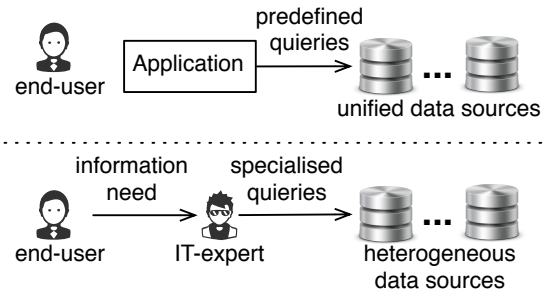

Fig. 1. Existing approaches to data access

use case is provided by Statoil[9], and concerns more than one petabyte of geological data. The data is stored in multiple databases which have different schemata, and the user has to manually combine information from many databases in order to get the results for a single query. In general, in the oil and gas industry, IT-experts spend 30–70% of their time gathering and assessing the quality of data [4]. This is clearly very expensive in terms of both time and money. The Optique project aims at solutions that reduce the cost of data access dramatically. More precisely, Optique aims at 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[10] with a generic architecture that can be easily adapted to any domain that requires scalable data access and efficient query execution for OBDA solutions.

The main bottleneck in the use cases discussed above is data access being limited to a restricted set of predefined queries (cf. Figure 1, top). Thus, if an end-user needs data that current applications cannot provide, the help of an IT-expert is required to translate the information need of the end-user to specialized queries and optimize them for efficient execution (cf. Figure 1, bottom). This process can take several days, and given the fact that in data-intensive industries engineers spend up to 80% of their time on data access problems [4] this incurs considerable cost.

The approach known as "Ontology-Based Data Access" (OBDA) [18,2] has the potential to address the data access problem by automating the translation process from the information needs of users to data queries (cf. Figure 2, left). The key idea is to use an ontology that presents to the user a conceptual model of the problem domain. The user formulates their information requirements (that is, queries) in terms of the ontology, and then receives 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 relationships between the terms in the ontology and the corresponding data source fields.

In complex domains, a complete specification of the ontology and the mappings will typically be expensive to obtain, suggesting to start from partial specifications that are incrementally refined and expanded according to users' needs. Moreover, in some

[9] `http://www.statoil.com`

[10] Optique's solutions are going to be integrated via the Information Workbench platform [8].

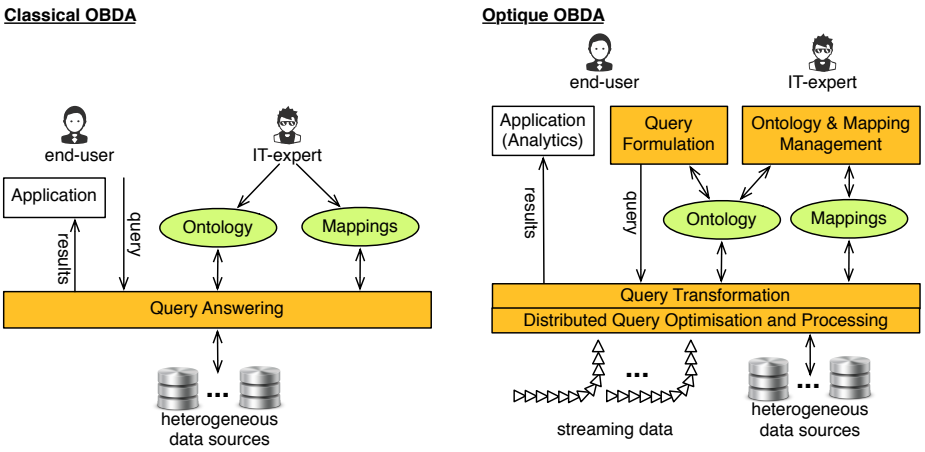

Fig. 2. Left: classical OBDA approach. Right: the Optique OBDA system

applications, changes in the ontology and/or in the schemata of the data sources (and thus in the mappings) are likely to happen. Thus, some means for bootstrapping and maintenance of ontology and mappings is required. The classical OBDA approaches fail to provide support for these tasks.

In the Optique project we aim at developing a next generation OBDA system (cf. Figure 2, right); more precisely, the project aims at a cost-effective approach that includes the development of tools and methodologies for semi-automatic bootstrapping of the system with a suitable initial ontology and mappings, and for updating them "on the fly" as needed by a given application. This means that, in our context, ontologies are dynamic entities that evolve *(i)* to incorporate new vocabulary required in users' queries, *(ii)* to accommodate new data sources, and *(iii)* to repair defects in ontologies and mappings. In all the cases, some way is needed to ensure that changes in the ontology and mappings are made in a coherent way. Due to this requirement, ontology *debugging* technologies will be a cornerstone of the system.

Besides ontology and mapping management, the Optique OBDA system will address a number of additional challenges, including: *(i)* user-friendly query formulation interface(s), *(ii)* processing and analytics over streaming data, *(iii)* automated query translation, and *(iv)* distributed query optimisation and execution in the Cloud. We will not, however, discuss these issues in this paper and refer the reader to [5] for details.

The remainder of the paper is organized as follows. First, we discuss different ontology defects that may arise and will need to be debugged (Section 2). Then we discuss how such defects can occur during the operation of the Optique OBDA system (Section 3), and how they will be dealt with.

2 OBDA Systems: Components and Defects

2.1 Components of OBDA Systems

An OBDA setting includes three central components: *data sources*, an *ontology*, and *mappings* (from data sources to the ontology).

Data sources. A data source consists of a data schema and a number of corresponding data instances. A typical example for a data source is a relational or semi-structured database.

Example 1. Consider the two data sources in Figure 3 (left): Source 1 (or S1 for short) contains a unary table about production wells with the schema $\mathbf{PWell}$ and says that the well 'w123' is a production well. Source 2 (or S2 for short) contains a unary table about exploration wells with the schema $\mathbf{EWell}$ and says that 'w123' is an exploration well. ■

Ontology. In the context of OBDA, it is usual to consider the ontology to be a Description Logic (DL) ontology [20] (or, equivalently, an OWL ontology). A DL ontology consists of a finite set of axioms that are usually in the form of set inclusions between two (possibly complexly defined) *concepts* that represent classes of objects. The ontology captures general knowledge about the domain of interest, such as generalizations, relational links, etc.

Example 2. Consider the ontology in Figure 3, left, (in the box). It describes (a part of) the oil production domain and consists of three concepts: *(i)* the concept $Well$ represents the class of *wellbores*,[11] *(ii)* the concept $PWell$ represents the class of *production wells*,[12] and *(iii)* the concept $EWell$ represents the class of *exploration wells*.[13] This ontology says that *production wells cannot be exploration wells* (denoted as $EWell \sqsubseteq \neg PWell$), *wells are exploration wells* ($Well \sqsubseteq EWell$), and *wells are production wells* ($Well \sqsubseteq PWell$). ■

Mappings. Mappings associate data from the data sources with concepts in the ontology.[14] A mapping m has the form

$$m : \mathbf{q}(\boldsymbol{x}) \rightsquigarrow N(\boldsymbol{x}),$$

where $\mathbf{q}$ is a query over the data sources, and N is an element of the ontological vocabulary, i.e., a concept or property name. Intuitively, $\mathbf{q}$ returns constants (resp., pairs of constants) from the data sources and *propagates* them into concept (resp., property) N (that is, *instantiates* N with them).

[11] A wellbore is any hole drilled for the purpose of exploration or extraction of natural resources, e.g., oil.

[12] *Production wells* are drilled primarily for producing oil or gas.

[13] *Exploration wells* are drilled purely for exploratory (information gathering) purposes in a new area.

[14] The mappings we are considering here are usually referred to as *global-as-view* mappings [13].

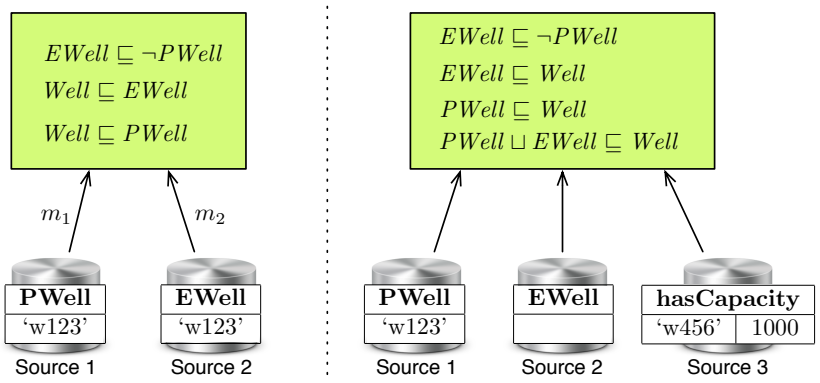

Fig. 3. Example of ontology and mapping defects. PWell stands for Production Well and EWell for Exploration Well.

Example 3. Consider Figure 3, left, again. The example includes two mappings, m_1 and m_2, which are depicted as arrows and defined as:

$$m_1 : \mathbf{PWell}(x) \rightsquigarrow PWell(x), \qquad m_2 : \mathbf{EWell}(x) \rightsquigarrow EWell(x).$$

Note that m_1 connects S1 with the concept $PWell$, while m_2 connects S2 with $EWell$: m_1 says that tuples from the table **PWell** are instances of the concept $PWell$, and m_2 analogously for $EWell$. I.e., m_1 instantiates $PWell$ and m_2 instantiates $EWell$. ■

Summing up Examples 1–3, we have the following OBDA setting depicted in Figure 3, left:

$$(\{S_1, S_2\}, \{EWell \sqsubseteq \neg PWell,\ Well \sqsubseteq EWell,\ Well \sqsubseteq PWell\}, \{m_1, m_2\}).$$

Another OBDA setting is illustrated in Figure 3, right, and we will comment on it in the following section.

In what follows, we present a number of defects that can occur in an OBDA setting and that may require debugging. We observe that several such defects have been individually pointed out and studied in literature (see, for instance, [17,19,16,11]). The Optique Project aims at facing these issues as a whole, and at individuating practical solutions for addressing them.

2.2 Logical and Modeling Defects

We distinguish two types of defects, namely logical and modeling.

The three types of logical defects that are usually discussed in the literature (see, for example, [22,15,21]) are *inconsistency*, *unsatisfiability*, and *incoherency*. Traditionally, these notions are applied to ontologies, while in the OBDA scenarios, as we will illustrate below, they may involve all three components of OBDA settings, that is, data sources, ontologies, and mappings.

Inconsistency of OBDA settings. An OBDA setting is *inconsistent* if it contains contradictory facts, so that there is no model of the ontology that is consistent with the data and the mappings.

Example 4. The OBDA setting in Figure 3, left, is inconsistent. Indeed, the well 'w123' is an instance of both concepts $PWell$ and $EWell$, and as the ontology asserts that $PWell$ and $EWell$ are disjoint, this makes the OBDA setting inconsistent. ■

The inconsistency in Example 4 could have been caused by a mistake in one of the data sources: either Source 1 or Source 2 provides wrong information about the well 'w123'. Perhaps, this well was an exploration one at the beginning, but then became a production one, but the information in Source 2 has not been brought up to date. A possible solution would be to update the offending data source. In Figure 3, right, there is an updated Source 2.

Unsatisfiability and Incoherency of Ontologies. This two types of defects are tightly related. A concept in an ontology is *unsatisfiable* if it cannot be instantiated without causing inconsistency, and an ontology is *incoherent* if an unsatisfiable concept occurs in it.

Example 5. Continuing with the OBDA setting in Figure 3, left, consider the concept $Well$. Observe that if we instantiate the concept with some object, say, 'w234', then it will turn out that 'w234' is both exploration and production well, which is inconsistent. Hence, the concept $Well$ is unsatisfiable and the ontology is incoherent. ■

Ontology incoherence is invariably indicative of an ontology design error. In Example 5, the two axioms stating that *wells are exploration wells* and *wells are productive wells* are in conflict with both our intuition and with the axiom stating that $PWell$ and $EWell$ are disjoint. Repairing this error will require one or more of these axioms to be modified or deleted. A possible repair plan is to replace the two counterintuitive axioms with axioms that make intuitive sense: $EWell \sqsubseteq Well$ (*exploration wells are wells*), and $PWell \sqsubseteq Well$ (*productive wells are wells*). Figure 3, right, contains a repaired version of the ontology.

Empty Mappings in OBDA settings. 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.

Example 6. Consider the OBDA setting in Figure 3, right. Besides the mappings m_1 and m_2 from Example 3, it includes the following mapping:

$$m_3 : \mathbf{PWell}(x), \mathbf{hasCapacity}(x, y) \rightsquigarrow PWell(x).$$

that describes how to populate the concept $PWell$ by joining tuples from tables $\mathbf{PWell}$ and $\mathbf{hasCapacity}$ on the well's ID and projecting out the capacity value. Observe that the mapping m_3 is empty in this setting: when applying the mapping to the data

sources, no object will be propagated into the concept *PWell*, since there is no x such that it appears in both table **PWell** and table **hasCapacity**.

Now consider the following mappings:

$$m_4 : \mathbf{q}_1(x) \rightsquigarrow PWell(x);$$
$$m_5 : \mathbf{q}_1(x) \rightsquigarrow EWell(x);$$

Since the ontology states that $PWell \sqsubseteq \neg EWell$, it follows that the query q_1 has to be empty, i.e., its evaluation over the data sources must return the empty tuple. ■

The defect with the mapping m_3 in Example 6 could be caused by *(i)* a mapping-design error, that is, tables $\mathbf{PWell}$ and $\mathbf{hasCapacity}$ are incorrectly related in m_3, or *(ii)* incompleteness of data sources. In the former case, m_3 should be deleted or repaired, while in the later case the problem should be solved at the data source level. Moreover, mappings m_4 and m_5 show that combining the knowledge about the mapping and the ontology is a key aspect towards the formal analysis of the OBDA specification.

Modeling defects are less intuitive than logical ones. A typical modeling defect is *redundancy* [7].

Redundancy. We distinguish three types of redundancy: *redundant axioms*, *concepts*, and *mappings*. Intuitively, an axiom, concept, or mapping is redundant in an OBDA setting if the deletion of it from the setting results in a logically *equivalent* setting.

Example 7. Recall the OBDA system in Figure 3, right. To observe the phenomenon of redundant axioms, note that the axiom $\alpha = PWell \sqcup EWell \sqsubseteq Well$ (*both production and exploration wells are wells*) would be redundant in this setting. Indeed, the pair of axioms $PWell \sqsubseteq Well$ and $EWell \sqsubseteq Well$ already state the same thing, i.e., they are logically equivalent to α, and so adding α would be vacuous.

To observe the phenomenon of redundant concepts, assume that the ontology implies that two concepts, say $PWell$ and $ExWell$ (standing for Exploited Well) are equivalent, i.e., $PWell \sqsubseteq ExWell$ and $ExWell \sqsubseteq PWell$. Then, these two concepts are synonymous, and we may prefer to remove one of them from the ontology.[15] Even if the ontology does not imply the logical equivalence of $PWell$ and $ExWell$, but the mappings for these concepts are the same, then the two concepts may be de facto synonymous.

To observe the phenomenon of redundant mappings, consider the mapping m_3 (Example 6) and the following mapping m_5:

$$m_6 : \mathbf{PWell}(x) \rightsquigarrow PWell(x).$$

Note that, in the presence of m_6, m_3 becomes redundant. Indeed, m_3 instantiates the concept $PWell$ with *some* objects from the table $\mathbf{PWell}$, while m_6 instantiates $PWell$ with *all* the objects found in $\mathbf{PWell}$. Thus, m_3 can be harmlessly dropped. ■

[15] This is not always the case as synonyms may capture differences in vocabulary usage amongst different user groups.

To conclude this section, we would like to note that it is not trivial to understand whether a modeling defect is actually a defect and hence requires debugging. For example, as noted above, redundancy can be intentionally introduced in an ontology by an ontology engineer. Thus, the necessity of debugging such errors depends on the application, and should be decided on a case-by-case basis.

3 Supporting the Life Cycle of OBDA Systems

Essential functionalities, required to support the life cycle of an OBDA system, are:

- Detection of defects,
- OBDA debugging,
- Ontology and mapping bootstrapping,
- OBDA evolution,
- OBDA transformation.

We will now discuss these functionalities in more detail.

An OBDA system should be able to analyze itself w.r.t. both logical and modeling defects as presented above. Thus, the system should be equipped with an OBDA *analyser*: a routine that takes an OBDA setting as input, and returns a set of defects as output. Based on the result of such analyses, the system should be able to debug itself. Since, as we discussed in the previous section, there is no universal way to debug an OBDA system, it is natural for the debugging to be semi-automatic. Thus, the system should be equipped with an OBDA *debugger*: a routine that takes an OBDA setting and its defects as input, and returns a debugged version of the setting, that is free from the input defects. For examples of tools that perform these tasks, we refer the reader to [16,12].

Clearly, 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, an OBDA system should be equipped with an OBDA *bootstrapper*: a routine that takes a set of database schemata and possibly instances over these schemata as input, and returns an ontology and mappings connecting it to the input schemata.

As was discussed above, OBDA systems are not static objects and are subjects of frequent changes, that is, evolution. Natural types of evolution are:

- **Adding/deleting a new concept together with a mapping.** It might be the case that the ontology lacks some concepts, or some concept should be dropped from the ontology, e.g., when it is synonymous with another concept. For example, consider the ontology in Figure 3, right. Assume that a new concept, e.g., $OilPr$ (*Oil Producer*), is needed. Then one might add this concept to the ontology and create a mapping that instantiates the concept.
- **Adding/deleting an ontological axiom.** An ontology may be missing relations between concepts. For example, one may want to assert that the concept $OilPr$ is a subconcept of $Well$. This can be done by adding the axiom $OilPr \sqsubseteq Well$ to the ontology.

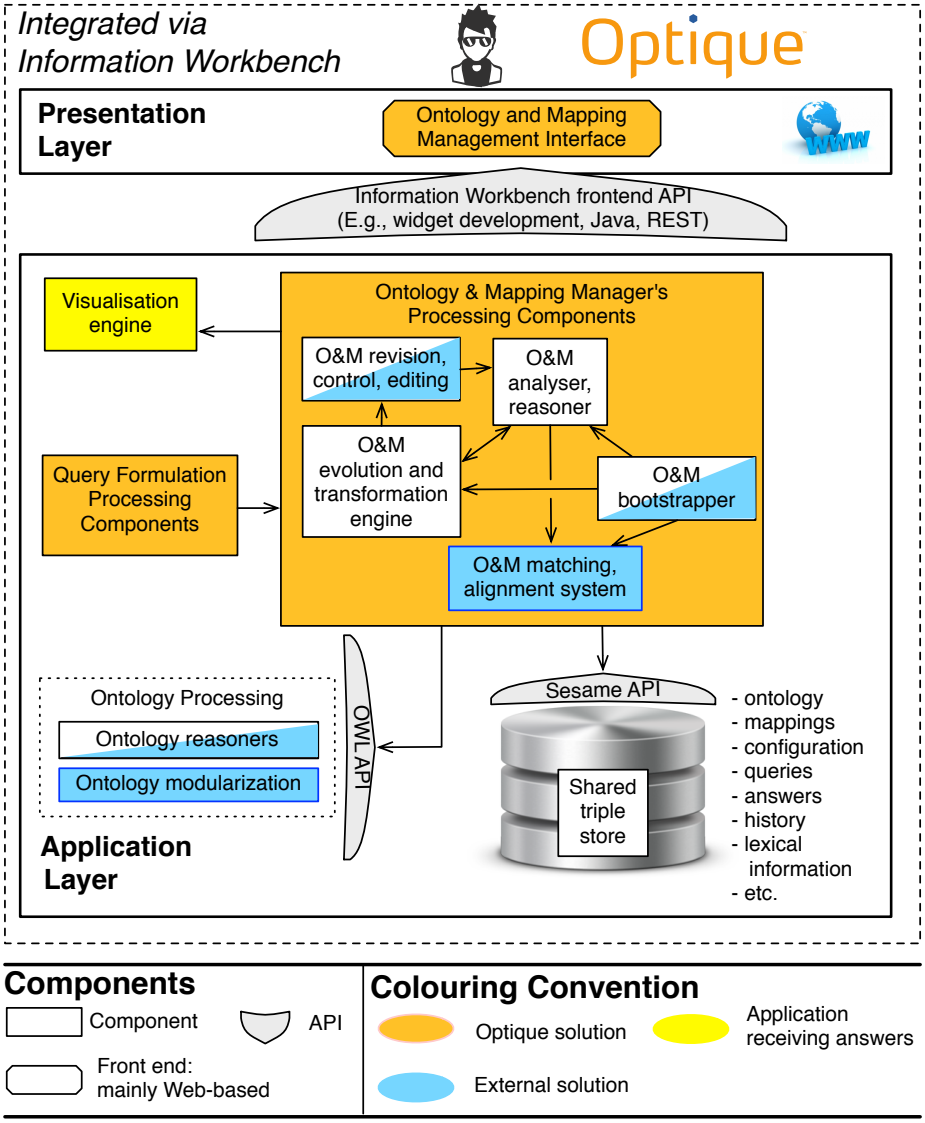

Fig. 4. Ontology and Mapping Management component of the Optique OBDA system

– **Mapping modification.** Mappings can be added, changed, or deleted. For example, if a new data source is added, one can create mappings from the new data source to some existing concepts. Another scenario for mapping modifications is optimization, that is, one may adjust mappings based on the constraints in the data sources in order to improve the performance of query processing.

A task related to both bootstrapping and evolution is ontology transformation. For example, one may want to reuse an existing third party ontology or mappings instead of (or combined with) bootstrapping them from the data sources. It is possible that the existing ontology or mappings are in a language that the system at hand does not support. Thus, one will have to transform them into the supported language. This may involve changes in the syntax and/or expressivity of the ontology.

Another reason for a transformation is optimization. One may want to improve the overall performance of an OBDA system by restricting the expressiveness of the ontology language, e.g., by moving from an OWL 2 ontology O to an OWL 2 QL ontology O'. This would in general require "approximation" of O using the weaker OWL 2 QL language [14,1].

To connect the five functionalities above, we observe that bootstrapping, evolution, and transformation functionalities naturally introduce errors in OBDA settings, while the detection functionality can detect the errors, and the debugging one can repair them.

In the following section, we present the Optique OBDA system, which will provide all of the life cycle supporting functionalities described above.

4 Optique OBDA Ontology and Mapping Manager

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. Functionalities of the O&M manager are intended for IT-experts rather than end-users. The manager includes five subcomponents:

- The *Bootstrapper* extracts an initial ontology and mappings from data sources, as discussed above. In Section 4.1, we will present our initial ideas on how to implement the bootstrapper.
- The *Matching and alignment system* performs ontology alignment.
- The *Analyser and reasoner* checks ontologies for defects.
- The *Evolution and transformation engine* performs debugging on defects found by the analyser.
- The *Revision, control and editing system* supports versioning (which can be based on, e.g., ContentCVS [10]) and editorial processes for both ontologies and mappings. It can also act as a hub, coordinating interoperation between the analyser and evolution engine.

Also, the O&M manager interacts with other components of the Optique OBDA system. In particular,

- It accesses the *Shared triple store*, where the ontology and mappings are physically stored. It can both read the ontology and mappings and update them when needed.
- The analyser, alignment system, and evolution engine have access to reasoning capabilities, e.g., external ontology reasoners, ontology modularization engines, etc.
- The *Query Formulation Component* can call the O&M manager whenever a user decides to add a new concept or axiom. We refer to this as query-driven ontology construction.
- Finally, the O&M manager is connected to a *Visualisation engine*.

4.1 Directions

In the development of the Optique OBDA system, we plan to exploit existing techniques from ontology evolution, e.g. [3,6], ontology modularisation, e.g. [23], and develop our own novel techniques. For example, a possible bootstrapping technique could be the following three step procedure:

- Step 1: Bootstrap direct[16] and R2RML[17] mappings from relational schemata of the data sources. These mappings naturally give ontological vocabularies over the schemata.
- Step 2: Construct a simple ontology over these vocabularies by extracting ontological axioms from the integrity constraints of the schemata.
- Step 3: Align the simple ontologies and the vocabularies with a state of the art ontology using an existing system, e.g. LogMap [9].

5 Conclusions

We have presented a selection of possible logical and modeling errors in OBDA systems and the main challenges to be faced in supporting the life-cycle of OBDA systems. Current approaches and methods only partially address the issues related to the construction, maintenance, and transformation of an OBDA specification. Although the EU project Optique is still at an early stage, we aim to turn our preliminary ideas into novel solutions in the very near future, and to evaluate their effectiveness in our industry use-cases. This will provide us with invaluable feedback to inform ongoing research and development of enhanced ontology and mapping management components.

Acknowledgements

The research presented in this paper was financed by the Seventh Framework Program (FP7) of the European Commission under Grant Agreement 318338, the Optique project. Horrocks, Jiménez-Ruiz, Kharlamov, and Zheleznyakov were also partially supported by the EPSRC projects ExODA and Score!

References

1. Botoeva, E., Calvanese, D., Rodriguez-Muro, M.: Expressive approximations in DL-Lite ontologies. Proc. of AIMSA 2010 pp. 21–31 (2010)
2. 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), 43–53 (2011)
3. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of DL-Lite Knowledge Bases. In: International Semantic Web Conference (1). pp. 112–128 (2010)

[16] http://www.w3.org/TR/2011/WD-rdb-direct-mapping-20110324/
[17] http://www.w3.org/TR/r2rml/

4. Crompton, J.: Keynote talk at 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
5. Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Möller, R., Özep, 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)
6. Grau, B.C., Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D.: Ontology Evolution Under Semantic Constraints. In: KR 2012 (2012)
7. Grimm, S., Wissmann, J.: Elimination of Redundancy in Ontologies. In: ESWC (1). pp. 260–274 (2011)
8. Haase, P., Schmidt, M., Schwarte, A.: The Information Workbench as a Self-Service Platform for Linked Data Applications. In: COLD (2011)
9. Jiménez-Ruiz, E., Grau, B.C.: LogMap: Logic-Based and Scalable Ontology Matching. In: International Semantic Web Conference (1). pp. 273–288 (2011)
10. Jiménez-Ruiz, E., Grau, B.C., Horrocks, I., Llavori, R.B.: Supporting Concurrent Ontology Development: Framework, Algorithms and Tool. Data Knowl. Eng. 70(1), 146–164 (2011)
11. Keet, C.M., Alberts, R., Gerber, A., Chimamiwa, G.: Enhancing web portals with ontology-based data access: the case study of south africas accessibility portal for people with disabilities. In: Proc. of OWLED 2008. vol. 2008 (2008)
12. Lehmann, J., Bühmann, L.: ORE - A tool for repairing and enriching knowledge bases. In: Proc. of ISWC 2010. Springer (2010)
13. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: PODS. pp. 233–246 (2002)
14. Pan, J.Z., Thomas, E.: Approximating OWL-DL ontologies. p. 1434 (2007)
15. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL Ontologies. In: WWW. pp. 633–640 (2005)
16. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A.: Validating ontologies with OOPS! In: Proc. of EKAW 2012, pp. 267–281. Springer (2012)
17. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In: Proc. of EKAW 2004. pp. 63–81. Springer (2004)
18. Rodriguez-Muro, M., Calvanese, D.: High Performance Query Answering over DL-Lite Ontologies. In: KR (2012)
19. Roussey, C., Corcho, O., Vilches-Blázquez, L.M.: A catalogue of OWL ontology antipatterns. In: Proc. of K-CAP 2009. pp. 205–206. ACM (2009)
20. Sattler, U., Calvanese, D., Molitor, R.: Relationships with other Formalisms. In: Description Logic Handbook. pp. 137–177 (2003)
21. 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)
22. Stuckenschmidt, H.: Debugging OWL Ontologies - A Reality Check. In: EON (2008)
23. Vescovo, C.D., Parsia, B., Sattler, U.: Logical Relevance in Ontologies. In: Description Logics (2012)