Exposing relational data on the Semantic Web with CROSS

Pierre-Antoine Champin[1], Philippe Thiran[2], Geert-Jan Houben[3], and Jeen Broekstra[4]

[1] LIRIS, Université Claude Bernard Lyon 1,
pchampin@liris.cnrs.fr,
[2] Facultés Universitaires Notre-Dame de la Paix, Namur,
pthiran@fundp.ac.be,
[3] Vrije Universiteit Brussel,
Geert-Jan.Houben@vub.ac.be,
[4] Technische Universiteit Eindhoven,
j.broekstra@tue.nl

Abstract. With the advent of the World Wide Web, a growing amount of heterogeneous data sources is becoming available, thus requiring wrapping mechanisms for a unified access by end-users or applications. OWL, an ontology language for the Web, is a good candidate for coping with structural and semantic heterogeneity between data sources. For this purpose, we propose CROSS, an open-source prototype which aims at exposing relational databases and additional knowledge about them in OWL. The key feature of CROSS is that it follows a declarative approach since it relies on the target language, OWL. In this approach, the relational structure is first converted into OWL which can then be employed by the user to declare additional OWL statements about the database. The conversion rules are inspired by previous work on database reverse engineering. However, the originality of our declarative approach is that those rules are used without any exception; we show in the paper with a number of examples how additional knowledge can be added without ever questioning the rules. We also discuss a number of directions for further work, such as the choice of the target language and the integration of the CROSS prototype with the Sesame framework.

1 Introduction

One of the challenges of the Semantic Web (SW) vision is to integrate the huge amount of information already available on the standard Web. This involves first to expose this information in the languages of the SW (XML, RDF, OWL), but also to take advantage of these languages to add as much semantics as possible to the data sources in order to improve their usability by SW agents.

On the other hand, the database community has long been studying methods and tools for expressing more knowledge about databases than the relational languages allow. One purpose of those methods and tools is data integration and query mediation among distributed and heterogeneous sources. OWL is a good candidate to fulfil that need, offering an expressive unifying language for representing both relational structures and additional knowledge about them.

This paper presents the CROSS wrapper[5], an open-source prototype which aims at exposing an OWL description of a relational database, as well as enabling to enrich that description with additional OWL statements about the database. One of the intended applications is the mediation of such wrapped relational databases with other wrapped relational databases or other OWL data sources. Indeed, OWL has a much richer semantics than relational languages, which is a benefit to cope with structural and semantic heterogeneity between data sources. We furthermore believe that the use of OWL for representing both the relational structures and the additional knowledge makes the CROSS prototype both powerful and easy to deploy with respect to other approaches.

1.1 Related work

There are a number of different approaches and proposals for exposing relational structures into more expressive formalisms: object models [1, 2], description logics [3] and of course SW technologies. Among the latter ones, we distinguish two trends: *rule-based* approaches require the user to express a set of conversion rules in a dedicated language, and then apply those rules to the relational database. *Declarative* approaches, on the other hand, focus on the target language (which is declarative, hence their name); they first convert the relational structure into that language, which can then be employed by the user to declare additional knowledge about the database.

The tool presented in [4], D2R MAP, implements a rule-based approach for exporting a relational database to RDF (hence possibly to OWL). It provides its own rule language, where each rule associates an SQL query with a template RDF description. That tool is only concerned with relational data and RDF instances; it makes the assumption that classes and properties were defined elsewhere. However, it seems technically possible to also generate an OWL ontology from the relational schema, provided that the latter is available to SQL queries through catalog tables.

[5] propose a declarative approach based on RDF Schema (RDFS), where both the schema and the data are converted, and customization is expected to be done by adding RDFS statements. This approach suffers, as the authors acknowledge, a scalability problem since it generates very dense graphs. It also focuses on a single table, and does not take advantage of constructs such as foreign keys.

[6] proposes another declarative approach, based on OWL and very similar to the one presented in this paper. It is however mainly focused on schema conversion; neither data conversion nor customization by adding OWL statements are addressed.

The approach proposed by [7] also uses OWL as its target language for both schema and data. The authors focus more on the conversion process than on

[5] `http://liris.cnrs.fr/~pchampin/dev/cross`

customizing the resulting OWL; furthermore, their approach uses the meta-modeling capabilities of OWL-Full, which prevents the use of decidable inference on the resulting ontology.

1.2 Structure of the paper

The CROSS prototype presented in this paper follows a declarative approach to expose relational databases and additional knowledge about them in OWL. The motivations of those choices are developed in section 2. We then present in section 3 the wrapper itself and the possibilities it offers in term of customizing the OWL view of the database. Finally, section 4 presents some perspectives and future work, before the conclusion in section 5.

2 Motivations

This section presents the motivations and rationale underlying our approach and its implementation.

2.1 Declarative approach for ease of use

One of the design rationales of our approach has been to make the wrapping of a relational database as *easy* as possible for the user that is unwilling to spend a lot of time on configuration, while allowing more demanding users to customize the wrapping according to their needs. This appears to us as a critical feature, for the Semantic Web can only have the same success as the Web if it also shares its simplicity of use.

This has led us to prefer a declarative approach, where the wrapping can be done fully automatically, while configuration and customization can be (but do not need to be) performed afterwards. Another advantage of this kind of approach is that it only requires users to learn the target language (OWL in our case), while rule-based approaches require them to learn the rule language as well. This reduces the learning cost, and the risk of semantic discrepancies between the expected and the actual result of the conversion. Declarative approaches also allow customization to be performed incrementally, with the possibility of testing the effect of every new declaration rapidly (because the whole conversion does not have to be run again).

Besides, declarative languages are indeed considered better than rule languages for the purpose of *knowledge representation*, from the perspective of designing as well as maintaining. This argument is relevant to the problem of wrapping relational structures because we consider that any configuration or customization of the wrapper is the assertion of *additional knowledge* about the data source: knowledge which was previously only implicitly contained in the relational structure. In a rule-based approach like D2R [4], such knowledge would be less explicit, not only because the language is rule-based but also because some of the knowledge would be held by the SQL queries, and some of it, held by the RDF statements produced by the rules. We hence believe that, although approaches of that kind are strictly more expressive than CROSS (which could be implemented with them), their lack of legibility paradoxically makes them less flexible from a user's point of view.

2.2 Locations of additional knowledge

The knowledge and constraints that the relational DBMS can not handle are generally concealed in the code, user interface, and documentation of applications using the database [8]. Changes in the structure or semantics of the database have to be reflected in the application, which is feasible (if not always easy) in a centralized context.

In the context of the Web, the problem becomes harder, because the data sources and the agents using them may be under separate and independent control, or belong to different organizations willing to keep their *autonomy*. It is thus critical that wrappers do not only act as model converters, but also supply as much additional knowledge as possible in an explicit form. By providing that knowledge to autonomous agents rather than requiring them to internally implement it, we improve the robustness of the overall system with regard to evolutions in the structure and semantics of data sources.

Of course, nothing prevents third-party ontologies to import the OWL description provided by the CROSS wrapper and add their own statements about the classes, properties, and instances it defines. The difference between the additional knowledge provided by the wrapper and such external ontologies is that the content of the former is endorsed by the source owner; it is hence assumed to describe "constant" knowledge about the database, while the latter may on the other hand state *contextual* knowledge, valid and/or useful only for a particular purpose: interoperability with another CROSS wrapper or independently designed ontology, user-friendly presentation, etc. (see figure 1).

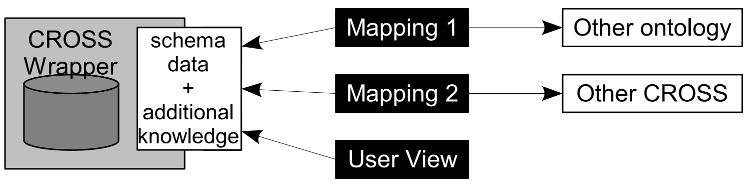

Fig. 1. Third-party ontologies (in black) extending the output of the CROSS wrapper

2.3 The choice of OWL

The choice of OWL as a target language could be questioned; the complexity of reasoning, even with OWL-Lite, is considered by some people as prohibitively high for scalable applications.

Our position is that RDF Schema is not expressive enough, considering the kind of knowledge we have to express (see section 3.3). On the other hand, implementations exist for OWL-Lite and OWL-DL[6]. We hence chose initially not to make any *a priori* restriction over the expressiveness of the allowed language (as long as decidable reasoning is available). Further experimentations with real databases will allow us to determine precisely how much expressiveness we need.

2.4 Running example

In the next section, we will use a small relational database as a running example to illustrate a number of points. That database is represented in figure 2. It is composed of five tables. In each table, the primary key is composed of the column with bold labels. Foreign keys are described explicitly, below the table data. The semantics of each table and column should be quite explicit from their name.

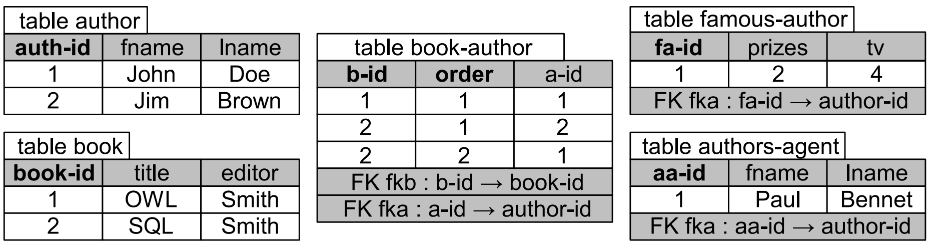

table author

auth-id	fname	lname
1	John	Doe
2	Jim	Brown

table book

book-id	title	editor
1	OWL	Smith
2	SQL	Smith

table book-author

b-id	**order**	a-id
1	1	1
2	1	2
2	2	1
FK fkb : b-id → book-id		
FK fka : a-id → author-id		

table famous-author

fa-id	prizes	tv
1	2	4
FK fka : fa-id → author-id		

table authors-agent

aa-id	fname	lname
1	Paul	Bennet
FK fka : aa-id → author-id		

Fig. 2. Our running example database

3 Converting and enriching

In this section we present the CROSS wrapper and its intended use. We first present the general principles underlying the conversion process, which produces three components. We then focus on the first two components (schema and data) and how they are automatically produced. Afterwards, we demonstrate how the manually generated third component (additional knowledge) allows us to tailor and refine the OWL view of the relational database. We then briefly discuss how external ontologies can also use those additional statements.

3.1 General principles

Translation vs. Wrapping It is important to first emphasize the difference between translation and wrapping, for CROSS uses both notions in different

[6] `http://www.mindswap.org/2003/pellet/`

contexts. We use the word *translation* to refer to a static conversion, made once and for all, usually off-line. On the other hand, the term *wrapping* refers to dynamically converting data on-demand, in response to a particular query.

In the CROSS prototype, the relational schema is translated. We consider this approach viable since the source does not change very often. On the other hand, data are wrapped, since they are likely to change frequently, and they are generally too big for reasonably allowing to store the result of a one-shot translation and update it regularly.

Components The OWL view of the wrapped database is divided into three components, namely: the Schema, the Data, and the Additional Knowledge (AK). The first two are quite explicitly named: they are automatically produced, respectively from the relational schema and from the data. On the other hand, the AK component contains all the knowledge which was not automatically inferred from the relational structures and has been manually added by the user inside the wrapper.

As previously stated (section 2.1), our approach prevents the users to have to interfere in the conversion processes. Instead, they can rely on the deterministic fashion in which those processes produce the Schema and Data components to express their own knowledge about the database; subsequently the Additional Knowledge component will allow OWL reasoners to infer more facts from the other two components.

3.2 The Schema and Data components

This subsection describes the automatic conversion processes generating the Schema and Data components. Those processes are largely inspired by previous work on database reverse engineering [1, 9, 10]. Those approaches describe conversion rules for the general case, acknowledging the fact that they should be adapted or even ignored in some particular contexts. In CROSS, however, the rules described hereafter are never questioned. Adaptation is still possible by adding statements in the AK component, but not by changing or discarding the rules.

Prerequisites There are a number of prerequisites that the relational database must fulfil for the CROSS approach to be applicable. First, every table must have a *primary key*. That is required to provide a unique identifier for every row-instance (see below).

Second, all foreign keys should be explicit in the schema. This requisite is not as strong as the previous one: the approach will be able to handle a database without any foreign key, but the produced OWL will obviously be impoverished.

We know that those requirements (even the first one) are not always fulfilled by real databases. However, we consider that extracting the implicit knowledge about the source database is a part of the reverse engineering effort necessary to its integration into any mediation architecture.

Finally, it is worth noting that the CROSS approach does *not* assume that the database was in any particular normal form.

Cells Besides the common notions describing relational structures (tables, columns, keys, rows), CROSS uses the notion of *cell*. A cell can be described as the intersection between a row and a column, and holds the value of that row for the corresponding column. Rows with a null value for a given column have no cell for that column.

While most approaches consider the values to be directly linked to the row, we will show that cells provide more flexibility for expressing additional knowledge about the relational structure (see section 3.3).

Conversion rules As stated before, the conversion rules used by CROSS are very similar to those recommended by other approaches. Each table t is converted into a class, each row of the table into an instance of that class (a *row-instance*). To each table is also associated an object property same-t which is both symmetric and functional. Furthermore, each row-instance of that table has the property same-t pointing to itself[7].

In CROSS, column values are not directly linked to rows, but held by cells. The latter are then represented in OWL by *cell-instances*. Therefore, to each column correspond (1) an object property, intended to link the row-instance to the cell-instance corresponding to this column, and (2) a datatype property, intended to link the cell-instance to its literal value (see figure 3). In the Schema component, appropriate axioms about the properties are generated to reflect general attributes of relational columns (e.g., their datatype range, cardinality restriction stating that they can have at most one value) as well as specific constraints expressed in the schema (e.g., whether the column is nullable and/or unique).

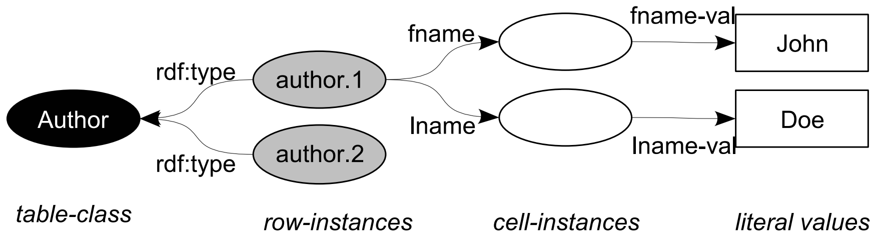

Fig. 3. Example: rows and cells in the Data component

[7] Note that owl:sameAs can not be used instead of same-t, for it is not an object property; so we can not, e.g., make it a subproperty of another object property, as described in section 3.3.

Finally, each foreign key is converted into an object property, intended to link two row-instances, as can be seen in figure 4. As with columns, appropriate axioms about the property are added to the Schema component, especially a cardinality restriction when the foreign key is declared to be unique.

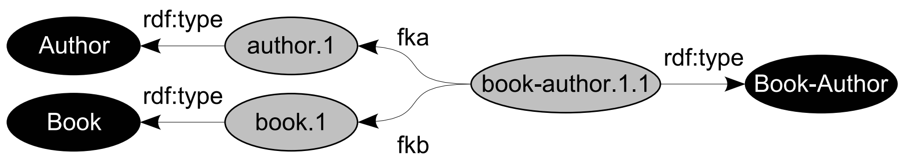

Fig. 4. Example: two foreign keys (book.fka and book.fkb) in the Data component

Let us emphasize again that the rules just described are not used in the same way for the Schema component and the Data component; namely, the former is produced by a static translation while the latter is generated on demand by the wrapper (see section 3.1).

3.3 Additional knowledge

The Schema and Data component, being automatically extracted from the relational database, only constitute a conversion of the relational structure. The language change does not convey any added value. However, what it allows is to make *explicit* the knowledge otherwise extracted by a user (after reading the documentation; understanding the labels of tables and columns; reverse-engineering the schema, queries and applications using them; etc.). Furthermore, explicit OWL-DL knowledge can not only be reused by other users but also by inference engines, to draw additional conclusions about the first two components. The Additional Knowledge (AK) component is the place of choice for that kind of statements.

Relating to other ontologies It is possible to relate the defined classes and properties to terms from other, widely accepted ontologies. In our running example, it could for instance be relevant to declare that author.fname is equivalent to foaf:firstName [11] or that book.title is equivalent to dc:title [12].

Compound attributes Relational tables are flat. However, they sometimes contain columns which can be grouped as a compound attribute. This is the case with columns table.fname and table.lname which can be considered as attributes of a complex “full-name” object.

This can be achieved by the aggregation mechanism described in [13]: stating that table.fname and table.lname are equivalent properties, together with the fact that they are functional, forces to infer that both cell-instances are the same individual (see figure 5). Making the attribute belong to a particular class, and/or making the merged column-object-properties a synonym of a more explicitly named property (e.g. fullName) is of course possible.

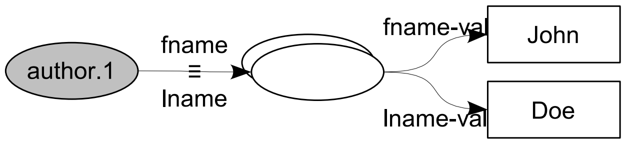

Fig. 5. Cell-instances aggregation to form a compound attribute

Cardinality restriction While many-to-many relations can be represented with a table (see for instance book-author in our example), limiting the cardinality of such relations is not possible in the relational model. Adding that knowledge is however trivial in OWL by using cardinality restrictions on the properties corresponding to the foreign keys (more precisely their inverse property). It could be stated, for example, that a book must have at least one author, or that an author must have written at least one book.

Subclasses from tables Modelling a subset of the objects represented by a table can be done by declaring another table, with its primary key being a foreign key to the former table. This is for example the case for table famous-author in our example, which represents a subclass of authors.

To make this subclass relationship explicit, one can state that the property corresponding to the foreign key is a sub-property of same-t, where t is the "superclass" table. This will result in (a) inferring the subclass relationship between the two classes representing the tables, and (b) aggregating instances of the subclass into their corresponding instance of the superclass. In our example, one would declare that famous-author.fka is a sub-property of same-author, so that class Famous-author was inferred to be a subclass of Author, and that row-instance famous-author.1 was aggregated (with its other properties) with row-instance author.1.

It must be noted that the pattern described above does not *always* represent a subclass relationship. In our example, table authors-agent follows exactly the same pattern, but does not represent a subset of the authors. It is the responsibility of the user to decide whether to add or not the statements making the subclass relationship inferable.

Subclasses from data Subsets of the objects represented by a table are not always represented by another table: in many cases, a column with a controlled set of values is used, each value representing a subset. In our example, the column book.editor can illustrate this scenario: it can only contain a limited set of values, partitioning the books into subset of books by the same editor.

The ability of OWL-DL to cope with single values (owl:hasValue construct) provides a straightforward solution to this problem: one can define, for each possible value of the column, the class of all instances having that value for the column-datatype-property. In our example, the class BookEditedBySmith can easily by defined as the set of books having value "Smith" in column book.editor.

New instances from data The previous paragraph about compound attributes has shown how cell-attribute can be useful. This approach can be extended when one wants to represent instances of classes which are totally implicit in the relational schema. Let us consider column book.editor again, but assume now that we wish to represent editors as instances of a new class Editor, defined in the AK component.

It is straightforward to state that every cell-instance pointed at by the column-property book.editor is an instance of Editor. What is more difficult is to state that every two such instances holding the same value are actually the *same* editor; though it may be tempting to declare the datatype property associated with book.editor to be inverse-functional, that is not possible in OWL-DL (only object properties can be inverse-functional). This limitation of OWL-DL is a very strong one with respect to converting relational data to OWL, since it prevents to capture the semantics of one of the most basic relational notions, namely the "UNIQUE" constraint.

We see two work-arounds to the aforementioned problem. The first one is to replace every literal value by an individual with an appropriate URI (constructed from the datatype URI and a canonical lexical representation), and use object properties instead of the datatype properties, thus escaping the limitation of the latter. This is not quite satisfying, since for interoperability reasons, we could not possibly leave the literal values. We would thus be forced to keep both representations (individual and literal) for every cell of the database, which would be both redundant and expensive.

The second work-around is the following: for each possible value (e.g. "Smith") of column book.editor, (1) add in the AK component a subclass of class Editor, and a fresh instance smith of that class, (2) define the class BookEditedBySmith as in the previous example, (3) force all instances of that class to be related to smith with an appropriate property. The drawback of that work-around is that it requires to know in advance all the possible values of the column, hence to update the AK component every time the *data* changes.

Mixing schema and instances The last two scenarios show an interesting feature of OWL-DL, namely its ability to mix schema components with instances. This feature is all the more interesting in that it is missing from most reverse-engineering approaches for converting relational structures to more expressive languages [8]. The last scenario, however, exhibits some limitations of that feature, primarily due to the limitations concerning literal values – a serious handicap when dealing with databases, which we will address in section 4.1.

4 Perspectives and future work

4.1 Alternative target languages

As we have mentioned earlier, the choice of OWL-DL has been pragmatically motivated by the fact that (1) it is highly expressive and (2) implementations of inference engine are available. However, we also observed that its expressiveness is limited when it comes to data (section 3.3). Furthermore, inference mechanisms for OWL-DL are highly complex, and many authors argue in favor of less expressive sub-languages with more efficient inference algorithms.

As we already discussed in section 2.3, we wait for more feedback on real databases to settle for any loss of expressiveness: we have to experimentally check which constructs are actually needed and which can be left aside. An interesting direction is proposed by [14], where the efficiency of logic programming engines can be combined with the legibility of description logic constructs (with heavy limitations on the way they can be used and combined, however).

As noted in the previous section, OWL has some limitations when it comes to literal values. Although we proposed some work-arounds for that problem, we are also investigating extensions of OWL aiming at improving its handling of concrete data. [15], for example, describe such an extension, where restriction of basic types (e.g. "any integer greater than 10") can be expressed. The authors show that their extension preserves decidability in many description logics –not only in OWL-DL. It is hence not contradictory with our intention to use an otherwise less expressive language.

Sesame Integration The Sesame framework [16] is an RDF database framework that allows scalable storage, querying and inferencing with large RDF data sets. One of the central architectural components of Sesame[8] is the SAIL (Storage And Inferencing Layer) API, which is an abstraction layer on top of a physical storage system, such as an RDBMS, a file system, or some other source of RDF data. On top of this SAIL, Sesame provides many useful tools for manipulating the data, such as the SeRQL [17] query language, and pluggable components for reasoning, such as a rule-based RDF Schema reasoner and OWLIM [18], an OWL reasoner.

The setup is ideally suited for integrating with CROSS: the CROSS approach can be implemented as a SAIL component, allowing Sesame's query engines and reasoner to operate on top of the CROSS wrapper (and thus on top of the legacy data) without the need for duplication of the data. Thus, CROSS will be enriched

[8] for a technical description of the Sesame architecture, see `http://www.openrdf.org/doc/sesame2/user`

by having query and OWL reasoning support, while Sesame is enriched with a generic approach for integrating legacy databases.

We plan to investigate the realization of such an integration in detail. Specific questions that remain to be solved are, for example, how the reasoner interacts with the CROSS wrapper, and if an RDF query language such as SeRQL is expressive enough for our purposes.

4.2 Assisting the user in expressing additional knowledge

Although we showed in section 3.3 how frequent scenarios can be handled in the AK component, those solutions are not always trivial, especially for a novice user to OWL. It would hence be interesting to design a "wizard" or assistant software, helping users in expressing their additional knowledge as OWL statements, at least for the frequent scenarios described above (and possibly some others).

Furthermore, those scenarios usually correspond to a particular pattern in the relational schema. We already noted that those patterns do not guarantee that the scenario applies. However, they could be automatically detected, and marked in the resulting OWL, using OWL annotations. Those annotations could then be use by the assistant to *suggest* the application of the scenario.

Finally, this assistant could take benefit of techniques such as case-based reasoning to improve its suggestions, relying on similarity measures (rather than exact pattern matching) and previous experiences.

5 Conclusion

We have presented CROSS, an open-source prototype which aims at exposing relational databases and additional knowledge about them in OWL. The approach adopted in CROSS is divided in two distinct steps: an automatic conversion of the schema and data into OWL statements, and the manual addition of OWL statements capturing the design knowledge that was only implicit in the relational structures (and generally hidden in applications code, documentation, user interfaces or queries). We call this kind of approach *declarative* because the users only employ a declarative language (OWL) to express that implicit knowledge. In other approaches, on the other hand, users have to employ rule languages; we have argued that this makes the additional knowledge less explicit, harder to express and to maintain.

A consequence of that declarative approach is that the conversion rules used in the first step are never questioned. This is an originality of CROSS, although the rules themselves are largely inspired by the previous work on database reverse engineering, in which they are not considered to be applicable in *any* case. However, we have demonstrated with a number of examples that frequent scenarios in database design could be handled by adding statements, and without removing any.

Finally, we have discussed the choice of the target language, as OWL-DL suffers from highly complex inference mechanisms and a poor handling of concrete data (literal values). We have also proposed further extensions of CROSS.

We plan to integrate it in Sesame, and RDF database framework, which would benefit from a generic back-end for legacy databases, and bring on top of CROSS its query capabilities. We also plan to develop an assistant software for helping users to express their additional knowledge in OWL.

References

1. Ramanathan, S., Hodges, J.: Reverse engineering relational schemas to object-oriented schemas. Technical Report MSU-960701, Reverse Engineering Relational Schemas to Object-Oriented Schemas (1996)
2. Lim, E.P., Lee, H.K.: Export database derivation in object-oriented wrappers. Information & Software Technology **41**(4) (1999) 183–196
3. Borgida, A., Lenzerini, M., Rosati, R.: Description Logics for Databases. In Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: Description Logic Handbook, Cambridge University Press (2003) 462–484
4. Bizer, C.: D2R MAP - A Database to RDF Mapping Language. In: 12th International World Wide Web Conference (Posters). (2003)
5. Korotkiy, M., Top, J.L.: From Relational Data to RDFS Models. In Koch, N., Fraternali, P., Wirsing, M., eds.: International Conference on Web Engineering. Volume 3140 of LNCS., Springer (2004) 430–434
6. Buccella, A., Penabad, M.R., :Rodriguez, F.J., Faria, A., Cechich, A.: From Relational Databases to OWL Ontologies. In: 6th Russian Conference on Digital Libraries, Pushchino, Russia (2004)
7. de Laborda, C.P., Conrad, S.: Relational.OWL - A Data and Schema Representation Format Based on OWL. In Hartmann, S., Stumptner, M., eds.: Asia-Pacific Conference on Conceptual Modelling. Volume 43 of CRPIT., Australian Computer Society (2005) 89–96
8. Hainaut, J.L.: Introduction to database reverse engineering. Technical report, University of Namur (2002) Available at `http://www.info.fundp.ac.be/~dbm/publication/2002/DBRE-2002.pdf`.
9. Hainaut, J.L., Henrard, J., Hick, J.M., Roland, D., Englebert, V.: Database Design Recovery. In Constantopoulos, P., Mylopoulos, J., Vassiliou, Y., eds.: CAiSE. Volume 1080 of LNCS., Springer (1996) 272–300
10. Andersson, M.: Extracting an entity relationship schema from a relational database through reverse engineering. Int. J. Cooperative Inf. Syst. **4**(2-3) (1995) 259–286
11. Brickley, D., Miller, L.: FOAF Vocabulary Specification. `http://xmlns.com/foaf/0.1/` (2005) See also `http://www.foaf-project.org/`.
12. DCMI Usage Board: DCMI Metadata Terms. DCMI Recommendation, `http://dublincore.org/documents/2005/06/13/dcmi-terms/` (2005)
13. Dean, M., Schreiber, G.: OWL Web Ontology Language. W3C Recommendation, `http://www.w3.org/TR/owl-ref/` (2004)
14. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: 12th International World Wide Web Conference, ACM (2003) 48–57
15. Pan, J., Horrocks, I.: OWL-Eu: Adding Customised Datatypes into OWL. In Gómez-Prez, A., Euzenat, J., eds.: 2nd European Semantic Web Conference. Number 3532 in LNCS, Springer (2005) 153–166

16. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Horrocks, I., Hendler, J., eds.: 1st International Semantic Web Conference. Number 2342 in LNCS, Sardinia, Italy, Springer (2002) 54–68 See also `http://www.openrdf.org/`.
17. Broekstra, J.: SeRQL: A Second Generation RDF Query Language. In: Storage, Querying and Inferencing for Semantic Web Languages. Number 2005-09 in SIKS Dissertation Series. Vrije Universiteit (2005) 67–86 See also `http://www.openrdf.org/doc/SeRQLmanual.html`.
18. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM - a Pragmatic Semantic Repository for OWL. In: International Workshop on Scalable Semantic Web Knowledge Base Systems, New York City, USA (2005) See also `http://www.ontotext.com/owlim`.