ChinaRxiv

AI translation · View original & related papers at
chinarxiv.org/items/chinaxiv-202211.00206

KnowID: An Architecture for Efficient Knowledge-Driven Information and Data Access (Postprint)

Authors: Ruben Fillottrani, Pablo, Maria Keet, C., Maria Keet, C.

Date: 2022-11-18T00:00:00+00:00

Abstract

Modern information systems require the orchestration of ontologies, conceptual data modeling techniques, and efficient data management so as to provide a means for better informed decision-making and to keep up with new requirements in organizational needs. A major question in delivering such systems, is which components to design and put together to make up the required "knowledge to data" pipeline, as each component and process has trade-offs. In this paper, we introduce a new knowledge-to-data architecture, KnowID. It pulls together both recently proposed components and we add novel transformation rules between Enhanced Entity-Relationship (EER) and the Abstract Relational Model to complete the pipeline. KnowID' s main distinctive architectural features, compared to other ontology-based data access approaches, are that runtime use can avail of the closed world assumption commonly used in information systems and of full SQL augmented with path queries.

Full Text

Preamble

KnowID: An Architecture for Efficient Knowledge-Driven Information and Data Access

Pablo Rubén Fillottrani[1],[2] & C. Maria Keet[3]†

[1]Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Buenos Aires 8000, Argentina
[2]Comisión de Investigaciones Científicas, Provincia de Buenos Aires, Argentina
[3]Department of Computer Science, University of Cape Town, Rondebosch 7700, South Africa

Keywords: Extended entity-relationship diagrams; Abstract relational model; Ontology-based data access

ChinaRxiv 中X

Citation: P.R. Fillottrani & C.M. Keet. KnowID: An architecture for efficient knowledge-driven information and data access. *Data Intelligence* 2(2020), 487-512. doi: 10.1162/dint_a_{00060}

Received: October 8, 2019; **Revised:** April 24, 2020; **Accepted:** April 26, 2020

Abstract

Modern information systems require the orchestration of ontologies, conceptual data modeling techniques, and efficient data management to provide better informed decision-making and keep up with new organizational requirements. A major question in delivering such systems is which components to design and assemble to create the required "knowledge to data" pipeline, as each component and process involves trade-offs. In this paper, we introduce a new knowledge-to-data architecture called KnowID. It integrates recently proposed components and adds novel transformation rules between Enhanced Entity-Relationship (EER) and the Abstract Relational Model to complete the pipeline.

KnowID' s main distinctive architectural features, compared to other ontology-based data access approaches, are that runtime use can avail of the closed world assumption commonly used in information systems and full SQL augmented with path queries.

1. Introduction

Traditional data management comprises a sequence from requirements to conceptual data model, transformation into a relational model, and creation of a physical database schema, with each stage leaving behind the artifact produced in the preceding step. As valuable information is represented in early pipeline models, ideas for reusing outputs from the first three stages have been proposed over the years, such as query-by-diagram with Unified Modeling Language (UML)-like notation [1] and conceptual queries with Object-Role Modeling (ORM) [2] in the mid-1990s. From the mid-2000s, theory, techniques, and tools advanced to reuse the conceptual model at runtime in conjunction with large data stores, creating a "knowledge to data" pipeline. This combination, or pipeline, is known by various terms, including ontology-based data access and (virtual) knowledge graphs [3]. We illustrate a key advantage of such a pipeline with an example about an information request concerning customers.

Example 1. A Marketing department member, say Juan, wants to query the company' s information systems about customers. If no canned queries exist and Juan cannot write SQL or does not know which database(s) store customer data, he must either explore the data himself or ask a database administrator for help. Both options cost time, and time is money. Data scientists reportedly spend at least 80% of their time on data discovery and integration [4].

Imagine now there is one model representing what data are stored in the systems, possibly displayed graphically. Then Juan himself can find what he can query, reducing effort for this otherwise time-consuming data discovery task. With an "intelligent" system, he does not need to know where and how exactly that data are stored—just that it is somewhere—and can simply pose the query, say, "list all customers." Here, the scenario diverges from query-by-diagram approaches, for in the knowledge-to-data pipeline, the model is used at runtime to answer the query. To see this, assume Joanne is an IndividualCustomer that is a Customer; formally, we have IndividualCustomer ⟨Joanne⟩ in the database (i.e., a table named IndividualCustomer with a row containing Joanne) and Customer in the knowledge layer (e.g., a Web Ontology Language (OWL) ontology).

When Juan fires the query "list all customers," he will observe different results:

- **Default relational database installation:** Returns {}, i.e., an empty answer, as there are no explicitly declared instances of Customer in the database.
- **The "knowledge to data" pipeline (with reasoner):** Returns {Joanne}, thanks to inference from the subsumption that inferred Customer ⟨Joanne⟩ .

An end-user like Juan would expect {Joanne} as the answer, but we only obtain that with the second option, where human intelligence in understanding the situation is woven into the system to produce that behavior.

The most popular approach to build such intelligence into the system is Ontology-Based Data Access [5, 6]. It uses the open world assumption (OWA) rather than the closed world assumption (CWA) of databases that end-users are more familiar with, contains a computationally costly mapping layer to link knowledge to data, and supports only a fragment of full SQL (unions of conjunctive queries). For example, with data { ⟨Joanne, PhD⟩ , ⟨John, MSc⟩ } and a query "who doesn' t have a PhD?" , the answer under OWA would be {} and under CWA would be {John} because absence is treated as negation.

Another option is to store everything in a database, but it typically has only limited or no support for automated reasoning, and if so, it is implemented with carefully crafted stored procedures and triggers (e.g., [7, 8]). Alternatives addressing these issues are being investigated. A recent alternative proposed extending the relational model into an "Abstract Relational Model" (ARM) with special object identifiers and a strict extension to SQL for path queries (SQLP) [9]. It addresses typical database users' expectation of CWA and supports full SQL. It avoids the computationally costly mapping layer of ontology-based data access through computationally cheap transformations. However, this ARM+SQLP option falls short on the knowledge (i.e., ontology or conceptual data model) layer in the knowledge-to-data pipeline because it lacks that knowledge component.

We aim to extend this partial knowledge-to-data pipeline of ARM+SQLP into the knowledge layer by adding a conceptual data model or application ontology

and relevant model management features, such that runtime use can remain within the closed world assumption and users can still use full SQL. This extension is worked out in detail for the ARM to/from Enhanced Entity-Relationship (EER) transformation through a set of rules, which differ from regular EER-to/from-Relational Model (RM) transformations due to several ARM peculiarities and the richer set of constraints ARM can represent compared to RM. The generalization from that can then avail of other extant recent research results to complete a generic knowledge-to-data architecture, which we call KnowID: Knowledge-driven Information and Data access. KnowID' s functionality is thus similar to ontology-based data access systems: it allows queries to be posed at the knowledge layer—supporting users in what to query without labor-intensive figuring out of how and where—that will be evaluated over an "intelligent" database making use of knowledge represented in the conceptual data model or ontology.

Architecturally, a distinct practical advantage is achieving this through automated transformations rather than (manual or automated) specifications of non-trivial mappings in a separate mapping layer. A further practical advantage is support for path queries, which user experiments have shown makes query formulation faster with at least the same level of accuracy or fewer errors [10, 11].

The remainder of the paper is structured as follows. Section 2 presents preliminaries and related work, including an overview of four principal knowledge-to-data approaches to contextualize the architecture and a summary of relevant ARM key components. Section 3 discusses design considerations for the extension and introduces the novel KnowID architecture. Section 4 provides new technical details of the extension: transformations from EER to ARM. Section 5 briefly motivates and discusses the architecture' s genericity. Section 6 presents evaluation, and Section 7 concludes.

2. Preliminaries

Accessing data through knowledge requires background knowledge about various pipeline components spanning computer science subfields. We therefore introduce key terms and related work first, then detail one approach—ARM with SQLP—in greater depth.

2.1 Key Components of a Knowledge-to-Data Pipeline

We focus on combining intensional knowledge (K) represented as formalized descriptions about entity types, relationships among them, and constraints holding for them—e.g., modeled in a conceptual data model or an ontology—with large amounts of data (D) representing instances. Surveying the literature, we distilled four core approaches, summarized and illustrated with exemplars.

- **K D** -Knowledge with Data: Store K and D in one system, knowledge representation (KR)-oriented, informally: "push the envelope of AI logic

to manage some data" ; e.g., put K in the ontology' s "TBox" , D in the "ABox" (in Description Logics terminology [14]), and store both in a single OWL file [15] for use with Semantic Web infrastructure.

- **K D** -Knowledge mapping Data: Store K in a KR-oriented way, as for K D, but delegate D to external storage in a database and link the two through a mapping layer (M) that maps terms from K to queries over D. Informally, this is the approach of "use each where it' s good at" ; e.g., Ontology-Based Data Access (OBDA) that links an OWL file to a Relational Database Management System (RDBMS) through mappings [6], for which multiple tools and use cases exist [3].
- **D→K** -Data transformation Knowledge: Exploit data, database schemas, and the relational model layer back up to its originating conceptual model or application ontology formalized in a suitable logic, informally: "extend the database with some AI notions" ; this option is currently incomplete, but partial theoretical foundations are described in [9, 10, 16].
- **D K** -Data with Knowledge: Store both K and D in one system, database-oriented, informally: "exploit database technologies and push the envelope thereof" ; e.g., the OntoMinD system [7] has separate tables for knowledge and data, plus additional triggers and stored procedures in the RDBMS to cater for automated reasoning.

The respective core architectures are depicted simplistically in Figure 1 [Figure 1: see original paper]. Design decisions for constructing specific architectures include open vs. closed world, query language (e.g., SPARQL, SQL, or both, or some extension), mapping choice (if applicable), query types supported, expressiveness of logic for representing K (e.g., OWL to QL, OWL to RL), and language/storage system for D (e.g., SQL, RDF, Datalog). Actual instances may be more elaborate with additional sub-components.

However, their respective core approach to solving runtime use of knowledge and data is essentially unchanged. Variants include, e.g., conceptually adding a "second TBox" in the knowledge layer in OBDA, which links the OBDA ontology (de facto a conceptual data model) to a domain ontology [17], or there may be an additional intermediate query language for generalizability [18], yet the overall approach remains K D.

A specific K D design is OBDA, with implementations such as Ontop [5]. Ontop permits installations using OWL to QL or RDFS as ontology language for K, Quest as automated reasoner, R2RML and its native mapping layer languages in M, and supports several RDBMSs for D including UnityJDBC, IBM Websphere, MS SQL Server, and Oracle. An example installation might use OWL to QL and an Oracle database, as in the Slegge system at Statoil [19]. A similar drill-down from generic principal architecture to actual system configuration is possible with other architectures.

D K systems have been proposed that exploit relational, object-relational, and

object-oriented databases. They have various trade-offs, such as a comprehensive database schema storing all possible OWL axioms but no reasoning beyond regular database querying [8], or only a fragment of OWL (e.g., DL-Lite_R) with automated reasoning [7]. Their actual architectures tend to be less elaborate than K D orchestrations; see [7] for a good overview.

The only principal approach incomplete in both overall design and implementation is D→K. We therefore examine it more closely in the next section.

2.2 Comparison of Architectures

We compare the four architectures on key points that significantly impact overall complexity of any particular practical design. These distinctive features are listed in Table 1 and discussed below.

First, note that among options K D, D→K, and D K, one must handle automated reasoning specially because they lack the native ABox of the ontology as in K D. This can be achieved through either query rewriting (i.e., reasoning on-the-fly) or data completion (pre-computing deductions and materializing inferences). K D generally uses query rewriting (including Ontop), while D→K and D K (including OntoMinD [7]) use data completion.

Second, other non-trivial issues involve or affect automated reasoning: (1) expressiveness of language for K; (2) query types supported over D; and (3) algorithm class used (e.g., tableaux vs. stored procedures).

Within items (1) and (2), options differ regarding: (i) OWA vs. CWA, (ii) unique name assumption or not (both affect computational complexity of reasoning problems for K' s representation language [20]), and (iii) query evaluation over D. While OWA/CWA balance tends to be consistent within an approach, they differ across the four approaches, reflecting their origins (AI or DB). These within-architecture variations for actual pipeline instances mean no single computational complexity figure exists for each architecture.

Third, one must appreciate the distinction between ontological principles and engineering. While philosophical and mathematical differences between classes, instances, and values are clear, one can "play"with them in back-end architecture implementation. For instance, take customer Joanna and class IndividualCustomer: if Joanna is stored in a database, it is not an individual but a value, so to pass it to an ontology' s ABox, that value must be converted to an object. If IndividualCustomer is stored in a database, it is not a class but a value, so it cannot have, e.g., a set of members as instances because values lack such instances; hence, they must be treated differently from cells storing instances-as-values.

This sort of manipulation of classes, instances, and values, also called entity recasting, is typical for K D and D K, whereas K D and D→K stay true to their respective formalisms.

Finally, as a general consideration, domain experts'data access at both relational

and conceptual layers improves query accuracy and time (see [10] and references therein). Therefore, "syntactic sugar" layers atop principal architectures are common, such as early systems ACE for OWL with K D [21] and WONDER for OBDA with K D [22] providing natural language and graphical interfaces. More recent systems may combine ease of user access and systems management, such as Optique [23]. We are not concerned with presentation layer quality here but note it is a common extension for any of the four approaches.

2.3 Preliminaries: ARM+SQLP

The Abstract Relational Model (ARM) generalizes the relational model (RM) by including an abstract domain of entities OID for object identifiers and a new self attribute for each relation in an ARM that, through an underlying theory of referring expressions, remains virtual and thus does not cost another column in a database table. Details are described in [9, 10, 16]. This seemingly simple extension allows various constructs beyond primary and foreign keys, such as declaring disjointness constraints, explicit inheritance, and path functional dependencies.

Definition 1 (Relations and Constraints). Let REL, AT, and CD be sets of relation names, attribute names (including self), and concrete domains (data types), respectively, and let OID be an abstract domain of entities (surrogates), disjoint from all concrete domains. An abstract relational model schema Σ is a set of relation declarations of the form:

`table T(A1 D1, ⋯, An Dn, Q1, ⋯, Qm)`

where T REL, Ai AT, Di (CD {OID}), and Qi are constraints attached to relation T. We write REL(Σ) to denote all relation names declared in ARM schema Σ. Ai is abstract if Di is OID, and concrete otherwise, and self must be abstract for each T. Constraint Qi has one of the following forms:

1. (primary keys) `primary key (self)`
2. (foreign keys) `constraint N foreign key (Ai) references T1`, where Ai is an abstract attribute in T' s definition
3. (inheritance) `isa T1`
4. (covering constraints) `covered by (T1, ⋯, Tk)`
5. (disjointness constraints) `disjoint with (T1, ⋯, Tk)`
6. (path functional dependencies) `pathfd (Pf1, ⋯, Pfk)$\rightarrow$Pf` where each Ti REL. A path Pf is either self or a dot-separated sequence of attribute names.

Primary keys and named foreign keys ("N" in item 2) are supported by standard SQL. Inheritance and disjointness constraints are satisfied when all (resp. no) self-values occurring in T occur as self-values in some (resp. all) Ti in inheritance (resp. disjointness) cases. A path functional dependency is satisfied when any pair of T-tuples agreeing on each Pfi' s value also agree on Pf' s value.

The identifier aspects of objects with self and OID are handled by referring

expressions, which is rather involved and detailed in [9, 10]; for extending ARM+SQLP, it suffices to know it works. Likewise, ARM-to-RM [9] and RM-to-ARM [10] transformation algorithms are internal to the data layer and do not affect the information and knowledge layer (this paper' s focus)—we may assume they are in place. The basic ARM formalization provides reasoning about constraints as logical consequences. That is, for some Ti in Σ, Σ (Q Ti) denotes that constraint Q for T logically follows from constraints in Σ even when not explicitly stated. A simple example: when T1 has attribute A1 as PK yet A1 also appears in an FK constraint to T2 where it is also PK, this implies constraint Q = `isa T2` for T1. ARM can be formalized in a Description Logic (DL) where the problem of deciding when Σ (Q T) holds with n-aries, e.g., the PTIME decidable CFDI -nc in ARM schemata, can be reduced to reasoning about logical consequence [9].

This approach has a complementary query language called SQLP [9], an extension to full SQL allowing queries over paths, availing of those virtual identifiers in the background, and shown to improve querying when used with ARM [10] (not this paper' s focus). Note that since SQL and RM already operate under CWA, ARM+SQLP as its extension with identifiers and paths does not affect CWA. We illustrate this with an example.

Example 2. Consider a section of the sample ARM diagram evaluated by users in [10], depicted in Figure 2 [Figure 2: see original paper] on the left-hand side, which tried to emulate an RM look-and-feel (the right-hand side resembles the ARM sample diagram notation in [25] and emphasizes its graph-like structure). One relation definition is:

```
table professor ((self OID, pnum INT, pname STRING, office STRING, department OID),
                 primary key (self),
                 constraint dept foreign key (department) references department,
                 pathfd (pnum) -> self,
                 disjoint with (course, class, department))
```

Other ARM relation definitions follow the same pattern. A sample query: "Find the distinct names of all professors who work at the CS department that have taught a class of a course offered by another department." This can be written in SQLP for ARM as:

```
SELECT distinct pname
FROM professor p
WHERE p.department.dname = 'CS' AND
      class.course.department.dcode != class.p.department.dcode
```

Recall that in RM, all attributes must be concrete, and only primary and foreign key constraints are allowed. In contrast, ARM includes self as the primary key of every table, with object identifiers from an abstract domain. This means ARM generalizes RM in two ways: first, there is no explicit choice in representing primary keys for each relation—we just know it exists; second, a set of declarative constraints is available to model domain data. In this sense, ARM is strictly

conceptual model-like but does not complete the knowledge-data column, as ARM still contains various design decisions rather than being purely at the conceptual "what" -layer of representing information and knowledge. Those design decisions are mainly: primary and foreign key choices and naming, data types, the choice of "a new relation for each entity type" for entity types in a hierarchy in a conceptual model, and it assumes other choices, such as the FK side of a 1:1 cardinality relationship.

This final step of adding the knowledge layer tends to make the whole pipeline theoretically and technically involved. We first deliberate options for adding that knowledge layer to the ARM+SQLP approach in the next section, then add it.

3. Design Considerations for Adding a Knowledge Layer to ARM+SQLP

The idea of having a complete knowledge-to-data pipeline is intuitively clear, but what does it mean with respect to its distinguishing features and requirements such that one can say "this architecture is one of those, but that one is not" ? There are three key components:

1. **Knowledge:** Formal/logic-based representation at the type/class-level (i.e., with entities that can be instantiated). This type-level theory should be independent of systems design aspects and contain only the "what" of the universe of discourse, not the "how" components (so, e.g., no PK/FK and OO object identifiers in the model).
2. **Structured data:** Representations of individuals (i.e., entities that cannot be instantiated further); e.g., data stored in a relational database or RDF triple store (but not unstructured text documents).
3. **Automated reasoning support:** At minimum, querying data using vocabulary elements from the knowledge layer, availing of formally represented knowledge in some way (not just a graphical query interface for stored data like query-by-diagram).

Holding ARM+SQLP against these key components, it falls short on the knowledge layer, as its ARM specification includes design decisions, as discussed previously (PK/FK etc.). To complete this approach for knowledge-to-data to also include the knowledge layer, there are two broad strategies:

a. Add an ontology or conceptual data model, suitably formalized in first-order logic or a fragment thereof (e.g., OWL or another Description Logic), and link it through a mapping layer to the data.
b. Construct an algorithm similar to forward pipeline or reverse engineering between conceptual models and relational models, but adjusted for ARM rather than RM.

Both ARM and conceptual data modeling language ORM [26] have been formalized in a member of the CFD family of DLs [27, 25], making Option (a)

seem doable. However, like all DLs, CFD logics operate under OWA, whereas databases assume CWA, and linking OWA with CWA components in one system is far from trivial theoretically and technically. Option (b) could operate fully within CWA while still permitting formalization in some DL to use standard automated reasoners, provided it is carefully orchestrated.

This extension is sketched in Figure 3 [Figure 3: see original paper] and elaborated in Sections 4 and 5. As seen from the figure, it remains within CWA in crucial querying stages, since query formulation uses only knowledge layer vocabulary and the query itself—in SQLP throughout—is fully evaluated within the database world' s CWA. Such Option (b) is more doable theoretically than Option (a), although it remains to decide how to handle which conceptual modeling language, and transformation algorithms between that and ARM must be designed. The choice of conceptual modeling language turns out not to matter, because constraints representable in ARM can be mirrored in popular conceptual data modeling languages; they are interchangeable either through direct 1:1 transformations or through a joint metamodel, as recently demonstrated in [28], or a common core profile can be designed, as proposed in [29]. The former —those transformations—are addressed in the next section.

4. Transformations Between ARM and Conceptual Models

Given that transformations between EER and RM are well known, we base our bi-directional algorithms on that and adjust for ARM' s peculiarities. Further, because ARM has several unique constraints that regular RM lacks, we present bi-directional transformations from ARM to EER first, then from EER to ARM. These transformations bridge the gap between knowledge and data layers, where the former has no known complete semantic formalization (i.e., including weak entity types), so no soundness results can be given. Also, modeling features of both languages are diverse due to different aims, making transformations necessarily incomplete. More details are provided in respective transformation descriptions below.

4.1 From ARM to EER Diagram

Reverse engineering from ARM to EER follows the same ideas as usual components, notably as presented in [30], such as classification of relations and attributes, 1:n relationships, and a "dangling attribute" indicating a weak entity type. Key differences compared to regular RM-to-EER reverse engineering algorithms appear in rules I-A, I-B, IV, V, and VI.

I. For each T such that Σ contains declaration `table T(A1 D1, ⋯, Am Dm, Q1, ⋯, Qn)`, define basic ERD construct (entity type or relationship type) BT as follows:

A. Choose T' s primary key as a Qi such that `pathfd (Ai$_{1}$, ⋯, Ai )$\rightarrow$self` where k≤n, with indexes i on A denoting they are part of constraint Qi. Order between Ai in this constraint is irrelevant, so

suppose they are already ordered such that all attributes participating in FK constraints in T are placed at the beginning, and attributes not belonging to any FK constraint are placed at the end. Let h be the index such that all attributes Ai with j≤h are also declared as FK (i.e., there is a constraint of form `constraint foreign key (Ai ) references Tj`), whereas for remaining Ai (h<j≤k) no such FK constraint exists.

B. If h=0 (i.e., no PK attributes are also FK), then T represents a "strong relation" (sensu [30]). This step decides whether it is a strong entity type or strong relationship type. If there exists a candidate key constraint `pathfd (Ai$_{1}$, ⋯, Ai ')$\rightarrow$self` where all Ai ' participate in FK constraints in T (i.e., there is `constraint foreign key (Ai ') references Tj` for all j, 1≤j≤k'), then set BT to be a new relationship type with participation of all basic constructs Tj. Otherwise (i.e., no such candidate key constraints exist), let BT be a new entity type.

C. If h=k (i.e., all PK attributes are also FKs), then all Aj are included in foreign key constraints `constraint foreign key (Aj) references Tj`, and T represents a "regular relation" . Set BT to be a new relationship in which those Tj participate that were reverse engineered from their respective Tj.

D. If 0<h<k then some PK attributes are included in FK constraints and some are not, so T represents a "weak relation" . Then let BT be a new entity type with partial key attributes determined by all dangling key attributes Aj (1≤j≤h<k).

Build the initial ERD with all BT.

II. Assign all attributes Ai of PK Qi where h<j≤k (i.e., PK attributes that are not also FK) to BT and declare them as members of the identifier (or partial identifier) for BT.

III. Let Al be any attribute not appearing in PK constraint Qi (i.e., l≠ij for all 1≤j≤k). If Al appears in FK constraint `constraint foreign key (Al) references Tl`, then add participation to entity type Tl. In the latter case, include 1:n participation from BT to Tl if BT is a relationship type, or to a new binary relationship type if BT is an entity type. If Al does not appear in any FK constraint, assign it to BT as a regular attribute, ignoring its respective Dl.

IV. For each Qi = `isa T1`, add a specialization relationship between BT and BT1.

V. For each Qi = `covered by (T1,⋯,Tp)` used in conjunction with `isa Ti` (1≤i≤p and p$ $2), declare the covering constraint over the corresponding isa in the ERD.

VI. For each Qi = `disjoint with (T1,⋯,Tp)` used in conjunction with `isa Ti` (1≤i≤p and p$ $2), declare the disjointness constraint over the corresponding isa in the ERD.

This transformation has a linear number of steps in the size of relation declaration clauses. Step I.A selects the primary key and partitions attribute sets (no ordering is necessary). A single scan of the table declaration clause suffices for steps I.B, I.C, I.D, II, and IV. Tagging non-PK attributes in step III can use an appropriate data structure. A dictionary of isa-relationships built during step IV enables constraining each member in steps V and VI in constant time.

After this process, some ARM specification information is not in the generated ERD because EER does not support those features. In particular, all functional path dependencies other than those with consequent self, all datatype declarations for attributes, candidate key attributes, and all disjointness constraints not associated with any ISA relationship are ignored. Thus, the transformation may not be information-preserving at the model specification level. However, this information may be attached to the ERD as non-graphical constraints later used in other tasks requiring formal representation, such as reasoning or exporting to another language.

We now briefly illustrate a possible reverse engineering of the ARM diagram from Example 2.

Example 3. Continuing from Example 2, we transform that ARM diagram into an EER diagram, with outcome included in Figure 4 [Figure 4: see original paper], following the procedure described above. For instance, take the professor relation from Example 2. By rule I, we create entity type Professor (Rule I-A selects PK and relevant pathfd `pathfd (pnum) -> self`, and notices h=0 (rule I-B), so it is a strong relation), and add pnum as identifier to Professor (rule II). Add other attributes not in FK constraints as regular attributes to Professor, and for the FK attribute (department), create a new binary relationship to Department (rule III). This binary relationship must be named, and rules applied to the department relation for the Department entity type in the EER diagram. Rules IV-VI do not apply because there is no isa in the relation specification. This concludes processing the rule sequence for this ARM relation.

4.2 From EER Diagram to ARM

The algorithm from ERD to ARM follows standard principles, such as FK on the n-side of 1:n multiplicity on a relationship, but distinguishes itself from staple algorithms on three main aspects: pathfds with self for each entity type when generating ARM relations, other pathfd assertions, and especially disjointness constraints among relations for entity types not in an isa hierarchy. These key differences from well-known EER-to-RM algorithms appear in rules I-A, I-C, II-A, II-C, III, IV, V, and VI.

As notation convention, an entity type is denoted by E, its attributes as A1, ⋯,An, and participating in relationships R1,⋯,Rm connecting to E1,⋯,Em in the EER diagram. Transformations assume referring expression machinery to handle identification properly, as described in [9].

I. For each strong E with identifier composed by attributes A1,⋯,Al and regular attributes Al+1,⋯,An (1≤l≤n):

A. Create in Σ ARM relation TE with declaration `table TE(self OID, A1 anyType, ⋯, An anyType, An+1 OID, ⋯, An+m OID, Q1, ⋯, Qp)` where anyType is used until datatype is set by designer or retrieved from reverse engineering step, and An+i (0≤i≤m) are new attributes different from those existing in the ERD.

B. Add to TE FK constraints `constraint Ri foreign key (An+i) references TEi` for all i (0≤i≤m).

C. Add to TE PK constraint `pathfd (A1, ⋯, Al)$\rightarrow$self`.

II. For each weak E with partial identifier composed by attributes A1,⋯,Al and regular attributes Al+1,⋯,An (1≤l≤n), related through weak relations Ri (1≤i≤k≤m):

A. Create ARM relation TE for E such that `table TE(self OID, A1 anyType, ⋯, An anyType, An+1 OID, ⋯, An+m OID, Q1, ⋯, Qp)` where anyType is as before, and An+i (0≤i≤m) are new attributes different from those existing in the ERD.

B. Add to TE FK constraints `constraint Ri foreign key (An+i) references TEi` for all i (1≤i≤m).

C. Add to TE PK constraint `pathfd (A1, ⋯, Al, An+1, ⋯, An+k)$\rightarrow$self`.

III. For each subsumption (E1 ISA E) in the ERD, take the "separation" option for transformation, so add to relation TE1 in ARM the constraint `isa TE`, shorthand for the pathfd constraint that self of E1 is an FK referencing self in E.

IV. For each covering constraint declared in a hierarchy in the ERD, on E1,⋯ ,En (n$ $2) that are entity subtypes of E, add to TE the constraint `covered by (TE1, ⋯, TEn)`.

V. For each disjointness declared in a hierarchy in the ERD, on E1,⋯ ,En (n$ 2)$that are entity subtypes of E, add to each TEi (1 i $n)$ the constraint `disjoint with (TE1, ⋯, TEn)`.

VI. Disjointness between ARM relations constructed from entity types not in a hierarchy in the ERD (and thus implicitly disjoint) can be computed based on comparing their respective pathfd constraints with consequent self for each Ti, because each relation in ARM resulting from an Ei (not in a hierarchy) of the ERD will have a different PK due to different identifiers in the corresponding ERD.

Except for step V, all previous transformation steps are linear in ERD element size since each deals with a specific element type (strong entities, weak entities, isa' s, etc.) and output does not depend on this size. In step V, a single disjointness constraint generates as many constraint clauses as participating

entities, so this step is quadratic. The resulting ARM after this process is a very weak database representation since several implementation details are missing, like attribute types, non-key functional dependencies, and general disjointness constraints. Instantiation can be done automatically with external annotations (not present in the ERD) or manually with expert help.

5. Using KnowID Beyond EER

The "extended ARM+SQLP" option for D→K, i.e., the KnowID architecture shown in Figure 3, covers the knowledge-to-data pipeline, worked out for EER diagrams at the knowledge layer, relational database as structured data, and SQLP as query language. As a generic architecture, two central components require clarification to justify genericity claims: the query component with SQLP, and formal representation beyond EER (addressing other items in the Knowledge and Information Management box in Figure 3). Given paper scope and that mentioned theory and techniques have been introduced elsewhere, we discuss these topics only from an architectural angle, not logic and algorithm details.

5.1 SQLP over ARM and EER Diagrams

Querying data through knowledge is an essential knowledge-to-data pipeline component. SQLP queries have been shown invariant under vertical partitioning, as that amounts to lossless join decomposition [10]. For an SQLP query over an ARM diagram, the query remains exactly the same irrespective of whether the graphical display depicts each class with attributes "inside" the class or with (functional) relations positioned like another class (i.e., graph-like notation), because formally it amounts to the same path formalization thanks to self and OIDs. We visualize this in Figure 2 with two ARM graphical notations: left diagram has RM look-and-feel, while right is flattened, showing attributes as (special) relationships they are.

Because EER-to-ARM (and vice versa) is transformation-based, SQLP for ARM can propagate to its use with EER diagrams. The SQLP query does not use relationship names explicitly, so any naming assignment step for new names in reverse-engineered EER diagrams has no effect on formulated queries. The relevant vocabulary of EER's entity types, relationships, and attributes remains largely the same with respect to ARM, so query shape can be the same. More precisely: the written, textual version of the query can be exactly the same, provided one accounts for ERD relationship names being ignored except for m:n cardinality relationships (which become relations in ARM). Thus, cognizant that CWA is normally assumed with EER already, this can indeed be maintained in KnowID thanks to a transformation-based approach and the option to remain with SQLP. We illustrate this in the following example.

Example 4. Continuing our running example from Example 3 and the ERD of Figure 4, a sample query could be: "Find the distinct names of all professors who work at the CS department that have taught a class of a course offered

by another department." This can likewise be executed over an ARM or EER diagram by availing of the ARM-to-EER transformation link and constructing paths "from-the-n-to-the-1-direction-into-an-attribute." Moreover, this should be easier when queried over the ERD because relationship names present in the EER diagram loosely map onto verbs in the information request. In this case: "Find the distinct names of all professors who work at the CS department that have taught [teach] a class [part] of a course offered by another department." This is not the case in an ARM-as-RM-lookalike diagram. Alternatively, in a graphical interface, one can select the path including relationships in the graphics, making it visually clearer that such a path exists. Such choices are left to interface design stages.

5.2 Flexibility in KnowID Input

We now examine the Knowledge and Information Management box in Figure 3, with its four core sub-processes depending on precise input of "conceptual data model or application ontology C." Input may be a conceptual data model in UML class diagram notation or an Object-Role Modeling (ORM) diagram. Also, several EER formalizations exist in various logics, invariably using OWA for automated reasoning rather than CWA common for database and conceptual modeling settings, which must be addressed.

Those four steps are briefly explained and elaborated below:

1. If C is not in EER, convert it to EER using a mapping or metamodel-driven approach (see, e.g., [31, 32] and references therein).
2. If the EER diagram is not yet formalized, use one of the logic-based reconstructions to formalize it (elaborated below).
3. Taking the formalization as input, compute inferences (deductions); if there are undesirable deductions or unsatisfiable classes, revise the model and classify again; repeat until deductions, if any, are acceptable.
4. Materialize the deductions (if any), i.e., modify the EER diagram by adding acceptable deductions; one may double-check modifications by classifying again, which should return an empty deduction list.

The model resulting from completing step 4 is the one transformed into ARM and used for querying data.

To determine a suitable logic for step 2 or for having the conceptual model formalized initially, note it ought to formalize at least the following modeling language features in a suitable logic, because they can be handled in ARM and/or RM: entity type (weak and strong), n-ary relationship, attribute, basic cardinality constraints (0..n, 0..1, 1, 1..n) and identifier, entity type subsumption, disjointness, and covering/total. Remaining EER features like higher number restrictions (e.g., 2..4) and multivalued attributes generally transfer directly into physical schema only (if at all), and this "feature gap" is thus not specific to this ARM architecture, so it is acceptable to set aside.

A good fit for the desired features list is DLRifd [33] of the Description Logics family, but it is ExpTime-complete in subsumption reasoning. A DLRifd disadvantage is lack of tools, so users likely turn to OWL 2 DL [15] or a fragment thereof [34] and DL reasoners thanks to ample tooling support, despite feature mismatch (among others, no n-aries, no multi-attribute identifiers). KnowID as an approach permits either. For better performance, one could drop covering/total and use CFDI -nc [24], which is in PTime. Others, notably those in the computationally better-behaved DL-Lite family [35], have logic-based EER reconstructions (e.g., [35]) but offer even fewer features to modelers. Because classification reasoning is separated from querying and can be done "offline" (not at runtime), there is no need to opt for low expressiveness.

Regarding step 4, materializing deductions is not new and was a feature in an earlier Protégé ontology development environment [36]; e.g., if A ISA B is deduced but not asserted in the EER diagram, one adds it explicitly even though it is "redundant" because implicitly present. The advantage of materializing all deductions at the knowledge layer is that the reasoning service is no longer needed at runtime, saving processing time. We deem this acceptable since conceptual models or ontologies do not change frequently. Regarding deductions, in OWA they differ from CWA, but research on reasoning over conceptual models widely ignores this difference (i.e., de facto accepts there is no good alternative). Here, we follow the same approach for this not-at-runtime step. Note this does not affect the CWA behavior users expect during system interaction because it is effectively "closed off" with this fourth step.

5.3 Flexibility in KnowID' s Data Layer

The KnowID architecture in Figure 3 has a box labeled "Database schema(s) S" with only one RM listed. Practically, there may be an SQL federator at the database schema layer to provide a global schema at implementation level, as possible with OBDA systems. Such a federator may deliver an RM as global view, which is, for both basic ARM+SQLP and thus also the proposed KnowID, the starting point for principal novelties making up KnowID. Therefore, KnowID' s data storage component is as flexible as current theories, techniques, and tools for relational database systems and any other data storage that can be queried directly or indirectly with SQL.

Secondly, the ARM+SQLP component within KnowID in Figure 3 includes a Data completion step using data and an ARM (which could also be the formalized EER diagram using constraints supported in EER-to-ARM conversion). Data completion informally means computing deductions first, then materializing them in the database. Returning to Example 1 illustrates this process and its operation within KnowID.

Example 5. Recall the database stores IndividualCustomer ⟨Joanne⟩ and the conceptual data model or ontology contains IndividualCustomer ISA Customer. The data completion process fires an algorithm checking data against knowledge.

In this case, it proceeds as:

1. Observation: There is an instance of IndividualCustomer, being {Joanne}, declared in database table IndividualCustomer.
2. Action: Use schema-to-RM-to-ARM transformation logs to look up what the corresponding relation in the ARM model says about IndividualCustomer.
3. Observation: It is a Customer, as readable in the constraint `isa Customer` declared in the IndividualCustomer relation.
4. Action: Deduce that instances of IndividualCustomer are also instances of Customer, availing of isa constraint semantics in the ARM specification.
5. Action: Use ARM-to-RM-to-schema transformation log to append the Customer table (originally generated from Customer) with Customer ⟨Joanne⟩ .

Now the answer to "list all customers" will indeed be {Joanne}, as expected from human understanding.

Trade-offs exist between this data completion approach and on-the-fly query rewriting typically incorporated in K D architecture. Determining the best option for a particular application depends on update frequency, whether optimizations are used in data completion (e.g., incremental [37]), and tolerance for errors due to data out-of-dateness. With (near) continuous updates, data completion tends to be costly because it keeps re-computing and storing implied assertions, but if updates are not continuous or an incremental approach to data completion can be taken [37], it is favored over costly query rewriting, which is exponential in query size [38] compared to data growing only polynomially in size for queries [39], although this polynomial blowup may still be unacceptable for large datasets. Optimization of rematerialization after conceptual model updates is another aspect for future work.

Finally, one could take the SQL result and use R2RML [40] or similar [41] to generate RDF triples for integration with a Web-based knowledge graph, already possible with K D and K D approaches.

6. Evaluation

We evaluated KnowID in two ways:

- Proof-of-concept implementation of transformation rules
- Comparison against OBDA (K D) as main contender, plus K D and D K, dealt with consecutively below

The proof-of-concept tool evaluation aimed to ascertain experimentally whether transformation rules were correct, using test models in EER and ARM covering all cases. A preliminary flexible tool was designed and implemented, taking EER and ARM diagrams serialized in JSON, transforming them at the back-end into an array, applying relevant transformation rules, and returning an ARM

or EER diagram plus a log file recording the transformation (see Figure 5 [Figure 5: see original paper]) [42]; code, examples, and screenshots are available at http://www.meteck.org/KnowID.html. In total, 83 tests were run with the latest version, designed to cover all rules plus unsupported features to verify rejection during transformation. Results showed transformation rules defined in Section 4 are indeed concise and correct. It also clarified that some transformation aspects are easier algorithmically when stated more explicitly in model serialization. A notable example is the isa constraint (rule III in Section 4.2) when computing disjointness among ARM relations (rule VI in Section 4.2): brute-force computation among all pathfd constraints to self with simple FK check is easier to implement than unpacking isa shorthand notation.

We investigated experimental evaluation of KnowID against a K D instance (OBDA), such as Ontop, on one or more task performances. Aside from implementation optimizations, from respective architectures, variables determining overall complexity (recall Table 1), and their strengths, it is easy to construct a setup guaranteeing one or the other wins on performance. Therefore, the comparison evaluation aim here is highlighting crucial factors for a knowledge-to-data system that will make one or the other win when actual instances are compared—i.e., examining practical consequences of theoretical generic differences. The three key factors are:

1. **Data stability:** Do they (i) continuously change a lot throughout the database or (ii) change intermittently, rarely, or append-only?
2. **Schema stability:** Do they (i) remain unchanged once the system is set up or (ii) have to change based on evolving business needs and usage optimizations?
3. **Query types posed over data:** Are they (i) at most (unions of) conjunctive queries (UCQs) or (ii) also other SQL query types (with or without paths)?

If data change continuously, schemas remain stable, and one needs at most UCQs, OBDA will win, whereas for intermittent, rare, or append-only data updates and UCQs+any other SQL (or SQLP) query, KnowID will win (irrespective of schema stability). Reasons are: With many database updates, KnowID consumes many resources re-computing data completion, which is not an issue for OBDA because it performs inferences on-the-fly for that query only. While those on-the-fly inferences are computationally costly, there is a tipping point where they outperform data completion in practice (recall Section 5.3). For query types, it would be easy to construct a user query list including non-UCQ queries, on which OBDA fails outright and KnowID wins since it supports full SQL and thus certainly obtains a 100% pass rate on user queries. Finally, creating the mapping layer in OBDA is resource-intensive with human-in-the-loop, hence time-consuming to maintain when ontology or database schema changes. Conversely, KnowID transformations are computed automatically and can be recomputed on-the-fly when changes occur, whether conceptual or implementation-level, since transformation algorithms are speci-

fied in both directions.

Configuring experiments pitting KnowID against other approaches from Section 2.1—K D with OWL and D K with OntoMIND—would also be easy to make one or the other win, but for different reasons. It is well-known that K D with only an OWL file shows noticeable performance degradation when there are tens or hundreds of individuals declared in the ontology (depending on axioms), precisely why other architectures were devised; this is no real contest. D K implementations typically have no reasoning component, so a query requiring it will lose against KnowID by returning fewer tuples than logically implied (recall {} vs. {Joanne} from Example 1). The one D K system with some reasoning via triggers and stored procedures, OntoMIND [7], does not support n-aries and cardinalities in its ontology language (DL-Lite_R), and also uses data completion. Since DL-Lite_R is a fragment of our EER language, there is less to compute both in the conceptual model and data completion process, so it should be faster but also retrieve fewer tuples than it ought on carefully constructed queries.

Overall, KnowID as a D→K instantiation evaluates positively against K D and D K regarding the knowledge-to-data pipeline in any case, and depending on prospective implementation characteristics, positively against K D as well.

7. Conclusions and Future Work

This paper presented a novel "knowledge to data" pipeline called KnowID, combining new transformation rules between EER and the Abstract Relational Model with recently proposed components for data, information, and query components. KnowID' s main distinctive features are that runtime use can avail of the closed world assumption with an RDBMS, relational and abstract relational model, and EER as declarative specifications, plus full SQL with path queries for expressive compact queries formulated over knowledge.

We are currently investigating the solution space for implementing transformation rules as a first step toward realizing KnowID as a usable and scalable software system. With rules verified working in implementation, we are considering multiple usage scenarios: graphical interfaces for both conceptual model/ontology and ARM model, usability and human readability of transformation log in anticipation of knowledge-based querying with SQLP, and compatibility with other tooling infrastructures. Current code and examples, plus anticipated updates, are available from http://www.meteck.org/KnowID.html.

Author Contributions

P.R. Fillottrani (prf@cs.uns.edu.ar) and C.M. Keet (mkeet@cs.uct.ac.za) jointly developed the theory. C.M. Keet coordinated manuscript writing and devised examples. Both authors made meaningful and valuable contributions in revising and proofreading the resulting manuscript.

Acknowledgements

We thank David Toman for feedback on an earlier draft. We also thank the 20 students of 7 capstone projects for exploring the solution space in implementing transformation rules.

References

[1] T. Catarci & G. Santucci. Query by diagram: A graphical environment for querying databases. *ACM SIGMOD Record* 23(2)(1994), 515. doi:10.1145/191843.191976.

[2] A.C. Bloesch & T.A. Halpin. ConQuer: A conceptual query language. In: *Proceedings of ER' 96: 15th International Conference on Conceptual Modeling*, 1996, pp. 121–133. doi:10.1016/0169-023X(95)00005-D.

[3] G. Xiao, L. Ding, B. Cogrel & D. Calvanese. Virtual knowledge graphs: An overview of systems and use cases. *Data Intelligence* 1 (2019), 201–223. doi:10.1162/dint_a_{00011}.

[4] M. Stonebraker & I.F. Ilyas. Data integration: The current status and the way forward. *IEEE Data Engineering* 41(2)(2018), 3–9. Available at: http://sites.computer.org/debull/A18june/p3.pdf.

[5] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro & G. Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web Journal* 8(3)(2017), 471–487. doi:10.3233/SW-160217.

[6] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini & R. Rosati. Linking data to ontologies. In: Spaccapietra S. (eds.) *Journal on Data Semantics X*, 2008, pp. 133–173. doi:10.1007/978-3-540-77688-8_5.

[7] L. Al-Jadir, C. Parent & S. Spaccapietra. Reasoning with large ontologies stored in relational databases: The OntoMinD approach. *Data & Knowledge Engineering* 69(2010), 1158–1180. doi:10.1016/j.datak.2010.08.001.

[8] F. Zhang, Z.M. Ma & W. Li. Storing OWL ontologies in object-oriented databases. *Knowledge-Based Systems* 76(2015), 240–255. doi:10.1016/j.knosys.2014.12.020.

[9] A. Borgida, D. Toman & G.E. Weddell. On referring expressions in information systems derived from conceptual modeling. In: *Proceedings of ER' 16*, 2016, pp. 183–197. doi:10.1007/978-3-319-46397-1_{14}.

[10] W. Ma, C.M. Keet, W. Oldford, D. Toman & G. Weddell. The utility of the abstract relational model and attribute paths in SQL. In: C. Faron Zucker, C. Ghidini, A. Napoli & Y. Toussaint (eds.) *Proceedings of the 21st International Conference on Knowledge Engineering and Knowledge Management (EKAW' 18)*, 2018, pp. 195–211. doi:10.1007/978-3-030-03667-6_{13}.

[11] M. Junkkari, J. Vainio, K. Iltanenan, P. Arvola, H. Kari & J. Kekäläinen. Path expressions in SQL: A user study on query formulation. *Journal of Database Management* 22(3)(2016), 22. doi:10.4018/JDM.2016070101.

[12] C.M. Keet. *An introduction to ontology engineering*. Available at: http://open.uct.ac.za/bitstream/handle/11427/28312/OEbook.pdf?sequence=1&isAllowed=y.

[13] P.R. Fillottrani & C.M. Keet. Dimensions affecting representation styles in ontologies. In: *The 1st Iberoamerican Conference on Knowledge Graphs and Semantic Web (KGSWC' 19)*, 2019, pp. 186–200. doi:10.1007/978-3-030-21395-4_{14}.

[14] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi & P.F. Patel-Schneider (eds.). *The Description Logics Handbook - Theory and Applications*. 2nd Edition. Cambridge: Cambridge University Press, 2010.

[15] B. Motik, P.F. Patel-Schneider & B. Parsia. OWL 2 web ontology language structural specification and functional style syntax, W3C recommendation. Available at: http://www.w3.org/TR/owl2-syntax/.

[16] D. Toman & G.E. Weddell. *Fundamentals of physical design and query compilation, synthesis lectures on data management*. Williston, VT: Morgan & Claypool Publishers, 2011. doi:10.2200/S00363ED1V01Y201105DTM018.

[17] D. Calvanese, T.E. Kalayci, M. Montali, A. Santoso & W. van der Aalst. Conceptual schema transformation in ontology-based data access. In: C.F. Zucker, C. Ghidini, A. Napoli & Y. Toussaint (eds.) *Proceedings of the 21st International Conference on Knowledge Engineering and Knowledge Management*, 2018, pp. 50–67. doi:10.1007/978-3-030-03667-6_4.

[18] E. Botoeva, D. Calvanese, B. Cogrel, J. Corman & G. Xiao. A generalized framework for ontology based data access. In: C. Ghidini, B. Magnini, A. Passerini & P. Traverso (eds.) *Proceedings of AI*IA' 18*, 2018, pp. 166–180. doi:10.1007/978-3-030-03840-3_{13}.

[19] E. Kharlamov, D. Hovland, M.G. Skaeveland, D. Bilidas, E. Jiménez-Ruiz, G. Xiao ···& A. Waaler. Ontology based data access in Statoil. *Web Semantics: Science, Services and Agents on the World Wide Web* 44(2017), 3–36. doi:10.1016/j.websem.2017.05.005.

[20] A. Artale, D. Calvanese, R. Kontchakov & M. Zakharyaschev. DL-Lite without the unique name assumption. In: *Proceedings of the 22nd International Workshop on Description Logic (DL 2009)*, 2009, pp. 1–13. Available at: http://ceur-ws.org/Vol-477/paper_{11}.pdf.

[21] N.E. Fuchs, K. Kaljurand & T. Kuhn. Discourse representation structures for ACE 6.6, Technical Report, ifi-2010.0010, Department of Informatics, University of Zurich, Switzerland, 2010. Available at: http://attempto.ifi.uzh.ch/site/pubs/papers/drs_{report}_{66}.pdf.

[22] D. Calvanese, C.M. Keet, W. Nutt, M. Rodríguez-Muro & G. Stefanoni. Web-based graphical querying of databases through an ontology: the WONDER system. In: S.Y. Shin, S. Ossowski, M. Schumacher, M.J. Palakal & C.C. Hung (eds.) *Proceedings of ACM Symposium on Applied Computing (ACM SAC' 10)*, 2010, pp. 1389–1396. doi:10.1145/1774088.1774384.

[23] A. Soylu, E. Kharlamov, D. Zheleznyakov, E.J. Ruiz, M. Giese, M. Skjaeveland, D. Hovland ···& I. Horrocks. Optiquevqs: A visual query system over ontologies for industry. *Semantic Web* 9(5)(2018), 627–660. doi:10.3233/SW-180293.

[24] D. Toman & G.E. Weddell. On adding inverse features to the description logic CFD8nc. In: *PRICAI 2014: Trends in Artificial Intelligence - 13th Pacific Rim International Conference on Artificial Intelligence*, 2014, pp. 587–599. doi:10.1007/978-3-319-13560-1_{47}.

[25] J.S. Jacques, D. Toman & G.E. Weddell. Object-relational queries over CFDInc knowledge bases: OBDA for the SQL-Literate. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 1258–1264. Available at: http://ceur-ws.org/Vol-1577/paper_{10}.pdf.

[26] T. Halpin & T. Morgan. *Information modeling and relational databases*. 2nd Edition. San Francisco, CA: Morgan Kaufmann, 2008.

[27] P.R. Fillottrani, C.M. Keet & D. Toman. Polynomial encoding of ORM conceptual models in CFDI -nc. In: D. Calvanese & B. Konev (eds.) *Proceedings of the 28th International Workshop on Description Logics (DL' 15)*, 2015, pp. 401–414. Available at: http://ceur-ws.org/Vol-1350/paper-50.pdf.

[28] C.M. Keet & P.R. Fillottrani. An ontology-driven unifying metamodel of UML Class Diagrams, EER, and ORM2. *Data & Knowledge Engineering* 98(2015), 30–53. doi:10.1016/j.datak.2015.07.004.

[29] P.R. Fillottrani & C.M. Keet. Evidence-based languages for conceptual data modeling profiles. In: T. Morzy et al. (eds.) *The 19th Conference on Advances in Databases and Information Systems (ADBIS' 15)*, 2015, pp. 215–229. doi:10.1007/978-3-319-23135-8_{15}.

[30] R.H. Chiang, T.M. Barron & V.C. Storey. Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data & Knowledge Engineering* 12(2)(1994), 107–142. doi:10.1016/0169-023X(94)90011-6.

[31] P.R. Fillottrani & C.M. Keet. Conceptual model interoperability: A metamodel-driven approach. In: A. Bikakis et al. (eds.) *Proceedings of the 8th International Web Rule Symposium (RuleML' 14)*, 2014, pp. 52–66. doi:10.1007/978-3-319-09870-8_4.

[32] Z.C. Khan, C.M. Keet, P.R. Fillottrani & K. Cenci. Experimentally motivated transformations for intermodal links between conceptual models. In: J. Pokorny' et al. (eds.) *The 20th Conference on Advances in Databases and*

Information Systems (ADBIS' 16), 2016, pp. 104–118. doi:10.1007/978-3-319-44039-2_8.

[33] D. Calvanese, G. De Giacomo & M. Lenzerini. Identification constraints and functional dependencies in description logics. In: B. Nebel (ed.) *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, 2001, pp. 155–160.

[34] B. Motik, B.C. Grau, I. Horrocks, Z. Wu, A. Fokoue & C. Lutz. OWL 2 Web Ontology Language Profiles, W3C recommendation (27 Oct. 2009). Available at: http://www.w3.org/TR/owl2-profiles/.

[35] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov & M. Zakharyaschev. Reasoning over extended ER models. In: C. Parent et al. (eds.) *Proceedings of the 26th International Conference on Conceptual Modeling (ER' 07)*, 2007, pp. 277–292. doi:10.1007/978-3-540-75563-0_{20}.

[36] J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson ···& S.W. Tu. The evolution of Protégé: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies* 58(1)(2003), 89–123. doi:10.1016/S1071-5819(02)00127-1.

[37] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu & J. Banerjee. Rdfox: A highly scalable RDF store. In: M. Arenas (ed.) *Proceedings of the International Semantic Web Conference (ISWC' 15)*, 2015, pp. 3–20. doi:10.1007/978-3-319-25010-6_1.

[38] G. Gottlob, S. Kikot, R. Kontchakov, V.V. Podolskii, T. Schwentick & M. Zakharyaschev. The price of query rewriting in ontology-based data access. *Artificial Intelligence* 213(2014), 42–59. doi:10.1016/j.artint.2014.

[39] C. Lutz, D. Toman & F. Wolter. Conjunctive query answering in the description logic EL using a relational database system. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI' 09)*, 2009, pp. 2070–2075. Available at: http://ijcai.org/Proceedings/09/Papers/341.pdf.

[40] S. Das, S. Sundara & R. Cyganiak. R2RML: RDB to RDF mapping language. Available at: https://www.w3.org/TR/r2rml/.

[41] Ó. Corcho, F. Priyatna & D. Chaves-Fraga. Towards a new generation of ontology based data access. *Semantic Web* 11(1)(2020), 153–160. doi:10.3233/SW-190384.

[42] P.R. Fillottrani, S. Jamieson & C.M. Keet. Connecting knowledge to data through transformations in KnowID: system description. Submitted to a journal, 2019.

Author Biography

Pablo Rubén Fillottrani is a Professor with the Department of Computer Science and Engineering, Universidad Nacional del Sur, Bahía Blanca, Argentina,

and independent researcher at Comisión de Investigaciones Científicas de la Provincia de Buenos Aires. He heads LISSI, Software Engineering and Information Systems R&D Lab. His research interests are in ontology engineering, semantic Web, knowledge representation, and information integration with applications in digital government and software engineering. Pablo obtained his PhD at Universidad Nacional del Sur. ORCID: 0000-0003-0906-867X

C. Maria Keet is an Associate Professor with the Department of Computer Science, University of Cape Town, South Africa. Her research interests are in knowledge engineering, including ontology engineering, natural language generation, and ontology-driven conceptual modeling, resulting in over 100 publications. She wrote a textbook on ontology engineering. Maria obtained her PhD at the KRDB Research Centre, Free University of Bozen-Bolzano, Italy, in 2008. She also worked as systems engineer in the IT industry for three and a half years. ORCID: 0000-0002-8281-0853

Note: Figure translations are in progress. See original paper for figures.

Source: ChinaXiv —Machine translation. Verify with original.