Serdica J. Computing **13** (2019), No 3–4, 123–138

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

A SET OF CONCEPTUAL GUIDELINES FOR REVERSE ENGINEERING USING ONTOLOGY-BASED MATERIALIZED VIEWS

Oumar Sy

Abstract. Data reuse and meta-data handling remain tricky problem for both the designers and managers, especially when schemas reverse engineering is on demand and that some data are stored in materialized views. In this paper, we tackle such problem by using ontology-based meta-materialized views. Indeed, ontologies, which are semantics-based, ensure the stability of the underlying schemas of the data repositories and to ease the overall access and processing of the data and meta-data. Our proposal is sustained by a set of conceptual guidelines and outlined through a case study example.

1. Introduction. Legacy databases and advanced databases systems, on one hand, and the ontologies, on the other hand, must permanently be up to date. In such context, reverse engineering is helpful for the understanding of these systems. However, data reuse remain tricky problem for both the designers and managers, because of the need of handling meta-data. Nonetheless, already in the early 90s, CASE (Computer-Aided systems-Engineering) tools were used to maintain and enhance existing systems [1].

ACM Computing Classification System (1998): H.2.1, H.2.2, H.2.3, H.2.8, K.6.1.
Key words: relational database, conceptual modelling, reverse engineering, materialized view, ontology.

Fortunately, the use of ontologies for creating more intelligent and effective enterprise information systems has increased considerably in recent years [2]. Indeed, ontologies allow the specification of high-level formal conceptualizations [3]. However, most of the proposed approaches, e.g., [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], are not domain independent. Therefore, data reuse is not easy to achieve or not achieved at all because of the common trend to approach the problem of reverse engineering with object-oriented techniques.

Indeed, according to [3], reverse engineering consists of producing a semantic model based on the code in which the data model or the knowledge model (e.g., an ontology) is implemented. Moreover, to the best of our knowledge, only [9, 10, 11, 13, 14] have dealt with the problem of ontologies' materialization. However, rather than the use of materialized views with ontologies for reverse reengineering, the authors only discussed ontologies' storage, without conceptualization of materialized views in the design process, and it is not clear how ontologies' models are linked to the business-data. Conceptual modelling approaches of ontologies for data integration and reuse have been well developed in [4, 15, 16], but materialized views were not considered. In [6] ontologies are not mentioned. In addition, despite efforts made in the field of reverse engineering in [3], ontologies' storage within DB repositories has not been sufficiently investigated.

In brief, without loss of generality, the posed problematic is how reverse engineering – rebuilding – the semantic-based data model, e.g. with EER or UML, of a given universe U. On this basis, we advocate that the most reliable solution reside on the use of ontologies, assuming the existing of a well-designed meta-model that links the business-data to the ontologies. Accordingly, we tackle the problem of reverse engineering from the semantic models point of view, by focusing our concern on ontologies-based materialized views. To this end, we propose an approach powered by a set of conceptual guidelines for schemas reverse engineering by using ontology-based materialized views for reverse engineering, we name Ontology-Based Materialized Views for Reverse Engineering (OBMV4RE).

Starting example: Let U = {R, E, L} where, respectively, R, E, and L represent Rice Seed, Cereal Seed and Plot-land, and U is the universe of discourse of the application domain.

Let VARIETY (V), STAMP (S), PURITY (P) and GERMINATION (G) be the properties of E, as shown in Fig. 1. Knowing the value of S and P, the database manager may update column G of the underlying table E, using the following instruction:

```
UPDATE E SET G=93 WHERE (S=‘J’) AND (P=’A’);
```

Moreover, one would like to propagate those update to all data related with the updated table. This is feasible through triggers, based on keys and foreign keys. A simplified syntax of such triggers is as follows:

```
CREATE TRIGGER <trigger_name>
BEFORE | AFTER INSERT |UPDATE | DELETE
ON <table_name> FOR EACH ROW
 <trigger_body>
```

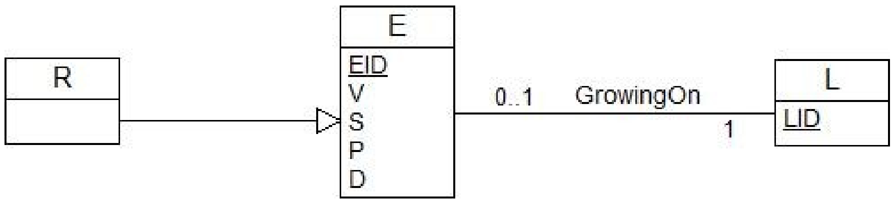


Fig. 1. UML sub-schema of Cereal Seeds ontology

Therefore, let R_i (K_i, A_i, ..., A_k) and R_j (K_j, B_i, ..., B_l, #K_i) be two n-ary relations that represent the concepts C_i and C_j of a given UML-based semantic model, with $C_i \leq C_j$ (meaning that C_j inherits of all the properties and features of C_i). Clearly, $T_j \subseteq T_i$, where T_i and T_j are respectively instances of R_i and R_j.

Now, let us denote by M_v (EID, V, G, LID) a materialized view where LID represents the plot-land identifier. Accordingly, a state of M_v contains the instances of all cereal seeds and plot-lands where they have been seeded. ∎

The rest of the paper is organized as follows. In Section 2, we give a theoretical background comprising conceptual foundations of the notions of view and of materialized view, the notion of ontology, and their use in schemas reverse engineering process and reengineering, as well. Section 3 presents the related work. In Section 4, we discuss the OBMV4RE approach based on a set of conceptual guidelines, and present a case study overview. Section 5 concludes and outlines the direction of our future work.

2. Background knowledge.

In this section, we present the notions of reverse engineering and materialized views.

2.1. Reverse engineering.

Reverse engineering originates from the hardware system analysis [1]. In the field of software systems, several formulations have been proposed for defining the notion of reverse engineering. Moreover, according to [1], there is a confusion between reengineering and ‘‘reverse engineering”. In the aim of clarification, these authors proposed a categorization of the notion of “reverse engineering” using six terms:

1. *Forward engineering* — Forward engineering is the traditional process of moving from high-level abstractions and logical, implementation—independent designs to the physical implementation of a system;
2. *Reverse engineering* — *Reverse engineering* is defined as “a process of analyzing a subject system to identify the system’s components and their interrelationships and to create representations of the system at a higher level of abstraction”;
3. *Redocumentation* — *Redocumentation* is the creation or revision of a semantically equivalent representation within the same relative abstraction level;
4. *Restructuring* — *Restructuring* is the transformation from one representation form to another at the same relative abstraction level, preserving the subject system’s external behavior;
5. *Design recovery* — *Design recovery* is a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify meaningful higher level abstractions beyond those obtained directly by examining the system itself;
6. *Reengineering* — *Reengineering* is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

The relationships between these terms have been rigorously related and defined based on a three-level’s abstraction architecture (see Fig. 2.), conceptually quite similar to the ANSI/X3/SPARC architecture dedicated to databases’ schemas design.

In [2], the notion of reverse engineering is defined as “a process which includes the extraction of information from source codes, and documentations; the abstraction of the extracted information; and the representation of the obtained abstraction”.

However, according to [17], there is no universally accepted definition of software re-engineering, commonly called reverse engineering.

Thus, in our concern, we consider reverse engineering as the process for the production of a conceptual model based on the code and or the metadata in which the “system object”, namely both the data and the ontology, has been implemented. Moreover, we emphasize that “*reverse engineering*” is the opposite process of *"forward engineering"* which is the traditional top-down process for databases' schemas design starting from the requirements analysis to the internal level, based on the ANSI/X3/SPARC architecture.

More precisely, "*reengineering*" is a process between or within abstraction levels while "*reverse engineering*" is a process that covers the entire three-level architecture, starting from the internal level up to the views' level, i.e. the well-known external level, also called level of views according to the ANSI architecture.

For a better understanding, borrowing the schema given in [1], these two processes are illustrated in Fig. 2.

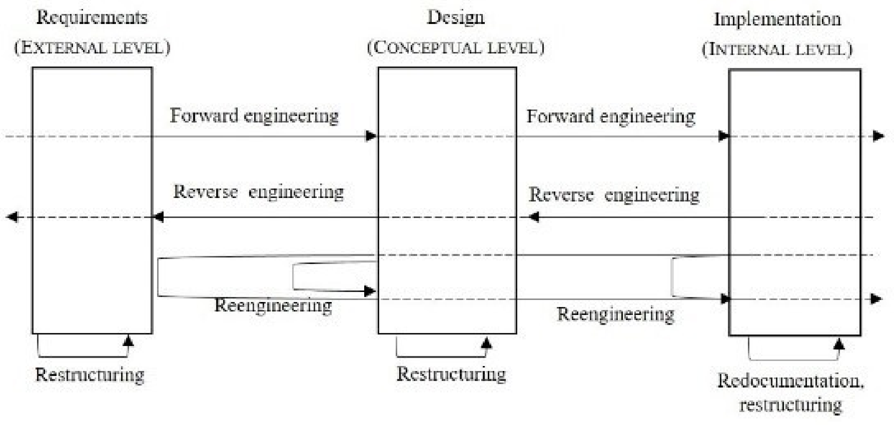


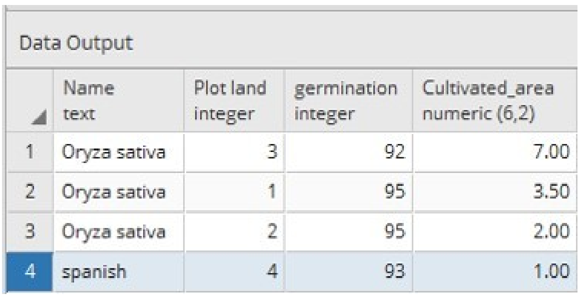
Fig. 2. Reverse engineering and related processes between or within abstraction levels [1]

Such an architecture stands as the cornerstone in our approach, when aiming to conciliate databases (DB) and ontologies, without redefinition of terms, concepts and methodologies [18, 19]. In [19] a meta-model designed in a single lifecycle efficiently ensure the reverse engineering of both the DB and the domain ontology by using materialized views because of its high semantic-based abstraction degree. Indeed, a suitable approach for reverse engineering is to store the roles' names in the underlying DB [19], even though UML does not provide an "explicit marking of two arbitrary associations as inverses of each other" [16].

2.2. Data view vs. materialized view. A typical data view is a simple named SPJ query executable by the database system. Such named query, which consist of a set of n-ary tuples, is launched each time a user accesses it. For example, the SQL code below returns, if any exists, the name N, the germination G and the cultivated area of all rice seeds and the plot of lands where they are growing.

```
        SELECT "N" As "Name", "LID" AS "Plot land", "G" As
Germination, "Cultivated_area" FROM "R", "E"
        WHERE "R"."EID"="E"."EID";
```

The relational data view [20] is shown at Fig. 3.



Data Output

	Name text	Plot land integer	germination integer	Cultivated_area numeric (6,2)
1	Oryza sativa	3	92	7.00
2	Oryza sativa	1	95	3.50
3	Oryza sativa	2	95	2.00
4	spanish	4	93	1.00

Fig. 3. Data view dvGrowingSeeds

Now, let assume that a new tuple of an existing seed variety, e.g. corn, has been sowed in the plot of land number 5. Subsequently, whenever some data are added, the user must re-run the query dvGrowingSeeds above, e.g. for knowing the current state of the cultivated cereals and the data they relate to.

Better still, according to [17, 21, 22], "all views that are theoretically updatable must be updated by the system". Thus, materialized views can be useful by caching the result of queries and allow refreshing them, once a user needs to access their content.

As example, the dvGrowingSeeds data view, theoretically updatable, and that we call mvGrowingSeeds, is materialized as follows:

```
        CREATE VIEW public."mvGrowingSeeds"
        WITH (check_option=local) AS
         SELECT "N" As "Name", "LID" AS "Plot land", "G" As
Germination, "Cultivated_area" FROM "R", "E"
         WHERE "R"."EID"="E"."EID";
```

The above view is easily maintainable up to date based on foreign keys and triggers. However, difficulties can occur for the practical updatability with respect to the data and meta-data, especially when the queries become more complex. In such cases, the answer may take more time. Thus, novelty in the methodologies, rather than in the concepts, is necessary for efficient solutions. Nevertheless, such complex queries are not in consideration in this work. Our proposal is the use of ontology-based materialized views to permit the reverse engineering, for recovering or improving conceptual schemas.

As an example, let us assume that we want to add seed of wheat (W) and, at the same time, a new tuple of sowed wheat. Given the sub-schema (see Fig. 4) that describes the context, the materialized view above (mvGrowingSeeds) is still updatable and it allows the rebuilding, i.e. reverse engineering, the semantic model of Fig. 1., which by the same time represents its external schema.

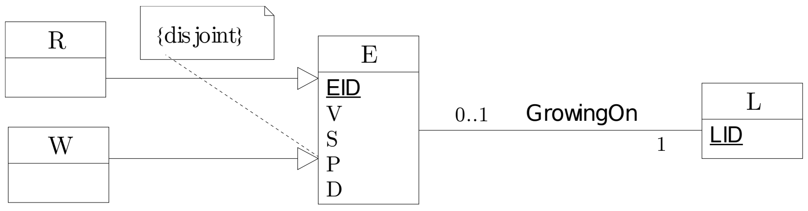


Fig. 4. UML sub-schema of Cereal Seeds ontology (updated)

3. Related work. As previously emphasized, only [9, 10, 11, 13, 14] tackled ontologies' materialization. Thus, we concentrate ourselves on these closely related works to ours and give a precise insight in the following.

Globally, in [9, 10, 11], reasoning over "large ABoxes" by "abstraction refinement" was the focus. More precisely, arguing that "updates of ontology definitions are equivalent to the updates and new definitions of rules, whereas existing maintenance techniques only address the update of ground facts", these works "present a technique for the incremental maintenance of materialized ontologies".

Mainly, the above works are based on logic databases, using "Description Logic Programs (DLP)" as equivalent to the "OWL's Abstract Syntax" [9, 10, 11].

For illustration, let us consider the following properties at Table 1, where the first column contains OWL's representations and those on the second column, the DLP's representations.

Table 1. "OWL's Abstract Syntax" vs. "Description Logic Programs (DLP)"

N°	OWL	DLP
1	domain(C_i)	C_i(x):-P(x, y).
2	range(R_j)	R_j(y):-P(x, y).
3	Transitive	P(x, z):-P(x, y), P(y, z).

However, according to our definition [19] formalizing an ontology as $O^D = \{\Sigma, \tau_{\leq}, A, \Omega, \rho, \varphi\}$, these properties fully hold as follows:

1. To each relation R_i the function ρ assigns a domain ρ_{dom}: $R_i \rightarrow \Sigma \times \Sigma$;
2. To each relation R_i the function ρ assigns a range ρ_{range}: $R_i \rightarrow \Sigma \times \Omega$;
3. The transitive property (line 3) is abstracted with the subsumption relation $\tau_{\leq}$ such that if $\tau_{\leq}(C_i, C_j)$ and $\tau_{\leq}(C_j, C_k)$ then $\tau_{\leq}(C_i, C_k)$.

Indeed, RDB's Attributes (Columns) and Foreign keys are equivalent to Resource Description Framework (RDF)'s Property (rdfs: Domain | rdfs: Range), while Relations (Tables) correspond to RDF's classes (rdfs: Class). Better still, each concept C_i either is represented by a unary relation, or relates to a binary relation. All these foundations of our approach are recalled in Section 4.1, and are sustained with Proposition 1 and corollaries 1, 2, 3, as well as with a semantic UML data model.

Further, for differentiating TBox axioms (intensional predicates) from ABox assertions (extensional rules), we emphasize that the former are represented by unaries and binaries relations, while assertions simply correspond to tables' facts. We also point out that all other properties, such as roles, functional dependencies, domain and or functional constraints and multiplicities are encompassed as semantics in the UML data model, which stands as conceptual schema of the TBox. We recall that TBox means Terminological Box, and ABox refers to Axioms Box.

Moreover, in [9, 10, 11], on one hand, materialized views were not used for reverse reengineering, and on the other hand, the content – state – of an ontology ABox, obviously, is larger than the number of axioms in its TBox. Besides, the comparison that these authors made between "the number of different concept names" and "the number of different individuals" is fuzzy.

Incremental maintenance of materialized ontologies was also discussed in [13, 14].

The work in [13] focused on reasoning over ontologies through materialized views. However, the work is especially declined for the Web Semantic ontologies, namely RDF/RDFS and the well-known Ontology Web Language (OWL), while our interest relates to RDBs and conceptual modeling.

In [14], the authors are mainly interested on ontology-based materialization views, but the technical approach is still based upon logic databases, and the ontology-based materialized views are not used or related to schema reverse engineering as in [9, 10, 11], whereas such purpose is the main goal of this paper.

Nonetheless, as demonstrated in [19], the semantic meta-model we conceived is suitable for some kind of reasoning by using the roles and the multiplicities which are fundamental features in semantic models. The main difference between RDBs and Semantic Web Languages (SWL) such as OWL resides in the fact that the former are Close World Assumption (CWA) oriented while OWL is mainly used for reasoning under the Open World Assumption (OWA).

Accordingly, we propose a set of conceptual guidelines for OBMV4RE with the aim to create the missing link between the use of materialized views, ontologies, and reverse engineering. Those guidelines are sustained with theoretical foundations, as depicted in the following section.

4. OBMV4RE: Toward a set of conceptual guidelines.

4.1. Ontology-based materialized view. In the Web Semantic community literature, ontology is defined as a "formal, explicit specification of a shared conceptualization" [23]. However, there is no consensually accepted definition of the notion of ontology. Thus, in this paper, an ontology is understood as "a formalization of the universe of the discourse as an organized structure, and constrained by a set of axioms, according to the knowledge domain" [19]. Thus, following [24], we defined an ontology as "a formalization of the universe of the discourse as an organized structure, and constrained by a set of axioms, according to the knowledge domain" [23]. More precisely, we advocate an ontology as a 6-tuple set, formalized as $O^D = \{\Sigma, \tau_{\leq}, A, \Omega, \rho, \varphi\}$, where:

- Σ is a set of concepts $\{C_1, ..., C_n\}$ of the knowledge domain, and assigned with the subsumption relation $\tau_{\leq}$ (C_i, C_j) abstracting the Is-A relationship;
- A is a set of attributes of the universe of discourse, describing of the concepts in Σ;
- Ω is a set of unary and binary relations;
- ρ is a relation over Ω assigning to each $R_i \in \Omega$ a domain ρ_{dom}: $R_i \rightarrow \Sigma\times\Sigma$ and a range ρ_{range}: $R_i \rightarrow \Sigma\times\Omega$;
- φ is a set of axioms and rules that Σ and Ω must hold.

Thus, based on our above definition, we give an insight of the domain ontology as a "system of categories" through the following example.

Example: Let Σ = {Maintainer, Rice seed, Cereal seed, Corn seed, Plot land, Rice crop, Farm land, Research station, Seed Variety, E, Contractual, Private Seed Operator} The semantic UML data model at Fig. 5 give an insight of the domain ontology and the data as well.

Consequently, an ontology-based materialized view (OBMV), is a view built on the basis of O such that it contains semantic-based knowledge, e.g. rice seed is a cereal seed, or rule-based knowledge, e.g. a plot of land is either a "rice crop", a "farm land" or a "research station". Now, since views are built upon concept-based relations over Ω which involve attributes of A, let F^c be the FROM clause of the SPJ query. We define an OBMV as follows.

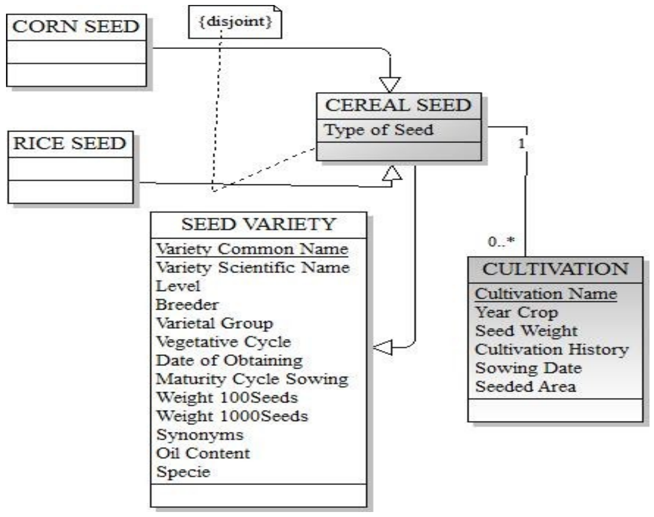


Fig. 5. Ontology-Business data subschema: cultivated cereal seeds

Definition — A view V materialized with the CREATE VIEW instruction is an OBMV if and only if F^c satisfies the following condition:

$$F^c \supset R_i \in \Omega \wedge \exists C_j \in \Sigma \mid \tau_{\leq} (C_i, C_j) \vee \tau_{\leq} (C_j, C_i) \qquad (1)$$

This means that F^c contains at least one concept-based relation and it exist a concept C_j such that $C_j \tau_{\leq} C_j$ or conversely.

Proposition 1 — Based on the properties of the subsumption relation $\tau_{\leq}$, an OBMW is an updatable view.

Corollary 1 — A well-defined OBMV is an OBMV such that its F^c contains two concept-based relations R_i, $R_j \wedge \exists C_i, C_j \in \Sigma \mid \tau_{\leq} (C_i, C_j) \vee \tau_{\leq} (C_j, C_i)$.

Better yet, the semantic model M of an ontology contains at least one taxonomic relation.

Corollary 2 — The semantic external model M^{OV} of an OBMV can be rebuilt using reverse engineering based on properties of the subsumption relation $\tau_{\leq}$.

Corollary 3 — The ontology-based model M^O of a given universe Σ can be reconstructed by reverse reengineering according to a unique well-defined OBMV and M^{OV}.

The foundation of the above Proposition and corollaries relies on the relational schemas R_i (K_i, A_i, ..., A_k) and R_j (K_j, B_i, ..., B_l, #K_i) that represent

C_i and C_j, where $C_i \leq C_j$ (or conversely) and allowing the underlying UML-based sub-model to be redesigned according to reverse engineering principles. Moreover, the taxonomic relation $\tau_{\leq}$ intrinsically implements the well-known functional dependencies (FD) in RDBs. Better still, any FD graph constitutes a theoretic access structure to the DB content, including the met-data of the DB system. Thus, the relational database (RDB) schema can be rebuilt too.

4.2. OBMV4RE: A set of conceptual guidelines. In the aim to develop an efficient methodology for reverse engineering using ontology-based materialized views, we propose a first set of useful conceptual guidelines as follows:

- *Guideline 1*: We assume that the DB catalog contains meta-data related to the ontology meta-model [19], which completes and enriches the DB schema.
- *Guideline 2*: Build an OBMV $M^{OV.}$

Such an ontology meta-model called KBSM (Knowledge-based Semantics Data Model) was proposed in [19].The KBSM (see Fig. 6) is a common meta-schema establishing a logical link between RDBs and ontologies.

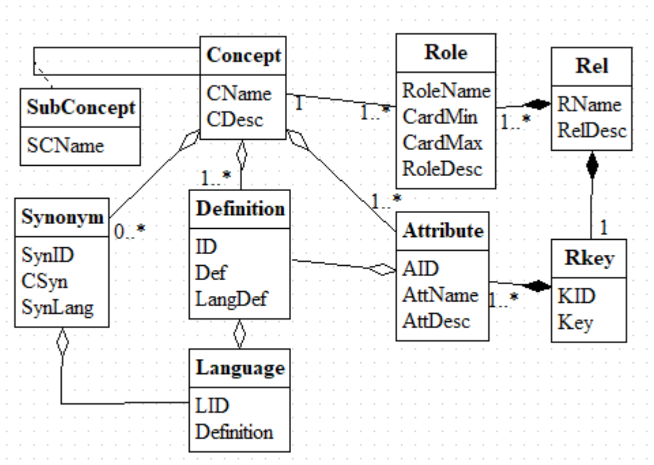


Fig. 6. Knowledge-based Semantics Data Model (KBSM) of Domain ontologies [19]

- *Guideline 3*: Based on Corollary 2, building of a semantic external model of M^{OV}, by reverse engineering;
- *Guideline 4*: According to Corollary 3, and based on the meta-data of the DB catalog enriched by the KBSM whose stored instances

contain relationships' names, roles' names and multiplicities that are the core of a conceptual model, re-build the global model M^O.

4.3. OBMV4RE: A case study example. An overview of our application domain (certified cereal seeds) is given at Fig. 7. In this context, e.g., given a variety V_i of certified rice seed, one would like to know its breeder (maintainer) and the plot of land where it is growing or where it has been produced.

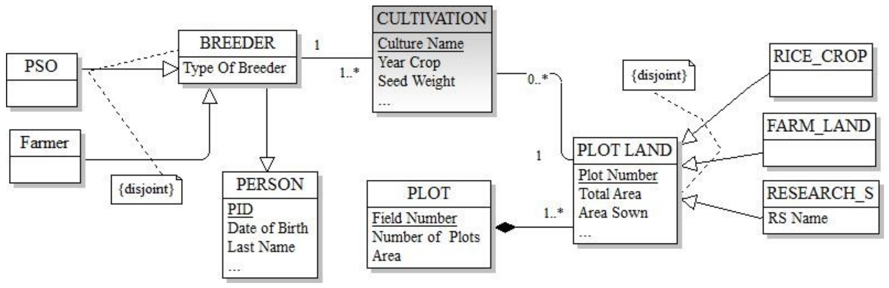


Fig. 7. Ontology-Business data subschema: A case study overview

Accordingly, let mvBreeders be the materialized view that stores all farmers (name, country, e-mail) who are rice seed breeders (rice name). The SQL code is the following:

```
        CREATE VIEW public."mvBreeders"
        WITH (check_option=local) AS
         SELECT "LName", "Email" FROM "Breeder" As "B"
        WHERE EXISTS (SELECT 1 FROM "Cultivation" AS "C"
  WHERE "C"."PID"="B"."PID" AND "B"."TYPE"='F');
```

As the relation Breeder in the FROM clause of the mvBreeders view is concept-based and it is subsumed by the concept Person, then mvBreeders is an OBMV. Thus, mvBreeders is updatable. Furthermore, based on the KBSM's DB (see Fig. 6) and corollaries 2 and 3, the semantic models of Fig. 5 and Fig. 7 can be rebuilt.

5. Conclusion. In this paper, we tackled the tricky problem in information systems and software systems maintenance, namely data and meta-data reuse, especially schemas reverse engineering based on ontology-based materialized views.

Thus, we first gathered the literature for a suitable support of related works. Next, we selected the works which are concerned with at all or at least one of our concern, namely the notions of "ontology", "materialized views" or "reverse engineering". Better still, we have shown that most of the works, which have dealt with one or more of these topics, were not inclusive, because the reverse engineering based on ontologies materialized was left out, but nonetheless on demand. Accordingly, we only retained those related to both the ontologies and the materialized views, in the aim to point out the differences between these works and ours. Moreover, we clarified one again the use and the signification of the term "reverse engineering" often confused with reengineering.

Finally, for improving the process of reverse engineering, we proposed useful methodology named OBMV4RE to overcome the existing drawbacks. Such our proposal is based on a set of conceptual guidelines with theoretical bases and supported with a coherent semantic metamodel with which they are compatible and fully applicable to ensure the reverse engineering of any database schema integrated to its underlying domain ontology.

On this basis, examples of data models are given in UML and SQL code. However, these guidelines are not yet implemented as a complete system. Thus, the next step in our future work is the experimentation of their theoretical foundations based of advanced conceptual schemas as ontology-based ones, for allowing full comparison with existing methods. To this end, we started with the .NET Framework and the C# language. Indeed, the ADO.NET framework allows the creation and the distribution of shared data, by using the DataSet which is one of core elements of the Framework. Better still, XML that is used for data exchange over the Web is the default serialization format of ADO.NET, such that we are looking at taking the data sources to the Web.

REFERENCES

[1] Chikofsky E., J. Cross. Reverse engineering and design recovery: A Taxonomy. *IEEE Software*, **7** (1990), No 1, 13–17.

[2] Reynares E., M. L. Caliusco, M. R. Galli. A set of ontology design patterns for reengineering SBVR statements into OWL/SWRL ontologies, *Expert Systems with Applications*, **42** (2015), 2680–2690.

[3] Gómez-Pérez A., D. Rojas-Amaya. Ontological Reengineering for Reuse. In: D. Fensel, R. Studer (eds). Knowledge Acquisition, Modeling

and Management. EKAW'99, *Lecture Notes in Computer Science*, **1621** (1999), 139–156.

[4] Ristic S., S. Aleksi, M. Celikovi, V. Dimitrieski, I. Lukovi. Database reverse engineering based on meta-models. *Central European Journal of Computer Science*, **4** (2014), No 3, 150–159.

[5] Zhang Q., A. Karcher. System Reverse Engineering to Requirements and Tests. In: Proceedings of the Seventh International Conference on Systems (ICONS), Saint Gilles, Réunion Island, 2012, 35–38.

[6] Müller H. A., J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, K. Wong. Reverse engineering: A roadmap. In: Proceedings of the Conference on the Future of Software Engineering (ICSE'00), Limerick, Ireland, 2000, 47–60.

[7] Shatnawi A., A.-D. Seriai, H. Sahraoui, Z. Alshara. Reverse engineering reusable software components from object-oriented APIs, *Journal of Systems and Software*, **131** (2016), No C, 442–460. DOI: 10.1016/j.jss.2016.06.101.

[8] Ali M. A., A. A. A. Fernandes, N. W. Paton. MOVIE: An incremental maintenance system for materialized object views. *Data & Knowledge Engineering*, **47** (2003), No 2, 131–166.

[9] Glimm B., Y. Kazakov, T. Liebig, T.-K. Tran, V. Vialard. Abstraction Refinement for Ontology Materialization. In: P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz, C. Goble (eds). The Semantic Web—ISWC 2014. *Lecture Notes in Computer Science*, **8797** (2014), 180–195.

[10] Brenner M., B. Glimm. Incremental Materialization Update via Abstraction Refinement, In: Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, 2017. `http://ceur-ws.org/Vol-1879/paper19.pdf`, 15 July 2020.

[11] Glimm B., Y. Kazakov, T.-K. Tran. Ontology Materialization by Abstraction Refinement in Horn *SHOIF*. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, 2017, 1114–1120.

[12] Koch C., D. Lupei, V. Tannen. Incremental View Maintenance For Collection Programming. In: Proc. of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'16), San Francisco, CA, 2016, 75–90. DOI: 10.1145/2902251.2902286.

[13] Volz R., S. Staab, B. Motik. Incremental Maintenance of Materialized Ontologies. In: R. Meersman, Z. Tari, D. C. Schmidt (eds). On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. *Lecture Notes in Computer Science*, **2888** (2003), 707–724.

[14] Volz R., S. Staab, B. Motik. Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases. In: S. Spaccapietra, E. Bertino, S. Jajodia, R. King, D. McLeod, M. E. Orlowska, L. Strous (eds). Journal on Data Semantics II. *Lecture Notes in Computer Science*, **3360** (2005), 1–34.

[15] Alberts R., E. Franconi. An Integrated Method using Conceptual Modelling to Generate an Ontology-based Querying Mechanism. In: 9th OWL: Experiences and Directions Workshop (OWLED), Heraklion, 2012. `http://webont.org/owled/2012/papers/paper_9.pdf`, 15 July 2020.

[16] Zedlitz J., N. Luttenberger. Conceptual Modelling in UML and OWL-2. *International Journal on Advances in Software*, **7** (2014), No 1–2, 182–196.

[17] Rick F. van der Lans. Is Actian PSQL a Relational Database Server? A Technical Whitepaper. R20/Consultancy, 2014. `https://www.r20.nl/WhitepaperPSQLMarch2014V2.pdf`, 15 July 2020.

[18] Sy O., M. Lo, D. Duarte. Integrating Ontologies in Database Scheme: Ontology-Based Views Conceptual Modeling. In: Proceedings of the 6th International Conference on Signal-Image Technology and Internet-Based Systems, SITIS, Kuala Lumpur, Malaysia, 2010, 269–276.

[19] Sy O., D. Duarte, G. Dal Bianco. Ontologies and Relational Databases Meta-Schema Design: A Three-Level Unified Lifecycle. In: 5th International Conference on Control, Decision and Information Technologies (CoDIT), Thessaloniki, Greece, 2018, 518–523.

[20] Codd E. F. A relational model of data for large shared data banks. *Communications of the ACM*, **13** (1970), No 6, 377–387.

[21] Codd E. F. Is Your DBMS Really Relational? *Computerworld*, 14 October 1985, 1–9.

[22] Codd E. F. Does Your DBMS Run by the Rules? *Computerworld*, 21 October 1985, 49–64.

[23] Studer R., V. R. Benjamins, D. Fensel. Knowledge Engineering: Principles and methods. *Data & Knowledge Engineering*, **25** (1998), No 1–2, 161–197.

[24] Guizzardi G. On Ontology, Ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. In: O. Vasilecas, J. Eder, A. Caplinskas (eds). Databases and Information Systems IV: Selected Papers from the Seventh International Conference DB&IS'2006. *Frontiers in Artificial Intelligence and Applications*, **155** (2007), 18–39.

Oumar Sy
Section d'Informatique
U.F.R de Sciences Appliquées et de Technologie
Université Gaston Berger de Saint-Louis
CP. 32000, BP. 234, Saint-Louis, Sénégal
e-mail: oumar.sy@ugb.edu.sn

Received October 22, 2019
Final Accepted January 28, 2020