Mediating Schema Ontology for Linked Open Data Cloud Integration Using Bottom-Up Schema Mapping

Heru Agus Santoso[1(✉)], Su-Cheng Haw[2], and Chien-Sing Lee[3]

[1] Faculty of Computer Science, Dian Nuswantoro University, Semarang, Indonesia
heru.agus.santoso@dsn.dinus.ac.id
[2] Faculty of Computing and Informatics, Multimedia University, Jalan Multimedia, 63100 Cyberjaya, Malaysia
[3] School of Engineering and Technology, Sunway University, Petaling Jaya, Malaysia

Abstract. Flexible data integration from heterogeneous sources, formalization of database schema, and consequent support and facilitation of knowledge discovery have become crucial to gaining competitive advantage. Consequently, its validity ("building the right system"), verification ("building the system right"), and efficiency are key factors, which enable more effective intelligent mining and recommendations. The Linked Open Data Cloud (LoD Cloud) is Cloud-based open data, which can be accessed using open standard protocols, such as HTTP, URL and URI. Enabling automated ontological information integration has commonly involved ontology as a mediator, which reconciles and provides a unified view of data structure, and as a mapping interface with the source database schema. Our approach to ontology versioning creates a mediating ontological layer, which encompasses a set of ontological properties which become updated based on queries by the user. We extend our prior work in terms of the change of data type property, object property and IS-A relationship are included in this paper and method/behavioral changes. The experimental results indicate 87% Precision, 100% Recall and 93% F-measure. These indicate that our approach has successfully captured changes in the database schema as changes in mediated ontology. We have also addressed the problem of change alignment when overlapping topics are discovered in mapping ontologies via formal validation and optimized via description logics consistency checking.

Keywords: Linked open data cloud · Ontology-based data integration · Mediating schema · Database · Semantic web

1 Introduction

Corporate strategy increasingly requires dynamic and evolving evaluation of multifactorial considerations, both internally and externally. Porter's [1] Five Forces, for example, has recommended competition in the industry, potential of new entrants into the industry, power of suppliers, power of customers, and threat of substitute products as

S.-C. Haw and K. Sonai Muthu (Eds.): CITIC 2022, 10, pp. 369–391, 2022.
https://doi.org/10.2991/978-94-6463-094-7_29

examples of such multifactorial considerations. Subsequently, [2] highlights the role of chance, demand-supply factors and government as external influencing factors. Within such environments, there is a need to represent, and validate/verify different versions in requirements and information systems evolution, in order to ensure flexible data integration, and to develop a means to formalize database schema, and consequently, support and facilitate knowledge discovery.

Assuming that operational and conceptual validation [3] have been carried out to suit the intended objectives [4–6] point out that relational database (RDB) components can be associated with a set of ontological properties. Ontological properties extend the capability of traditional databases [7], as they enable more efficient query and data integration. Given a set of ontological properties and a database query, the query over a database firstly needs to be re-written. It will then compile both query properties and ontological properties into a first order query. After query translation, it is delegated to database management systems (DBMS). However, the ontology itself is subjective as relationships between entities may differ among users, developers and experts. Furthermore, the ontology is subject to continuous changes. For example, the schema holding domain application might be modified due to the addition of new information, which was previously unavailable. As the ontology will change over time, there is a need to manage the change. Ontology versioning reflects the changes in the versioning process, and helps minimize the side effects of the changes to ontology, application or other elements [8].

In this paper, we approach these concerns, from a systems analysis and design perspective, an ontological perspective, and a formal methods perspective. In systems analysis and design, Waterfall, Parallel, V-model, Rapid Prototyping Development methodologies (RAD), and agile methodologies are popular. V-model emphasizes greatly on quality, while agile methodologies focus on design thinking and decomposition. Dennis et al. [9] adds on an internal value-based V model, emulating design thinking.

A formal description of concepts and their relationships in a given domain, ontology enables machines to integrate across different applications [10], not only based on linguistic mapping, but also schematic (structural and semantic) knowledge. Subsequently, ontology as mediating schema in data integration has gained broad acceptance [7, 11, 12].

The role of ontology during data integration can be either at design time or execution time. Examples of the role of ontology during design time are discussed in [13], where during execution time can be found in [12, 14]. They provide the following advantages: (1) a framework for flexible data integration, (2) a means to formalize database schema, and (3) a means to facilitate knowledge discovery.

The Linked Open Data Cloud (LoD Cloud) contains open data published on the Cloud, accessible via protocols, such as HTTP and URI. Represented as a structured (ontological) graph, LoD listed in Table 1, consists of entities and their relationships to real-world objects, enabling acquisition [15].

Figure 1 depicts a sample schema (entities and their relationships), of the Bibo ontology, part of the DBpedia LoD. The schema enhances knowledge acquisition in bibliographic applications. Many industrial applications use this method [10, 11]. However, a crucial challenge is how to facilitate the integration of application data and LoD Cloud, as the ecosystem grows in an efficient way.

Table 1. Example of LoD cloud

LoD	#Entities	#Fact
DBpedia	~1.95 million	~103 million
YAGO	~1.10 million	~5 million
Wordnet	119.957	207.016

Fig. 1. Bibo ontology facilitates integration with LoD [15]

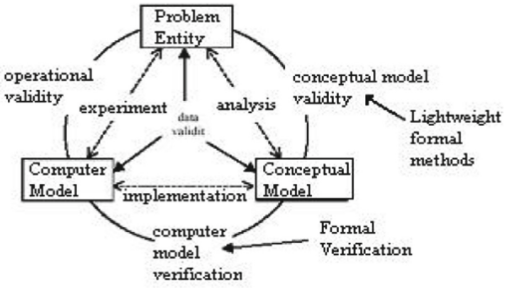


Fig. 2. Formal techniques [3]

Knowledge-aware applications such as recommender systems, robots and smart IoT can harvest knowledge from such structured representations. However, interoperability is required among the different Semantic Web layers, prior to integration. This gives rise to the question of versioning, and its validity, verification, and accuracy during the mapping and integration process. Formal methods require a high level of expertise, as it simulates the actual scenario, similar to a digital twin, and carries out theorem proving (Fig. 2), usually, to detect abnormalities. Hence, verification ("building the system right") while verification ("building the right system") are equally important. Both are complementary, as each provides input to and supports the other. In this paper, we are more concerned with automated or semi-automated verification.

Kuhn et al. [3] suggest two approaches: (a) lightweight formal methods, and (b) encapsulating the complexity of the formal models. Lightweight methods would automate analyses based on simplified specification approaches, so that requirements and designs can be tested and verified/validated early in the development cycle. Hiding complexity would require feedback avenues accessible only to the experts.

Commonly, ontology plays two main roles in versioning and data integration: (1) as a mediator, which reconciles and provides a unified view of data structure, and (2) as a mapping interface, with the associated data at the source [16]. In the first scenario, once a dataset is associated with the LoD Cloud using ontology to facilitate queries, it can then provide a wider interpretation of the dataset; improving data integration [14]. The initial version of the ontology, i.e., the mediating schema (the ontology which mediates between schemas), can subsequently be developed, using a 'schema-to-ontology' mapping approach.

Bottom-up change involves changes in ontology, which can be found through changes in conceptual structure, ontology metadata, corresponding sets of terms associated to the underlying ontology, mapping and so on [17, 18]. The limitation of existing approaches are lacking of directly deal with changes in the database schema. Our prior work mainly addressed concept change detection [19]. We capture the changes by analyzing the relationship between the changes in schema properties such as columns, constraints, tables and so on, with the associated ontological entities by means of a mapping approach. Metadata vocabulary, i.e., Ontology Metadata Vocabulary (OMV) [20] is incorporated. Compared to [19], this paper includes more ontological components i.e., datatype property, object property and more importantly, IS-A relationships. In addition, formalization on the ontological model related to schema mapping and formal verification are provided.

2 Related Work

In this paper, the related work is focused on change detection in the context of ontology versioning. Kendall and McGuinness [21] explained that ontology can be regarded as conceptual schemata, which provide representation of knowledge in a reusable and formal way. Several advantages of using ontology in data integration with LoD Cloud are the possibility for automatic inference over dataset aligned to the ontology, supporting integration, instance classification and so on. Generally, ontology itself is classified based on its intended purpose, i.e. (1) upper-level ontology, e.g., SUMO ontology, (2) domain ontology, which represents knowledge of a specific domain, such as CIDOC, and (3) application ontology, the ontology used in a specific application, such as ontology for mediating schema in data integration application. *EvoRDF* [18] can handle sequence of changes is extended. The approach employs a set of inference techniques to identify and track down problems in ontology. The method successfully maintains the correctness, completeness and usability of ontology. On the other hand, Jurisch and Igler's [22] study focused on the problem of change alignment when overlapping topics are discovered by ontologies. Detecting which properties in ontologies have been changed after some period of time is the main problem. The solution they propose is to utilize RDF embedding in their mapping scheme. He-Gook et al. [23] propose a transformation of RDB-RDF mapping with rules optimization approach. The rule defined used to produce semantic information of column's key constraints.

3 Method

Metadata plays a key role as it stores and describes the structure of the ontology. Furthermore, as schema, e.g., database schema and ontology are considered to have some commonalities [24], then both database schema and ontology provide a set of properties and constraints to the meaning of data. We propose using a bottom-up change detection approach because new information in the application domain are reflected as changes in ontology metadata and subsequently, the ontology.

Consequently, since versioning should include verification mechanisms [17], this paper proposes additional verification to ensure that the adapted changes to the new version of the ontology are guaranteed to produce a consistent ontology version. The additional verification extends from our earlier work [19], in terms of the change of data type property, object property and IS-A relationship and method/behavioral change. Studies on ontological properties, related to database modeling, are presented in Sect. 3.2.

The mapping operation used to determine matching elements, are performed by means of a set of rules we derived, to address the change of schema properties in relation to the additional ontological components i.e., classes, data type property, object property and IS-A relationships. These differentiate our approach from prior related literature as explained above.

Our approach applies Uschold's [25] creation of high-level disjoint axioms and the effective use of domain-range property, and answers Jurisch and Igler's [22] problem of change alignment when overlapping topics are discovered by ontologies. Where He-Gook et al. [23] have applied backward-inference to infer *RDF* triples, we have used OWL and description logic. They have also used pruning optimization to compute the closure of *RDF* triples. We have however, optimized using DL-SPARLQL.

3.1 Fundamental Definitions

Explaining the association between ontology and database schema properties, we define ontology model as follows.

Definition 1: Ontology Model

An ontology model consists of sets of concepts, properties, data properties, restrictions, axioms and instances, and a concept hierarchy denoted by C, OP, DP, R, A and ch respectively. Ontology O is 6-tuples $= <C, OP, DP, RS, A, ch>$, where:

(1) C is a set of concepts c in ontology O, and $C = DM \cup RN$, where:

- DM is a set of domain concepts dm_i in ontology O,
- RN is a set of range concepts rn_i in ontology O,

(2) OP is a set of object properties op_i in ontology O. If instances of c_1 are related to instances of c_2 through op_i then $c_1 \in DM$ and $c_2 \in RN$,

(3) DP is a set of data properties dp_i in the form: $dp_i = (pi_i, pd_i, pr_i)$, where:

- p_{ii} is property identity,
- pd_i is property domain,
- pr_i is property range,
- pr_i primitive data types taken from XML Schema Definition (XSD),

(4) *RS* is a set of restrictions, $RS = \{rs_1, rs_2, \ldots, rs_n\}$, where rs_i is a restriction of property p_i. OWL has two main types of property restrictions: value constraints and cardinality constraints. Rs_i can be a quantifier restriction that are $\forall$ (universal quantification) or $\exists$ (existential quantification), cardinality restriction, *hasValue* restriction and so on.
(5) A is a set of axiom a(*op*); *a*(*op*) is axiom which occurs in object property *op*,
(6) *ch* is a concept hierarchy. The structure of ontology can be seen as a concept hierarchy, whereby the node represents a concept and the edge represents an object property.

Definition 2: Graph Representation of Ontology

Ontology can be seen as a graph *G*, denoted by $G = (N, E)$, where *N* is a set of nodes representing a set of classes, and *E* is a set of edges connecting a set of object properties including their restrictions.

For instance, in OWL ontology, *owl:Thing* is in the highest level of hierarchical structure of OWL ontology. Hence, every user-defined class is a subclass of *owl:Thing*. In a subsumption relation involving a set of classes, the change operation will either make a class become more generalized or more specialized with respect to hierarchical structure in representation of the ontology. A class becomes more generalized if it moves up in an *IS-A* relationship, whereas a class becomes more specialized if it moves down in an *IS-A* relationship. For example, the addition of class Person to the mediating schema can be expressed with axiom $Person \sqsubseteq \top$. Since $Author \sqsubseteq \top$ while Author can be considered subsumed by *Person* denoted by $Author \sqsubseteq Person$, the operation *AddConceptHierarchy(Author, Person,* $\top$*)* can be performed. Here, $\top$ refers to *owl:thing*. The operation makes class Author move down in the concept hierarchy *ch*. Therefore, it becomes a more specialized concept. On the other hand, if the mediating schema needs to be subtracted with *Person*, then the operation *SubConceptHierarchy(Author, Person,* $\top$*)* can be carried out.

Definition 3: Structural Changes in Ontology Version

Structural change in ontology versioning is a set of structural differences between two versions of ontology caused by addition and subtraction operations. Given two versions of ontology, V_1 and V_2, the structural changes from V_1 to become V_2, are a set of atomic change operations *OntoChange*, defined as:

$$V_2 = OntoChange(V_1),$$

where:

- V_1 is an initial version of ontology
- V_2 is a new version of ontology

Table 2. Change operation

Structural change	Atomic change operation
Addition	*AddClass*, *AddObjectPropery*, *AddDataProperty*, *AddRestriction*, *AddAxiom*, *AddConceptHierarchy*
	AddObjectPropery = {*AddObjectProperty*, *AddSubClass*, *AddSubProperty*, …} *AddDataPropery* = {*AddDomain*, *AddRange*, …}, *AddRestriction* = {*AddHasValue*, *AddCardinality*, …} *AddAxiom* = {*AddSubClassOf*, *AddEquivalentClass*,.}
Subtraction	*SubClass*, *SubObjectPropery*, *SubDataProperty*, *SubRestriction*, *SubAxiom*, *SubConceptHierarchy*
	Each addition operation also has corresponding subtraction operation

- *OntoChange* is a set of atomic operations, which cause the changes in ontological properties.

Table 2 presents the addition and subtraction of structural change operations and their associated atomic operations. We use a finer-grained change representation, i.e., DL-axioms. For instance, the class collection can be added as the new class to the mediating schema. On the other hand, *collection_*d_m occurs in the mediating schema. Since collection and *collection_*d_m are considered equivalent classes, the addition of collection should be followed by the addition of the equivalence axiom between collection and *collection_*d_m using *AddEquivalentClass()*. The equivalent axiom is used to tell a reasoner that they are equivalent classes.

Definition 4: Type of Structural Change

Given the predicate *ontoStructDiff()* to compare ontological properties between V_1 and V_2, the type of structural change among the two versions where *ontoStructDiff()* $\neq$ *{}* is a triple $<\Delta_1, \Delta_2, \Delta_3>$ such that Δ_1 is the addition of EA to V_1 to become V_2 where $EA = V_2 \backslash (V_2 \bigcap V_1)$; Δ_2 is subtraction of V_1 with ES to become V_2 where $ES = V_1 \backslash (V_1 \bigcap V_2)$; Δ_3 is composite-change in V_1 with $(EA \bigcup ES)$ to become V_2.

As defined in Definitions 3 and refer to Table 2, structural change in ontology can be represented as addition and subtraction operations. With respect to a set of atomic change operations such as the ones presented in Table 2, in ontology versioning, the modification operation can be regarded as a composite change operation involving addition and/or deletion operations. For instance, renaming concept C_1 to C_2 in V_1, can be regarded as the subtraction of V_1 with C_1, followed by the addition of C_2 to V_1. Therefore, the modification operation can be carried out by means of a set of addition and subtraction operations. Based on the ontology model as defined in Definition 1, structural difference, (which is also called low level delta), involves the following ontological properties, i.e., class, object property, data property, restriction, concept hierarchy and axiom.

3.2 Rules to Determine a Set of Mapping Elements Between RDB and the Ontological Properties

Implementation of change detection in *RDB* schema which can be reflected as the change of ontological entities, needs a mapping approach to define the relationship between *RDB* and ontology. This subsection discusses the rules used to determine a set of mapping elements between RDB and the ontological properties:

Definition 5: RDB Components

RDB consists of a set of tables and their relationships, denoted by $RDB = (R, REL)$, where R is a set of tables and REL is a set of relationships explained as follows:

(1) R is a set of tables r_i, where $r_i = (A(r_i), pk(r_i), ROWS(r_i), CONS(r_i))$ *and* $r_i \in R$, for $i = 1, 2, \ldots, n$, where:

- $A(r_i)$ is a set of attributes in r_i, where $A(r_i) = \{at_{i1}, at_{i2}, \ldots, at_{in}\}$,
- $pk(r_i)$ is the primary key. The present of primary key is the requirement for a table to mapped to an ontological class.
- $ROWS(r_i)$ is a set of rows in r_i,
- $CONS(r_i)$ is a set of restrictions in r_i, where $CONS(r_i) = \{MA_i, NA_i, CH_i, FK_i\}$, where MA is a set of mandatory attribute (*ma*) restrictions, NA is a set of nullable attribute (*na*) restrictions, CH is a set of check (*che*) restrictions and FK is a set of foreign keys *fk*.

(2) REL is a set of table relationship of the form: $REL = \{ISA, NON\_ISA\}$, where:

- ISA = set of generalization–specialization relationships, a table may be considered as generalization of several tables, and vice-versa,
- NON_ISA = set of non-generalization–specialization relationships.

Definition 6: Mapping Between RDB Components and Ontological Properties

Tables, attributes, constraints and table relationships which exist in *RDB* can be mapped respectively to classes, data properties, restrictions and object properties of the OWL ontology. For a given table r_i, let $PA(r_i)$, $PRes(r_i)$, $PPkey(r_i)$ and $Pfkey(r_i)$ be the predicate to obtain attributes, restrictions or constraints, primary key and foreign key from table t_i respectively. Rules to map *RDB* components with ontological entities can be explained as follows:

(1) Mapping table r_i, where $r_i \in R$, for $i = 1, 2, \ldots, n$:
Table r_i can be mapped to OWL classes, defined by:

$$\forall r_i{:}r_i \in R\,|PPkey(r_i)| = 1 \rightarrow owl{:}class$$

(2) Attribute $AT(r_i)$:

- All attributes except foreign key can be mapped to OWL *DatatypeProperty*, defined by:

$$\forall at_i{:}at_i \in PA(r_i) \wedge Pfk(r_i) \rightarrow owl{:}DatatypeProperty$$

- Foreign Key, establishing relationships among instances from different tables can be mapped to object property, defined by:

$$\forall fk{:}fk \in Pfkey(r_i) \rightarrow owl{:}objectProperty$$

(3) Constraints $CONS(r_i)$:

The constraints of table r_i which are obtained from $CONS(r_i)$ can be mapped with OWL ontology restriction.

(4) Mapping table relationship *REL*:

In RDB schema, concept hierarchy cannot be directly represented. A table may be considered as a generalization table if every instance of the table is also an instance of at least one of the other tables in a database, although further analysis is needed to confirm this. The other way to identify subsumption relation in RDB is by incorporating concept hierarchy as the background knowledge. Given r_1, r_2 as the tables which correspond to respectively class_1 and class_2 of the provided background knowledge, $Plevel(r_i)$ is a predicate to determine the level of r_i which is associated to $class_i$ in the concept hierarchy of the background knowledge. If $Plevel(r_i) > Plevel(r_i)$ then $class_2$ becomes the subclass of class_1 and vice versa. Therefore, the *ISA* mapping of table r1, r2 can be defined as follows:

$$\forall r_1, r_2{:}(Plevel(r_1) > Plevel(r_2)) \vee (Plevel(r_2) > Plevel(r_1)) \rightarrow rdfs{:}SubClassOf$$

3.3 Model of Ontological Database

In this section, the ontology model of the database is represented using a set of DL axioms. DL axioms model an ontology of database modeling using a set of concepts and roles. In DL-Litecore, concept and roles are specified using the following form [26]:

$$\begin{aligned} B &\rightarrow A|\exists R \qquad R \rightarrow P|P^{-} \\ C &\rightarrow B|B \qquad\quad E \rightarrow R|R \end{aligned} \tag{1}$$

where: A, B, C are atomic concept, basic concept and general concept respectively, while P, R and E are atomic roles, basic role and general role respectively. Basic concept has two types of concept constructors, i.e., atomic concept A and unqualified existential restriction $\exists R$. General concept C has two types of concept constructor, i.e., basic concept B and its negation $\neg B$. The constructor of basic role R consists of atomic role P and its inverse P^{-}, while the type of general role E consists of basic role R and its negation $\neg R$. The fragments of the DL-Lite family have different expressivity. For instance, $\text{DL-Lite}_{\text{core}}$ which is augmented with the capability of specifying role inclusion is referred to as $\text{DL-lite}_{\text{R}}$, $\text{DL-Lite}_{\text{core}}$ which is augmented with the capability of specifying functionality based on role is referred to as $\text{DL-lite}_{\text{F}}$. In the *DL* knowledge base, the knowledge of the domain of interest consists of intentional and extensional knowledge, which is denoted by:

$$\sum = <T,\ A>,$$

where:

- T is the intensional knowledge of DL ontology (*TBox*),
- A is the extensional knowledge of DL ontology (*ABox*).

TBox T, the intensional knowledge, is defined by a set of concepts and role inclusions expressed in the form: $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$ where C_1 and C_2 are types of concepts and R_1 and R_2 are roles in the DL knowledge base. Extensional knowledge ABox A is defined by a set of membership assertions. It specifies the assertions about instances or individuals which are expressed in the form: Concept assertion: $C(a)$ and Role assertion: $P(a, b)$, where a, b are constants.

DL-Lite can be used to capture the conceptual modeling of entity relationship (ER model) and Unified Modeling Language (UML) class diagram [26]. For this study, the initial version of the mediating schema ontology is extracted on top of the database *OpenBiblio* [27] system. Therefore, the main building blocks of the ontology, i.e., ontological properties of the database modeling, are a set of classes and object properties, *OBclass* and *OBproperty*, where *OBclass* = {*Author*, *Biblio_field*, *Collection_dm*, *Biblio*, *Material_Type_d*$_m$, *Mbr_classify_d*$_m$, *Member*, *Biblio_hold*, *Member_account*, …} *and OBproperty* = {*hasAuthor*, *hasField*, *hasMaterialType*, *hasHolder*, *hasAccount*, …}. More precisely, DL-lite is indeed rich enough to capture the basic ontology of database modeling as it provides the following properties [26]:

(1) ISA relationship

ISA relationship axiom $C1 \sqsubseteq C2$ states that concept $C1$ is subsumed by concept $C2$. For example, *Author* $\sqsubseteq$ *Person* states that all instances of class *Author* belong to class *Person*, as *Author* represents a more specific class than class *Person*. The concept subsumption expressed in OWL is as follows:

$$<owl{:}Class\ rdf{:}ID = \text{“}Author\text{”}>$$
$$<rdfs{:}subClassOf\ \ rdf{:}resource = \text{“}\#Person/\text{”}>$$
$$</owl{:}Class>$$

(2) Concept disjointness

Concept disjointness $C1 \sqsubseteq \neg C2$ states that concept $C1$ is disjoint with concept $C2$. For example: *biblio* $\sqsubseteq \neg$*Author* states that all instances of class biblio are not members of class author. The concept disjointness expressed in OWL is as follows:

$$<owl{:}Class\ rdf{:}iD = \text{“}biblio\text{”}>$$
$$<owl{:}disjointWith\ rdf{:}resource = \#\text{“}Author/\text{”}>$$
$$</owl{:}Class>$$

(3) Role typing

Role typing axiom $\exists R \sqsubseteq C$ states that components or members of role R are instances of concept C. For example, $\exists$*hasAuthor* $\sqsubseteq$ author states that there exists someone in object property *hasAuthor* which is instance of author. The domain of object property *hasAuthor* is expressed in OWL as follows:

$$<owl{:}ObjectProperty\ rdf{:}ID = \text{“}hasAuthor\text{”}>$$
$$<rdfs{:}domain\ rdf{:}resource = \text{“}\#author\text{”}/>$$
$$</owl{:}Class>$$

(4) Mandatory participant

Mandatory participant axiom C $\sqsubseteq \exists R$ states that instances of concept *C* participate in role *R*. For example, *biblio* $\sqsubseteq$ *hasPublication* states that all member of class *biblio* should be produced by the member of domain of role *hasPublication*, i.e., *Author*. Example:

$$<owl{:}Class\ rdf{:}ID = \text{“}Author\text{”}><owl{:}restriction>$$
$$<owl{:}onProperty\ rdf{:}resource = \text{“}\#hasPublication/\text{”}>$$
$$</owl{:}restriction>$$
$$</owl{:}Class>$$

(5) Non-mandatory participant

Non-mandatory participant $C \sqsubseteq \neg\, \exists R$ states that instances of concept *C* do not participate in role *R*. For example, biblio $\sqsubseteq \neg$ hasAccount states that all members of class biblio do not participate in the object property relationship hasAccount.

Definition 7: TBox of Ontology Extracted from Database

Let *DB* be a database from which the properties of ontology are extracted from, e.g., *OBclass* and *OBproperty* are respectively the set of extracted classes and object properties from *DB*. The *TBox* of the ontology which represents the ontological structure of database modeling, consists of concept inclusions in the form $C1 \sqsubseteq C2$ and role inclusions in the form $R1 \sqsubseteq R2$. The *TBox* models ontological properties of database modeling, such as *ISA* relationship, concept disjointness, role typing, mandatory and non-mandatory participant.

3.4 Bottom-Up Change Detection Approach

The initial version of mediating schema ontology for data integration application with *LoD* Cloud was extracted on top of the associated database. Hence, the ontology is closely related to the database. When the database schema changes, there is the need to investigate important changes to be adapted to the ontology. Figure 3 depicts mediating schema implementation framework in data integration.

The role of mediating schema in data integration scenario is explained as follows: given a *SPARQL* query, a mediating schema, local *Openbiblio* database [27] and British Library linked open Data (BL LoD), the query can be used to find a set of instances stored in Openbiblio database and *BL LoD* which match to a given pattern defined in the *SPARQL* query. In this situation, the query should return more complete answer as it also returns a set of instances from multiple sources. After some period of time, the schema of *Openbiblio* may change, therefore there is the need to investigate changes in database schema needed to be adapted to the ontology. Other challenge is the semantic issue of schema which usually does not use common vocabularies, e.g., in *Openbiblio* $collection\_d_m$ is used instead of collection which is commonly used in the bibliography domain. Therefore, a reference bibliographic ontology in needed. A reference ontology also provides the most important semantic relations between two concepts of different ontologies which are subsumption ($\subseteq$) and equivalence ($\equiv$) relations.

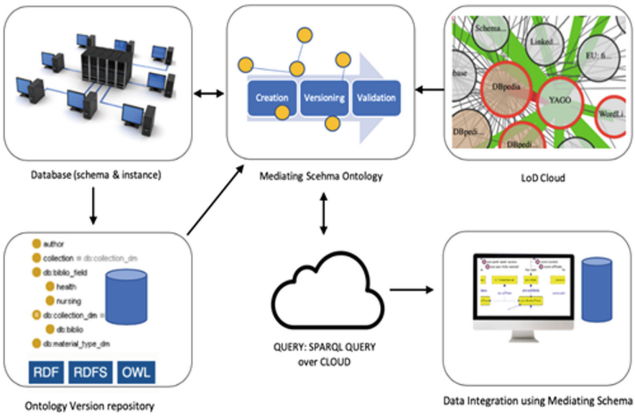


Fig. 3. Data integration framework using mediating schema

Considering ontology versioning needs metadata, Ontology Metadata Vocabulary (*OMV*) can be used to provide compatibility information between ontology and the database. Once the ontology is created on top of an RDB, the metadata also contains information about the database components associated with ontological entities. *OMV* attributes store information such as the number of classes, properties, individuals, and so on.

(1) Ontology Metadata

Ontology metadata such as *OMV* is standard machine-processable vocabulary to describe ontology, improving the reuse and sharing of ontology. The proposed approach employs the ontology entity of *OMV*. Precisely, the ontology entity attributes (*OA*) which are useful in version identification and change detections are employed.

(2) Version Identification

Version identification is essential in ontology versioning as the versioning approach can only be applied in any version of the same ontology. *OWL* provides several primitives to store version information at different levels of granularities, i.e. at the ontology level, class level, or property level as listed below:

- *<owl:Ontology rdf:about = ""/>* is the primitive which contains annotations about ontology, including version information.
- *<owl:priorVersion>* is used to indicate the earlier version of the ontology.
- *<owl:backwardCompatibleWith>* is used to relate version compatibility as it identifies a certain prior ontology and backward compatibility of the current version.
- *<owl:incompatibleWith>* is used to define version incompatibility among ontology versions.

When the version granularity at the ontology level is not sufficient, one may store the version information at class level and property level without the creation of a new version in a different ontology file. Version at the class level

or property level can be annotated after the *rdf:id* and *rdf:* Resource primitive in <*owl:class*> or <*owl: DatatypeProperty*> primitive of the new version.

(3) Change Detection Algorithm

Let V_1 be the initial version of the mediating schema ontology created on top of the relational database *RDB*. *RDBC* is the particular *RDB* components used for the source dataset to create V_1, so that $RDBC \subseteq RDB$. Since *RDBC* is the subset of *RDB*, then *RDBC*, which consists of tables, attributes and constraints denoted respectively by *R*, *AT* and *CONS* can be mapped to classes, data type properties and OWL restrictions or object properties. Relationships *REL* present in RDBC, which consists of ISA and NonISA, can be mapped with subclass axioms and object properties respectively. The term initial difference denoted with $diff_0$ is used to determine the state in which the changes of V_1 have not occurred.

Let *A*, *S* be the set of addition *R*, *AT*, *CONS* components and the set subtraction *R*, *AT, CONS* components in *RDBC* respectively when new database schema is used. Let $diff_1$, $diff_2$ be the differences between the numbers of *RDBC* metadata and the values of ontology attribute in OMV. $RDBC_1$, $RDBC_2$, $diff_1$, $diff_2$ are defined as follows:

- Schema components addition $RDBC_1 = RDBC \bigcup A$
- Schema components subtraction $RDBC_2 = RDBC/S$
- $diff_1$ is the difference between the numbers of *RDBC* metadata and *OA* values caused by *A*
- $diff_2$ is the difference between the numbers of *RDBC* metadata and *OA* values caused by S

Having all the above requirements for bottom-up change detection in hand, the algorithm presented in Algo 1 is used to determine the changes occurring in database schema which can be adapted in V_1 to become V_2.

Algorithm 1: *Change_detection*

1. $RDBC_1, RDBC_2 = \{\}$,
2. Input (*OMV*, RDBC)
3. Begin
4. Read (*RDBC*)
5. $RDBC_1 = RDBC \cup A$
6. $RDBC_2 = RDBC / S$
6. $diff_1 = |RDBC_1|$; $diff_2 = |RDBC_1|$
7. If $diff_1 \vee diff_2 \neq 0$ then detected_change= "a set of structural changes are detected"
8. If $diff_1 > 0$ then change_type:= addition;
9. If $diff_2 > 0$ then change_type:= subtraction;
10. End
11. // Output: detected_change, change_type

In line 4 of the algorithm, *Read RDBC* is performed by means of reverse mapping (from ontological properties to *RDB* components). It is used to identify a set of particular *RDB* components occurring in V_1. The result of *Read RDBC* is the ontological properties of V_1 which are present in the new system. The changes in database schema, which should be reflected as the changes in the mediating schema ontology, can be detected if $diff_1 \vee \text{diff}_1 \neq 0$. Furthermore, the type of change can be detected as additional properties to the ontology if $diff_1 > 0$, whereas the type of change can be detected as subtraction of properties in the ontology if $diff_2 > 0$. For instance, for the addition of table r_i to the database, where concept c_i is associated with table r_i, we can say that V_1 requires c_i addition to become V_2. On the other hand, the subtraction database with table r_i, where $r_i \in RDBC$ and r_i is associated with concept c_i in V_1, then we can say that concept c_{i} needs to be subtracted from V_1 to become V_2.

4 Results and Discussion

Having all the above requirements for bottom-up change detection in hand, the algorithm presented in Listing 1 is used to determine the changes occurring in database schema which can be adapted in V_1 to become V_2. The algorithm is implemented on top of the *OpenBiblio* database schema for experimental simulation. *OpenBiblio* is open source software for library system automation. The database compatible with MySQL5.5. We analyze the change in *OpenBiblio* versioning, while our analysis focuses only on the online public access catalog or OPAC functionality of the application. OPAC is access tool and resource guide to the collection of a library.

4.1 Change Analysis

Based on the previous definitions and the mapping between RDB and ontological entities, the change in the legacy database, which can be reflected as the change of ontology in the context of ontology versioning, is summarized as follows:

- The change in table r_i: $r_i \in R$ can be reflected to the change of concept or class in the ontology.
- The change in attributes $AT(r_i)$ can be reflected in the change of dp_i where $dp_i \in DP$. It may involve p_i which is stored in rdf:about =" ", property domain, or XSD primitive datatype.
- The change of foreign key $Pfkey(r_i)$ can be reflected in the change of object property relationship.
- The change in data constraints $CONS(r_i)$ can be reflected in the change in the restriction to the ontology.
- The change in generalization-specification (*Is-A relationship*) can be reflected in the change in concept hierarchy *ch* in the ontology. If table r_i is associated with concept c_i in ontology O, then the change of *ch* can be caused by: (1) r_i subtraction, (2) addition of new table r in an *Is-A* relationship or (3) the change of Is-A relationship in which concept c_i becomes more generalized if r_i moves up in the *Is-A* relationship, or concept c_i becomes more specialized if r_i moves down in the *Is-A* relationship.

For example, we have determined the change in a set of tables R in *OpenBiblio Ver. 0.7.2*. The initial version of V_1 was created on top of the *OpenBiblio* database using a mapping approach. The initial difference $diff_0$ (as the state in which the changes have not occurred yet) is the difference between the numbers of *RDBC* components and the ontological entities. In terms of the changes occurring in R, they consist of the addition of four tables, thus $diff(R) = 4$. Among them, three tables: *checkout_privs, member_fields*, $member\_fields\_d_m$, denoted by r_1, r_2 and r_3 respectively are considered as new classes in V_2. Based on Definition 6, the database table can be mapped with *owl:class*, thus, we can compute $diff_1$. If $diff_1 > 0$, the changes should be reflected as changes in ontology O.

4.2 Evaluation

This section presents the evaluation of the proposed algorithm, to measure the relevance of the changes in the legacy database to be reflected to the change in the mediating schema ontology in the versioning process. The detectable changes in the legacy database, which can be reflected as the change in ontology, involve the changes in (1) set of tables R, (2) set of attributes $AT(r_i)$, (3) set of foreign key $Pfkey(r_i)$, (4) constraints $CONS(r_i)$ and (5) *ISA* relationships. Set-based measures, such as *Precision* ($\mathcal{P}$) and *Recall* ($\mathcal{R}$) are often used for performance evaluation of detection algorithm [28], for instance, they are used to determine the fraction λ (where $0 \leq \lambda \leq 1$) which refers to the relevancy of specific items detected by the algorithm from a given collection of items. Therefore, $\mathcal{P}$ and $\mathcal{R}$ can be used to evaluate the quality of the matching algorithm, i.e., $\mathcal{P}$ corresponds to the probability of the legacy database changes, which are detected by the algorithm, and are relevant with the change in the ontology in the versioning process. $\mathcal{R}$ corresponds to the probability of relevant changes in the legacy database to changes in the ontology detected by the algorithm. F-measure ($\mathcal{F}$) combines $\mathcal{P}$ and $\mathcal{R}$.in harmonic mean.

Let dc be any database component from a given legacy database, then ω and $\acute{\omega}$ are defined as follows:

$$\omega = \left\{ \begin{array}{c} 1\text{: if } dc \text{ is relevant} \\ 0\text{: otherwise} \end{array} \right\}, \tag{2}$$

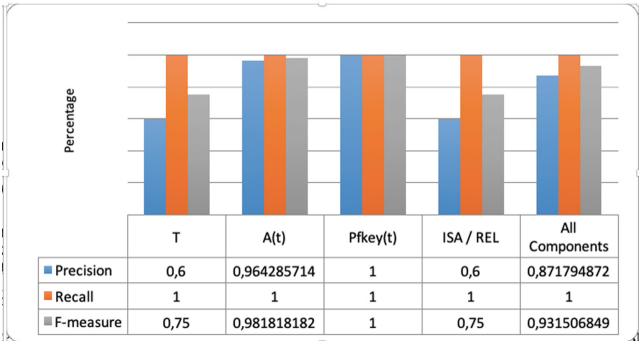
$$\acute{\omega} = \left\{ \begin{array}{c} 1 : \text{if the algorithm detects } dc \\ 0 \ : \ \text{otherwise} \end{array} \right\} \tag{3}$$

$\mathcal{P}$ is the probability of $\omega = 1$ where $\acute{\omega} = 1$. On the other hand, $\mathcal{R}$ is the probability of $\acute{\omega} = 1$ where $\omega = 1$. Hence, $\mathcal{P}$, $\mathcal{R}$ and $\mathcal{F}$ in this paper can be obtained as follows:

$$\mathcal{P} = \frac{|\{ \text{ relevant changes} \}| \cap |\{ \text{ occurred changes} \}|}{|\{ \text{ occurred changes} \}|} \tag{4}$$

$$\mathcal{R} = \frac{|\{ \text{ relevant changes} \}| \cap |\{ \text{ occurred changes} \}|}{|\{ \text{ relevant changes} \}|} \tag{5}$$

$$\mathcal{F} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$



	T	A(t)	Pfkey(t)	ISA / REL	All Components
Precision	0,6	0,964285714	1	0,6	0,871794872
Recall	1	1	1	1	1
F-measure	0,75	0,981818182	1	0,75	0,931506849

Fig. 4. Precision, Recall and F-measure

Figure 4 shows the *Precision*, *Recall* and *F-measure* of the algorithm in determining the relevancy of changes in the *OpenBiblio* database.

The *Precision* of the detected changes to R, $AT(r)$, $Pfkey(r)$ and *ISA* are 0.60, 0.96, 1.00 and 0.60 respectively, while there were no changes occurred in $CONS(r)$. That means the changes to table, attribute, foreign key and *ISA* are correctly captured as changes to ontological entities in the versioning process. Furthermore, the changes are also detectable by the algorithm since its Recall is 1.00.

4.3 Formal Validation

Flouris et al. in [29] stated that ontology versioning should provide a mechanism to validate all versions of the ontology. Therefore, validation is an important task in ontology versioning. Formal validation presented in this section is used to ensure that the relevant changes detected by the algorithm can be captured as a new version of the original ontology. DL validation plays a significant role in maintaining the consistency of the ontology [29]. As DL is used as the logical foundation of Web ontology, the term consistency in this paper is referred to as the logical consistency. Consistency checking in this work is performed to maintain the logical consistency of the ontology when the change in the associated database should be captured in the ontological model.

Recall that the detection algorithm identifies the changes that occurred in the associated database. The operation for determining the matching elements between RDB components and ontological entities are carried out by means of a set of mapping rules presented in Sect. 3.2. Next, a set of relevant changes is then associated with a set of atomic change operations (such as the ones presented in Table 1), which can be seen as ontological changes to be adapted to the new version of ontology.

We refer to DL semantics to provide logical validation of the new ontology version. Based on DL knowledge base, the change in ontology is consistent iff the new ontology version has at least one model interpretation I which satisfies each axiom of the resulting ontology. DL semantics is specified by means of interpretation, usually presented as Tarski-style of interpretation I in the form: $I = (\Delta^I, .^I)$, where

- Δ^I is the interpretation domain, which should be a non-empty domain,
- $.^I$ is the interpretation function which maps every individual o to a set $o^I \subseteq \Delta^I$, each concept and each role occurring in knowledge base to a set $C^I \subseteq \Delta^I$ and $R^I \subseteq \Delta^I \times \Delta^I$ respectively. For example, I is a model of a concept inclusion, denoted by $I \vDash C \sqsubseteq D$, iff $C^I \subseteq D^I$. In other words, it can be said that I is the model of concept inclusion $C \sqsubseteq D$ iff each instance of concept C is also instance of concept D.

The validation of the new ontology version in this paper is addressed by the ontological properties of database modeling presented in Sect. 3.2, i.e., IS-A relationship, concept disjointness, role typing and mandatory or non-mandatory participant. They are presented as follows:

(1) Validation after concept subtraction

This is based on the fact that the deletion of a table in the associated RDB can be reflected to a concept subtraction in the new version of ontology, and it needs validation. For instance, to produce a new ontology version, the subtraction of the ontology with concept C_1 is needed. Then the validation is used to check whether a number of axioms need to be removed. Let T be TBox; C_2 be a concept; $C_1 \sqsubseteq C_2$, $C_1 \sqsubseteq \neg C_2$, $\exists R \sqsubseteq C_1$, $C_1 \sqsubseteq \exists R$, $C_1 \sqsubseteq \neg \exists R$ be respectively IS-A relationship, concept disjointness, role typing, mandatory and non-mandatory participant axioms; T' be TBox after validation and α_i be axioms in T. The algorithm used to identify and subtract inclusion axioms involving C_1 to produce new version of the ontology is given as follows:

***Algorithm 2**: Validation to ontological properties of database modeling*

***Input:** T, C_1, $dbAxioms = \{C_1 \sqsubseteq C_2,\ C_1 \sqsubseteq \neg C_2,\ \exists R \sqsubseteq C_1,\ C_1 \sqsubseteq \exists R,\ C_1 \sqsubseteq \neg \exists R\}$.*

***Output:** T'*

```
1.  Begin
2.          T':= T
3.          For each axiom αi ∈ T
4.          do
5.                  If C1 in αi |αi ∈ dbAxioms
6.                  then T':= T' \ {αi}
7.          endfor
8.          return T'
9.  end
```

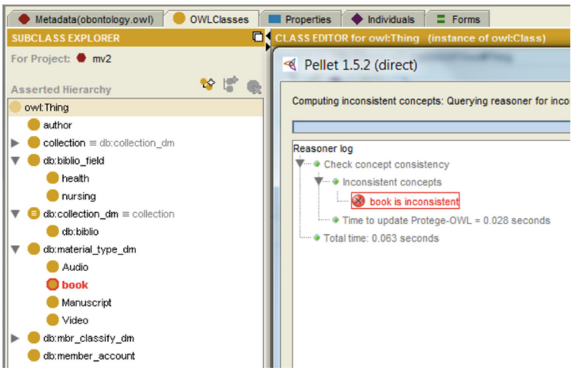


Fig. 5. Concept inconsistency after the addition of book in TBox

(2) Validation after concept addition

Concept addition to ontology may lead to inconsistency. The new version of ontology is inconsistent if there is no interpretation I which satisfies each axiom of the ontology [30]. Note that, interpretation is a way of understanding a concept as a set of instances.

Definition 7: Inconsistency of TBox

Let T be a DL TBox. T is inconsistent if there is no interpretation I which satisfies DL axioms in T.

Recall that the initial version of the ontology is extracted on top of RDB. In the spirit of the Semantic Web, the extracted ontology can be aligned with any available domain ontology in the Web. For instance, we are interested to align concept collection_dm with concept collection from the Bibliographic ontology. Once they are aligned, then concept collection_dm and collection become equivalent concepts. The reason for the alignment is not only to use collection as a common term in the domain knowledge, but also to provide useful hierarchical information for the concept collection. The following excerpt of T' explains such alignment:

$$\begin{aligned} T' = \{&(1)\ \text{book} \sqsubseteq \text{material\_type},\\ &(2)\ \text{material\_type} \sqsubseteq \neg\ \text{collection\_dm},\\ &(3)\ \text{collection\_dm} \sqsubseteq \text{collection},\\ &(4)\ \text{collection} \sqsubseteq \text{collection\_dm}\} \end{aligned}$$

Axiom (1) states that book is material_type, axiom (2) states that material_type is disjoint with collection_dm and axiom (3) and (4) state the equivalence of collection and collection_dm after alignment. For instance, book is the new concept extracted from RDB as the sub class of material_type expressed in axiom (1). Concept inconsistency occurs when T' should be adapted to TBox, as there exists book which is subsumed by collection in T. This is because material_type and collection_dm are two disjoint concepts. The inconsistency was detected by the reasoner as depicted in Fig. 5.

Resolving the inconsistency as depicted in Fig. 3, we can subtract *T'* with axiom {book ⊑ material_type} before adapting *T'* to TBox. The use of the external reasoner, usingthe blackbox approach, aims to provide information on whether certain combinations of axioms can cause inconsistency. However, in the blackbox approach, the result still needs user interpretation and justification due to the fact that the ontology reasoner itself cannot easily explain the root of the problem causing inconsistency.

In addition, inconsistency in TBox is also closely related to incoherency. In model-based diagnosis, maintaining logical consistency is firstly performed by deriving a minimal set of DL axioms causing the inconsistency [30]. Next, it is followed by the process of reducing irrelevant axioms causing incoherency. Ontology is said to be incoherent if there exists unsatisfiable concepts in TBox. Unsatisfiable concept is the concept which is interpreted as an empty concept in a given TBox. Therefore, coherency evaluation of a given TBox will produce a set of unsatisfiable concept [31].

Definition 8: Incoherency in TBox

Given T as the TBox of ontology, T is incoherent if there exists an unsatisfiable concept in T. Incoherency may lead to TBox inconsistency.

Subtraction of unsatisfiable concept during the creation of a new version is important to maintain the consistency of the ontology. Inconsistency of a set of axioms affects other axioms in *T*. Inconsistency in *T* is highly undesired [31] as inconsistent axioms often carry no knowledge as no meaningful inference can be obtained. Validation of logical consistency by removing unsatisfiable concepts is based on the principle of minimal change, which states that the resulting ontology after validation should be as close as possible with the initial ontology. Schlobach [32] stated that the following three steps are necessary to remove the unsatisfiable concept, i.e.: (1) determining which set of axioms caused concept unsatisfiability, (2) analyzing the smallest set of axioms, which should be corrected, and (3) performing correction. To determine logical consistency by examining concept incoherency, a minimal set of concept definitions with respect to unsatisfiability and incoherency are defined with Minimal Unsatisfiability Preserving Sub TBox (MUPS) and Minimal Incoherency Preserving Sub TBox (MIPS). MUPS and MIPS are defined as follows [35]:

Definition 9: Minimal Unsatisfiability Preserving Sub TBox (MUPS)

Let T, T' and C be TBox, Sub TBox and concept respectively, where T',C ⊆ T. T' is MUPS for concept C if C is unsatisfiable with respect to T', and C is satisfiable with respect to T" where T" ⊆ T'.

Based on Definition 9, MUPS contains at least one axiom involving a concept in T' which needs to be corrected or subtracted to resolve any conflict in the ontology. On the other hand, MIPS is the minimal set of axioms which make the involving concept in the set of axioms unsatisfiable. Hence, MIPS can be used to determine minimal conflict set causing incoherence in the ontology.

Definition 10: Minimal Incoherency Preserving Sub TBox (MIPS)

Let T, T' be TBox and Sub TBox respectively, where T' ⊆ T. T' is MIPS for T if T' is incoherent and T" ⊆ T' is coherent.

Computing MUPS is useful to determine a set of axioms related to unsatisfiability of a concept, while computing MIPS provides the smallest set of TBox preserving concept

unsatisfiability in a given TBox. However, the use of both MUPS and MIPS is mainly to be focused on a set of axioms, which potentially causes inconsistency in the knowledge base. Several tools for ontology validation by means of computing MUPS and MIPS are available on the Web such as DION, MUPSter and so on. For example, given a MIPS as follows:

$$T' = \{(1).\ \text{student} \sqsubseteq \neg\ \text{author},$$
$$(2).\ \text{student}; \sqsubseteq \text{author}\}$$

The first axiom states that student is a disjoint concept with author, while the second axiom states that student is author. For instance, the second axiom is added due to the need of change in T with respect to the change of information in the associated database. Initially, T is coherence. Once concept assertion is performed, i.e., certain individuals are asserted into both student and author, due to the fact that some students are also authors. Thus, T becomes incoherent. Therefore, incoherency is a potential source of inconsistency. To resolve this incoherency, T needs to be subtracted with the second axiom. This is crucial to maintain coherency since ontology which plays the role of mediating schema can be associated with arbitrary instances from diverse sources in data integration settings.

(3) Validation after change of ISA relationship

The validation after change of ISA relationship is illustrated as follows, for example, given an excerpt of T:

$$\begin{aligned} T' = \{&\text{journal} \sqsubseteq \text{periodical},\\ &\text{book} \sqsubseteq \text{nonperiodical},\\ &\text{journal} \sqcap \text{book} \sqsubseteq \bot,\\ &\text{periodical} \sqsubseteq \text{collection},\\ &\text{nonperiodical} \sqsubseteq \text{collection}\} \end{aligned}$$

T' is subtracted with ISA axiom $\beta = \{\text{book} \sqsubseteq \text{collection}\}$. As axiom β is not explicitly expressed in T', thus, the validation should firstly find the implicit knowledge in T', i.e., a set of axioms which has the same consequence with β. For this purpose, the pinpointing technique can be used to determine a minimal set of axioms expressing β. Further details on pinpointing technique can be found in [34]. Given the function Pinpointing(β, T'), it can yield a set of axioms {(1) Book $\sqsubseteq$ nonPeriodical, (2) nonPeriodical $\sqsubseteq$ Collection}. Hence, the validation can be carried out by focusing on updating or removing axiom (1) or (2), or both axioms.

5 Conclusion and Future Work

The Semantic Web framework is one of the solutions to address the issue of interoperability and integration over Web-based applications. In such types of applications, ontology plays a critical role. To maintain the ontology, bottom-up change detection by analyzing the change in system behavior reflected by the changes in the database

and its metadata is crucial. This paper proposes structural change detections involving class, object property, datatype property and ISA relationship. Our study using Open-Biblio database, Precision, Recall and F-measure are used to determine the relevance of the changes occurring in the database to be reflected as the changes in the ontology model. The results are challenging, with 87% of Precision, 100% of Recall and 93% of F-measure respectively. These results indicate that the detectable changes occurring in the database are relevant to be reflected as the changes to the ontology. Our future work will be focus on validation method of structural changes that reflect semantic changes.

Acknowledgments. This research work was supported by Directorate of Hinger Education, Ministry of Education and Culture of Indonesia.

References

1. Porter, M.E. (1979) "How Competitive Forces Shape Strategy", *Harvard Business Review*, March/April 1979.
2. Porter, M.E. (1980) Competitive Strategy, Free Press, New York, 1980.
3. D. R. Kuhn, D. Craigen & M. Saaltink, (2003). Practical Application of Formal Methods in Modeling and Simulation. Summer Computer Simulation Conference, pp. 726–731
4. C. Pinkel (2018): RODI: Benchmarking relational-to-ontology mapping generation quality. Semantic Web, vol. 9, no. 1, pp. 25–52.
5. H. A. Santoso, Z. T. Abdul-Mehdi, and S.-C. Haw (2009): Semantic Enhancement Framework for e-Government Using Ontology Versioning Approach. p. 6.
6. T. Pankowski (2018): Schema Transformations and Query Rewriting in Ontological Databases with a Faceted Interface. in Theory and Practice of Model Transformation, pp. 76–91.
7. D. Calvanese and E. Franconi (2012): First-Order Ontology Mediated Database Querying via Query Reformulation, S. Flesca, S. Greco, E. Masciari, and D. Saccà, Eds. Cham: Springer International Publishing, pp. 169–185.
8. S. D. Cordoso, M. Da Silveira, and C. Pruski (2020): Construction and Exploitation of an Historical Knowledge Graph to Deal With The Evolution of Ontologies. Knowledge-Base Systems, vol. 194.
9. Dennis, A., Wixom, B. H. and Roth, R. M. (2012). Systems Analysis and Design. (5th edition). Wiley.
10. M. Benedikt, B. Cuenca Grau, and E. V. Kostylev (2018). Logical foundations of information disclosure in ontology-based data integration. Artificial Intelligence, vol. 262, pp. 52–95.
11. M. Bienvenu, S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev, Eds. (2018): Ontology-Mediated Queries: Combined Complexity and Succinctness of Rewritings via Circuit Complexity. J ACM, vol. 65, no. 5, p. 28:1–28:51
12. D. Calvanese, P. Liuzzo, A. Mosca, J. Remesal, M. Rezk, and G. Rull (2016) : Ontology-based data integration in EPNet: Production and distribution of food during the Roman Empire. Engineering Applied Artificial Intelligence, vol. 51, pp. 212–229.
13. Moreno, N., Navas, I. & Aldana, J.F. (2003). Putting the Semantic Web to Work with Database Technology. *Data Engineering Bulletin* (Vol. 26, No. 4, p. 49 – 54). Washington, DC, USA: IEEE Computer Society.
14. G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati (2018): Using Ontologies for Semantic Data Integration. S. Flesca, S. Greco, E. Masciari, and D. Saccà, Eds. Cham: Springer International Publishing, pp. 187–202.

15. X. Ji, S. Pan, E. Cambria (2020): A Survey on Knowledge graphs: Representation, Acquisition and Applications. CoRR abs/2002.00388.
16. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati (2009): Conceptual Modeling for Data Integration. in Conceptual Modeling: Foundations and Applications, A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 173–197.
17. F. Zablith (2015), "Ontology evolution: a process-centric survey," Knowledge Engineering Review, vol. 30, no. 1, pp. 45–75.
18. H. Kondylakis and N. Papadakis (2018): EvoRDF: evolving the exploration of ontology evolution. Knowledge Engineering Review, vol. 33.
19. H.-A. Santoso, S.-C. Haw, and C.-S. Lee (2010): Change Detection in Ontology Versioning: A Bottom-Up Approach by Incorporating Ontology Metadata Vocabulary. in Database Theory and Application, Bio-Science and Bio-Technology, pp. 37–46.
20. J. Hartmann (2005): Ontology Metadata Vocabulary and Applications. In On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops, pp. 906–915.
21. E. F. Kendall and D. L. McGuinness (2019): Ontology Engineering". Synthesis Lectures on Semantic Web: Theory and Technology, Vol.9, No.1.
22. M. Jurisch and B. Igler (2018): RDF2Vec-based Classification of Ontology Alignment Changes. ArXiv180509145 Cs.
23. J. He-Gook, I. Dong-Hyuk, K. Hyoung-Joo (2020): Semantics-preserving optimisation of mapping multi-column key constraints for RDB to RDF transformation, Journal of Information Science, Vol 46, SAGE Publishing.
24. A. Calì, G. Gottlob, and A. Pieris (2012): Ontological query answering under expressive Entity–Relationship schemata. Inf. Syst., vol. 37, no. 4, pp. 320–335.
25. M. Uschold (2016): Finding and Avoiding Bugs in Enterprise Ontologies. In Know@LOD-2016, Heraklion Greece, vol. 1586, p. 13.
26. D. Calvanese et al., "Ontologies and Databases: The DL-Lite Approach," in Reasoning Web. Semantic Technologies for Information Systems: 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures, S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, and R. A. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 255–356.
27. OpenBiblio." [Online]. Available: http://obiblio.sourceforge.net/. [Accessed: 06-Feb-2022].
28. J. Euzenat, "Semantic Precision and Recall for Ontology Alignment Evaluation," C, p. 6.
29. G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou, "Ontology change: classification and survey," Knowledge Engineering Review., vol. 23, no. 2, pp. 117–152, Jun. 2008.
30. X. Fu, G. Qi, Y. Zhang, and Z. Zhou, "Graph-based approaches to debugging and revision of terminologies in DL-Lite," Knowledge-Based System, vol. 100, pp. 1–12, May 2016.
31. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family," J. Autonomous Reasoning, vol. 39, no. 3, pp. 385–429, Oct. 2007.
32. S. Schlobach, Z. Huang, R. Cornet, and F. van Harmelen, "Debugging Incoherent Terminologies," J. Autonomous Reasoning., vol. 39, no. 3, pp. 317–349, Oct. 2007.