Hindawi
Mobile Information Systems
Volume 2018, Article ID 1359174, 13 pages
https://doi.org/10.1155/2018/1359174

Research Article

Methodology for Automatic Ontology Generation Using Database Schema Information

JungHyen An[1] and Young B. Park[2]

[1]*Artificial Intelligence and Information Architecture, Department of Computer Engineering Graduate School, Dankook University, Yongin, Republic of Korea*
[2]*Department of Software Science, Dankook University, Yongin, Republic of Korea*

Correspondence should be addressed to Young B. Park; ybpark@dankook.ac.kr

Received 14 December 2017; Accepted 4 March 2018; Published 2 May 2018

Academic Editor: Jeongyeup Paek

An ontology is a model language that supports the functions to integrate conceptually distributed domain knowledge and infer relationships among the concepts. Ontologies are developed based on the target domain knowledge. As a result, methodologies to automatically generate an ontology from metadata that characterize the domain knowledge are becoming important. However, existing methodologies to automatically generate an ontology using metadata are required to generate the domain metadata in a predetermined template, and it is difficult to manage data that are increased on the ontology itself when the domain OWL (Ontology Web Language) individuals are continuously increased. The database schema has a feature of domain knowledge and provides structural functions to efficiently process the knowledge-based data. In this paper, we propose a methodology to automatically generate ontologies and manage the OWL individual through an interaction of the database and the ontology. We describe the automatic ontology generation process with example schema and demonstrate the effectiveness of the automatically generated ontology by comparing it with existing ontologies using the ontology quality score.

1. Introduction

An ontology is a model language that can build models, which support the conceptual integration of the distributed domain data and the inference of relationships among the concepts as a result of activities such as concept analysis and domain modeling using the standard methodology [1]. In particular, the importance of ontology is recognized in areas such as knowledge engineering, context awareness, knowledge integration, and knowledge management and modeling.

When an existing ontology cannot be reused, it needs to be newly developed. The process of developing an ontology involves creating attributes and constraints, creating a model, and applying it to domain data [2]. This process is like designing the requirements of a software architecture. As with software development, ontology development needs to discuss domain concepts, relationships, and constraints with domain experts [3–6].

Since this process consumes a lot of manpower, methods to automatically define an ontology model by defining a domain in the form of the metadata that can characterize the domain and apply rules to the metadata are currently studied. Yahia et al.'s work automatically generates ontologies based on XML data sources [7]. The following studies, including Dey et al., conceptually classify fuzzy data and describe how to generate an ontology and the rules to generate an ontology [8–10]. The Clonto Framework automatically generates an ontology through a suffix tree clustering algorithm in a document that describes the domain information [11].

Methodologies to automatically generate an ontology through metadata must preprocess the metadata for generating an ontology through a domain into a template for applying an ontology-generating rule [12]. The generated ontology model does not focus on how to manage when many individuals occur. Individual inputs into the generated ontology model can

be stored in a table in one of the databases in a triple form that consists of an object and a subject. Using this approach, it is possible to provide efficient management and query functions for individuals of the corresponding schema [13, 14]. Individual is the basic component of an ontology. The role of individuals in an ontology is to classify objects according to their class, which is the concept of a domain [15]. Individuals in OWL correspond to constants in first-order logic and instances in the Resource Description Framework.

In this paper, we propose a methodology to automatically generate an ontology model based on the database metadata and convert it into a database tuple when many individuals occur in the generated ontology. This methodology reads an OWL-DL-level ontology based on the schema information, which is the metadata of the relational database, and converts the individual of the ontology into a relational database.

A relational database is one of the common methods to structurally store data in a domain [16]. As a result, the schema of the database storing the domain data has characteristics of the corresponding domain. In addition, a database table is a conceptual model that can contain similar data in the domain. As a result, the methodology to generate an ontology from a built database has the advantage that the generated ontology can better express the characteristics of each domain region for a wider range of domain regions.

The ontology quality metric was applied to determine whether the automatically generated ontology through the database schema was sufficient for actual domain applications. A good ontology is impossible to evaluate because an ontology has different characteristics depending on the applied domain, but it is possible to determine how suitably the ontology fits into the domain [17]. In this paper, we define a metric of how well an ontology can reflect the domain knowledge, compare the automatically generated ontology according to our process to the other ontologies, and show the effectiveness of the automatic ontology-generating method using the database schema.

The remainder of the paper is organized as follows: Section 2 introduces existing papers on the data construction for automatic ontology generation and individual management. Section 3 describes the automatic ontology generation process. Section 4 describes the process of managing an ontology individual using a database management system. Section 5 shows the process and results of automatic ontology generation using the sample database schema. In Section 6, the ontology quality score is used to verify how the ontology expresses the domain by comparing it with other ontologies. The final section concludes with a discussion of future research.

2. Related Works

2.1. Automatic Ontology Generation Using Metadata. Frameworks such as TANGO [18] and TARTAR [19] automatically generate an ontology from the metadata that contain the structure and characteristics of the domain data. In the framework, the commonly found components in the data are organized in a tabular form, and the table is analyzed to generate the components of the ontology model. In the TANGO application, a table is analyzed, a semiontology is generated based on each table, and a semiontology is connected to generate a kernel ontology to finally generate an ontology. TANGO supports functions such as multiple-source query processing, semantic web creation, and superimposed information generation to use application. TARTAR automatically transforms tabular data such as HTML, PDF, and EXCEL into a formal (structural and semantic) template and provides it to users through an internal engine. At this time, a table attribute ontology of the OWL format linked to each table data is automatically generated.

Long [20] has realized an agent that interprets table data by recognizing tabular data and each table attribute and generating an ontology of the RDF format to realize an agent-based approach methodology for table recognition and interpretation. The study explains how to extract these tables from text files, evaluation of table analysis tasks, and the Table Analysis Framework based on the RDF. Among them, the RDF-Based Blackboard Framework generates an RDF file through the annotation of different printed tables and analyzes the table through the generated RDF.

The following studies on automatic ontology generation based on the relational database define each component of the database and ontology as a notation and generate the ontology based on the database through the rule using the relation of each component [21, 22]. These researchers used the Jena Framework to read and analyze the metadata of a database in a program, which was written in the Java language, and applied the rule to create an ontology model. As a result, they used Jena to implement the ontology model and generated documentation and RDF graphs.

Alalwan et al. [23] explained the overall process and rules to automatically generate an OWL ontology from a database schema to merge the data from each database using ontologies in a distributed database environment. The rules applied to the automatic creation of ontologies in the paper are based on the rules of this study; they are integrated and generalize the conditions of the rules. In a study, the rule for class fragmentation related to a layering of the class generated by referring to a database table is defined in the following formula:

$$\mathrm{Rule}_{C_2}\begin{cases}\text{Class condition:}\\ \exists x, y:\begin{pmatrix}\text{(a) } \mathrm{PK}(x, \mathrm{R}_1) \wedge \mathrm{PK}(y, \mathrm{R}_2) \wedge\\ \text{(b) } \mathrm{FK}(y, \mathrm{R}_2, x, \mathrm{R}_1) \wedge\\ \text{(c) } \exists v \in \mathrm{Dom}(x):\ (\mathrm{OCC}(v, x, \mathrm{R}_1) \wedge \neg \mathrm{OCC}(v, y, \mathrm{R}_2))\end{pmatrix}\\ \text{Class action:}\\ \begin{pmatrix}\text{(1) create class } \mathrm{C}_1 \text{ from } \mathrm{R}_1\\ \text{(2) create class } \mathrm{C}_2 \text{ from } \mathrm{R}_2\\ \text{(3) make class } \mathrm{C}_2 \text{ (subOf) } \mathrm{C}_1\end{pmatrix}\end{cases}, \tag{1}$$

where Rule_{C_2} defines class C_1 and C_2 when the table satisfies the class condition, and C_2 is a subclass of C_1. The primary key of one table must be set to a foreign key of another table, and the attribute data type of both tables must be identical. OCC denotes all tuples that belong to one table, and Dom denotes all attribute data types of the table. After the class is created based on the information in the table, the study suggests applying the rule defined in the following formula when a DatatypeProperty is created in the attribute of the table:

$$\mathrm{Rule}_{\mathrm{DP}_3}\begin{cases}\text{Datatype condition:}\\ \wedge_{i=1}^{2}\left(\mathrm{Attr}(x, \mathrm{R}_i) \wedge \mathrm{NonFK}(x, \mathrm{R}_i)\right)\\ \text{Datatype action:}\\ \text{create DP}(x) \text{ with Domain } C \text{ and Range dom}(x)\end{cases}, \tag{2}$$

where $\mathrm{Rule}_{\mathrm{DP}_3}$ defines a class to be a domain for all attributes that are not foreign keys in the table and to create a DatatypeProperty that sets the data type of each attribute as a range. In addition, the OWL is automatically generated through rules that define the properties and class relations of each property to integrate the distributed database.

2.2. Database-Based Large-Scale Ontology Data Management Methodologies. To manage ontology-based data, a methodology of storing the ontology model and individuals through an ontology-based database representation is suggested. OntoDB will refine the classes into triples, represent the existing ontologies in the form of individuals of the triples, and store them in the database to locate the entire model and data of the ontology in the database. OntoDB maps the metaschema of the ontology and the metadata of the database. This is a reverse process of generating an ontology through the metadata, which can guarantee high query performance based on the OBDB [24–26].

Shah and Rabhi [27] created an ontology workflow for intelligent big data analytics. Automatic service composition is used to automate the process and design the ontology and rules to infer the workflow of the data analysis process according to the attributes of the data set and the user requirements. In this case, big data are organized, and a large amount of data are clustered to the ontology model to apply it to the workflow based on the ontology model.

2.3. Metric-Based Ontology Quality Analysis and Scoring. The ontology quality in a thesis is evaluated by comparing the evaluation metrics of two categories and compared with the score of the ontology that covers the developed specific domain. The first category is about how well the generated ontology model can represent the knowledge of the domain. The second category is about how well the ontology model extracts data from the target knowledge base (KB) and applies it to the individual.

The notations and ontology quality metrics of the ontology model in the ontology quality evaluation are based on OntoQA [17], OntoMetric [28], and OntoClean [29]. The summation process and scoring were modified to suit the method of the paper based on the ontology evaluation and ranking research using OQuaRE [30] and Tartir and Arpinar [31]. We redefine the ontology model as a set of five components O: {C, P, SC, R, I} to use metrics for the ontology quality generated by automatic ontology generation using the schema. Each component constructs an ontology model with C (classes), P (properties), SC (subclasses), R (all relations between classes C_1 and C_2), and I (individual).

Two metrics of OntoQA were selected to score the ontology model categories: relationship richness indicates the number of relationships in the ontology and attribute richness defines the average number of attributes in the entire class and indicates the quantity of knowledge that the schema represents. The following formula defines the relationship richness as an ontology configuration notation:

$$\mathrm{RR} = \frac{|\mathrm{P}|}{|\mathrm{SC}| + |\mathrm{P}|}. \tag{3}$$

As a result of the formula, a percentage is calculated that shows how much relation each class has with other classes, excluding subclass relations in the entire ontology. When the calculated value approaches zero, the ontology becomes a vertical ontology with small relationships among classes, and a closer value to 1 corresponds to a more relevant ontology. The following formula defines the attribute richness:

$$\mathrm{AR} = \frac{|\mathrm{att}|}{|\mathrm{C}|}. \tag{4}$$

As a result of the formula, the average number of attributes per class is calculated. A higher value better corresponds to the ontology model that expresses the attribute of domain knowledge. Two metrics of OntoQA were selected

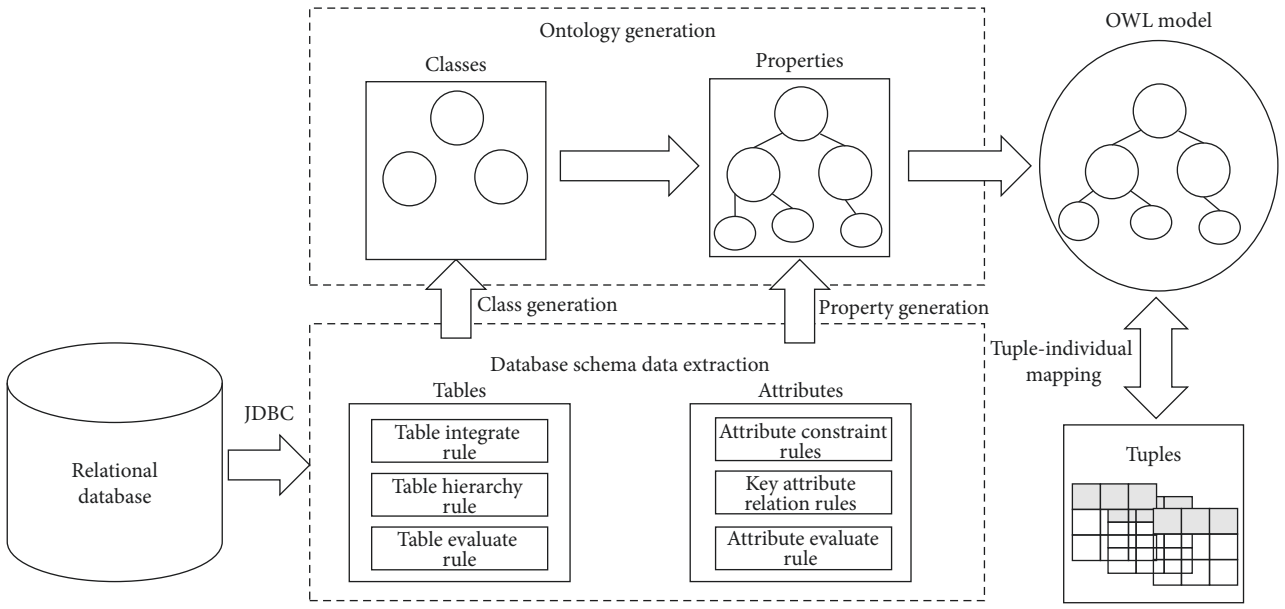

Figure 1: Automatic ontology generation from the RDB process.

to score the knowledge base category. The class richness is related to the degree to which individuals are distributed in a class. The average population compares the number of classes with the number of individuals and evaluates whether there are sufficiently many individuals for the class. The following formula defines the class richness:

$$\mathrm{CR} = \frac{|\mathrm{C}'|}{|\mathrm{C}|}. \tag{5}$$

The result of the formula is the percentage of classes that an individual has in the entire class. When the calculated value approaches zero, the corresponding ontology can be considered an ontology that cannot extract data from the knowledge base. The following formula defines the average population:

$$\mathrm{P} = \frac{|\mathrm{I}|}{|\mathrm{C}|}. \tag{6}$$

As a result of the formula, the average number of individuals per class is calculated. If the average number of individuals per class is not sufficient, the ontology model can be considered to have failed to extract all knowledge of the domain knowledge. OntoQA also analyzes the characteristics and quality of ontology through various metrics. Based on the analysis results, it shows how the ontology can represent the domain knowledge and how one can use the ontology accordingly.

3. Automatic Ontology Generation from Relational Database Schema

This section describes how to automatically generate an OWL ontology by importing a relational database schema. In detail, we describe the entire process of automatic creation of OWL ontology, required components of schema for the automatic generation, and applied rules to the ontology generation process using elements. The following process is performed to automatically generate the OWL ontology: (i) read and analyze the schema information of the database and convert it into the base information for the ontology generation. (ii) Create a class of ontologies based on the table information of the database. (iii) Based on the attribute information of the table, DatatypeProperty of the ontology class and ObjectProperty, which represents the relation among the classes, are created. Finally, the tuple in the database and the individual data of the OWL ontology are mapped using the generated ontology model. Figure 1 shows the automatic creation of an ontology using a database schema.

There are two reasons for mapping the tuple and OWL individual in the database. First, the mapping enables the ontology to use the tuple information for the actual integration and reasoning functions. The tuple data as owl: Namedindividual of the class created based on the table can be used to construct rules through the description logic and inference function of the ontology. Second, the mapping manages the newly created individual in the ontology in the database management system. In this case, the ontology is replaced by the tuple in the table by referring to the referenced table information when the ontology is automatically generated. This replacement enables one to use the existing relational database as an ontology and an ontology-based database instead of creating and using an ontology-based database. The methodology of converting a tuple into an individual is included in the automatic ontology generation process and described in this section. The opposite is explained in detail in Section 4. This process automatically generates an ontology based on schema information by entering schema data from the database. The OWL-DL XML file is generated as a result of the process. This XML file

contains information about the ontology model and the syntax for the individual.

Section 3.1 describes the notations of the relational database and basic configuration methodology that must be established to convert the database to an ontology. The configuration information of the database schema depends on the domain knowledge where the database is built and the intent of the designer who builds the database. As a result, we set the minimum database schema configuration constraints that can automatically generate an ontology using the database schema according to the process of the thesis. The notations in the database represent the components of the database and schema information that is read from the database in the actual automatic ontology generation application. Section 3.2 describes the rules applied when classes are created from the database tables. Section 3.3 explains the rules applied when we generate ontology properties from the attributes in a table. Finally, Section 3.4 explains the conversion of the database tuple into an individual in the generated OWL ontology model. The notations and some rules in this paper are adapted to the stepwise generation of the OWL ontology based on our previous studies [32] and the ontology database generation studies [23] for database integration.

3.1. Notations and Basic Constraints of the Database. To define the rules in the ontology automatic generation process, we defined each database component with the following notations.

T_x is represented by the table name x, which is a set of attributes in the database:

$$\mathrm{Attr}_y(T_x)\text{: Attribute name } y \text{ of table } T_x. \tag{7}$$

In other words, table x, which is a set of attributes, can be expressed by the following formula:

$$T_x = \cup \mathrm{Attr}_y \in T_x. \tag{8}$$

$\mathrm{PK}(\mathrm{Attr}_y, T_x)$ is represented by the primary key attribute $\mathrm{Attr}_y(T_x)$ in table T_x.

$\mathrm{FK}(\mathrm{Attr}_{y1}(T_1), \mathrm{Attr}_{y2}(T_2))$ is represented by the foreign key $\mathrm{Attr}_{y1}(T_1)$ in table T_1 and reference $\mathrm{Attr}_{y2}(T_2)$ in table T_2.

The four notations from the beginning are used to define the rules that create the basic ontology components. C_x is a more detailed representation of each component and used in the table to evaluate the rule to determine the details in the ontology component.

The database schema that can be applied to automatically generate the OWL ontology using the method in the paper must satisfy at least the following condition to consistently apply the automatic generation process by constraining the schema design method: the schema constraint in the automatic ontology generation process requires that the schema should be satisfied at least to automatically generate the ontology. The constraints are as follows:

(i) The subtables that inherit the characteristics of the parent with one-to-many relationships share the primary key with the parent.

(ii) A fragmented table with a one-to-one relationship has the primary key of the subtable of the parent table as a foreign key.

(iii) The schema of the database must satisfy the third normal form.

The first constraint classifies the general relationship between two tables of whether each table can be merged into one class during the ontology class creation. The second constraint determines the basic hierarchical structure of the ontology classes. The final constraint maintains the consistency of the relationship extraction among the tables. The database for the automatic ontology creation must be a relational database, and the database schema is represented by SQL-DDL. The ontology that is generated as an object is OWL-DL level and expressed in the XML file with a functional syntax.

3.2. Rules to Generate Ontology Classes from Database Tables. The first step of the automatic ontology generation is to create an ontology class based on the table information in the database. The information in the table is an annotation that indicates the name of the table, the dependency of the table with the foreign key and the primary key, and the purpose and characteristics of the table. An ontology class C_x is generated from table T_x that does not apply to the rule to be basically described. However, if the target table corresponds to the following two rules that determine that the conceptual separation of domain knowledge occurs at the schema design level, the merging or layering of the class is performed in the process of creating the class based on the table. The first rule is involved in the merger of classes. The following formula defines the class-merging rule:

$$\mathrm{PK}\left(\mathrm{Attr}_{y1}, T_2\right) = \mathrm{FK}\left(\mathrm{Attr}_{y1}(T_2), \mathrm{Attr}_{y2}(T_1)\right) \wedge \mathrm{PK}\left(\mathrm{Attr}_{y2}, T_1\right) \rightarrow \mathrm{PK}\left(\mathrm{Attr}_{y1}, T_2\right). \tag{9}$$

If the primary key of table T_2 is a foreign key of T_1 and the primary key of T_2 is a functional dependency of T_1, that is, if the primary key of T_1 determines the entire T_2, then T_2 can be considered one table of T_1. Because the primary key of T_2 is a functional subordinate to the primary key of T_1, it implies that T_1 generally determines its content in T_2 because all attributes of T_2 belong to T_1. In this case, T_2 has a one-to-one relationship with T_1.

The class hierarchy rule is involved in establishing a hierarchical structure between two classes. If two tables share the same key, one table becomes a lower-level table in the other table. In this case, the lower-layer table includes the attributes of the upper table, but the attributes of the upper table do not determine the attributes of the lower-layer table. The determination of a hierarchy in both tables depends on which key is referenced as a foreign key. The table that references a foreign key becomes a lower-level table. The subclasses created by applying this rule are the upper class and owl:subClassof relation. The following formula defines the class hierarchy rule:

$$\mathrm{PK}\left(\mathrm{Attr}_{y1}\left(\mathrm{T}_1\right), \mathrm{T}_1\right) = \mathrm{PK}\left(\mathrm{Attr}_{y2}\left(\mathrm{T}_1\right), \mathrm{T}_1\right) \wedge \mathrm{FK}\left(\mathrm{Attr}_{y1}\left(\mathrm{T}_1\right), \mathrm{Attr}_{y2}\left(\mathrm{T}_2\right)\right). \tag{10}$$

The table evaluation rule determines the table to which the transformed tuple belongs in the mapping procedure of the tuple and individual. When two tables are created as one table that satisfies the merging rule, the annotation of the class is the sum of the comments in the two tables. At the end of this step, the generated classes are inserted into the OWL ontology model based on the information of the tables. The OWL model at this stage includes classes with a basic hierarchy, and the property information is generated for these classes in the next step.

3.3. Rules to Generate Ontology Properties from the Database Attributes. In this step, ontology properties are created based on the attribute information of each table. OWL properties fall into two categories: DatatypeProperty, which is an attribute for the actual data that enter the individual, and ObjectProperty, which contains information about the constraints in the OWL and the relationships among the classes. owl:DatatypeProperty is created based on the attribute where the data value is stored in the table. Here, owl: DatatypeProperty has xsd:datatype as a range, where the class to which the attribute belongs is converted as a domain.

When a class is created based on two tables that do not correspond to the merging or layering rules in the previous step, the two generated classes will have an owl:Object-Property relation. The name owl:ObjectProperty is created with the "has a" prefix attached to the name of the foreign key attribute. The resulting owl:inverseProperty is automatically determined by the "is" prefix. The following formula defines an ObjectProperty generation rule based on a foreign key:

$$\mathrm{FK}\left(\mathrm{Attr}_{y1}\left(\mathrm{T}_1\right), \mathrm{Attr}_{y2}\left(\mathrm{T}_2\right)\right) \Rightarrow \mathrm{owl{:}\ ObjectProperty}\left(\mathrm{id} = \mathrm{has}Y_1\right), \quad \mathrm{domain{:}\ T_1,\ range{:}\ T_2}. \tag{11}$$

When the above rule is applied to create an Object-Property between two classes, the restriction on the ontology is determined according to the relationship cardinality of the two tables. The restriction is represented by the cardinality of how many classes a class can apply to an ObjectProperty of that type. Rule (12) defines the ontology restriction according to the table relationship:

$$\text{cardinality} = \begin{cases} \text{one to one restriction: 1 both} \\ \text{one to many restriction: 1 one} \\ \text{many to many: no restriction} \end{cases} \tag{12}$$

The attribute evaluation rule is involved in determining the owl:functionalProperty restriction when the conditions of the attributes of the table are not null and unique, or autoincrease. Table 1 lists the conditions that an attribute can have and the corresponding property constraints.

Table 1: Table evaluation rule.

Attribute conditions	OWL restriction
Primary key	owl:cardinality = 1
Not null	owl:mincardinality = 1
Unique	owl:maxcardinality = 1
Check-in	owl:one of
Unsigned	rdf:datatype = &xsd:nonNegativeInteger

At the end of this step, an OWL ontology model is created, which contains the classes and properties of the class. Each property of a class contains a DatatypeProperty that represents the attributes of the table and an Object-Property that represents the relationship of the foreign key among the tables.

3.4. Mapping Procedure for the OWL Individual and Tuples. In this step, the tuple of each table stored in the database is mapped to the ontology individual based on the generated OWL ontology model. The mapping matches each tuple to the individual and asserts the corresponding Namedindividual to the table to which the tuple belongs. The name of the individual that is mapped to each tuple is the index attribute value of the corresponding table. It is the attribute value of the corresponding tuple of properties to the generated individual. Figure 2 shows the mapping process of database tuples and individuals to the automatically generated ontology model.

An attribute value of an ObjectProperty other than a DatatypeProperty becomes an ObjectProperty value that is related to an individual of another class in an individual. In this procedure, we refer to the tuple of both tables. The mapping process is not performed for all tuples, but the user can select a tuple of the desired category and map to an individual. Thus, it is possible to select the data category and determine the relevance of each data and the conceptual meaning of the data set through the reasoning function in the ontology model of the data.

4. Database Schema-Based Large-Scale Ontology Individual Management

This section describes how to convert an individual into a mapped tuple and store it in a database management system to efficiently manage a large ontology individual in the OWL. The last step in the automatic creation of an ontology in Section 3 is to map the individuals in the generated ontology model and the tuples stored in the database and assert each individual into classes based on the table. The tuple transformation of the individual proceeds with the inverse transformation of the described process.

When large-scale ontology individuals continue to grow, the individuals asserted in the class in the OWL model are converted into tuples and stored in the database. In this process, an automatic ontology generation program parses each individual and analyzes the parsed individual using the mapping information in the program. After the individual data are processed, the program converts them into a tuple,

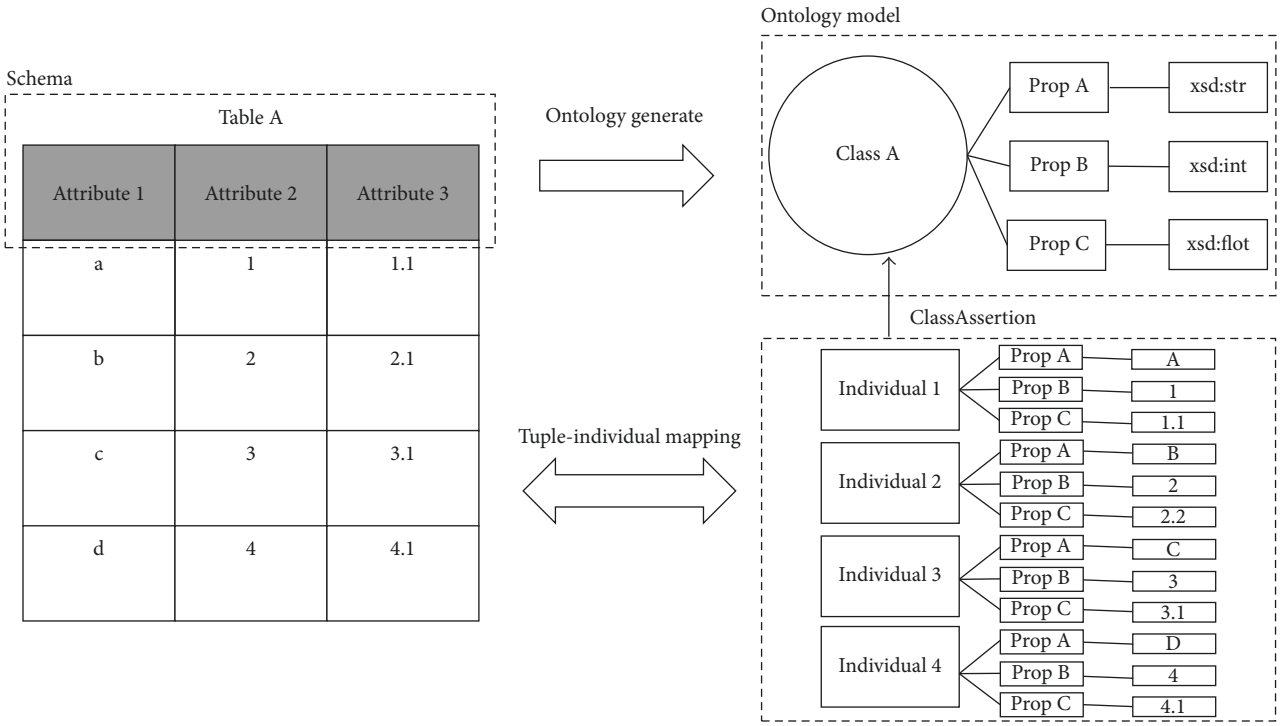

Figure 2: Mapping between a tuple and an individual process.

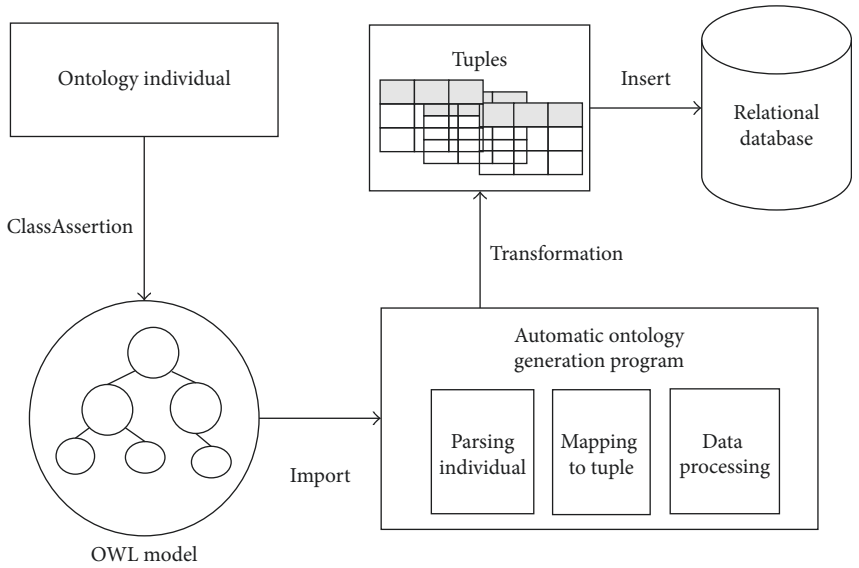

Figure 3: Process of inserting an individual to a database.

inserts the corresponding tuple into the table, and saves it. Figure 3 shows the process of storing an individual in a database.

By storing individuals in the database, one can manage large amounts of data while minimizing the increase in size of the ontology. The stored ontology individual can be reverted to the individual at any time using the mapping information. Using the methodology, we can minimize the cost of system implementation and ontology development by implementing the middleware for the interaction between the relational database and the ontology of a known type without building a database based on a specific ontology.

This process is the inverse process of tuple and individual mapping, which is the final step in the process of automatically generating the OWL ontology from the database schema. A manually created ontology allows the developer to choose the creation of an individual. However, because the process is done automatically, all tuples are converted into individuals. Thus, individuals can be stored in the form of tuples in the database to prevent too many individuals

```
smart_watchtable:
        Create table smart_watch(device_id, watch_idintnot null unsigned, detail_idint, watch_netadress varchar(20) not null,
        watch_own varchar(20), whatch_consumenergy float not null unsigned,
        Primary key(device_id),
        Unique index watch_id_index(watch_id ASC), watch_netaddress_index(watch_netaddress ASC),
        foreign key(device_id) references device(device_id),
        foreign key(watch_own) references people(people_name),
        foreign key(detail_id) references smart_watch_detail(detail_id),
        comment("name:smart_watch));

smart_watch_detail table:
        Create table smart_watch_detail(detail_id int, model_name varchar(40),model_marker varchar(20), model_spec varchar(40),
        Primary key(detail_id),
        comment(name:smart_watch_detail, oneOf:smart_watch));

smart_watch_sportmodel table:
        Create table smart_watch_sportmodel(watch_id int, sport_waterproof Boolean not null, sport_dirtproof boolean not null,
        sport_uniquespec varchar(20),
        Primary Key(device_id),
        CHECK(waterproof = = true, dirtproof = = true),
        foreign key(watch_id) references smart_watch(watch_id),
        comment(name:smart_watch_sportmodel));

smart_watch_classicmodel table:
        Create table smart_watch_classicmodel(watch_id int, watch_classicspec varchar(40), watch_classicnum int not null unsigned,
        Primary key(device_id),
        foreign key(watch_id) referenes smart_watch(watch_id),
        comment(name:smart_watch_classicmodel));

smart_watch_priminummodel table:
        Create table smart_watch_priminummodel(watch_id int, watch_priminumspec varchar(40), watch_primiumopt booleannot null,
        watch_priminumnum int not null unsigend.
        Primary key(device id),
        foreign key(watch_id) references smart_watch(watch_id),
        comment(name:smart_watch_priminummodel));
```

Figure 4: Smart_watch table SQL-DDL.

from accumulating in an ontology. As a result, this process extends the ontology management system in conjunction with the database.

One can use both SPARQL and SQL to query the database for interaction with the ontology model. If there are no required data for the query, the program searches for them in the remaining models and generates the necessary data for the query. To convert an individual into a tuple, the criterion to determine the tuple to convert to a table among the tables to which the merging rule is applied is the annotation information to which the table-evaluating rule in the merged class is applied.

5. Implementation

This section shows the process of automatically generating an OWL ontology using the actual database schema and the progress of the methodology through the intermediate output generated during the implementation. The target domain of the target database is for the smart home. Therefore, the schema of the target database contains tables and properties for storing data generated in the smart home. The entire schema consists of 85 tables, each of which contains one or more attributes. Each table has a hierarchical relationship according to the smart home unit through the foreign key. In this paper, we describe the automatic conversion process by selecting one of these tables as an example and outputting the result of the intermediate process of converting the corresponding table and attributes into classes.

An example table to illustrate the ontology autogeneration process contains the data for the smartwatch of the device in the smart home database. The smartwatch table is one of the tables that are managed by the device table and manages multiple smartwatch models and tables that contain detailed information on the smartwatch. The smartwatch table has attributes for managing the data of the entire smartwatch. The example in Figure 4 shows a smart_watch table and the tables managed by the smartwatch table of SQL statements.

The program reads the SQL-DDL, identifies the table and the attributes of the table, and converts the table into a class. Since the smart_watch_detail table satisfies the merging rule with the smart_watch table, a class is created. The name of the generated class will be the smart_watch class that governs the merger. The subattributes of the smart_watch_detail table are created as properties of the smart_watch class with the attributes of the smart_watch table, smart_watch_sportmodel, and smart_watch_classicmodel. Since the smart_watch_sportmodel table satisfies the layering rule, classes that are subclasses of the

```
Declaration(Class(:smart_watch))
SubCLassOf(:smart_watch:device)
AnnotationAssertion(rdfs:comment: smart_watch"name:smart_watchon of smart_watch_detail"@en)

Declaration(Class:smart_watch_sportmodel)
SubCLassOf(:smart_watch:smart_watch_sportmodel)
AnnotationAssertion(rdfs:comment: smart_watch_sportmodel"name:smart_watchon of smart_watch_sportmodel"@en)

Declaration(Class:smart_watch_primiummodel)
SubCLassOf(:smart_watch:smart_watch_primiummodel)
AnnotationAssertion(rdfs:comment: smart_watch_primiummodel"name:smart_watchon of smart_watch_primiummodel"@en)

Declaration(Class:smart_watch_classicmodel)
SubCLassOf(:smart_watch:smart_watch_classicmodel))
AnnotationAssertion(rdfs:comment: smart_watch_classicmodel) "name:smart_watchon of smart_watch_classicmodel"@en)
```

Figure 5: OWL functional syntax of the generated classes based on information of tables.

```
Declaration(DataProperty(:watch_id))
DataPropertyDomain(:watch_id ObjectSomeValuesFrom(owl:isdataproperty:smart_watch))
DataPropertyRange(:watch_idxsd:int)
DataPropertyRange(:watch_idowl:has_aspect:not_null)
DataPropertyRange(:watch_idowl:has_aspect:unique)

Declaration(DataProperty(:watch_netaddress))
DataPropertyDomain(watch_netaddressObjectSomeValuesFrom(owl:isdataproperty:smart_watch))
DataPropertyRange(:watch_netaddressxsd:string)
DataPropertyRange(:watch_netaddressowl:has_aspect:not_null)
DataPropertyRange(:watch_netaddressowl:has_aspect:unique)
DataPropertyRange(:watch_netaddressowl:has_aspect:unsigned)

Declaration(DataProperty(:watch_consumenergy))
DataPropertyDomain(:watch_consumenergy ObjectSomeValuesFrom(owl:isdataproperty:smart_watch))
DataPropertyRange(:watch_consumenergyxsd:float)
DataPropertyRange(:watch_consumenergyowl:has_aspect:not_null)
DataPropertyRange(:watch_consumenergyowl:has_aspect:unsigned)

Declaration(DataProperty(:model_name))
DataPropertyDomain(:model_name ObjectSomeValuesFrom(owl:isdataproperty:smart_watch))
DataPropertyRange(:model_namexsd:string)

Declaration(DataProperty(:model_maker))
DataPropertyDomain(:model_makerObjectSomeValuesFrom(owl:isdataproperty:smart_watch))
DataPropertyRange(:model_makerxsd:string)

Declaration(DataProperty(:model_spec))
DataPropertyDomain(:model_specObjectSomeValuesFrom(owl:isdataproperty:smart_watch))
DataPropertyRange(:model_specxsd:int)
```

Figure 6: OWL functional syntax of the properties of the smart_watch class.

smart_watch class are created. The name of the created class is identical, and the attributes of each table are created as properties. In the annotation of the generated class, comments of the tables are inputted by the table evaluation rule. Figure 5 shows the OWL functional syntax of the generated classes based on table information.

The properties of the generated classes are generated based on the attribute information in the table. An attribute with a foreign key relationship is involved in the creation of an ObjectProperty by applying a key attribute relation rule. The attributes other than the foreign key are generated as a DatatypeProperty with actual data by applying the attribute evaluation rule and the attribute constraint rule. In the SQL-DDL, the watch_own attribute of the smart_watch table is related to the people_name of the people table. Key attribute relation is as follows: when an object property is created based on the watch_own attribute in the rule, the has_watch property is created by prefixing the attribute name. The inverse object property of this property, which is have_watch, is automatically generated. Figure 6 shows the OWL functional syntax of the properties generated based on the attribute information.

After the property creation is completed, the smart_-watch class and properties are created based on the smart_watch table. Then, we map the tuples of the smart_home table to the individual. Mapping is based on the data in each tuple and refers to the data in the people table via the foreign key. Figure 7 shows the mapping results for the top three tuples.

The generated individuals are used as data of an artificial intelligence system such as the smart home environment analysis and correspondence method processing. When too many individuals are created, some unused individuals are inverted into tuples and stored in the database.

```
Declaration(Namedindividual(:smart_watch_1))
ClassAssertion(:smart_watch_1 :smart_watch)
ObjectPropertyAssertion(:has_watchJohn)
DataPropertyAssertion(:watch_id:smart_watch_1 "00000001"^^xsd:int)
DataPropertyAssertion(:watch_netaddress:smart_watch_1 "192.168.0.2"^^xsd:string)
DataPropertyAssertion(watch_consumenergy:smart_watch_1 "38.2"^^xsd:float)
DataPropertyAssertion(model_name:smart_watch_1 "model1"^^xsd:string)
DataPropertyAssertion(model_maker:smart_watch_1 "maker1"^^xsd:string)
DataPropertyAssertion(model_spec:smart_watch_1 "spec1"^^xsd:string)

Declaration(Namedindividual(:smart_watch_2))
ClassAssertion(:smart_watch_1 :smart_watch)
ObjectPropertyAssertion(:has_watchJohn)
DataPropertyAssertion(:watch_id:smart_watch_1 "00000002"^^xsd:int)
DataPropertyAssertion(:watch_netaddress:smart_watch_2 "192.168.0.3"^^xsd:string)
DataPropertyAssertion(watch_consumenergy:smart_watch_2 "64.5"^^xsd:float)
DataPropertyAssertion(model_name:smart_watch_2 "model2"^^xsd:string)
DataPropertyAssertion(model_maker:smart_watch_2 "maker2"^^xsd:string)
DataPropertyAssertion(model_spec:smart_watch_2 "spec2"^^xsd:string)

Declaration(Namedindividual(:smart_watch_3))
ClassAssertion(:smart_watch_1 :smart_watch)
ObjectPropertyAssertion(:has_watchJohn)
DataPropertyAssertion(:watch_id:smart_watch_3 "00000003"^^xsd:int)
DataPropertyAssertion(:watch_netaddress:smart_watch_3 "192.168.0.4"^^xsd:string)
DataPropertyAssertion(watch_consumenergy:smart_watch_3 "32.3"^^xsd:float)
DataPropertyAssertion(model_name:smart_watch_3 "model3"^^xsd:string)
DataPropertyAssertion(model_maker:smart_watch_3 "maker3"^^xsd:string)
DataPropertyAssertion(model_spec:smart_watch_3 "spec3"^^xsd:string)
```

Figure 7: Generated individuals of the smart_watch table.

6. Validation

In this section, we evaluate the existing ontologies and automatically generated ontologies according to the ontology quality score. Using the score, we prove whether the automatically generated ontology can represent the domain knowledge. The result also demonstrates the effectiveness of instance management through the tuple-individual transformation by comparing the rate of increase in the database size with the OWL file when the ontology individual consistently occurs. The ontology quality scoring is defined according to the automatic ontology generation based on the notations and the OntoQA formula of the elements that constitute the ontology model in the related study. The category of the generated ontology model that represents the knowledge of the domain is evaluated by the sum of the relationship richness and the attribute richness of OntoQA. The score for the ontology model can be calculated with the following formula:

$$\text{Score}_{\text{om}} = \frac{(|R|*|C|*100) + ((|SC| + |R|)*|P|)}{(|SC| + |R|)*|C|}. \tag{13}$$

The calculated value shows the degree of the generated ontology with more relations and the index of the average attribute for each class. This value enables us to evaluate whether the automatically generated ontology reflects the relationships and characteristics of the actual domain knowledge. The database schema, which is the base knowledge of the ontology generation by evaluating the ontologies, can be determined to be suitable as the metadata for the ontology generation. Furthermore, the automatically generated ontology is about the efficiency in extracting the base knowledge, and the score for the knowledge base is summed with the class richness and average distribution across all classes of OntoQA. The score for the knowledge base extraction can be calculated using the following formula:

$$\text{Score}_{\text{kb}} = \frac{(C'*100) + (|I|)}{|C|}. \tag{14}$$

The calculated value indicates how well the generated ontology can extract the value to the knowledge base of interest. Thus, it is possible to confirm whether the automatically generated ontology can extract and use the domain knowledge data. We can verify whether the ontology model can reflect the tuple in all databases when we map the tuple and individual of a database, which is the basis of the automatic ontology generation. To evaluate the quality, we use similar ontologies of the smart home environment to the domain knowledge of the thesis. BOnSAI [33] is a smart building ontology for context awareness and conceptual integration for ambient intelligence. The ThinkHome [15] ontology is a comprehensive ontology of the smart home for the energy efficiency of the future smart home. Table 2 shows the ontology components of 3 ontologies for the smart home, including the automatically generated ontology.

By calculating the quality score of each ontology using formulas (13) and (14), we derived the result of

Table 2: Summary of smart home ontologies.

Ontology name	Classes	Individuals	Subclasses	Object property	Data property
ThinkHome	264	427	551	237	207
BOnSAI	99	5	90	76	41
Automatic generation ontology	76	124	70	43	194

$\text{Score}_{om} = \{30, 46, 40\}$ and $\text{Score}_{kb} = \{19.6, 26\}$ in the order of Table 2. The automatically generated ontology from the database schema information is lacking in ontology construction compared to the existing ontology, but it is sufficiently available compared with the basic ontology because the knowledge base extraction rate is high based on the database. Figure 8 shows the quality score of each ontology.

To prove the usefulness of the method of transforming individuals into tuples and storing them in a database when an individual is continuously generated in an automatically generated ontology, a new individual is continuously generated in the above automatic generation ontology. We compared the capacity growth rate when individuals were continuously stored in the ontology model and when individuals were converted into tuples and stored in the database. Figure 9 shows the storage capacity growth rate when the number of individuals increases.

When the individuals of a thousand units were continuously generated, the individual was stored in the OWL file and the tuple was stored in the database. As a result, the relational database slowly increased in the capacity growth rate compared to the ontology model when the number of individuals constantly increased. The database could effectively manage individuals in a tuple through data compression and management.

An ontology application based on the automatically generated ontology was created to confirm that the automatically generated ontology was usable and the context awareness using ontology was confirmed. Through the data generated by each smart device, the smart home application controls the power consumption of the IoT devices in the home based on the location measured in the human wearable equipment. Figure 10 shows the change in power consumption according to the user activity time.

When the user was at home, the power consumption of devices such as AI speakers and lamps was high and the power consumption of the devices activated by the users, such as smart flower pot and CCTV, increased. This is a result of the application that recognizes the GPS context awareness from the smartwatch and adjusts the power consumption of each device.

7. Conclusion and Future Work

In this paper, we describe a method to automatically generate OWL-DL using the database schema and store individuals in a database management system when many individuals occur in the generated ontology model in the database, where the ontology is generated. The ontology quality score also defines an indicator of how well the ontology represents the domain and compares it with other ontologies, which proves that the automatically generated

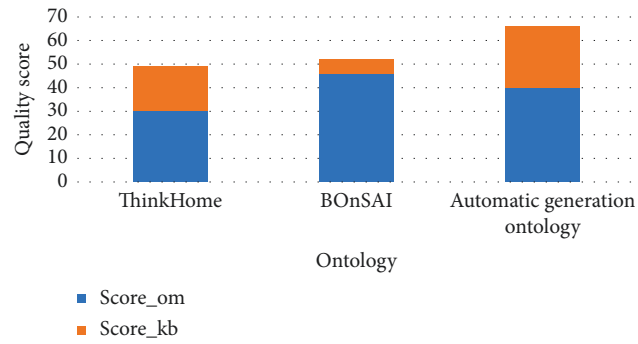

Figure 8: Ontology quality score.

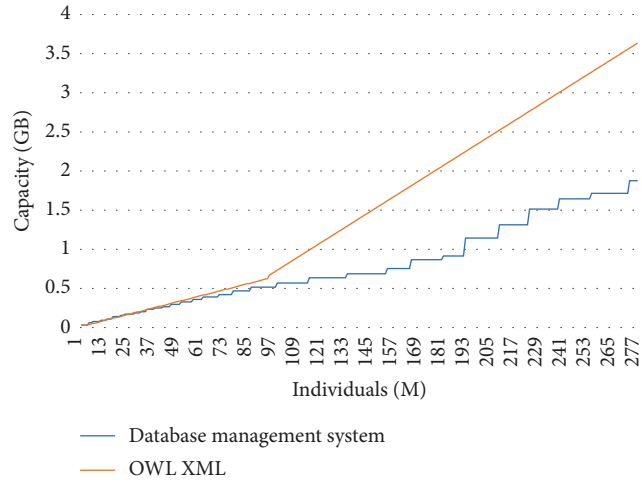

Figure 9: Storage capacity growth rate when the number of individuals increases.

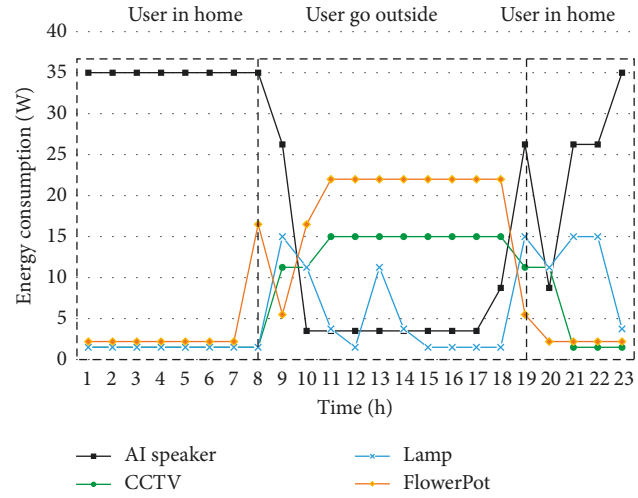

Figure 10: Power consumption in the automatic generation ontology-based smart home application.

ontology from the database schema can be used in the main application. In the proposed ontology model, the ontology can be efficiently managed by using the database schema and each individual can be integrated to reduce the overhead of the storage space.

The process of reading and analyzing the database schema information, automatically generating the ontology model using the analyzed information, and managing the instance through the database on which the ontology is created will be explained through an example. The ontology and the database can interact. In the ontology model, there is difficulty in expanding beyond the basic inclusion relation through the relation between the tables when we apply the automatic ontology generation method. In future studies, to apply the complex inference relation in the OWL, a constraint in the ontology model is considered based on the foreign key relationship between the database tables and the attribute information in a table and the optimization of the storage structure to store individuals to the database.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by The Leading Human Resource Training Program of Regional Neo Industry through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (no. NRF-2016H1D5A1909989). This research was also supported by the MISP (Ministry of Science, ICT and Future Planning), Korea, under the SW Master's Course of Hiring Contract Program (H0116-16-1015) supervised by the IITP (Institute for Information & Communications Technology Promotion).

References

[1] N. Guarino, "Formal ontology and information systems," *Proceedings of FOIS*, vol. 98, no. 1998, 1998.

[2] D. Ga, D. Djuric, and V. Deved, *Model Driven Architecture and Ontology Development*, Springer Science and Business Media, Berlin, Germany, 2006.

[3] D. Gašević, N. Kaviani, and M. Milanović, "Ontologies and software engineering," in *Handbook on Ontologies*, pp. 593–615, Springer, Berlin Heidelberg, Germany, 2009.

[4] N. F. Noy and D. L. McGuinness, *Ontology Development 101: A Guide to Creating Your First Ontology*, Stanford University, Stanford, CA, USA, 2001.

[5] B. Succar, "Building information modelling framework: a research and delivery foundation for industry stakeholders," *Automation in Construction*, vol. 18, no. 3, pp. 357–375, 2009.

[6] Y. Sure, S. Staab, and R. Studer, "Methodology for development and employment of ontology based knowledge management applications," *ACM SIGMOD Record*, vol. 31, no. 4, pp. 18–23, 2002.

[7] N. Yahia, S. A. Mokhtar, and A.-W. Ahmed, "Automatic generation of OWL ontology from XML data source," 2012, http://arxiv.org/abs/1206.0570.

[8] L. Dey, M. Abulaish, R. Goyal, and K. Shubham, "A rough-fuzzy ontology generation framework and its application to bio-medical text processing," *Advances in Intelligent Web Mastering*, pp. 74–79, Springer, Berlin Heidelberg, Germany, 2007.

[9] W. Chen, Q. Yang, L. Zhu, and B. Wen, "Research on automatic fuzzy ontology generation from fuzzy context," in *Proceedings of the Second International Conference on Intelligent Computation Technology and Automation (ICICTA)*, vol. 2, pp. 764–767, Zhangjiajie, China, October 2009.

[10] Q. T. Tho, S. C. Hui, A. C. M. Fong, and T. H. Cao, "Automatic fuzzy ontology generation for semantic web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 6, pp. 842–856, 2006.

[11] H.-T. Zheng, C. Borchert, and H.-G. Kim, "A concept-driven automatic ontology generation approach for conceptualization of document corpora," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 1, pp. 352–358, Sydney, Australia, December 2008.

[12] Y. A. Tijerino, D. W. Embley, D. W. Lonsdale, Y. Ding, and G. Nagy, "Towards ontology generation from tables," *World Wide Web*, vol. 8, no. 3, pp. 261–285, 2005.

[13] R. Agrawal, A. Somani, and Y. Xu, "Storage and querying of e-commerce data," in *Proceedings of the Very Large Data Bases (VLDB)*, pp. 149–158, Rome, Italy, 2001.

[14] S. Harris and N. Gibbins, "3store: efficient bulk RDF storage," in *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems (PSSS)*, Sanibel Island, FL, USA, October 2003.

[15] C. Reinisch, M. J. Kofler, F. Iglesias, and W. Kastner, "ThinkHome energy efficiency in future smart homes," *EURASIP Journal on Embedded Systems*, vol. 2011, no. 1, p. 104617, 2011.

[16] S. L. Osborn and T. E. Heaven, "The design of a relational database system with abstract data types for domains," *ACM Transactions on Database Systems (TODS)*, vol. 11, no. 3, pp. 357–373, 1986.

[17] S. Tartir, I. B. Arpinar, M. Moore, A. P. Sheth, and B. Aleman-Meza, *OntoQA: Metric-Based Ontology Quality Analysis*, Wright State University, Dayton, OH, USA, 2005.

[18] Y. A. Tijerino, "Ontology generation from tables," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE)*, Rome, Italy, December 2003.

[19] A. Pivk, P. Cimiano, Y. Sure, M. Gams, V. Rajkovič, and R. Studer, "Transforming arbitrary tables into logical form with TARTAR," *Data & Knowledge Engineering*, vol. 60, no. 3, pp. 567–595, 2007.

[20] V. Long, "An agent-based approach to table recognition and interpretation," Ph.D. thesis, Macquarie University, Sydney, NSW, Australia, 2010.

[21] S. Zhou, H. Ling, M. Han, and H. Zhang, "Ontology generator from relational database based on Jena," *Computer and Information Science*, vol. 3, no. 2, p. 263, 2010.

[22] J. Bakkas and M. Bahaj, "Generating of RDF graph from a relational database using Jena API," *International Journal of Engineering and Technology*, vol. 5, no. 2, pp. 1970–1975, 2013.

[23] N. Alalwan, H. Zedan, and F. Siewe, "Generating OWL ontology for database integration," in *Proceedings of the Third International Conference on Advances in Semantic Processing (SEMAPRO)*, Sliema, Malta, October 2009.

[24] H. Dehainsala, G. Pierra, and L. Bellatreche, "OntoDB: an ontology-based database for data intensive applications," in *Proceedings of the International Conference on Database*

Systems for Advanced Applications (DASFAA), vol. 7, Suzhou, China, March 2007.

[25] S. Jean, H. Dehainsala, D. Nguyen Xuan, G. Pierra, L. Bellatreche, and Y. Aït-Ameur, "OntoDB: it is time to embed your domain ontology in your database," in *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, Suzhou, China, March 2007.

[26] L. Al-Jadir, C. Parent, and S. Spaccapietra, "Reasoning with large ontologies stored in relational databases: the OntoMinD approach," *Data & Knowledge Engineering*, vol. 69, no. 11, pp. 1158–1180, 2010.

[27] T. Shah, F. Rabhi, and P. Ray, "Investigating an ontology-based approach for Big Data analysis of inter-dependent medical and oral health conditions," *Cluster Computing*, vol. 18, no. 1, pp. 351–367, 2015.

[28] A. Lozano-Tello and A. Gómez-Pérez, "OntoMetric: a method to choose the appropriate ontology," *Journal of Database Management*, vol. 2, no. 15, pp. 1–18, 2004.

[29] N. Guarino and C. A. Welty, "An overview of OntoClean," in *Handbook on Ontologies*, pp. 201–220, Springer, Berlin Heidelberg, Germany, 2009.

[30] A. Duque-Ramos, J. T. Fernández-Breis, M. Iniesta et al., "Evaluation of the OQuaRE framework for ontology quality," *Expert Systems with Applications*, vol. 40, no. 7, pp. 2696–2703, 2013.

[31] S. Tartir and I. B. Arpinar, "Ontology evaluation and ranking using OntoQA," in *Proceedings of the International Conference on Semantic Computing (ICSC)*, Irvine, CA, USA, September 2007.

[32] J.-H. Ahn and Y. B. Park, "Rule extraction ontology generation from an adaptive IoT ecosystem database," in *Proceedings of the International Conference on ICT Convergence*, Jeju Island, Korea, October 2017.

[33] T. G. Stavropoulos, D. Vrakas, D. Vlachava, and N. Bassiliades, "BOnSAI: a smart building ontology for ambient intelligence," in *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, Craiova, Romania, June 2012.