IEIE Transactions on Smart Processing and Computing, vol. 5, no. 3, June 2016
http://dx.doi.org/10.5573/IEIESPC.2016.5.3.222

A Novel Approach for Accessing Semantic Data by Translating RESTful/JSON Commands into SPARQL Messages

Khiem Minh Nguyen, Hai Thanh Nguyen, and Hiep Xuan Huynh

College of Information and Communication Technology, Cantho University / Vietnam
{nmkhiem, nthai, hxhiep}@cit.ctu.edu.vn

* Corresponding Author: Hai Thanh Nguyen

Received April 20, 2016; Revised May 15, 2016; Accepted June 3, 2016; Published June 30, 2016

* *Extended from a Conference: Preliminary results of this paper were presented at the ICEIC 2016. This present paper has been accepted by the editorial board through the regular reviewing process that confirms the original contribution.*

Abstract: Linked Data is a powerful technology for storing and publishing the structures of data. It is helpful for web applications because of its usefulness through semantic query data. However, using Linked Data is not easy for ordinary users who lack knowledge about the structure of data or the query syntax of Linked Data. For that problem, we propose a translator component that is used for translating RESTful/JSON request messages into SPARQL commands based on ontology – a metadata that describes the structure of data. Clients do not need to worry about the structure of stored data or SPARQL, a kind of query language used for querying linked data that not many people know, when they insert a new instance or query for all instances of any specific class with those complex structure data. In addition, the translator component has the search function that can find a set of data from multiple classes based on finding the shortest paths between the target classes - the original set that user provide, and target classes- the users want to get. This translator component will be applied for any dynamic ontological structure as well as automatically generate a SPARQL command based on users' request message.

Keywords: Translator component, API, RESTful/JSON to SPARQL, Linked data search

1. Introduction

Linked Data [1], especially Resource Description Framework (RDF) [2], is a technology that aims to store graph databases effectively. In order to access the RDF repository, we need to use SPARQL [3] which is a special query language for manipulating the data in a Linked Data server. An ontology is the way to design a linked big data structure for a distributed system to allow users to use RESTful/JSON [4] requests to access servers.

Clients have met with difficulties over how to easily communicate with RDF databases. Clients could use a uniform resource identifier (URI) as the path that contains the data request and send it to the Linked Data server. However, the challenge is that the clients must know the graph structure and how they can represent the graph structure of the Linked Data source in a request message. Conversely, clients could use a SPARQL query as part of a request message to access Linked Data, but they still need to know the syntax of the SPARQL language and graph structure as well.

With RESTful/JSON technology, the ordinary users (who do not know much about graph databases or SPARQL syntax) send requests to an ordinary server (not a Linked Data server) and receive the result easily, because this is a popular technique and most of them are already familiar with it. However, there are numerous difficulties for them when contacting graph databases in the Linked Data server because they need to know how to construct a SPARQL statement instead using a simple query, like JSON strings.

In this paper, we propose a new approach that is an on-line syntactic and semantic translation service to help users more easily access a Linked Data server.

This translation service converts RESTful/JSON messages into SPARQL commands based on the ontology

of the target semantic data. Such a translation service enables RESTful web service clients to access ontology-based RDF repositories without knowledge of the semantic data ontology and without the need to issue pattern-matching SPARQL commands.

The users just send simple JSON strings requested by RESTful technology, and then the JSON strings are dynamically translated into SPARQL syntax. Users do not need to worry about how to use JSON strings to access a Linked Data server that contains a graph database. This problem is solved by the translation and is handled smoothly and seamlessly. This translation is useful because it supports a way to help people use a popular technique to easily connect with a new technique. Besides that, it is an automatic translation for every data structure defined by users, and makes it comfortable for them to access the Linked Data server.

For each data access, the translation service produces a minimal set of SPARQL commands by traversing the ontological structure of the semantic data, especially Object Properties and Datatype Properties. Besides that, we also provide a solution to the class hierarchy problem because its structure may cause SPARQL to return an empty result. There have two kinds of class in a hierarchy's structure (abstract class and concreate class) and both of them have a relationship with other classes that is described in the ontology definition. However, only concreate class has instances, while abstracts do not have instances. If the SPARQL command is built based on an abstract class, we may get an empty set of instances as a result and should avoid that.

The rest of this paper is organized as follows. In Section 2, we discuss the related work. A model of a translator component, and how to automatically generate a SPARQL statement, are introduced in section 3 and 4. Three basic operations are supported for this translator (1) searching specific instances of a class, (2) inserting a new instance of a class while enforcing the cardinality restriction specified in the ontology in order to maintain semantic consistency among the instances, and (3) a search function, that generates a SPARQL query based on the client's input data set: origin (query restriction) and target (query projection) sets. Some scenarios for functions of a translator are evaluated in the section 5. Finally, we conclude the paper in section 6.

2. Related Works

Linked Data is a challenging field in which many authors have attempted to propose many models for solving problems. There have been studies about Linked Data, such as database applications for interacting with and sharing multiple layers of distributed systems that are used to track human brain waves [5]. This research achieved a significant accomplishment in real-time forecasting of human awareness states in real life situations by combining intelligent sensors.

One of the studies of the hierarchy class issue in an ontology is a mapping method that connects a set of concepts such as the name of an entity, its relationships, etc. [6]. However, this method applies to multiple ontologies that map together, not to one specific ontology that has a complex structure of a hierarchy class. Another result for a specific area that uses ontology construction and reasoning using Web Ontology Language (OWL) also had a real experiment and solved a real-world issue [7].

Besides that, the study of automatically generating query samples based on an ontology structure also achieved accurate results. However, it was only research for conversion and integration among ontologies [8]. Other works can be found [9, 12]. Moreover, research into extension of the SPARQL Ontology Query Language with four types ("Adjacent", "Opposite", "Vertical" and "Contain") only solved the problem of IndoorSPARQL functions used to support quantitative spatial computations [10]. In addition, research into using SPARQL to create a graph from a relational database made a good contribution to applying information that is stored in a traditional database into Web Semantic [11].

One research effort into the relationship between RESTful and SPARQL is useful for generating semantic sensor data from existing data sources [14]. The authors just showed how to use a RESTful API to publish sensor data into a Linked Open Data Cloud. However, they did not mention SPARQL applications for fully supporting for ordinary users to easily manipulate semantic data in a Linked Data server, such as retrieving and searching for data. The approach proposed in our paper is a useful tool that helps users automatically translate simple JSON queries into SPARQL commands. Users only need to know how to indicate the query's information sent to the servers via RESTful technology. The inside process will handle the complex structure of a graph database and will generate a SPARQL statement.

Some studies also presented research on RDF. The authors in [12] revealed some approaches that reverse some of the complications of adding semantic annotations, exposing those patterns in the data. A simple model on an RDF kernel was also presented by Bloem at al. [13]. In addition, the works in [11] demonstrated a context-aware approach to keyword query interpretation, which addresses the novel problem of using a sequence of structured queries corresponding to interpretations of keyword queries. Similarly, a considerable number of studies have been attempted into proposed approaches to RDF query [15-18].

3. Modeling

This translator is a middle component that provide a helpful way for simple JSON strings supported by popular technique (RESTful) to access a complex data structure supported by a new technology (Linked Data). Generating a SPARQL statement from a JSON string request inside the translator is dynamic and seamless to users. It is an ideal way to help a user can use Linked Data without knowing about the SPARQL syntax or graph data structure.

In order to build a translator that translates RESTful/JSON into SPARQL command, we needed to base it on the structure of metadata in the ontology, which

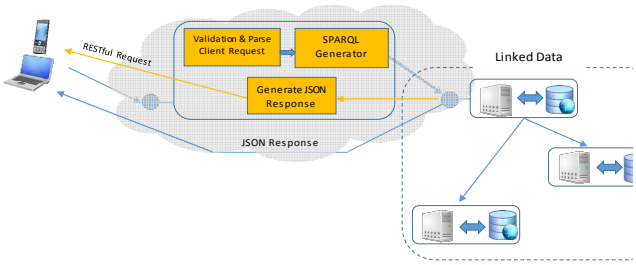

Fig. 1. The architecture of translator.

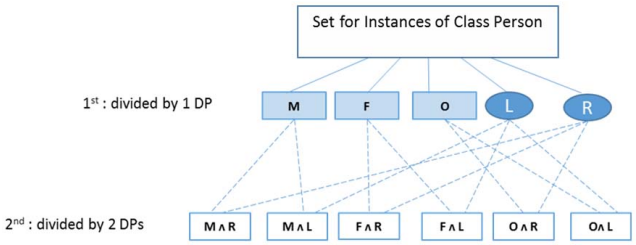

Fig. 2. The data type property set for Person.

specifies all of the classes as well as the relationships among these classes. With this translator component, the web service can easily interact with an RDF repository. The translator component has an interface that accepts client access and sends the request message. After that, our translator will validate or parse the client request and generate the SPARQL, connect Linked Data server to query information, construct the result in a JSON format, then send the response data to clients. In real world scenarios, the Linked Data server does respond with the result directly to clients (As seen in Fig. 1).

The translator is designed based on the ontology because it provides a specification of a conceptualization. It describes the concepts as well as the relationships among them, for an agent.

A consistency ontology will define the vocabulary and it is used for sharing in a coherent and consistent manner. In an ontology, we find some main components, as follows.

The definition of the ontology is like a formal specification of a program. The structure of the ontology is presented for objects, concepts and other entities that exist in some area and defines the relationships that hold among them.

Classes in an ontological structure are understood as a sets of individuals.

Object properties are connections between pairs of individuals.

Datatype properties connect individuals with literals.

Individuals represent actual objects from the domain.

3.1 Object Property

Every Object Property (OP) in the ontology is a mapping from individuals of a Domain Set that contains one single class to individuals of a Range Set consisting of multiple classes. Following that, this object is also a relationship between two individuals that belong to these two classes.

It is formed in mathematics like this:

$$y = f(x)$$

in that,

f is Object Property
x is a class in the Domain of the OP
y is set of classes in the Range of the OP.

In this case, f the takes a role of a relationship that connects instances of class x with instances of the set of classes y.

In an RDF repository, the data are stored in triple format (subject - predicate - object). When a subject and an object are two instances that belong to a class in the Domain and Range of the OP respectively, the OP becomes the predicate in the triple which connects the subject and the object. One specific instance of one class may have a relationship with more than one instances of different classes with the same OP. To find the relationship among classes, we need to use the OP as the part that connects them. Then, we apply these paths to generate pattern matching in SPARQL to retrieve the data.

There are two specific cases for cardinality of the OP:

Cardinality includes 0.
Cardinality does not include 0 (n>1).

The OP's cardinality decides the connection between two instances of two classes that are the part of the Range and Domain of this Object Property.

If the OP has cardinality that includes 0, the OP's Domain will be one where it has unreachability with the OP's Range, so the path goes through this OP can be broken down. In other words, the instance in the Domain may not connect with any instances in the Range. Generating the triple pattern with this OP should be optional, because it has no data matching or we may receive incorrect data. Using this triple in a SPARQL statement may cause the empty data result.

For an OP cardinality is 1...n, the OP's Domain is always reachable with the OP's Range, the path goes through this OP always exists. In this case, we are always sure that any instance in the Domain connects with at least one instance in the Range. This kind of OP is always required in generating the triple when we insert or get data.

3.2 Datatype Property

The Datatype Property (DP) in the ontology creates a partition for the set of instances that belongs to one class. In other words, it is considered as the attributes of the class. Each DP value belongs to one specific primary data type (DT) such as string, literal, double, etc.

In this case, describing how the data type property divides the set of instances of the class, may look like a graph, but the form is similar to a tree. For example, the DP Gender will divide instances of the Person class into two subsets: Male and Female (As seen in Fig. 2).

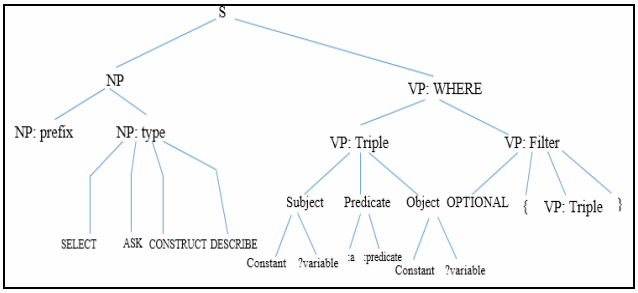

Fig. 3. The grammar tree of SPARQL.

3.3 Class Hierarchy

If one class belongs to an OP's domain (or range), all its descendants will correspond to the OP's domain (or range) due to the inherit relationship.

Let A and B be ancestors in two different hierarchy classes. If A is connected to B via OP1, then the inheritance of OP1 cannot be applied to a descendant of A when this already has another OP that connects to a descendant of B.

e.g. $B = OP_1 (A)$

$B_1 = OP_2 (A_1)$ (A_1 is a subclass of A, B_1 a subclass of B)

OP_2 can be replaced by OP_1.

We assume that all the ancestors of the *leaves* in any class hierarchy are *abstract classes*, i.e., they do not have instances. Only the classes of the last level have instances ("concrete classes"). Therefore, Data Type properties will only partition the classes in the last level of a class hierarchy.

3.4 SPARQL Statement

A SPARQL statement, a special kind of query language, needs to be constructed to query the Linked Data in the server.

In order to design the triple pattern for querying the data based on the relationships among classes and relationships between them theirs DTs which are given in request's body. The SPARQL statement has a set of triples that follows a structure like this:

```
SELECT: ?s1, … ?si, … ?sI
WHERE
{
 [ t1 . … tk . … tK ]
  FILTER(?w1 == "v1")
  FILTER(?wl == "vl")
  …
  FILTER(?wL == "vL")
}
```

In the SELECT statement, each projection `variable ?si` belongs to a set of projection variables ("S"). Moreover, it corresponds to a pair of triples in the output SPARQL query body with the following structure:

```
?<individualOfAclass>  a : Cj
?<individualOfAclass> : <DataTypeProperty j> ?si
```

where: Cj is a class (or concept) of the ontology, and `<DataTypeProperty j>` is a DP of the ontology that connects Cj with `?si`.

The WHERE statement, it consists of two parts: a set of triples that describes the relationships among the classes and these classes with theirs data type properties, while other part is the filter statement that describes the comparison with specific value to select the appropriate instances.

With `WHERE { [ t1 . … tk . … tK ] }`

every ***tk*** is a triple that represents a step in the possible shortest path that connects any distinguishable pair of classes in the SPARQL query body. It has the structure (x, p, y) where x and y are names of SPARQL variables that correspond to individuals of any class that belongs to ontological structure. This means that x and y could represent individuals of a class that belongs to set of connected classes. In addition, ***p*** is an OP of the ontology that connects x and y.

Each `tk` has a representation that corresponds to the following SPARQL triples:

```
?x a :<Class of the individual x>
?y a :<Class of the individual y>
?x  :p  ?y
```

In addition, it corresponds to a step in a shortest path from Cm to Cn, where Cm and Cn belong to a set of classes to connect. In some cases, tk could be an "`OPTIONAL`" triple due to the cardinality of the OP.

In the WHERE/Filter clause, each restriction `variable ?wl` belongs to a set of restriction variables ("W"). In additional, it corresponds to three triples in the output SPARQL query body with the following structure:

```
?<individualOfAclass> a :Cl .
?<individualOfAclass> :<DataTypeProperty1> ?wl .
 FILTER(?wl == "vl"),
```

where Cl is a class (or concept) of the ontology, and `<DataTypeProperty l>` is a DT of the ontology that connects Cl with `?wl` .

3.5 Solve with Hierarchy Class

With the hierarchy class in the ontology, the ancestors (called abstract classes) have no individuals directly. All of the individuals are in the concrete classes. To make SPARQL statements related to hierarchy classes, we should move all of the abstract classes, because those statements are generated based on the variable that represented individuals; but an abstract class does not have any individual. However, to guarantee the relationship of classes in the hierarchy classes, we need to move down all of the relationships to the concrete classes. In order to do that, we have two situations, as follows

- The relationship between hierarchy classes with a

single class. In this case, the concrete classes will inherit all relationships with that single class. Then, we move all the ancestor classes to make sure that generating a SPARQL statement just covers the classes that have individuals.

- The relationship between hierarchy classes with a hierarchy class. In this case, we allow the concrete classes in both hierarchy classes to inherit all of the relationships from their ancestors. The relationship between two concrete classes are set of relationships among their ancestors.

4. Generating SPARQL based on an Ontology Structure

Retrieving information must be based on all concepts (classes, OP, DP and cardinality) in the ontology, because this process has to be done by the SPARQL statement, which was built dynamically on the ontology. In order to do that, we need to follow all relationships among these given classes, and then find the instances of target classes that can connect with the given classes well. Then, we may have the path from any pair of classes (one in the given class, the other in the target classes. We can use given values in the user request to filter these instances. Generating the triple pattern follows the relationships among classes and DTs of each class. There are two kinds of triples that are generated from the request's parameters. One kind of triples aims to define the relationship among all of the instances via OPs which belong the classes in the ontological structure.

To indicate a relationship among classes, we use a shortest-path algorithm to find the path between the origin class and the target class. The others are used to define the relationships between these instances with their DTs.

They have some functions that were built in this translator, such as: optimal search, retrieve instance or insert new instance for one specific class. Remarkably, the search is a complex operator of the translator. It was designed as seen below:

```
T := Ø;
P ←findDistinguishPairsOfClass(); //one is
origin class and the other is target class
For (pair p : P)
fromClass ← p.getX();
toClass ← etY();
path ←findShortestPaths(fromClass, toClass);
T ← T U generateTriplesFromPath(path);
T' := Ø;
For (triple t : T)
Boolean op ← isTripleOptional(t);
If (op) Then
  T' ← T' U { OPTIONAL(t) }
Else
  T' ← T' U { t }
```

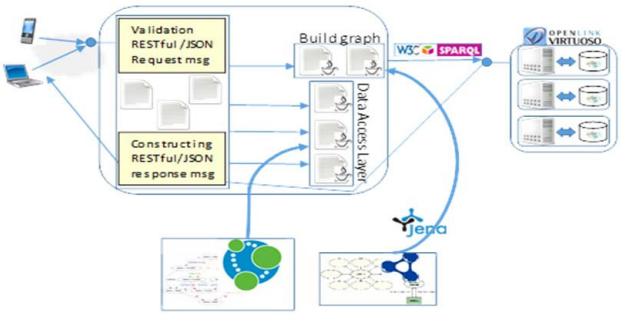

Fig. 4. Application model.

5. Examples

To read the structure of the ontology, we use the Jena API, an open source Semantic Web framework for Java. This API extracts data and writes to RDF graphs. In addition, we also use Neo4j, a kind of graph database for handling the graph of the ontology (as seen in Fig. 4).

The semantics of the Input structure

This is an HTTP request message. There are two kinds of methods: GET and POST.

- GET is used query the data of the class(es) in database.
- POST is used to insert new instances for a specific class in the database.

The URI of this kind of request is an HTTP schemed URI with the following components:

http://<entry-point >/<operation>? <Query String>

The semantics of the Output structure

The output is the result of one or more processes on the server side after the server handles the request from the client. Based on the request, the server will use the appropriate functions to generate the answer and respond to the client. Functions in the server are applied algorithms, as well as interactions with other servers or the cloud to find the best answer for the client.

The output is constructed in JSON format. Then, the client parses this response for the representation data.

The response process is also based on RESTful technology, because the result can be stored in a cache so it can be reused for a subsequence, similar request. This is helpful in reducing congestion in the network.

In a graph store (e.g. Virtuoso server), there is support for two categories: graph update and graph management.

- Graph update is used to add or remove triples of one graph in the graph store. This kind of operation just changes data of the existing graph with some statements, such as insert, delete, insert data, delete data, modify, load and clear. Delete and insert operations are specific cases of a modify operation that consists of a group of triples to be deleted and a group of triples to be added. These triples are constructed via query pattern. However, there is a difference between the "insert data/delete data" and "insert/delete" in that insert data and delete data do not take a template and pattern. The load operation uses to

reads the contents of an ontology representing a graph into a graph in the graph store whereas clear operation removes all the triples of a graph.

Graph management is used to create or delete a graph in the graph store. There are two kinds of statement: create and drop. Creating graph will create a new graph with a name specified by the URI whereas the drop operation removes the specified named graph from the graph store.

- The structure of a class hierarchy a complex step that needs to be handled. The problem is how to make a recursive process to "move down" all relationships from ancestors to descendants when that hierarchy class belongs to Domain or Range. There have three cases, as follows.

The Domain and the Range of the relationship are two classes that are ancestors belonging to two different hierarchy classes. In this case, we look down one level from these classes. All of the "next level" classes in the Domain will connect with all of the "next level" classes in the Range with the same Object Properties as their ancestors. From that, these "next level" classes have a set of Object Properties that is the sum of inherited the Object Properties and object properties of itself. This makes the process a loop until the Domain and the Range of the relationship is between all classes of the last level of these initial classes.

The Domain of the relationship is an ancestor class of a class hierarchy and the Range is a single class. In this case, we allow all of the "next level" classes to inherit all of Object Properties from the Domain class while the single class in the Range is still stable. Doing this step is a recursive method until the relationship is between classes of last level in hierarchy class and single class.

The Domain of the relationship is a single class, while the Range is one of the ancestors in a hierarchy class. In this case, we allow the single class in the Domain have relationship with all "next level" classes of current class in the Range. Repeat this step until the relationship connects between single classes with concrete classes in the class hierarchy.

For any Object Property that is related to a hierarchy class, we preprocess it to make sure that the Object Property just connects two concreate classes which that have instances (or data) directly.

5.1 Scenario 1: Search with Multiple Classes without Hierarchy Class

With the ontological structure of the the Brain Computer Interface (BCI) ontology, we have a Subject class, which has a set of instances; each instance is a person who takes part in collecting the EegBciRecord. So, the Subject class has a relationship "has Data Set" with the EegBciRecord class. Similarly, each instance of EegBciDevice is a certain device that is used to make an EegBciRecord. Each instance of the EegBciRecord has a specific channel that describes the structure of the record. EegBciRecord has the relationship "hasEegChannel" with EegChannel. Assume that the client will send a request to ask about the identification (ID) of all the instances of EegChannel. These records were of young males whose year of birth is 1989 and they were collected by a certain

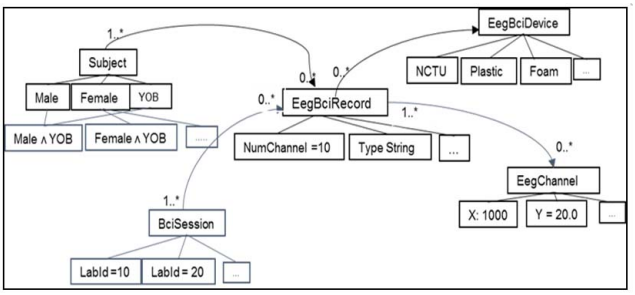

Fig. 5. A part of data structure of BCI ontology.

device which has organization name NCTU. The position of the device a distance from the center towards the right by about 3.5 millimeters (see Fig. 5).

The SPARQL statement is generated automatically as follows:

```
SELECT ?EegChannel_id
WHERE
 {
      ?Subject_id a bci:Subject .
      ?Subject_id bci:hasYearOfBirth ?Subject_hasYearOfBirth .
      ?Subject_id bci:hasGender ?Subject_hasGender .
      ?EegBciRecord_id a bci:EegBciRecord .
      ?Subject_id bci:hasDataSet ? EegBciRecord_id.
      ?EegChannel_id a bci: EegChannel.
      ?EegBciRecord_id bci:hasEegChannel ?EegChannel.
      ?EegBciDevice_id a: EegBciDevice
      ?EegBciDevice_id
bci:hasOrganizationName ?EegBciDevice_hasOrganizationName .
      ?EegBciDevice_id
bci:isUsedForGenerateEegBciRecord ?EegBciRecord_id.
      ?EegBciRecord_id a bci:EegBciRecord .
      ?EegBciRecord_id
bci:hasEegChannelData ?EegChannel_id .
      ?EegChannel_id a: ?EegChannel.
      FILTER (?Subject_hasGender= "Male")
          FILTER (?Subject_hasYearOfBirth= "1989")
          FILTER (?EegBciDevice_id hasOrganizationName= "NCTU") .
  }
```

5.2 Scenario 2: Search with Multiple Classes with Hierarchy Class

With the above example in Scenario 1, there have many types of record such as EegBciRecord, Eye GazeBciRecord, and MouseClickBciRecord. All of them belong to BciRecord. So, we have an abstract class of BciRecord and three concrete classes (EegBciRecord, EyeGazeBciRecord, and MouseClickBciRecord). Similarly, for each type of record, we also have a specific type of device that is used to collect a specific record that corresponds. There are three types of device such as EegBciDevice, EyeGazeBciDevice and MouseClickBci Device. All of the devices belong to BciDevice, a abstract class, that does not have instances directly (see Fig. 6)

In this case, we allow all the children of BciRecord to inherit all relationships with Subject. It means that Subject

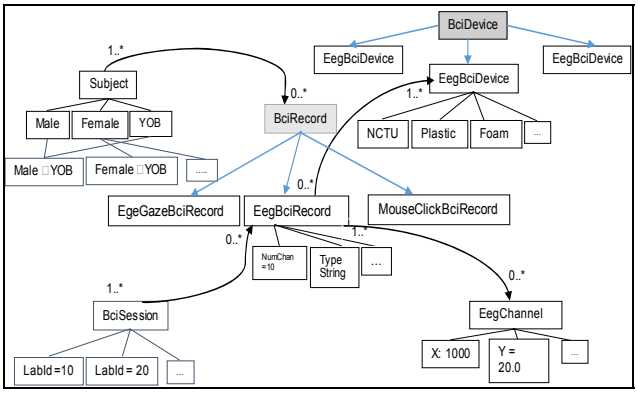

Fig. 6. A part of the data structure of BCI ontology with a hierarchy class.

will have a relationship with EyeGazeBciRecord, EegBciRecord and MouseClickBciRecord with same relationship as BciRecord.

The SPARQL statement is generated similar to the above SPARQL and this statement does not contain any abstract classes such as BciRecord or BciDevice.

5.3 Scenario 3: Insert New Instances for One Class

Following the ontology structure, we organize and store the data in the server (VUS) in the triple format (subject – predicate - object). In that data, the subject is the instance of one specific class in the ontology. The object can be instances of other classes that have the relationship with the subject or a DT (attribute) of this subject. Predicate is a relationship between subjects and objects corresponding with the OP between those two classes. To insert a new instance of the class in the ontology, we need to insert all of the triples for the relationship of this instance with the other instances in another class or the data type properties of itself.

To design the triple pattern for inserting a new instance, we need to follow the metadata structure. The relationship between one instance in one specific class and other instances in another class is expressed by the OP. Besides that, there has a set of triples that describes pattern matching for this instance with its attributes. This kind of pattern is based on the DP of the class that the new instance must belong to. However, some DTs are optional, and we do not need to insert all of them, or require clients to send parameters.

5.4 Scenario 4: Retrieve Instances of One Class

In the query operation, we find all the instances of one specific class that satisfy the given values of the DTs. However, any class in the ontology has a lot of DTs and some of them are required. We cannot make each query pattern for every specific class to retrieve the instances, especially when we have a new concept. It is difficult to define a new query pattern for new concepts because we need to be concerned with the structure.

Like the insert operation, we need to base it on the DTs of one specific class to define the set of triples that can be used to query all instances that belong to this class. From the required values and provided values, we try to incorporate them into the pattern matching to query the database as well as filter out suitable instances that satisfy the request.

6. Conclusion

In this paper, we presented translation that solved the problem of mismatches between two languages (RESTful/JSON and SPARQL) that are supported by two powerful technologies (web services and Linked Data, respectively). With this translation, clients can work with Linked Data easily. In addition, there are still no barriers to limit communication. This translator component also applies a graph database with an algorithm to solve the problem of the shortest path between any two nodes that correspond to any two classes in the ontology. Using the graph to handle the metadata structure is the best way to find the exact triple patterns that keep the classes connected as well as the data of these classes. Based on that, we can find the data effectively. Besides that, the translator component offers four main functions: 1) query specific class instances, 2) get values of data type properties, 3) insert new class instances and 4)search (multiple classes). Finally, the translator component can work with any ontological structure and query/search data in different resources with a federated query scheme.

References

[1] Christian Bizer, Tom Heath and Tim Berners-Lee, "Linked Data - The Story So Far," *International Journal on Semantic Web and Information Systems,* 2009. Article (CrossRef Link)

[2] Bastian Quilitz, Ulf Leser, "Querying distributed RDF data sources with SPARQL," in *ESWC'08 Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, 2008. Article (CrossRef Link)

[3] Jorge P´erez, Marcelo Arenas, and Claudio Gutierrez, "Semantics of SPARQL" in *Semantic Web Information*. Article (CrossRef Link)

[4] Roy T. Fielding, Richard N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology,* 2002. Article (CrossRef Link)

[5] John K. Zao, Tchin-Tze Gan, Chun-Kai You, Cheng-En Chung, Yu-Te Wang, Sergio José Rodríguez Méndez, Tim Mullen, "Pervasive brain monitoring and data sharing based on multi-tier distributed computing and linked data technology," *Frontiers in Human Neuroscience,* 2014. Article (CrossRef Link)

[6] Ying Wang, Weiru Liu, and David Bell ,"*A Concept Hierarchy based Ontology Mapping Approach*" School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast,

Belfast, BT7 1NN, UK, 2013. Article (CrossRef Link)
[7] Vinu, Sherimon and Reshmy Krishnan, "ontology construction and reasoning using owl: a case study from seafood domain", SENRA Academic Publishers, British Columbia Vol. 8, No. 2, pp. 2979-2984, June 2014 Online ISSN: 1920-3853; Print ISSN: 1715-9997. Article (CrossRef Link)
[8] Carlos R. Rivero, Inma Hernández, David Ruiz, and Rafael. University of Sevilla, Spain, "Generating SPARQL Executable Mappings to Integrate Ontologies," in *Proceeding ER'11 Proceedings of the 30th international conference on Conceptual modeling*, 2011. Article (CrossRef Link)
[9] Tuan-Dat Trinh, Ba-Lam Do, Peter Wetz, Amin Anjomshoaa, Elmar Kiesling and Amin Tjoa, "A Drag-and-block Approach for Linked Open Data Exploration" in *ISWC 2014*, 2014. Article (CrossRef Link)
[10] Can Li , Xinyan Zhu , Wei Guo , Yi Liu , Liang Huang, "*Research on Extension of SPARQL Ontology Query Language Considering the Computation of Indoor Spatial Relations*", The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XL-4/W5, 2015 Indoor-Outdoor Seamless Modelling, Mapping and Navigation, 21–22 May 2015, Tokyo, Japan. Article (CrossRef Link)
[11] Ayoub Oudani, Mohamed Bahaj, Ilias Cherti. "*Creating an RDF Graph from a Relational Database Using SPARQL*", Manuscript submitted May 11, 2014; accepted October 20, doi: 10.17706/jsw.10.4.384-391. Article (CrossRef Link)
[12] Ilaria Tiddi, Mathieu D'Aquin and Enrico Motta, "Walking Linked Data: A graph traversal approach to explain clusters" in *ISWC 2014*, 2014. Article (CrossRef Link)
[13] Peter Bloem, Adianto Wibisono, Gerben De Vries, "Simplifying RDF Data for Graph-Based Machine Learning" in *11th ESWC 2014*, 2014. Article (CrossRef Link)
[14] Heiko Müller, Liliana Cabral, Ahsan Morshed, and Yanfeng Shu "From RESTful to SPARQL: A Case Study on Generating Semantic Sensor Data", The 12th International Semantic Web Conference (ISWC2013). Article (CrossRef Link)
[15] Vries, G.K.D., de Rooij, S.d'Amato, C., Berka, P., Sv´atek, V., Wecel, K., eds., "A fast and simple graph kernel for RDF" in *EUR Workshop Proceedings., CEUR-WS.org (2013)*, 2013. Article (CrossRef Link)
[16] Haizhou Fu and Kemafor Anyanwu, "Effectively Interpreting Keyword Queries on RDF Databases with a Rear View" in *ISWC 2011*, 2011. Article (CrossRef Link)
[17] Carlos Viegas Damasio and Filipe Ferreira, "Practical RDF Schema reasoning with annotated Semantic Web data," in *The ISWC 2011*, 2011. Article (CrossRef Link)
[18] Gregory Todd Williams and Jesse Weaver, "Enabling fine-grained HTTP caching of SPARQL query results", in *The ISWC 2011*, 2011. Article (CrossRef Link)
[19] Roi Blanco, Peter Mika and Sebastiano Vigna, "Effective and Efficient Entity Search in RDF data", in *The ISWC 2011*, 2011. Article (CrossRef Link)
[20] Mohamed Morsey, Jens Lehmann, Sören Auer and Axel-Cyrille Ngonga Ngomo, "DBpedia SPARQL Benchmark Performance Assessment with Real Queries on Real Data" in *The ISWC 2011*, 2011. Article (CrossRef Link)

Khiem Minh Nguyen has been a lecturer at Can Tho University since November, 2011. He obtained his B.S degree in Information System from Can Tho University (CTU), Vietnam in 2011 and his M.S degree in Computer Science from National Chiao Tung University (NCTU), Taiwan in 2015. At the present, he researches as a PhD student in Bournemouth University (BU), United Kingdom. He is interested in Database Management Systems, Linked Data, Social Networks and 3D Animation. Besides that, he also participates in some fields such as Cloud Computing, Artificial intelligence and Mobile Application.

Hai Thanh Nguyen has been a lecturer at Can Tho University since April, 2009. He received his B.S degree in Informatics from Can Tho University (CTU), Vietnam and his M.S in Computer Science and Engineering from National Chiao Tung University (NCTU), Taiwan in 2009 and 2014, respectively. At the present, he is a PhD student in Computer Science of Pierre and Marie Curie University (UPMC), Paris, France. He has been very excited by Deep Learning, Bioinformatics and Social Networks. Another field which he is also very interested is Simulation-based Data Mining. Simulation-based Data Mining Solutions are applying to agriculture in Vietnam. Besides Data Mining, he also works on Cloud Computing, Mobile Data Management and Artificial intelligence.

Hiep Xuan Huynh is an associate professor in computer science (informatics) at College of Information and Communication Technology, Cantho University, Vietnam. He obtained his Ph.D degrees in informatics from Polytechnics School of Nantes University in 2006. His research interests are IoT, interestingness measures in data mining, deep learning, cellular automata, modeling decisions and recommender system. Contact him at hxhiep@ctu.edu.vn