Graphical Query Language as SPARQL Frontend

Guntis Barzdins[1], Sergejs Rikacovs[1], Martins Zviedris[1]

[1] Institute of Mathematics and Computer Science, University of Latvia, Raina bulv. 29, Riga LV-1459, Latvia
Guntis@latnet.lv, Sergejs.Rikacovs@lumii.lv, Martins.Zviedris@lumii.lv

Abstract. It is well known that end-users have problems to write even simple SQL queries. The new SPARQL query language for RDF databases is a step in the right direction, but is still not suitable for end-users. This lead us to creating a more convenient approach in which end-users could retrieve structured data from the database through a graphical query language named GQL. GQL graphical query language is based on OWL ontology language and SPARQL query language for RDF data. GQL visualization format is based on UML graphical language. To achieve interoperability between all these techniques, a true subset approach did not work – minor modifications were required to achieve a functional solution. The proposed approach is applicable also to querying data from the legacy relational databases through database export to OWL/RDF format.

Keywords: Ontology, Database, Graphical query language

1 Introduction

In this paper we propose a graphical query language frontend for SPARQL that end-users can use to create simple or even rather complex queries. The perceived end-user simplicity of the proposed approach is based on tight integration of selected features of OWL, SPARQL and UML.

As we know OWL [1] is the de-facto standard for web ontology specification. Nevertheless there are certain pragmatic situations, like performance sensitive billion triple RDF stores, where ontologies are needed to define the database schema, but application of standard OWL semantics is not practical [2]. The reason for the need to departure from the standard OWL semantics is that the role of the ontology in the pragmatic scenarios considered in this paper is to provide data validation (constraint checking) and data explanation to user, rather than universal logic inference (enabled by the standard OWL semantics based on the Open World Assumption and rejection of the Unique Name Assumption).

Although a number of well-defined OWL subsets, such as OWL Lite [1], OWL DL, OWLSIF [3], OWLPRIME [4], etc. have been around for a while, the need for even more finely refined OWL subsets has lead towards introduction of the language profile concept in OWL 2, along with an extensive list of predefined profiles [5]. Nevertheless, none of the existing OWL profiles provides a pragmatic OWL subset , which would be compatible with the following three ubiquitously used real-world technologies:

- Provides meaningful database schema definition and data constraint validation means,
- Can be represented by graphic UML class diagrams,
- Supported by the existing high performance billion triples RDF stores,

The only OWL 2 profile which comes close to the above goal is OWL 2 RL profile [6]. For example, the OWL 2 RL profile ensures, that a reasoning engine only needs to reason with individuals that occur explicitly in the data part of the ontology. The main reason why we, nevertheless, introduce a new and mostly more limited OWL profile is that we want to cover all three above mentioned goals, including the compatibility with the existing high performance billion triples RDF stores.

Therefore first in this paper we describe a syntactic subset of OWL, which we will call **UML / OWL** subset. The semantics of this subset will be different from the standard OWL and will follow the more pragmatic UML semantics. Meanwhile, preserving of OWL syntax provides convenient compatibility with numerous OWL serialization formats as well as with the popular ontology editors such as Protégé. The selected subset satisfies the following properties:

- It contains only a basic subset of OWL syntactic constructs, which can be directly mapped to graphic UML class diagrams and cover the needs of the considered use-cases (the only advanced feature included is <<EnumeratedClass>>, which pragmatically is a widely used construct in almost any real-world database)
- The semantics of the selected OWL constructs can be defined through entailment rules which are supported by the existing high-performance RDF data stores capable of operating on billions of triples
- It is possible to define a graphical user-friendly query language (a SPARQL [7] pre-processor rooted in UML graphical notation) for this subset of OWL (see Section 3)

The approach described in this paper has been developed based on real application in the area of medical statistics with RDF triple-stores close to billion triples. For space saving purposes, in this paper we demonstrate the approach on the very basic university ontology.

2 UML / OWL Subset

The basic idea of the proposed **UML / OWL** subset is to use only those OWL DL constructs that can be adequately represented with UML class diagrams. More precisely, we use "UML profile for RDF and OWL" [8] to define constructs allowed in the specification of ontologies. Namely, the **UML / OWL** subset is defined to contain only the following stereotypes from the "UML profile for RDF and OWL" definition:

- <<owlClass>>
- <<rdfsSubPropertyOf>>
- <<rdfsSubClassOf>>

- <<owlProperty>> along with sub-stereotypes <<objectProperty>> and <<datatypeProperty>>
- <<equivalentClass>>
- <<equivalentProperty>>
- <<enumeratedClass>>.

The "UML profile for RDF and OWL" describes how these UML constructs map into OWL constructs. In this way we have introduced the new **UML / OWL** profile which defines both its graphic UML syntax and the traditional OWL serialization. Fig. 1 in graphic UML format shows an ontology belonging to the introduced **UML / OWL** profile (Fig. 1. uses also an additional stereotype <<inverseOf>> which is not part of **UML / OWL** profile and shall be considered only as a comment; the reason for not including <<inverseOf>> in the profile is its poor support in the existing RDF databases). In our approach, the end-user shall use only the graphical UML format of the ontology, while its corresponding OWL serialization should be used only for the technical purposes when interfacing with the RDF databases.

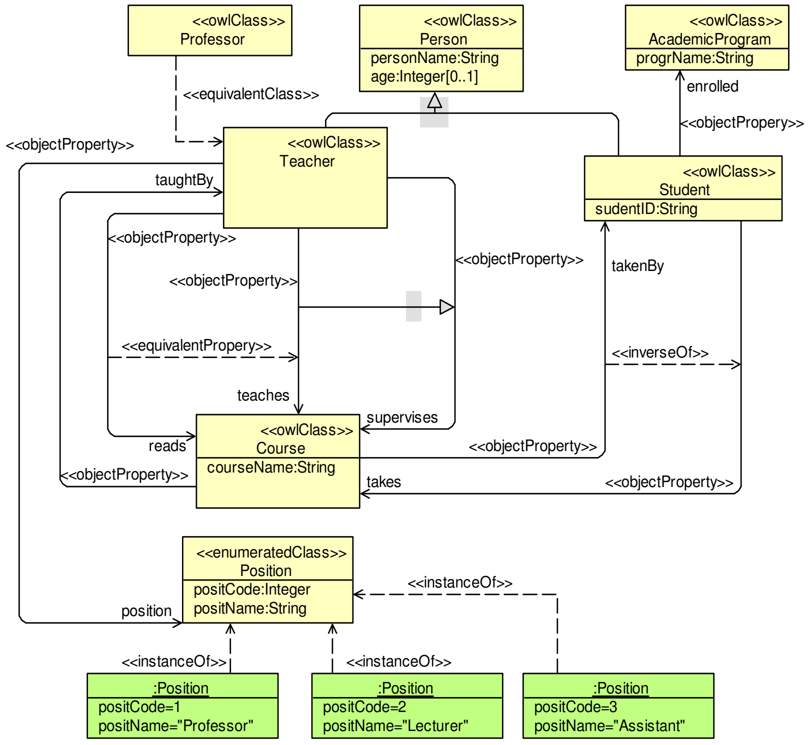


Fig. 1. Schema part of the simple university ontology (belonging to **UML / OWL** profile.)

In the context of the proposed **UML / OWL** subset we define two parts of the ontology description:

- **Ontology schema part** contains classes, properties and relations such as(rdfs:subclassOf, rdfs:subPropertyOf, owl:equivalentClass, owl:equivalentProperty). A special consideration is made regarding EnumeratedClass stereotype – the instances of the EnumeratedClass are also considered part of ontology schema. In other words, ontology schema includes everything shown in Figure 1. From the UML point of view this part contains classes, attributes, associations and corresponding relations.
- **Ontology data part** - is used to express simple statements about resources by means of data values, named properties and classes, defined in the schema part of the ontology. In UML terms this part contains instances (objects and links) of classes and associations that were defined in the schema part.

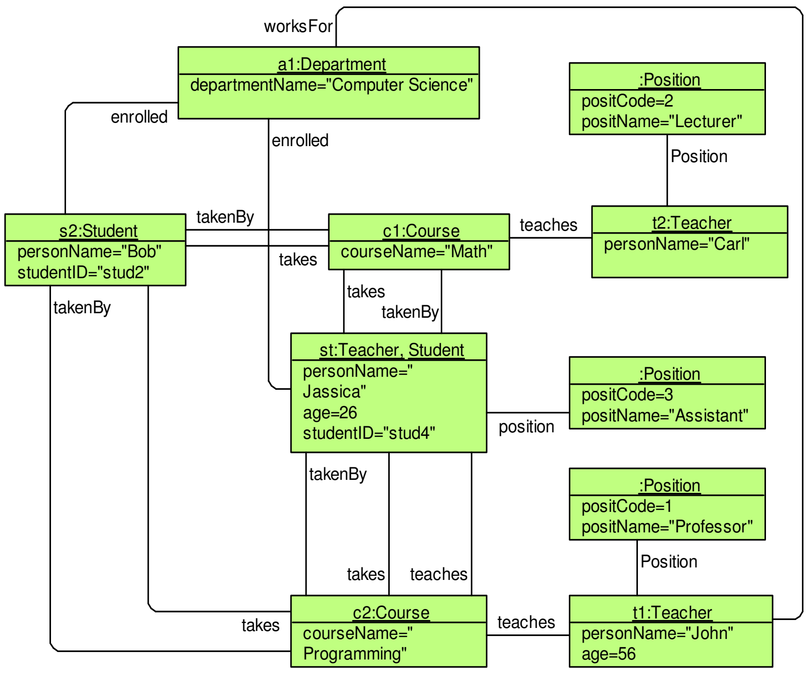


Fig. 2. Data part of the ontology schema shown in Figure 1 (belonging to **UML / OWL** profile.)

The OWL specification along with "UML profile for RDF and OWL" [8] defines several serialization formats for OWL ontologies along with mapping between these serialization formats. Fig. 3 shows a fragment of OWL/N-TRIPLE serialization of the ontology schema and data parts shown in Fig. 1 and 2. The shown OWL/N-TRIPLE serialization format is crucial for interfacing with RDF triple-sores and their standard query language SPARQL.

```
Position rdf:type owl:Class .
Position owl:equivalentClass _:ag0 .
_:ag0 rdf:type owl:Class.
_:ag0 owl:oneOf _:ali3 .
_:ali3 rdf:first p2 .
_:ali3 rdf:rest  _:ali2 .
_:ali2 rdf:first p3 .
_:ali2 rdf:rest  _:ali1 .
_:ali1 rdf:first p1 .
_:ali1 rdf:rest  rdf:nil .
...
p1 rdf:type Position.
p2 rdf:type Position.
p3 rdf:type Position.
...
Student rdf:type owl:Class.
Person rdf:type owl:Class.
Student rdfs:subClassOf Person .
...
studentID rdf:type owl:DatatypeProperty.
studentID rdfs:domain Student.
studentID rdfs:range XMLSchema#integer.
...
s1 rdf:type Student .
s1 studentID "1"^^XMLSchema#integer.
```

Fig. 3. A fragment of OWL/N-TRIPLE serialization of the ontology schema and data parts shown in Fig. 1. and Fig. 2.

The mapping between the graphic UML format and OWL/N-TRIPLE serialization is generally rather straight-forward. The only exception is the <<EnumeratedClass>> stereotype, which requires special attention in our approach. The regular OWL/N-TRIPLE serialization of this construct uses a special *oneOf* construct over a set of permitted values. Meanwhile to stick with UML interpretation of this construct, we also need to create the actual individuals for the enumerated class and link them through the regular *instanceOf* construct (which corresponds to *rdf:type* relation in RDF). For the purposes of this paper we will assume that such individuals are created for all EnumeratedClasses by some external procedure.

Due to rather limited set of features included in the **UML / OWL** profile, it is possible to define its semantics through a rather short list of RDFS-like entailment rules seen in the table below.

Table 1. RDFS-like entailment rules for **UML / OWL** profile.

1	uuu `rdfs:subPropertyOf` vvv . vvv `rdfs:subPropertyOf` xxx . `obj1 uuu obj2`	Obj1 `xxx` obj2 .
2	aaa `rdfs:subPropertyOf` bbb . uuu aaa yyy .	uuu bbb yyy .
3	uuu `rdfs:subClassOf` xxx . vvv `rdf:type` uuu .	vvv `rdf:type` xxx .

4	uuu rdfs:subClassOf vvv . vvv rdfs:subClassOf xxx . obj rdf:type uuu.	obj rdf:type xxx .
5	p1 owl:equivalentProperty p2 x p1 y	x p2 y
6	p1, owl:equivalentProperty p2 x p2 y	x p1 y
7	c1 owl:equivalentClass c2 x rdf:type c1	x rdf:type c2
8	c1 owl:equivalentClass c2 x rdf:type c2	x rdf:type c1

We have intentionally kept the list of necessary entailment rules for **UML / OWL** profile very short. This enables to implement **UML / OWL** profile much more efficiently than it would have been possible with more complete rule set approaching RDFS or OWL-Lite. One of the RDF data stores providing a good support for the introduced **UML / OWL** subset is OpenLink Virtuoso [9]. It allows storing ontologies along with corresponding data and provides querying facilities through the highly efficient subset of standard SPARQL (this SPARQL subset is adequate for the graphic query language to be introduced in the Section 3). In particular this data store provides a support for all of the abovementioned entailment rules.

Above described **UML / OWL** profile gives us very nice "glasses" providing graphic UML visualization of ontology (both *ontology schema part* and *ontology data part*). This kind of user-friendly visualization will be applied also in the next Section to define a user-friendly graphic ontology query language, which provides nice "glasses" also for generating SPARQL queries.

3 Graphical Query Language (GQL)

In this Section is described an original graphical query language (GQL) for querying ontology data part, defined in the previous Section. The GQL can be considered as a pre-processor for SPARQL because „behind the scenes" it produces the regular SPARQL queries. The beauty of GQL is that it is compatible with the graphic UML visualization of UML/OWL subset ontologies, thus providing a completely graphic end-user interface for both ontology schema part exploration, as well as for ontology data part querying. This is exactly the kind of end-user experience we want and with the described techniques are able to provide to our real-life medical end-users, for whom native SPARQL or native OWL serialization formats would be completely unacceptable. The situation here is somewhat similar to SQL queries in relational databases – end-users generally do not write such queries themselves, but rather relies on the database programmer created user-interfaces; the difference and novelty in our approach is that GQL substitutes the database programmer – GQL provides an automatic translation from the end-user graphic queries into SPARQL.

GQL is based on the UML / OWL subset introduced in the previous Section, along with its graphical ontology visualization illustrated in Fig.1 and Fig.2. The initial ideas and motivation for development of GQL can be traced to [10]. The main difference of our approach is that we propose more flexible and more powerful graphical forms for query formulation. It shall also be noted that many more graphical query languages [11], [12] have been proposed for direct querying of graph patterns in RDF databases, but these are not relevant to our approach, as we rely on high-level ontology schema visualization rather than on low-level data pattern visualization. A similar idea is also described in [13]. However, it is less expressive and queries are translated from SPARQL to SQL.

To describe GQL we will use as an example the simple university ontology schema and corresponding data part shown in Fig. 1 and 2. The graphic query follows analogy with queries in natural language in the sense that first we have to select the central concept (class) we will be querying about. In case of university ontology, a typical question (query) could be following: find all Teachers, which have position="Professor" and which have age>"45" and which work for department="Computer Science". The same query converted into the GQL graphical format is shown in Fig. 4.

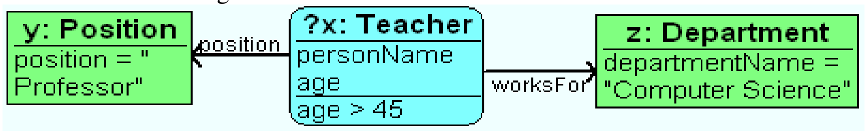


Fig. 4. GQL representation of query: find all Teachers, which have position="Professor" and which have age>"45" and which work for department="Computer Science"

In GQL the central concept (class) is depicted by the rounded rectangle with question mark preceding the variable in front of class name. Other concepts utilized in the query (rectangle) are regarded as „context concepts" and should be interpreted as existential conditions – in our example there „exist y,z such that x.position.y and x.worksFor.z". Finally, there can be added also filtering constraints such as Teacher age>"45", position="Professor", departmentName="Computer Science". Additionally, the graphic query lists the attributes of the central concept, which shall be included in the query answer – these attributes are listed inside the rounded rectangle of the central concept just below the class name. Table 2 shows the answer for the graphic query in Fig. 4. The graphic query in Fig. 4 actually translates into the SPARQL query shown below:

```
PREFIX uni:<http://www.owlontologies.com/University.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?personName ?age WHERE
  {?x rdf:type uni:Teacher.
  ?x uni:position ?y.
  ?y rdf:type uni:Position.
  ?x uni:worksFor ?z.
  ?z rdf:type uni:Department.
  ?x uni:age ?age.
  ?y uni:position ?position.
  ?z uni:departmentName ?departmentName.
  OPTIONAL {?x uni:personName ?personName.  }
  FILTER ( ?age > 45 && ?position = "Professor" && ?departmentName =
"Computer Science")}
```

Table 2. Answer for graphic query shown in Fig. 4.

personName	age
John	56

Now let us consider a more complex query. We want to extend the previous query so that it outputs also the names of Courses the Teacher teaches and we are interested only in those Courses, which are taken by some Student. In this query answer should include attributes from two classes, therefore we have to introduce a "context frame" (bold border) containing the second central concept Course, subordinated to the main central concept Teacher. The corresponding graphic query is shown in Fig. 5, Table3 contains the answer to this query and below is shown the corresponding SPARQL query:

```
PREFIX uni:<http://www.owlontologies.com/University.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?personName ?age ?courseName WHERE
   {?x rdf:type uni:Teacher.
   ?x uni:position ?y.
   ?y rdf:type uni:Position.
   ?x uni:worksFor ?z.
   ?z rdf:type uni:Department.
   ?x uni:age ?age.
   ?y uni:position ?position.
   ?z uni:departmentName ?departmentName.
   OPTIONAL { ?x uni:teaches ?a.
   ?a rdf:type uni:Course.
   ?a uni:takenBy ?b.
   ?a rdf:type uni:Student.
   }
   OPTIONAL {?x uni:personName ?personName.  }
   OPTIONAL {?a uni:courseName ?courseName.  }
   FILTER  ( ?age  >  45  &&  ?position  =  "Professor"  &&  ?departmentName  =
"Computer Science")}
```

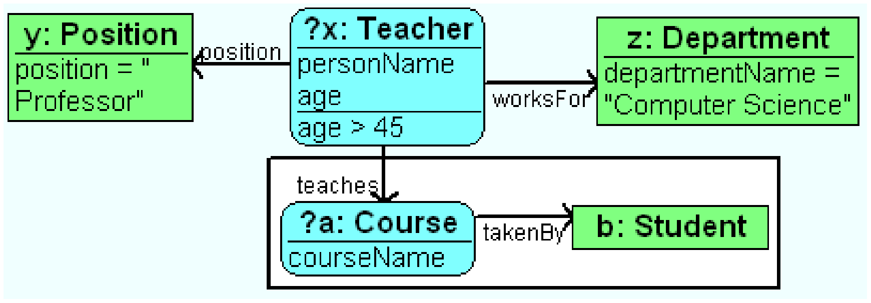


Fig 5. Graphic query with optional subordinate "context frame"

Table 3. Answer for graphic query shown in Fig. 5.

Teacher.personName	Teacher.age	Course.courseName
John	56	Programming

As we see from answer Table 3, it contains also the names of Teachers who do not teach any Courses. This is because the semantics of the bold context frame is that its content is optional. If we want to get in the answer table only those Teachers which do teach some Course, then we must replace the bold context frame with a double line context frame meaning that its content is mandatory. The corresponding query is shown in Fig. 6. The corresponding SPARQL query will be similar to the previous one, except that there will be no OPTIONAL keyword for the part corresponding to the context frame.

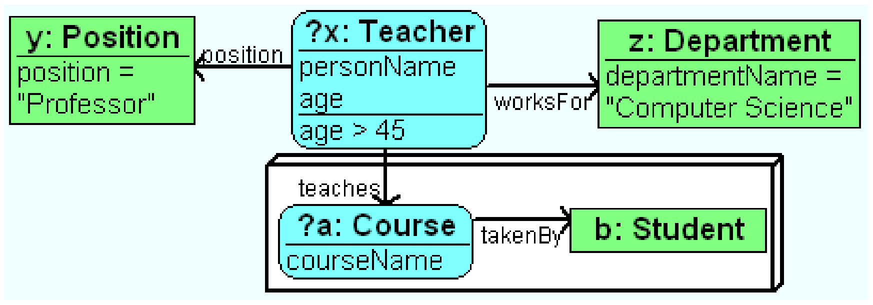


Fig. 6. Graphic query with mandatory subordinate "context frame"

Another important query case is when we want to find the Teachers which do not teach any Course. This is provided by the third "banned" variation of the context frame shown in Fig. 7. The corresponding SPARQL query will be same as the query in Fig. 5, except that it will contain additional statement `FILTER(!(bound(?a))`.

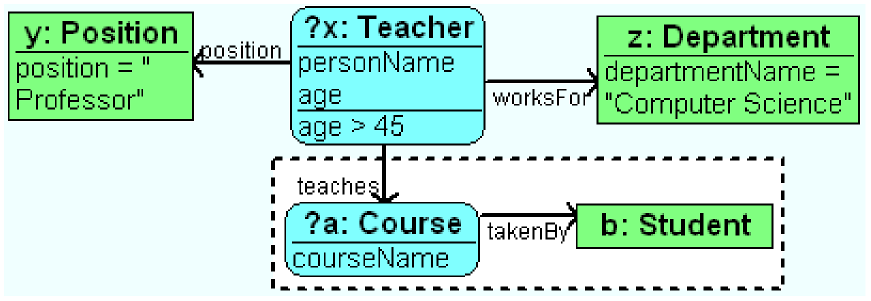


Fig. 7. Graphic query with banned subordinate "context frame"

As a "syntactical sugar", it is allowed to omit the context frame for subordinate central concepts as shown in Fig. 8, which means that all concepts in the rounded rectangles are mandatory. Therefore query in Fig. 8 is semantically equivalent to the query in Fig. 6. This format is more convenient when query involves many central concepts.

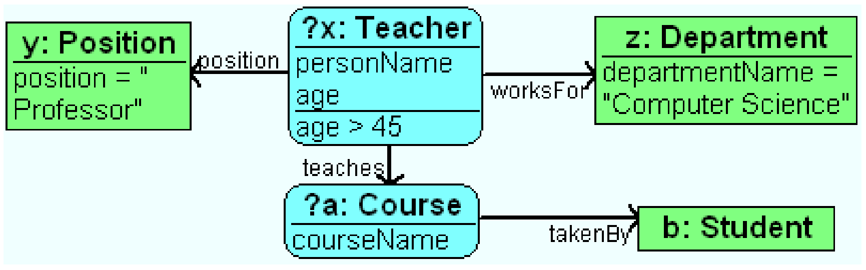


Fig. 8. Graphic query with mandatory subordinate "context frame" omitted

The last feature of GQL is the concept intersection. This feature is necessary if we want to find all Students which are also Teachers for some Course. The graphic format for such query is shown in Fig. 9 and the corresponding SPARQL query is:

```
PREFIX uni:<http://www.owlontologies.com/University.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?personName WHERE
   {?x rdf:type uni:Student.
   ?x rdf:type uni:Teacher.
   ?x uni:teaches ?y.
   ?y rdf:type uni:Course.
   OPTIONAL {?x uni:personName ?personName.  }}
```

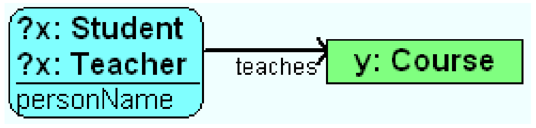


Fig. 9. Graphic query with concept intersection

Fig. 10 illustrates a more complex GQL query: find all Students younger than 27 years which are enrolled in the Computer Science department and take some Course thought by the Professor working for Computer Science department. Answer to this question is presented in Table 4.

Table 4. Answer for graphic query shown in Fig. 10.

Student.studentID	Student.personName	Student.age
stud4	Jessica	26

Finally, Fig. 11 illustrates one more complex GQL query: find all Students X (student ID, person name, age) and all Teachers Y (person name) such that:

- X enrolled in Computer Science Department and X younger than 27 years
- Y works for Computer Science Department and has position Professor
- Exist Course B such that X takes B and B is taught by Y

Answer to this question is presented in Table 5.

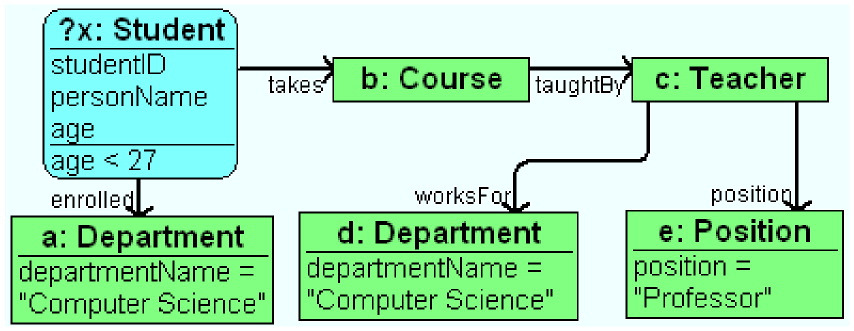


Fig. 10. An example of a more complex GQL query

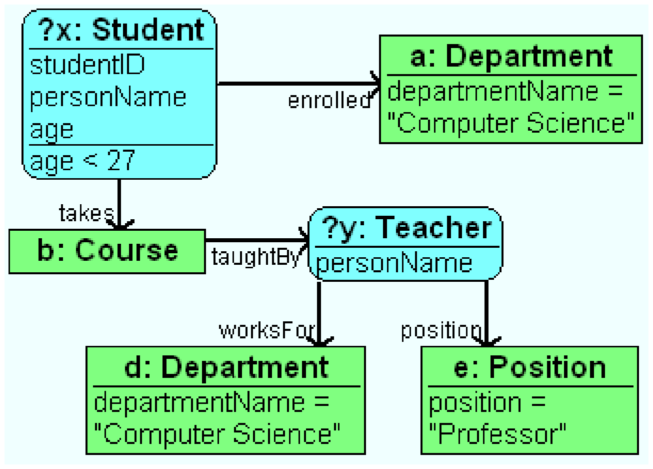


Fig. 11. An example of a more complex GQL query

Table 5. Answer for graphic query shown in Fig. 11.

Student.studentID	Student.personName	Student.age	Teacher.personName
stud4	Jessica	26	John

The above examples demonstrated the core features of GQL, which are sufficient for most end-user needs. Our intention is to eventually extend GQL with more graphical features in order to cover all SPARQL functionality relevant to the considered UML/OWL environment.

4 GQL Tool Support

We have also developed a tool which implements the described graphic query language GQL. Fig. 12 shows the functional structure of this tool. When started, the tool connects to the specified RDF database and snoops from it the schema part of the stored ontology through the set of SPARQL queries. At this point the tool can display the ontology schema part in the graphical format as shown in Fig. 1.

Fig. 12 illustrates the main steps supported by the tool: graphic query construction, translation of graphic queries into SPARQL, submitting queries to RDF database and retrieval of answers, formatting of the answers into the user-friendly format. Out of these steps, the end-user support for graphic query construction is the most elaborated and essential for the user part of the system and we will explain this step in more detail.

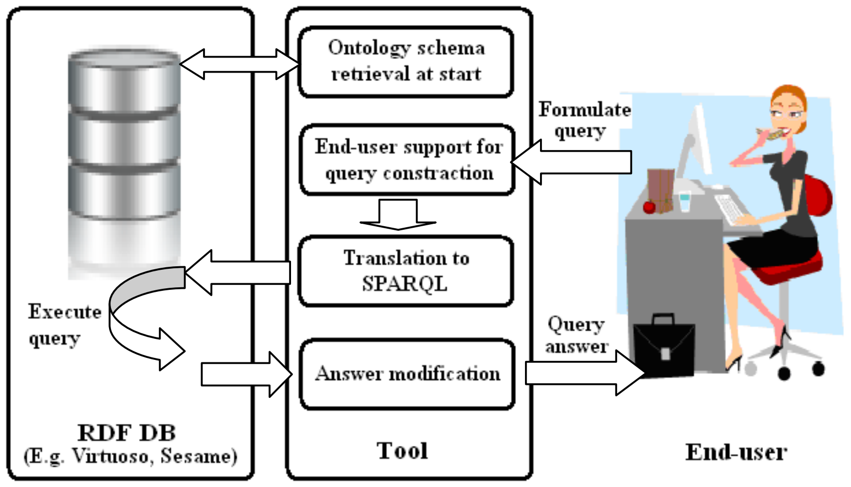


Fig 12. Structure of the tool implementing GQL

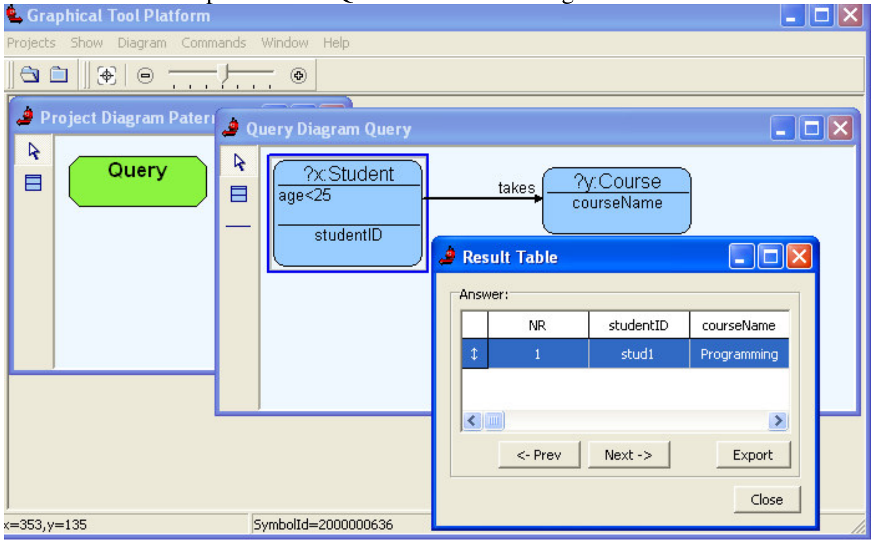
As the first step, user is offered to select the “central concept” from the list of all classes in the ontology schema – once selected, it is automatically depicted graphically. Then user can add some attribute conditions to the central concept – only attributes relevant to the central concept are shown and once selected, are automatically depicted graphically.

A special service is developed for adding subordinate central concepts (the ones belonging to subordinate context frames). Here user again can select any class from the ontology schema and the tool automatically calculates several shortest paths of ontology properties and classes, through which the new concept could be linked to the current central concept – the user only has to manually select one of the presented available paths and the intermediate properties and classes will be automatically added to the graphic query.

The tool also has special support for EnumeratedClasses. For these classes during query construction tool allows to set as condition only instances defined for this class in the ontology schema. For example, for concept Position in the ontology from Fig. 1, the tool would prompt end-user to select only one of the values:

1- Professor
2- Lecturer
3- Assistant

The GQL tool itself is implemented through the graphic editor toolset [14], [15] providing high-level graphic diagram presentation and editing primitives. A screenshot of the implemented GQL tool is shown in Fig. 13.



	NR	studentID	courseName
↕	1	stud1	Programming

Fig. 13. A screenshot of the implemented GQL tool.

5 Conclusion

The described methodology has been successfully implemented and tested in the practical application in the medical statistics domain with medical researchers as end-users. We have successfully transferred data from several relational databases into RDF database according to ontology format described in Section 2 and using transfer methodology described in [16].

We have also tested several RDF databases. Our initial implementation was based on Sesame (version 2.0-1) [17] RDF database, but it turned out to be too slow even for sub-million triples RDF database – most likely due to poor query execution planning for our automatically generated SPARQL queries. Meanwhile, by switching to OpenLink Virtuoso RDF database (version 05.08.3034) query execution performance improved radically and is now generally comparable to that of similar relational databases.

In the future we look forward for high performance RDF databases to support efficiently more entailment rules. For example, due to lack of support for *owl:inverseOf* entailment in OpenLink Virtuoso RDF database [18], currently we have to store separate triples for both directions of association. Other useful entailment would be support for symmetric properties. Once available, such features could be added to the described **UML/OWL** subset resulting in more straight-forward query construction and reduced storage data volumes.

References

1. Web Ontology Language (OWL). W3C, 2004. URL: http://www.w3.org/2004/OWL/
2. Sirin, E., Smith, M., Wallace, E. Opening, Closing Worlds – On Integrity Constraints. ISWC 2008, Workshop: OWL: Experiences and Directions
3. H. J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. J. Web Sem., 3(2–3):79–115, 2005.
4. Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle. In *24th International Conference on Data Engineering*. IEEE, 2008.
5. Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U. OWL 2: The next step for OWL. J. of Web Semantics, 6(4):309-322, November 2008.
6. OWL 2 Web Ontology Language : Profiles, http://www.w3.org/TR/2008/WD-owl2-profiles-20081202/
7. SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query
8. Ontology Definition Metamodel, http://www.omg.org/docs/ptc/07-09-09.pdf
9. Erling, O. : Towards Web Scale RDF. In : ISWC 2008, Workshop: Scalable Semantic Web knowledge Base Systems(SSWS2008).
10. Athanasis, N., Christophides, V., & Kotzinos, D. Generating On the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL). In the 3rd International Semantic Web Conference (ISWC2004), November 7-11, Hiroshima, Japan, pp. 486-501. (available at http://dblp.uni-trier.de/rec/bibtex/conf/semweb/AthanasisCK04)
11. Smart, P. R., Russell, A., Braines, D., Kalfoglou, Y., Bao, J. and Shadbolt, N. (2008) A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. In: *16th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2008)*, 29th September-3rd October 2008, Acitrezza, Catania, Italy.
12. Fadhil, A., Haarslev, V.: OntoVQL: A graphical query language for OWL ontologies. In: International Workshop on Description Logics. (2007)
13. Chen, H., Wang, Y., Wang, H., Mao, Y., Tang, J., Zhou, C., Yin, A., Wu, Z.: Towards a semantic web of relational databases: A practical semantic toolkit and an in-use case from traditional chinese medicine. In Cruz, I.F., et.al, eds.: 5th International Semantic Web Conference. Lecture Notes in Computer Science (4273), Springer (2006) 750–763
14. Bārzdiņš, J., Zariņš, A., Čerāns, K., Kalniņš, A., Rencis, E., Lāce, L., Liepiņš, R., Sproģis, A. GrTP: Transformation Based Graphical Tool Building Platform. In: MoDELS`07, Workshop: Model Driven Development of Advanced User Interfaces (MDDAUI-2007), available at http://ceur-ws.org, Vol 297.
15. Bārzdiņš, J. , Kozlovičs, S. , Rencis, E. The Transformation Driven Architecture. In: OOPSLA`07, Workshop: DSM'08, USA, Nashville, October 2008, pp.60-63.

16. G.Barzdins, E.Liepins, M.Veilande, M.Zviedris, "Semantic Latvia Approach in the Medical Domain", Proceedings of the 8th International Baltic Conference (Baltic DB&IS 2008), June 2-5, Tallin, Estonia, Tallinn University of Technology Press, 89.-102. pp.
17. Broekstra, J., Kampman, A., & Harmelan, F.v. (2002) Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In The Semantic Web - ISWC 2002, volume 2342 of Lecture Notes in Computer Science, pp. 54-68. (available at http://www.cs.vu.nl/~frankh/postscript/ISWC02.pdf)
18. RDF Inference in Virtuoso, http://docs.openlinksw.com/virtuoso/rdfsparqlrule.html