ITC 1/46
Journal of Information Technology and Control
Vol. 46 / No. 1 / 2017
pp. 118-137
DOI 10.5755/j01.itc.46.1.13998

SBVR Based Natural Language Interface to Ontologies

Received 2016/01/20 | Accepted after revision 2017/02/23

http://dx.doi.org/10.5755/j01.itc.46.1.13998

SBVR Based Natural Language Interface to Ontologies

Algirdas Šukys, Lina Nemuraitė, Rita Butkienė

Kaunas University of Technology, Department of Information Systems, Studentų St. 50-308, LT-51368 Kaunas, Lithuania; e-mails: algirdas.sukys@ktu.lt, lina.nemuraite@ktu.lt, rita.butkiene@ktu.lt

Corresponding author: algirdas.sukys@ktu.lt

The semantic search over ontologies allows user to retrieve more relevant results comparing with ordinary keyword based search systems. This type of search system is powered by ontologies and the most convenient interface to ontologies is natural language interface. In this paper, we present multilingual SBVR standard based natural language interface to ontologies, which allows writing questions based on concepts of SBVR vocabulary and transforms them to SPARQL queries using model transformations. The solution can also be used for questioning, when question mapping to ontology is not straightforward. The experimental evaluation of correctness using Mooney Natural Language Learning Data showed results, similar to other natural language interface solutions, answering questions in English and Lithuanian languages.

KEYWORDS: Natural language interface, semantic search, SPARQL, OWL ontology, SBVR business vocabulary, SBVR questions, ATL, Acceleo.

Introduction

The amount of information on the Web grows constantly nowadays. Information overload makes the search process tedious. Traditional keyword based Web search engines use HTML documents that are intended to render information for humans and do not represent semantics that computer can understand. Although such search engines help to find information, they give redundant results with keyword matches, leaving a lot of work for users to find relevant information.

The Semantic Web idea [17] is based on understanding the meaning of published information. The backbone of Semantic Web are ontologies that store entities representing real world objects, their relations, properties, etc. The search over ontologies is called semantic search. Due to capability to understand the intent of user's queries and even complex questions, semantic search returns results that are more precise.

One of the challenges in developing a system with semantic search function is an implementation of

usable graphical user interface. After the Semantic Web idea spread, a number of systems with interfaces to ontologies were created: Semantic Crystal [1], Ginseng [24], QuestIO [2], FREyA [3], ORAKEL [5], PANTO [20], Querix [8], etc. They vary from simple SPARQL interfaces to more sophisticated natural language interfaces (NLI). The study carried out by E. Kaufmann and A. Bernstein [1] have showed that various interfaces differ in their usability and users prefer querying ontologies using full sentences in natural language (NL). This usability study revealed the potential of NLIs for end-user access to the Semantic Web - this type of interface proved most useful and best-liked query interface.

The mandatory requirement of NLI is portability. Portable NLI can be adapted for questioning different ontologies. The process of adaptation is called configuration. It includes creating the lexicon and mapping NL phrases with ontology resources. As this work is time-consuming, most NLIs can be ported to different ontologies without or with minimum configuration efforts. In some situations, mapping can be complex. Ontologies in the Semantic Web are processed by machines and their structure can differ from how questions are formulated (e.g., using n-ary relations [23]). Therefore, robust NLI must have means to map NL phrases with combination of ontology resources also.

Questioning, using regular NL sentences, seems not difficult from the user's perspective. On the other hand, the ambiguity and complexity of NL makes it difficult to interpret user's questions by machine. An important notion here is habitability. It is a term, proposed by Watt [21], to define, how naturally and easy a user can express his thoughts using language restrictions. There are three groups of techniques that are usually used to deal with ambiguities of NL and improve habitability [18]: automatic ambiguity resolving, based on heuristics and ontology reasoning; clarification dialogues; query refinement.

Another feature that is taken into account in this paper is ability to adjust NLI for questioning in different languages. According to [13], 26.3% of internet users use English language and the rest part use other languages. Therefore, multilingualism of NLI is important.

In this paper, we present SBVR based NLI solution. Particular features of this solution is adaptability for different languages and ability to answer questions that cannot be directly mapped to the ontology. The rest of the paper is structured as follows. Section 2 analyses existing NLIs to ontologies. Section 3 presents components of our solution. Sections 4 introduces SBVR metamodel, SPARQL syntax metamodel, and transformation rules of SBVR to SPARQL. Section 5 presents the experimental evaluation. Finally, Section 6 draws conclusions and describes directions for future work.

The comparative analysis of existing NLIs to ontologies

In this section, existing NLIs to ontologies, which show promising results of answering questions, are analysed. In order to find commonalities and differences, the joint method of agreement and difference is used [25]. NLIs are analysed overviewing algorithms of parsing and transforming questions to ontology queries and compared on the following criteria: (1) portability; (2) automatic configuration; (3) mapping NL phrases with combination of ontology resources; (4) automatic ambiguity resolving; (5) clarification dialogs; (6) query refinement; (7) clear adaptability for different languages. In further text, the criterion number in parenthesis marks the mentions of analysed criterion.

The analysis starts from **QuestIO** [2]. This NLI does not require any user training and allows writing English questions of any length and form. QuestIO is portable (1), the lexicon is created automatically by generating gazetteer list from morphologically normalized ontological lexicalizations. Therefore, the approach can be applied for different ontologies without configuration (2). QuestIO cannot map NL phrases with combination of ontology resources (3).

Questions are interpreted identifying key concepts and searching for relations between them based on object properties of the ontology. The algorithm of analysis and transformation of questions includes the following steps: linguistic analysis, ontological gazetteer lookup, transformation to SeRQL query, executing query and displaying results. In the first step, tokenization, POS tagging and morphological analysis is performed. In the second step, annotations for all mentions of ontological resources are created from gazetteer list. In the third step, the most suitable

interpretation is found. Finally, question is transformed to query which is executed against the ontology. Disambiguation (4) is performed using ontology reasoning in order to derive all potential valid interpretations of the question. To find the most suitable interpretation, fuzzy string distance metrics and similarity scores are used. Clarification dialogs (5) are not used in this approach. However, it allows to refine the set of returned documents [18] (6).

The next NLI is **FREyA** [4]. It allows a flexible formulation of English questions, having no strict structures [3]. FREyA is designed by authors of QuestIO to have better understanding of semantic meaning of questions and provide concise answers. FREyA is portable NLI (1), requiring no configuration. The lexicon is derived from the semantic repository by executing the set of SPARQL queries [18] (2). FREyA cannot map NL phrases with combination of ontology resources (3).

The algorithm of translating question to query combines ontology reasoning and syntactic parsing. First of all, ontology based annotations, called ontology concepts (OC), are identified in the question. In the next step, syntax tree is created. Certain words in the syntax tree (e.g., nouns, noun phrases, etc.) are identified as potential ontology concepts (POC). The algorithm iterates through all POCs and tries to map them to OCs either automatically (4) or engaging the user (5). If some POCs cannot be resolved, the algorithm finds the closets OC for that POC by walking through the syntax tree and generates suggestions using ontology reasoning. Suggestions are ranked using string similarity metrics, synonyms, and other algorithms. Clarification dialog is generated for user to select the relevant suggestion. When all POCs are resolved, the query is interpreted as a set of OCs and transformed to SPARQL.

To improve habitability, FREyA also uses query refinement together with feedback mechanism. It allows user to confirm, if question is interpreted correctly or reformulate it if needed (6).

ORAKEL is the system, capable to understand composite semantic constructions, such as quantifications, conjunctions, and negations [5]. ORAKEL is portable (1), but, unlike QuestIO and FREyA, it requires configuration (2). The mapping of NL phrases with ontology is defined creating linguistic structures, called subcategorization frames (i.e., verbs with their arguments). Part of the lexicon (including proper names) is automatically generated from the underlying ontology. WordNet [6] is used to append lexicon with synonyms. ORAKEL allows to relate subcategorization frames with combination of several relations in the ontology and answer questions that do not directly correspond to one relation in the ontology (3).

The parsing process includes syntactic analysis of question and construction of semantic representation in terms of first order logic, enriched with query, count, and arithmetic operators. The syntactic analysis is performed using logical description grammar. First of all, parser selects elementary trees from the lexicon for each token. Parse tree is produced combining elementary trees. Then, the meaning of every word in the parse tree is analysed, semantic representation is created and translated into the query. In ORAKEL, ambiguities are resolved automatically (4) during parsing process. The algorithm selects only those elementary trees that fulfil ontological restrictions. Clarification dialogs (5) and query refinement (6) are not used in this approach.

PANTO [20] accepts English NL questions and is designed to be portable (1) for different domain ontologies without manual configuration (2). The lexicon is built automatically from ontology entities. As well as in ORAKEL, proper names are written to the lexicon. Users can enter their own synonyms, it helps to adapt system for specific domains. In this approach, NL phrases cannot be mapped with combination of ontology resources (3).

The parsing and transformation to SPARQL is performed by the query translator. First of all, questions are parsed using the statistical Stanford Parser [7]. Nominal phrase pairs (i.e., phrases or words and their relationships expressed by verb phrases, prepositions, etc.) are extracted from parse tree to form intermediate representation of question, called query triples. Then, query triples are mapped to ontology triples using lexicon. Simultaneously, parse tree is analysed to extract potential words for targets (i.e., variables after SELECT keyword) and modifiers (i.e., information for UNION and FILTER elements). Ontology triples, targets and modifies are finally used to generate SPARQL query. Questions are disambiguated automatically

(4), matching query triples to ontological triples. This step is performed employing semantic matching (i.e., using WordNet [6]) and morphological matching (i.e., using string metrics and heuristic rules).

PANTO does not use clarification dialogs (5) or query refinement (6).

Querix [8] is domain-independent NLI for the Semantic Web to answer NL questions in English. Querix is portable (1) and requires no manual configuration (2). The lexicon is constructed from the ontology automatically and is enriched using WordNet [6]. Querix does not have means to map NL phrases with combination of ontology resources (3).

The algorithm of question analysis starts from creating the syntax tree using Stanford Parser [7]. Word categories of syntax tree are used to compose the query skeleton. Then, a small set of heuristic patterns are used to identify triple patterns of question. After finding possible triples in the skeleton and combining them with ontology resources, SPARQL query is generated [8].

Querix does not try to resolve ambiguities of NL automatically (4), but asks for user for clarifications using dialogs (5). This approach does not use query refinement (6).

AquaLog is portable (1) question answering system [9], which interprets questions using terms and structure of the ontology. Although Garcia et al. [9] state that configuration time is negligible, AquaLog requires manual configuration (2). However, it cannot be configured to map NL phrases with combination of ontology resources (3).

The analysis of NL question starts translating it into a set of intermediate representations – query triples. Further, relation similarity service (RSS) is used to map query triples to ontology compliant triples. Ontology compliant triples are used to generate SPARQL query. Ambiguities are resolved automatically in RSS. The algorithm uses knowledge, encoded in the ontology and string metrics (4). Ambiguities can also be resolved interacting with users, using clarification dialogs (5). Query refinement (6) is not used in AquaLog.

The analysis is summarized in Table 1. It also includes SBVR based NLI, presented in further sections.

During the analysis, it was not found information about possibilities to adapt existing NLIs for different languages, what components are language dependent, etc.

All of the analyzed NLIs are portable. The lexicon of NLIs is often created semi-automatically. Part of the lexicon is generated from ontology lexicalization, and the rest is created manually by user. However, most NLIs do not have means to relate questions with combinations of several ontology relations.

For improving the habitability, most NLIs use algorithms to solve ambiguities automatically using heuristic rules or ontology reasoning. When ambiguities cannot be resolved automatically, users are involved showing clarification dialogs. FREyA additionally uses query refinement to improve the correctness of answering questions.

Table 1
Comparison of NLIs to ontologies

Criterion	QuestIO	FREyA	ORAKEL	PANTO	Querix	AquaLog	SBVR based NLI
1. Portability	+	+	+	+	+	+	+
2. Automatic configuration	+	+	-	+	+	-	-
3. Mapping NL phrases with combination of ontology resources	-	-	+	-	-	-	+
4. Automatic ambiguity resolving	+	+	+	+	-	+	+
5. Clarification dialogs	-	+	-	-	+	+	+
6. Query refinement	+	+	-	-	-	-	-
7. Clear adaptability for different languages	-	-	-	-	-	-	+

SBVR based natural language interface to ontologies

The created NLI to ontologies is based on the OMG's SBVR standard. SBVR is intended to specify business vocabularies and business rules using structured natural language and allows querying software models by writing questions. The foundation of SBVR is semiotic/semantic triangle, which is the theoretical basis for SBVR's linguistic based architecture and allows separating the expression from meaning [10]. This separation allows expressing same things differently and in different languages. For example, the same question written in different languages will have the same meaning and will be interpreted equally.

Another reason of using SBVR is semantic richness of SBVR specifications. It allows to describe generalizations of concepts, synonyms and use them for semantic search. Furthermore, SBVR concepts can have definitions given as rules that describe derivations of those concepts. Such definitions formally specify the derivation of concepts from other concepts, and can support inferences [19]. We see a good potential of SBVR definitions for bridging the gap between the way in which that data are stored (i.e., the ontology scheme) and the way, how user thinks about data and formulates questions. It allows questioning facts using simple formulations even though they are stored in more complex ontology structures (i.e., expressed through several object properties, derived from values of data properties, etc.).

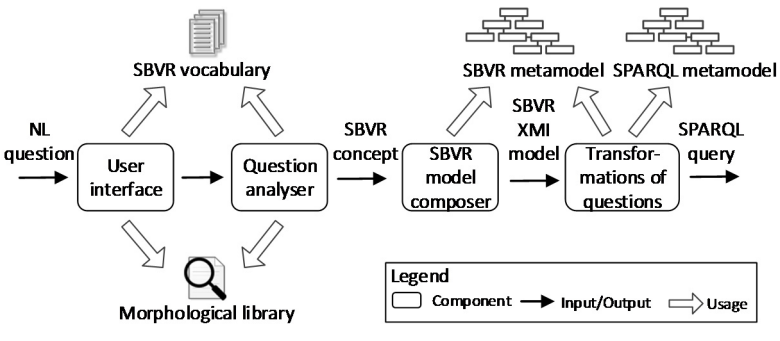
The main components of the solution are presented in Figure 1.

The system accepts NL questions that are written and analysed based on SBVR vocabulary, which is created in a configuration phase, using SBVR Structured language editor [22]. To help writing correct questions, autocomplete is provided for users. Suggestions are generated using SBVR vocabulary. This function is language dependent and requires morphological library to generate words in correct morphological form.

The analysis of questions is performed by question analyzer. This component is language dependent. The goal of the analysis is to find SBVR concept that question is based on. It is performed taking the following steps:

- *Tokenization* – question is splitted into separate tokens;
- *Morphological analysis* – tokens are analyzed morphologically by finding part of speech, lemma and other information using morphological analyzer. This step requires specific morphological analyzer, such as Stanford parser [7] for English;
- *Joining compound SBVR words* – this step is performed comparing tokens with words of SBVR vocabulary. Compound SBVR words (e.g., *large_state*, *works_in*, etc.) are searched in the question and joined into a single token in the question.
- *Identification of SBVR words* – each token that is found in SBVR vocabulary as term, verb or proper name is marked as SBVR word.
- *Clarification* – clarification is used when some words of the question are not recognized as SBVR word and morphological analysis does not provide any helpful information (e.g., word is name of place, surname, etc.). User can clarify unrecognized word

Figure 1
Components of SBVR based NLI



as: (1) synonym of other SBVR vocabulary word; (2) proper name of certain type; (3) stop word that should be skipped in further interpretation. Clarification dialog is also generated in cases of ambiguities when several equal interpretations of questions available. Certainly, some ambiguities can be resolved using context. For example, although Mississippi can mean state or river, but in question *What states border Mississippi?* it is obvious that state is meant.

- *Identification of SBVR concept* – this step is required to find concept (i.e., general or verb concept) that question is based on. Each interpretable question must be based on one or more SBVR concepts. For example, question *What states border Illinois?* is based on SBVR verb concept *state borders state*, while question *Find cities* is based general concept *city*.

Question analyzer also identifies the type of question (simple questions, questions for counting, etc.) and passes it as a parameter for query transformation component to use appropriate transformation rules.

After analyzing question and identifying SBVR concept, SBVR model of question is created by model composer component. If the identified concept has definition, that describes derivation rules of that concept, model composer uses the derivation. In the final step, SBVR model is transformed into SPARQL query using model transformations. Model composer and SPARQL transformation components are independent from language. Metamodels and transformation rules are presented in the following section.

Transforming SBVR questions to SPARQL queries

SBVR metamodel for meaning of question

SBVR metamodel allows formulating three types of meaning: concepts, propositions, and questions. SBVR metamodel fragment to represent meanings and detailed representation of questions is presented in Figure 2.

The meaning of question is formulated using specific SBVR semantic formulation – closed projection. According to the SBVR specification [10], a projection returns a set of things that satisfy projection's constraints. Projection introduces one or more variables to represent types of results. They are defined by general concepts that variables range over. For example, if one wants to see a list of persons, projection introduces the variable that ranges over general concept *person*.

A projection is constrained by a logical formulation, which projects variables using first order logic. A con-

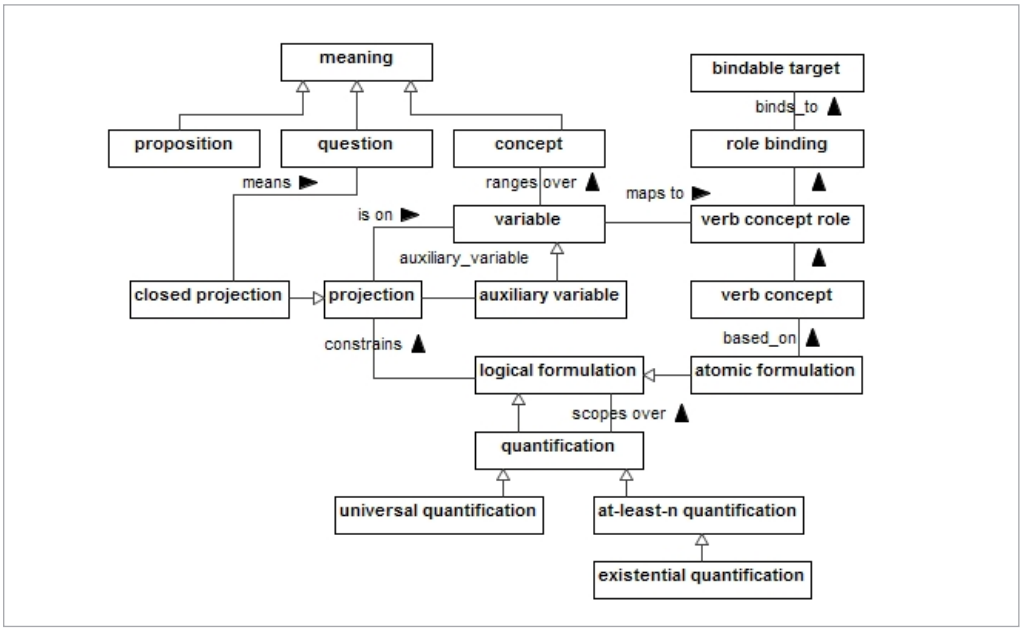


Figure 2
SBVR metamodel for representing meanings

straining logical formulation is based on a verb concept. Depending on the question, verb concept roles can be bound to particular bindable targets – variables or individual concepts to formulate particular meaning.

SPARQL 1.1 syntax metamodel

The SPARQL 1.1 syntax metamodel is based on the W3C specification [11]. Although SPARQL has four types of query (i.e., SELECT, ASK, DESCRIBE, and CONSTRUCT), we use only SELECT queries. Figure 3 presents top-level elements of this query type: SELECT clause, dataset clause, WHERE clause, and solution modifier.

The detalization of SELECT clause is presented in Figure 4. It is used to declare variables to appear in query results. It can be either simple variable or variable expressed using counting, minimum, maximum, or other aggregate functions.

WHERE clause is presented in Figure 5. It defines triple patterns that are used to formulate query results and create bindings of variables, defined in SELECT clause. Results are formed by matching triple patterns with RDF graph.

WHERE clause is expressed by GroupGraphPatternSub element and contains TriplesBlock, which holds TriplesSameSubjectLeft elements, representing tri-

Figure 3
Structure of SELECT query

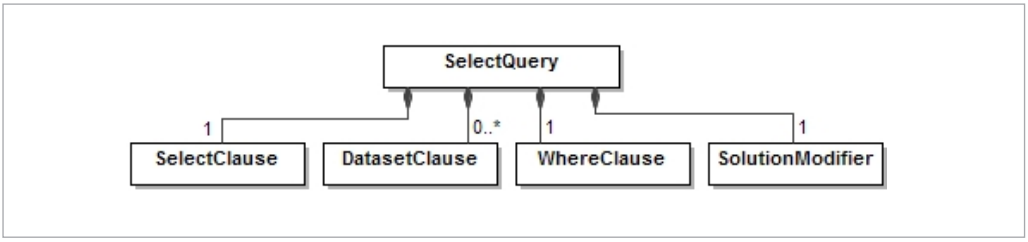


Figure 4
Structure of SELECT clause

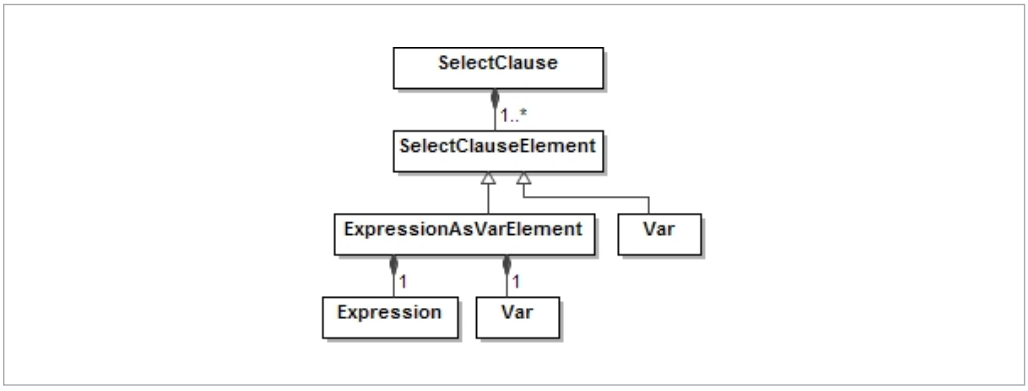
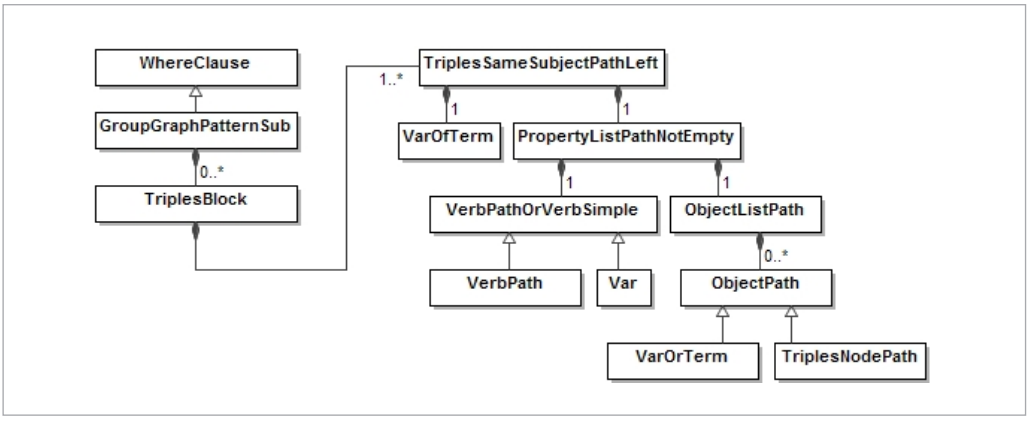


Figure 5
Structure of WHERE clause



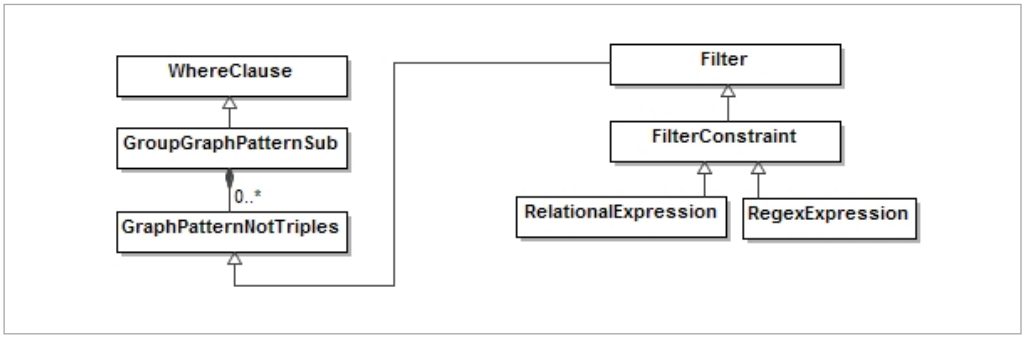


Figure 6
WHERE clause with Filter element

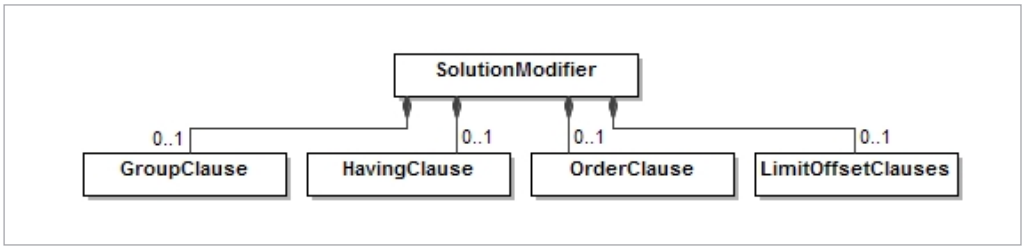


Figure 7
Structure of solution modifier

ple patterns. This element has a structure of subject, predicate, and object. In positions of subject and object, variables or graph elements (e.g., IRI references, blank nodes, RDF literals, numeric literals, Boolean literals, etc.) can be used. In a position of predicate, variables or IRI references, expressed by VerbPath element, are used.

WHERE clause can have Filter elements, expressed by GraphPatternNotTriples or FilterConstraint (see Figure 6). Note that SPARQL specifications describe many other types of filter constraints, such as functions of data type conversions, IN, NOT IN operators, aggregate, rounding functions, etc. In this model, we only include those types of constraints that were used in transformations for numeric comparison and string matching.

The last part of SELECT query is solution modifier. It is presented in Figure 7. This part is used after pattern matching for the following reasons: (1) divide results into smaller groups with GROUP BY modifier to calculate aggregate values; (2) filter grouped solution sets using HAVING modifier; (3) order results using ORDER BY modifier; (4) slice results using LIMIT and OFFSET modifiers.

Rules to transform SBVR questions to SPARQL

There are six types of questions that are transformed in the solution: questions to find individuals of certain type (e.g., *Find persons*); simple questions with roles bound to variables or individuals (e.g., *What states that border Illinois?*); counting questions (e.g., *How many states border Illinois?*); questions with cardinality restriction (e.g., *Find states that border at least 3 states.*); questions with numerical comparison (e.g., *Find cities that have population greater than 100000.*); questions to find minimum or maximum values (e.g., *Find state that has largest population.*).

We defined 9 model transformation rules to transform SBVR questions to SPARQL queries. Transformation rules are called by different algorithms (Figure 8 – Figure 13) depending on the type of question.

Figure 8
Algorithm for transforming questions to find individuals

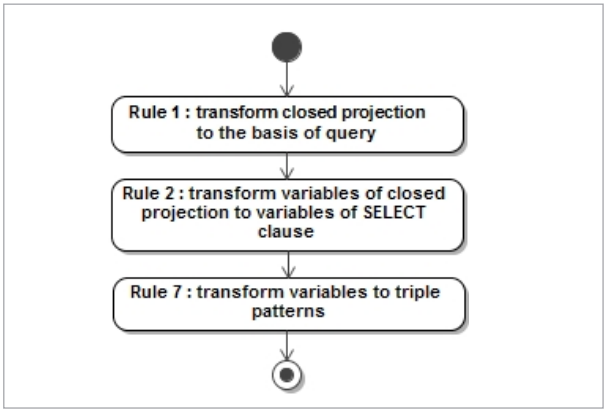


Figure 9
Algorithm for transforming simple questions

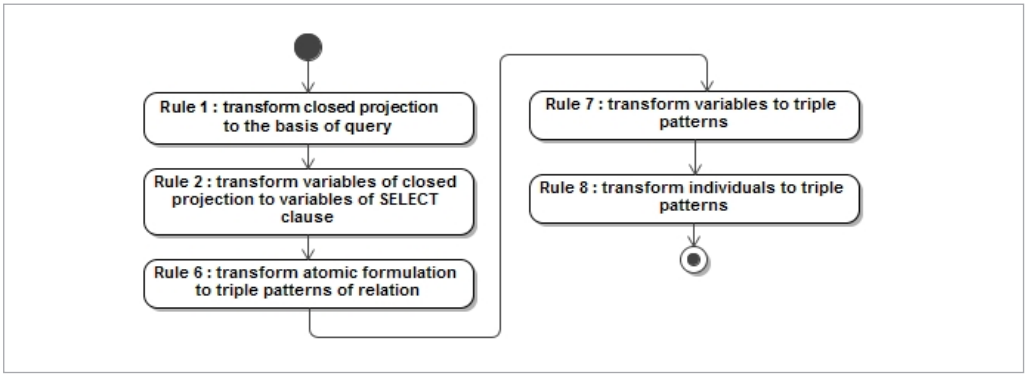


Figure 10
Algorithm for transforming counting questions

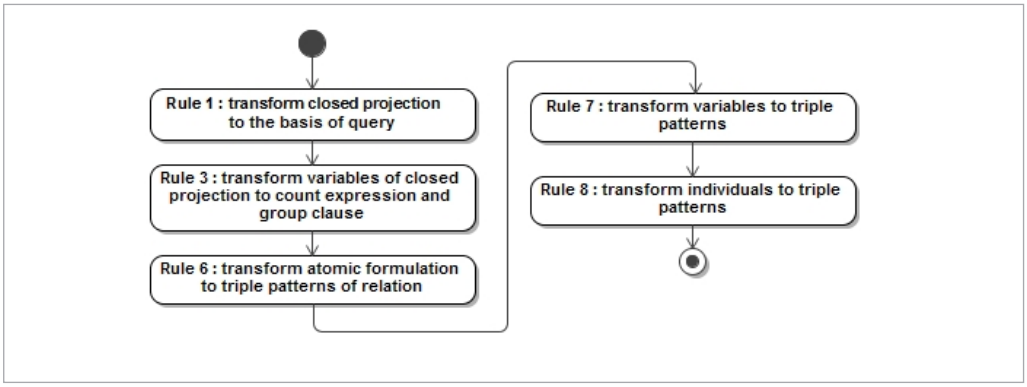


Figure 11
Algorithm for transforming questions with cardinality restrictions

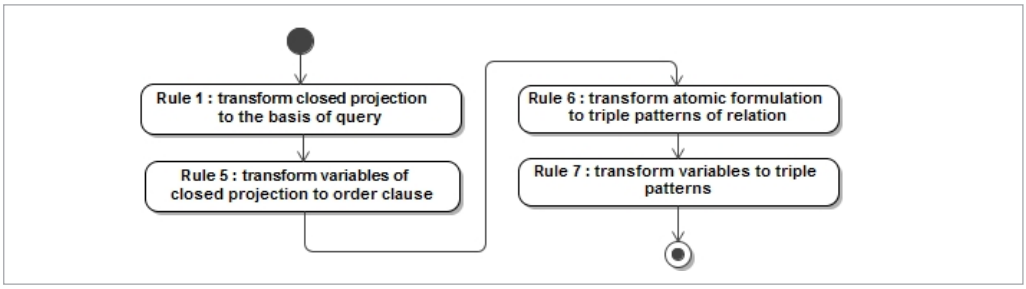
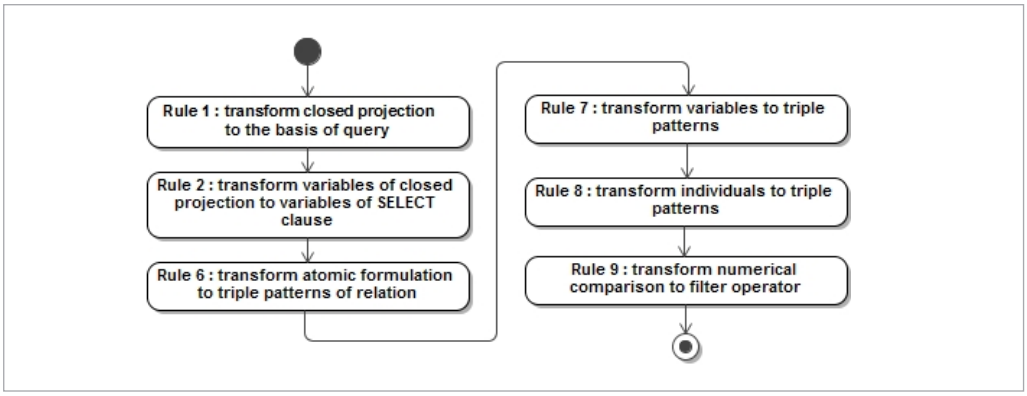


Figure 12
Algorithm for transforming questions with numerical comparisons



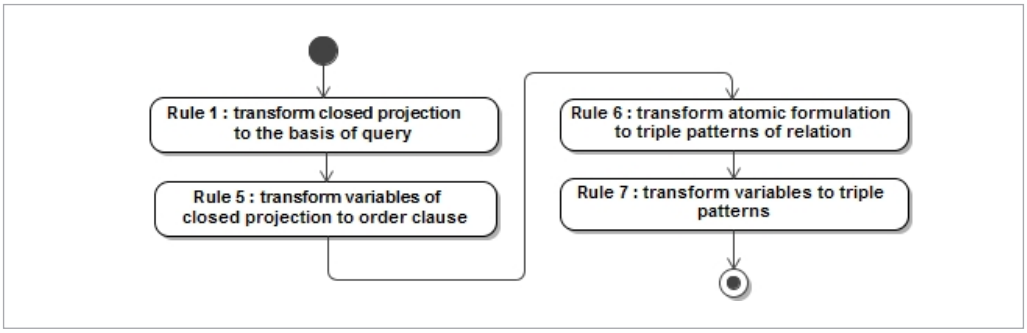


Figure 13
Algorithm of transforming questions with minimum or maximum restrictions

Rule 1: transform closed projection to the basis of query

This rule is called first for all types of questions. It uses closed projection to create top level element of the query with empty SELECT clause and WHERE clause elements that are filled calling subsequent rules. Steps of this rule are presented in Figure 14. Table 2 presents SPARQL model fragment, created by this rule. It also includes example question and SPARQL fragment.

Figure 14
Steps of Rule 1

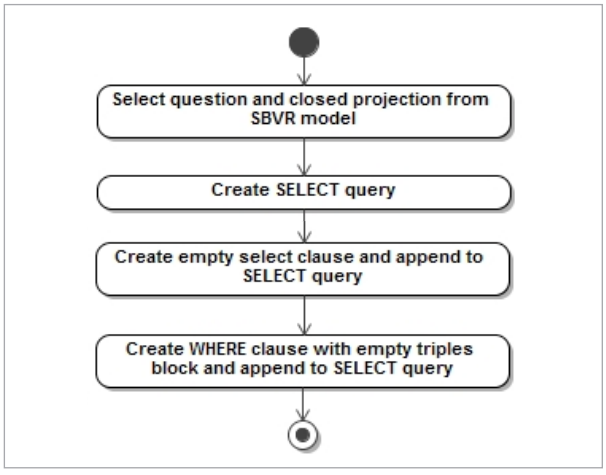


Table 2
Model fragment and example created by Rule 1

```
in: SBVR:ClosedProjection
out: SPARQL:
 SelectQuery (
  SelectClause ( ),
  WhereClause (
   GroupGraphPatternSub (
    TriplesBlock ( )
   )
  )
 )
```

```
SBVR: What rivers run_through states?
SPARQL:
  SELECT
  WHERE { ... }
```

Rule 2: transform variables of closed projection to variables of the SELECT clause

This rule (see Figure 15 and Table 3) transforms variables of closed projection to variables of SELECT clause. Names are set by expressions of general concepts that projection variables range over.

Figure 15
Steps of Rule 2

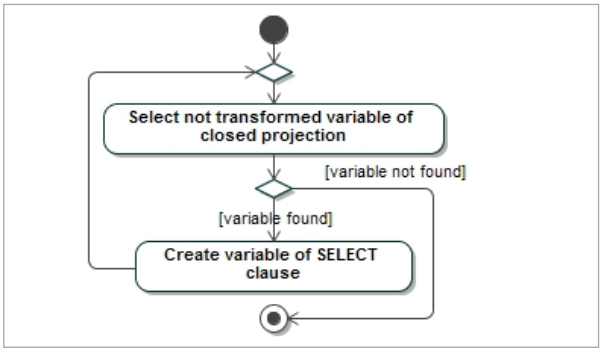


Table 3
Model fragment and example created by Rule 2

```
in: SBVR:ClosedProjection
out: SPARQL:
 SelectClause (
  foreach in.variable as v
   Var(name=v.rangedOver.expr)
 )
SBVR: What rivers run_through states?
SPARQL:
  SELECT
    ?river
    ?state
  WHERE { ... }
```

Rule 3: transform variables of closed projection to count expression and group clause

This rule (see Figure 16 and Table 4) is called transforming questions with counting. Since SBVR metamodel is not capable to represent counting, models for such questions are created in the same way as simple questions. Transformation accepts the parameter to indicate counting questions and call the appropriate rule. This rule takes the first variable of the closed projection and creates the counting expression. The second variable is transformed to group clause.

Figure 16
Steps of Rule 3

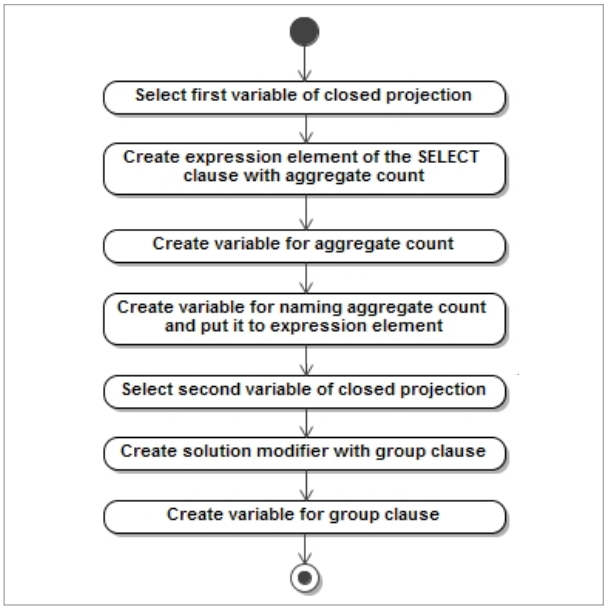


Table 4
Model fragment and example created by Rule 3

```
in: SBVR:ClosedProjection
out: SPARQL:
SelectClause(
 ExpressionAsVarElement1(
  AggregateCount(
   Var(name=in.Var[0].rangedOver.expr)
  ),
  Var(name=in.Var[0].rangedOver.expr +
   "_count")
 )
),
SolutionModifier(
 GroupClause(
  Var(name=in.Var[1].rangedOver.expr)
 )
)
```

```
SBVR: How_many rivers run_through Illinois?
SPARQL:
  SELECT (COUNT(?river_i) as ?river_count)
  WHERE { ... }
  GROUP BY ?state_i
```

Rule 4: transform variables of closed projection to group and having clauses

This rule (see Figure 17 and Table 5) is called to transform variables of closed projection, restricted by cardinality quantification. It creates COUNT function and solution modifier with GROUP BY and HAVING operators from first and second variables of closed projection.

Figure 17
Steps of Rule 4

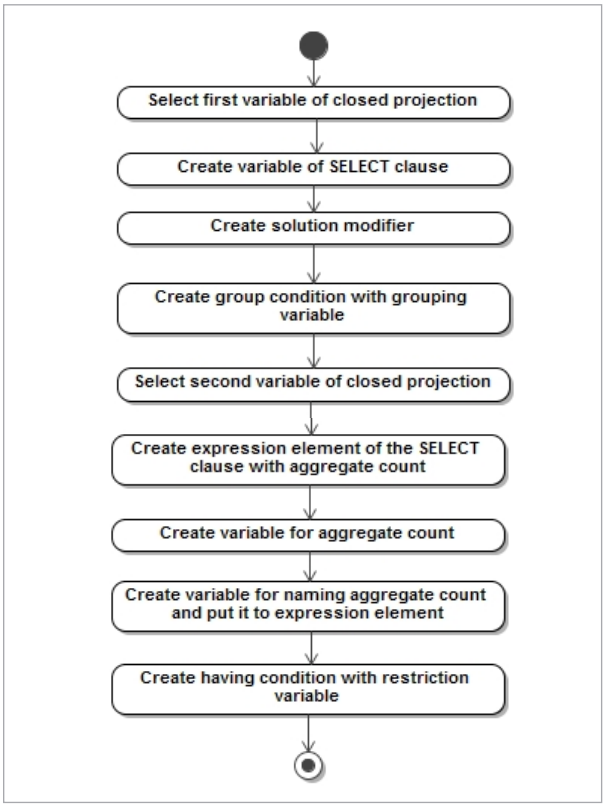


Table 5
Model fragment and example created by Rule 4

```
in: SBVR:ClosedProjection
out: SPARQL:
 SelectClause(
  Var(name=in.Var[0].rangedOver.expr),
  ExpressionAsVarElement1(
```

```
 AggregateCount(
  Var(name=in.Var[1].rangedOver.expr+"_i")
   ),
  Var(name=in.Var[1].rangedOver.expr+
   "_count")
 )
),
SolutionModifier(
 GroupClause(
  Var(name=in.Var[0].rangedOver.expr+"_i")
 ),
 HavingClause (
  Var(name=in.Var[1].rangedOver.expr+
   "_count")
  ComparisonSign
  INTEGER
 )
)
```

```
SBVR: Which rivers run_through at_least 3
      states?
SPARQL:
 SELECT
  ?river_i
  (count(?state_i) as ?state_count)
 WHERE { ... }
 GROUP BY ?river_i
 HAVING(?state_count >= 3)
```

Rule 5: transform variables of closed projection to order clause

This rule (see Figure 18 and Table 6) is called when the restricting atomic formulation is based on *is-property-of* verb concept (e.g., *population of city*), and the property is additionally restricted by minimum or maximum formulations. This rule creates variables of SELECT clause and solution modifier with order and limit clauses from variables of closed projection. Depending on whether it is a minimum or maximum restriction, ordering is ascending or descending.

Figure 18
Steps of Rule 5

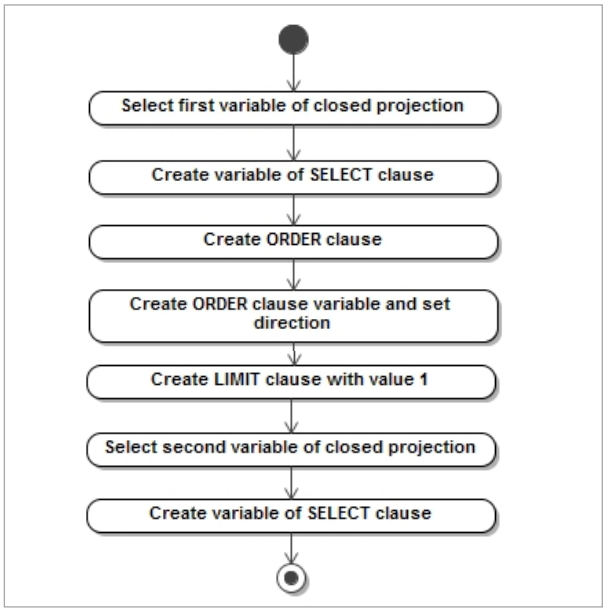


Table 6
Model fragment and example created by Rule 5

```
in: SBVR:ClosedProjection
out: SPARQL:
 SelectClause(
  Var(name=in.Var[0].rangedOver.expr),
  Var(name=in.Var[1].rangedOver.expr),
 ),
 SolutionModifier(
  OrderClause(
   OrderDirection,
   iriOrFunction(
    iri="xsd:float",
    argList=Var(name=
     in.Var[0].rangedOver.expr)
   )
  )
  LimitClause(
   integer=1
  )
 )
```

```
SBVR: What city has largest population?
SPARQL:
 SELECT
  ?city_i
  ?population_i
 WHERE { ... }
 ORDER BY DESC(xsd:float(?population_i))
 LIMIT 1
```

Rule 6: transforming atomic formulation to triple patterns of relation

Atomic formulations are based on verb concepts and are used to express restrictions of questions. Rule 6 (see Figure 19 and Table 7) transforms atomic formulation and its verb concept to two triple patterns, representing relation of verb concept in query.

The first one is the main triple pattern representing relation. It has variables in positions of subject, predicate, and object. The name of the predicate's variable is set by the expression of verb concept's verb symbol. Names of variables in positions of subject and object are set by roles of verb concept and suffixed with "_i". The second triple pattern is used to identify the relation by label.

When both triple patterns are created, they are appended to triples block. If questions use synonymous forms, preferred representations are used.

Figure 19
Steps of Rule 6

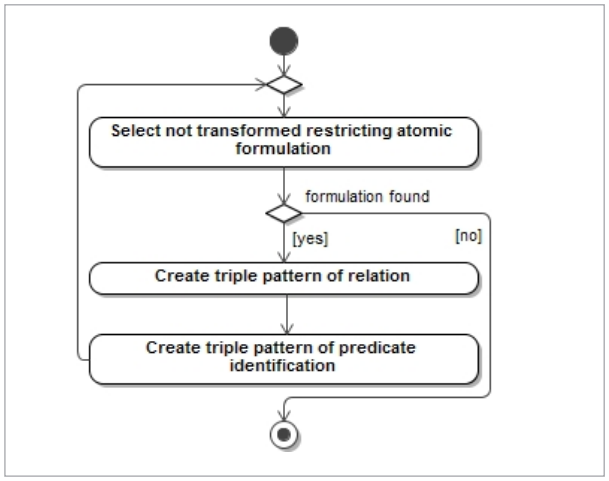


Table 7
Model fragment and example created by Rule 6

```
in: SBVR:AtomicFormulation
out: SPARQL:
WhereClause(
 GroupGraphPatternSub(
  TriplesBlock(
   TriplesSameSubjectPath(
    Var(name=in.verbConcept.role[0].expr +
     "_i"),
    PropertyListPathNotEmpty(
     Var(name=in.verbConcept.verbSymb.expr),
     Var(name=in.verbConcept.role[1].expr +
      "_i")
   )
  )
 ),
 TriplesSameSubjectPath(
  Var(name=in.verbConcept.expr),
  PropertyListPathNotEmpty(
   IRIREF=":label_sbvr",
   STRING_LITERAL=in.verbConcept.
    sentForm.expr + "@" + lang
  )
 )
)
```

```
SPARQL:
 ?city_i ?is_in ?state_i .
 ?is_in :sbvr_label "city is_in state"@en .
```

Rule 7: transform variables to triple patterns

Verb concept roles can be bound to variables or individual concepts for expressing meaning of question.

Bindings to variables are transformed to two triple patterns. The first one creates *rdf:type* relation between variable and its type and the second one identifies type by label. Rule 7 (see Figure 20 and Table 8) is applied for each role binding to variable.

Figure 20
Steps of Rule 7

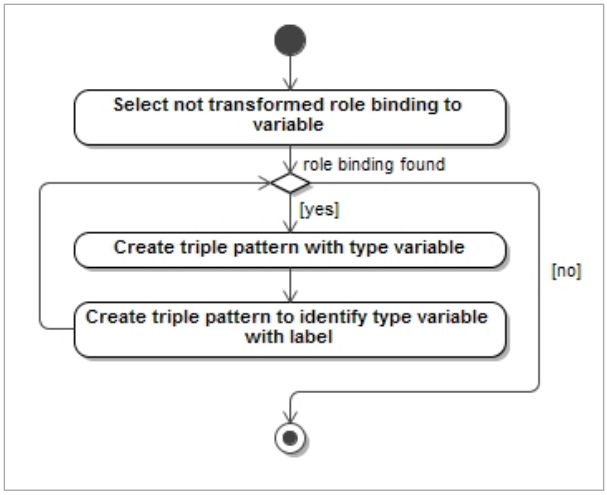


Table 8
Model fragment and example created by Rule 7

```
in: SBVR:Variable
out: SPARQL:
 TriplesSameSubjectPath (
  Var(name=in.rangedOver.expr + "_i"),
  PropertyListPathNotEmpty(
   IRIREF="rdf:type",
   Var(name=in.rangedOver.expr + "_c"),
  )
 ),
 TriplesSameSubjectPath (
  Var(name=in.rangedOver.expr + "_c"),
  PropertyListPathNotEmpty(
   IRIREF="rdfs:label",
   STRING_LITERAL=in.rangedOver.expr+"@"+
    lang
  )
 )
```

```
SBVR: What rivers run_through states?
SPARQL:
 ?river_i rdf:type ?river_c .
 ?river_c rdfs:label "river"@en .
 ?state_i rdf:type ?state_c .
 ?state_c rdfs:label "state"@en .
```

Rule 8: transform individuals to triple patterns

Rule 8 (see Figure 21 and Table 9) is used to transform role bindings to individual concepts. It creates three triple patterns with filter operator. The first two triple patterns are the same as Rule 7 creates. The third one

defines variable of searched individual label and filter element used to filter individuals by label.

Figure 21
Steps of Rule 8

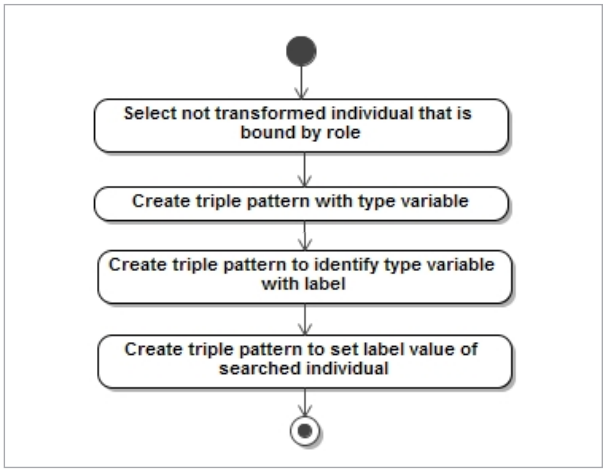


Table 9
Model fragment and example created by Rule 8

```
in: SBVR:IndividualConcept
out: SPARQL:
 TriplesSameSubjectPath (
  Var(name=in.general.expr + "_i"),
  PropertyListPathNotEmpty(
   IRIREF="rdf:type",
   Var(name=in.general.expr + "_c"),
  )
 ),
 TriplesSameSubjectPath (
  Var(name= in.general.expr + "_c"),
  PropertyListPathNotEmpty(
   IRIREF="rdfs:label",
   STRING_LITERAL=in.general.expr+"@"+lang
  )
 ),
 TriplesSameSubjectPath (
  Var(name=in.general.expr + "_i"),
  PropertyListPathNotEmpty(
   IRIREF="rdfs:label",
   Var(name=in.general.expr + "_v"),
  )
 ),
 RegexExpression(
  Var(name=in.general.expr + "_v")
  pattern=in.expr
 )
```

```
SBVR: What rivers run_through Illinois?
SPARQL:
 ?state_i rdf:type ?state_c .
 ?state_c rdfs:label "state"@en .
 ?state_i rdfs:label ?state_v .
 FILTER regex(?state_v, "Illinois")
```

Rule 9: transforming numerical comparison to filter operator

Rule 9 (Figure 22 and Table 10) defines transformation of questions with quantity restrictions, expressed by numerical comparisons of values of data properties. In SBVR models, numerical comparisons are expressed by atomic formulations based on particular verb concepts (e.g., *number1 is_greater_than number2*). This restriction is transformed to filter element in WHERE clause.

Figure 22
Steps of Rule 9

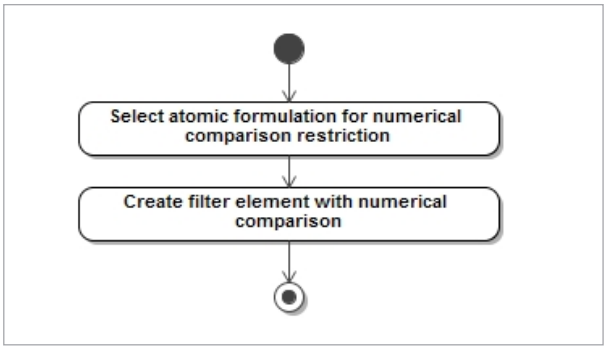


Table 10
Model fragment and example created by Rule 9

```
in: SBVR:AtomicFormulation
out: SPARQL:
 WhereClause(
  RelationalExpression(
   Var(name=in.Var[1].rangedOver.expr)
   ComparisonSym
   INTEGER
  )
 )
```

```
SBVR: What cities has population less_than
      30000?
SPARQL:
 SELECT
  ?city_i
  ?population_i
 WHERE {
  ...
  FILTER(?population_i < 30000)
 }
```

Experimental evaluation

Evaluating correctness of the solution

The goal of experimental evaluation is to investigate, if the created solution allows questioning ontologies

in different languages. We evaluated the correctness using precision, recall and F-measure parameters and also compared results with evaluation of other NLIs.

The prototype NLI was implemented using Java programing language. Presented transformations were implemented using ATL model transformation language. To generate textual query from SPARQL query model, we used Acceleo tool. The prototype was used to evaluate the correctness of our solution by measuring the ability to answer English and Lithuanian questions correctly. We used test data sets that are based on the Mooney Natural Language Learning Data created by Ray Mooney and his group from the University of Texas at Austin [12]. The original knowledge base was created using Prolog and has been used to evaluate NLIDBs. It was translated to OWL knowledge base and is published by the Dynamic & Distributed information Systems Group from University of Zurich [14] and is now often used to evaluate NLIs to ontologies.

To perform the experiment, two knowledge bases were used: geography and restaurant. The first one is designed for storing geographical information about the United States: states, cities capitals of states, borders of states, population, rivers, highest points, etc. This knowledge base contains a set of 880 questions. Its subset consists of 250 questions that semantically represent the whole set. The conceptual model of geography knowledge base is presented as class diagram in Figure 23.

The restaurant knowledge base contains information about restaurants, their ratings, locations, type of food, etc. It has 251 representative questions. The conceptual model of restaurant knowledge base is presented in Figure 24.

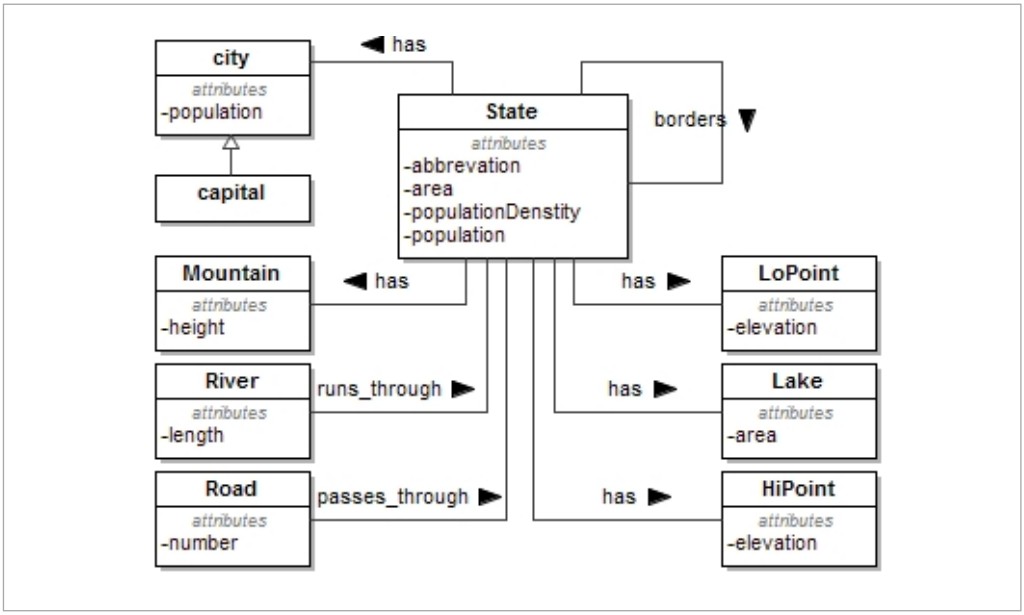


Figure 23
Conceptual model of geography knowledge base

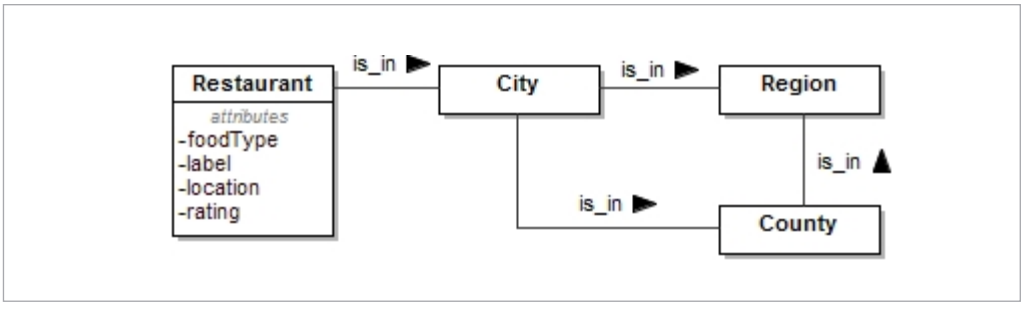


Figure 24
Conceptual model of restaurant knowledge base

Mooney knowledge bases contain only English questions to evaluate NLI. Therefore, we translated those questions to Lithuanian language to evaluate our solution not only in English but also in Lithuanian language.

In our solution, questioning is carried out using SBVR business vocabulary and business rules specifications, corresponding the ontology. These specifications were created in English and Lithuanian languages. We have also created derivation rules for concepts that are derived from their properties (e.g., *large_city* or *italian_restaurant*). Fragments of English and Lithuanian SBVR specifications are presented in Table 11 – Table 14.

Table 11

Fragment of SBVR specifications for geography knowledge base in English

```
city
population
    General_concept: number
    Concept_type: role
city has population
    Concept_type: property_association
```

```
It is necessary that major_city is city that
has population greater_than 300000.
```

Table 12

Fragment of SBVR specifications for geography knowledge base in Lithuanian

```
miestas
populiacija
    General_concept: number
    Concept_type: role
miestas turi populiaciją
    Concept_type: property_association
```

```
Būtina, kad didelis_mietas yra miestas, kuris
turi populiaciją didesnę_u  300000.
```

Table 13

Fragment of SBVR specifications for restaurant knowledge base in English

```
restaurant
rating
    General_concept: text
    Concept_type: role
restaurant has rating
    Concept_type: property_association
```

```
It is necessary that good_french_restaurant is
restaurant that has rating "good" and has
food_type "french"
```

Table 14

Fragment of SBVR specifications for restaurant knowledge base in Lithuanian

```
restoranas
reitingas
    General_concept: number
    Concept_type: role
restoranas turi reitingą
    Concept_type: property_association
```

```
Būtina, kad geras_prancūzi kas_restoranas yra
restoranas, kuris turi reitingą "geras" ir
gamina patiekalų_rū į "prancūzi kas".
```

After creating SBVR specifications, OWL ontologies were prepared by adding labels with SBVR expressions for ontology resources in order to establish the compliance between ontology resources and SBVR concepts using principles defined in [16].

During the experiment, English and Lithuanian questions were transformed into SPARQL queries using created transformations. One of the English questions and transformed query is presented in Table 15.

Table 15

Example question and transformed query

```
What is population of Dallas?
```

```
SELECT
 ?population_i
WHERE {
 ?city_i ?city_has_population ?population_i.
 ?city_has_population rdfs:label "city has
  population"@en .
 ?city_i rdf:type ?city_c.
 ?city_c rdfs:label "city"@en.
 ?population_i rdf:type ?population_c.
 ?population_c rdfs:label "population"@en
 FILTER regex( ?city_i, "Dallas")
}
```

Queries were executed against OWL ontology, and parameters of precision, recall and F-measure were calculated. The precision *PQ* is the number of questions for which the correct answer is returned (*CQ*) divided by number of questions which answers were returned at all (*AQ*). The recall *RQ* is the number of questions for which correct answers were returned (*CQ*) divided by the total number of questions (*TQ*) that can be answered by the knowledge base [15]. Formulas of calculating precision, recall, and *F*-measure are presented below:

$$PQ = \frac{CQ}{AQ} \qquad RQ = \frac{CQ}{TQ} \qquad FQ = 2 \times \frac{PQ \times RQ}{PQ + RQ} \quad (1)$$

Table 16
The results of evaluating correctness

Knowledge base	*TQ*	*AQ*	*CQ*	*PQ*	*RQ*	*FQ*
Geography	250	English questions				
		224	205	0,9151	0,82	0,8649
		Lithuanian questions				
		232	222	0,9569	0,888	0,9212
Restaurants	251	English questions				
		247	188	0,7611	0,749	0,7550
		Lithuanian questions				
		248	187	0,754	0,745	0,7495

The results of evaluating the solution are presented in Table 16.

This experiment showed that the solution answers questions in English and Lithuanian languages well. In the geography knowledge base, the created prototype was not able to answer questions with negations. We have not implemented negations, because Semantic Web uses open world assumption. Due to the imperfection of our natural language analysis algorithms we could not answer some English questions with grammatical structure that differs from the structure of SBVR concepts (e.g., *Through which states does the Mississippi run?*) and questions to find minimum or maximum values according to the specified criterion (e.g., *What is the smallest state by area?*). Our transformation rules could not transform questions with both comparison and minimum or maximum values (e.g., *Which states have points higher than the highest point in Colorado?*).

Results of restaurant knowledge base are worse, because it contains many questions to find addresses of restaurants that the prototype was not able to answer correctly. For example, the question *Where is Chinese food in Bay area?* was answered incorrectly by showing list of restaurants instead of their exact locations.

Clarification dialog made a significant impact on improving precision. It helped to answer questions with names of places, such as *What is the population of Seattle Washington?* where *Seattle Washington* is the composite name meaning city *Seattle* in state *Washington*.

Some other NLIs to ontologies were evaluated using geography knowledge base and showed similar results in English (see Table 17).

Table 17
Comparing correctness with other solutions

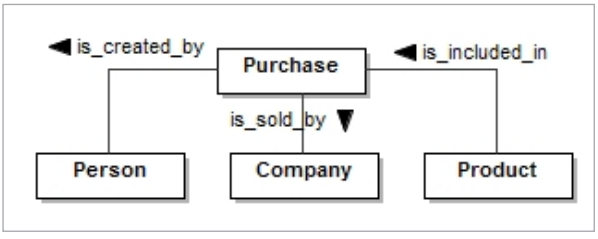
NLI	*PQ*	*RQ*	*FQ*
Querix	0,8608	0,8711	0,8659
PANTO	0,8805	0,8586	0,8694
FREyA	0,924	0,924	0,924
SBVR based NLI	0,9151	0,82	0,8649

Mapping question with combination of ontology resources

In this experiment, we investigate questioning capabilities when the structure of the ontology differs from language formulations used for writing questions. Particularly, we analyse n-ary relation case which occurs in practical applications. The example is adapted from [23] and is presented in Figure 25. It contains the relation class *purchase*, which is connected with classes *buyer*, *seller*, and *products* that are being purchased.

Relations of this ontology do not express very useful information for the user. For example, it is unlikely that the user will be interested which products were included in some purchase. Probably, the user will be interested in relations that are not declared in the

Figure 25
N-ary relation of purchases domain



ontology (e.g., what products were bought by certain person), but can be derived. The solution allows to describe derivations in SBVR specification and formulate questions using derived concepts. The SBVR specification of the example is presented in Table 18.

An example question and transformed SPARQL query are presented in Table 19.

Table 18

SBVR specification for describing n-ary relations of purchases domain

```
purchase
person
product
purchase is_created_by person
product is_included_in purchase
product is_bought_by person

It is necessary that product is_bought_by
person if product is_included_in purchase that
is_created_by person.
```

Table 19

Example question and transformed query

```
What products were bought by John Smith?
```

```
SELECT
 ?product_i
WHERE {
 ?product_i ?is_included_in ?purchase_i.
 ?is_included_in rdfs:label "product.
 is_included_in purchase"@en.
 ?purchase_i ?is_created_by ?person_i.
 ?is_created_by rdfs:label "purchase
 is_created_by person"@en.
 ?product_i rdf:type ?product_c.
 ?product_c rdfs:label "product"@en.
 ?purchase_i rdf:type ?purchase_c.
 ?purchase_c rdfs:label "purchase"@en.
 ?person_i rdf:type ?person_c.
 ?person_c rdfs:label "person"@en
 FILTER regex(?person_i, "John Smith")
}
```

Conclusions and future works

The research has showed that SBVR is capable to be used as a basis of NLI to ontologies. It allows formulating questions in different languages. Multilingualism is achieved by the nature of SBVR to separate meaning from expression.

Rules that we described to transform meaning of SBVR questions to SPARQL queries are independent from language. Therefore, adjusting NLI for certain language requires adapting only question interpretation algorithms and libraries of morphological analysis.

Another situation where it makes sense to use SBVR is when question mapping to ontology is not straightforward. In our solution, questions are formulated using SBVR vocabulary, which contains concepts having direct mappings to ontology resources as well as concepts that are derived from other concepts. These derivations are described in SBVR specification instead of creating additional resources (i.e., object properties) and derivation rules to the ontology.

The drawback of the solution are efforts required for customization to prepare SBVR specification and synchronize it with ontology. Our future work will be related with integrating new features to improve habitability, for example, feedback and query refinement.

Acknowledgements

The work is supported by the project VP1-3.1-ŠMM-10-V-02-008 ″Integration of Business Processes and Business Rules on the Basis of Business Semantics" (2013–2015), which is funded by the European Social Fund (ESF).

References

1. E. Kaufmann, A. Bernstein. Evaluating the Usability of Natural Language Query Languages and Interfaces to Semantic Web Knowledge Bases. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 2010, 8, 377–393.
2. D. Damljanovic, V. Tablan, K. Bontcheva. A Text-based Query Interface to OWL Ontologies. In: 6th Language Resources and Evaluation Conference (LREC), Marrakech, Marocco, 2008.
3. D. Damljanovic, M. Agatonovic, H. Cunningham. Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In: Proceedings of the 7th international conference on The Semantic Web: Research and Applications, 2010, Part I, 106-120.
4. D. Damljanovic, M. Agatonovic, H. Cunningham, K. Bontcheva. Improving Habitability of Natural Language Interfaces for Querying Ontologies with Feedback

and Clarification Dialogues. Web Semantics: Science, Services and Agents on the World Wide Web, 2013, 19, 1–21. https://doi.org/10.1016/j.websem.2013.02.002

5. P. Cimiano, P. Haase, J. Heizmann, M. Mantel. ORAKEL: A Portable Natural Language Interface to Knowledge Bases. Technical report, Institute AIFB, University of Karlsruhe, 2007.
6. C. Fellbaum. WordNet – An Electronic Lexical Database. MIT Press, 1998.
7. D. Klein, C. D. Manning. Accurate Unlexicalized Parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, 2003, 1, 423-430. https://doi.org/10.3115/1075096.1075150
8. E. Kaufmann, A. Bernstein, R. Zumstein. Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. In: 5th International Semantic Web Conference (ISWC 2006), 2006, 980–981. https://doi.org/10.1007/11926078_78
9. V. L. Garcia, E. Motta, V. Uren. AquaLog: An ontology-driven Question Answering System to interface the Semantic Web. In: Proceedings of Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, New York, US, 2006, 269-272. https://doi.org/10.3115/1225785.1225790
10. Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.3. OMG Document Number: formal/2015-05-07, 2013.
11. A. Harris, A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, March 21, 2013. http://www.w3.org/TR/sparql11-query/. Accessed on June 16, 2015.
12. Natural Language Learning Data. http://www.cs.utexas.edu/users/ml/nldata.html. Accessed on April 5, 2015.
13. Number of Internet Users by Language. Internet World Stats, Miniwats Marketing Group, June 30, 2016. http://www.internetworldstats.com/stats7.htm. Accessed on November 6, 2016.
14. OWL Test Data. https://files.ifi.uzh.ch/ddis/oldweb/ddis/research/talking-to-the-semantic-web/owl-test-data/. Accessed on April 5, 2015.
15. L. R. Tang, R. J. Mooney. Using Multiple Clause constructors in inductive logic programming for semantic parsing. In: Proceedings of the 12th European Conference on Machine Learning, Freiburg, Germany, 2001, 466-477. https://doi.org/10.1007/3-540-44795-4_40
16. J. Karpovič, G. Krišč iūnienė, L. Ablonskis, L. Nemuraitė. The comprehensive mapping of semantics of business vocabulary and business rules (SBVR) to OWL 2 ontologies. Information Technology and Control, 2014, 43(3), 289-302. ISSN 1392-124X.
17. T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. Scientific American, 2001, 28–37. https://doi.org/10.1038/scientificamerican0501-34
18. D. Damljanovic. Natural Language Interfaces to Conceptual Models. Ph.D. thesis, The University of Sheffield, Language Resources and Evaluation, 2011. http://etheses.whiterose.ac.uk/1630/. Accessed on April 5, 2015.
19. M. H. Linehan. SBVR Use Cases. In: Proceedings of 2008 International Symphosium on Rule Representation, Interchange and Reasoning on the Web (RuleML'08), ser. Lecture Notes in computer Science, vol. 5321, Berlin, Germany, 2008, 182–196. https://doi.org/10.1007/978-3-540-88808-6_20
20. C. Wang, M. Xiong, Q. Zhou, Y. Yu. Panto: A portable natural language interface to ontologies. In: The SemanticWeb: Research and Applications, 2007, 473-487. https://doi.org/10.1007/978-3-540-72667-8_34
21. W. C. Watt. Habitability. American Documentation, 1968, 19(3), 338-351. https://doi.org/10.1002/asi.5090190324
22. A. Šukys, L. Ablonskis, L., Nemuraitė, B. Paradauskas. A Grammar for Advanced SBVR Editor. Information Technology and Control, 2016, 45(1), 27-41. ISSN 1392-124X.
23. N. Noy, A. Rector. Defining N-ary Relations on the Semantic Web. W3C Working Group Note, 12 April 2006. http://www.w3.org/TR/swbp-n-aryRelations/. Accessed on Junuary 14, 2016.
24. A. Bernstein, E. Kaufmann, C. Kaiser. Querying the semantic web with ginseng: A guided input natural language search engine. In: 15th Workshop on Information Technologies and Systems, Las Vegas, NV, USA, 2005, 112–126.
25. J. S. Mill. A System of Logic. University Press of the Pacific, Honolulu, USA, 2002. ISBN 1-4102-0252-6.

Summary / Santrauka

The semantic search over ontologies allows user to retrieve more relevant results comparing with ordinary keyword based search systems. This type of search system is powered by ontologies and the most convenient interface to ontologies is natural language interface. In this paper, we present multilingual SBVR standard based natural language interface to ontologies, which allows writing questions based on concepts of SBVR vocabulary and transforms them to SPARQL queries using model transformations. The solution can also be used for questioning, when question mapping to ontology is not straightforward. The experimental evaluation of correctness using Mooney Natural Language Learning Data showed results, similar to other natural language interface solutions, answering questions in English and Lithuanian languages.

Palyginus su įprastomis raktiniais žodžiais grindžiamomis paieškos sistemomis, semantinė paieška ontologijose padeda vartotojams gauti tikslesnius rezultatus. Tokio tipo paieškos sistema yra paremta ontologijomis, o pati patogiausia sąsaja ontologijai yra natūralios kalbos klausimų sąsaja. Šiame straipsnyje pristatome daugiakalbę SBVR standartu grindžiamą natūralios kalbos sąsają ontologijoms, kuri leidžia rašyti klausimus naudojant SBVR žodyną ir transformuoja šiuos klausimus į SPARQL užklausas naudojant modelių transformacijas. Sprendimas taip pat leidžia pateikti klausimus, kurių struktūros susiejimas su ontologijos struktūra nėra paprastas. Eksperimentinis tyrimas, naudojant Mooney natūralios kalbos tyrimų duomenis, parodė rezultatus, panašius į kitų sprendimų, atsakant į klausimus anglų ir lietuvių kalbomis.