Terminological and Assertional Queries in KQL Knowledge Access Language

Krzysztof Goczyła, Piotr Piotrowski, Aleksander Waloszek, Wojciech Waloszek, and Teresa Zawadzka

Gdańsk University of Technology, Department of Software Engineering, Narutowicza 11/12, 80-233 Gdańsk, Poland
{kris,piopio,alwal,wowal,tegra}@eti.pg.gda.pl

Abstract. One of the directions of development of information systems in recent years is the evolution of data-based systems into the knowledge-based systems. As a part of this process there is ongoing work on a whole range of languages for accessing knowledge bases. They can be used in a variety of applications, however their main drawback is the lack of clearly defined algebra representing a theoretical basis for them. For instance, such a basis is the relational algebra for SQL. The paper presents a new language for accessing knowledge bases that is built on a solid, theoretical foundation – the algebra of ontological modules.

Keywords: query language, knowledge base, ontology, modularization.

1 Introduction

The recent years in the information science brought more and more discussion about the knowledge and knowledge-based systems. Although there is no clear definition of what the knowledge is (in fact, there are many different definitions), we all agree that the knowledge, unlike the data, requires *understanding* and *reasoning*. The problem of building knowledge-based systems is not a new one, but with the growth of the idea of Semantic Web it gained a new meaning. There appeared a very real need for providing the "engineering tools" that would allow for expressing certain facts and querying systems for conclusions arising from these facts. In practice, this need translates into a number of initiatives aimed at developing a new language to access the knowledge base [1], [2], [3]. Unfortunately, those languages, though useful in various applications, do not have a solid mathematical theoretical basis. In our opinion the development of a universal language for accessing knowledge bases must be based on assumptions similar to those adopted and practically proven in the case of SQL – beside the others, theoretical mathematical basis, language closure and availability of commands for creating, manipulating and controlling the knowledge. In [4] there the algebra of ontological modules (in other words s-modules) was presented that constitutes a theoretical basis for the KQL language presented in this paper. Section 2 briefly presents the basis of algebra of ontological modules, Section 3 describes the formal model of s-modular knowledge base, in Section 4 we present assumptions and rules for creating terminological and assertional queries in KQL while in Section 5 we provide a set of examples presenting the usage of the language.

2 The Algebra of s-modules

Recent years have brought increased interest in modularization of knowledge bases. This stems primarily from the need to provide users with ability to create and process modular knowledge bases, but also ability to execute more advanced operations such as extracting a part of a module, connecting this fragment to a fragment of another module, etc. Another goal is using methods of automatic (and invisible to the user) decomposition of a knowledge base in order to increase performance of inference process. As a response for such requirements the s-modular knowledge bases have been proposed. The s-modular knowledge base is a set of ontological modules and a set of inter-modular Horn rules. An s-module is a structure that represents all models (in terms of first-order logic) of a fragment of knowledge, regardless of how the part is formulated (e.g. regardless of the description language). One of the most important, in the cognitive as well as practical point of view, assumptions that were adopted is the fact that the user is interested in conclusions derived from a given fragment of knowledge, and not in the form of sentences that were used to create the module. As a consequence of this assumption the term of an s-module has been defined strictly semantically focusing solely on its first-order logic interpretation.

Definition 1 (s-module). A s-module M = (S,W) is a pair of a signature S (called a signature of the s-module) and a class W of interpretations. The two parts of M are denoted respectively as S(M) and W(M). Each interpretation from W is called a model of M. A signature S (the set of used names) $S = C \uplus R \uplus I$ consists of concept names (C), role names (R), and individual names (I) ($\uplus$ denotes disjoint union). ■

Due to the fact that with the notion of s-module defined in a such way we are losing the sentence structure (e.g. there is no typical division into a terminological part TBox and an assertional part ABox [5]), the formalism was called algebra of ontological modules [6] [7]. This, of course, does not mean that the sentences are not used to define s-modules. The definition of all basic s-modules (i.e. not created on the basis of other ones) clearly is derived from a defined set of statements. The statements are formulated using (possibly complex) concepts, roles and individuals with operators of a specific dialect of description logic $\mathcal{L}$ (using operators of $\mathcal{L}$ and names from the signature **S**, denoted respectively by $\mathcal{L}_C(\mathbf{S})$, $\mathcal{L}_R(\mathbf{S})$, $\mathcal{L}_I(\mathbf{S})$). In addition, algebra of ontology modules introduces a set of algebraic operations (similar to those defined in the Codd algebra [8]) that allow for creation of new s-modules from existing ones (note that, however, these new s-modules may not be expressible as a set of statements).

3 Formal Model of an s-modular Knowledge Base

Below, there is presented a framework of a formal (expressed in a description logic) model of an s-modular knowledge base (further denoted as a metamodel). It should be noted that the present metamodel provides expressiveness of language for defining s-modules that corresponds to expressiveness of OWL DL version 1.0 [11] and expressiveness of language for defining rules that corresponds to expressiveness of SWRL [12] with a limited set of predicates. The metamodel does not define ability of building s-modules with the use of s-module algebra. It was assumed that s-modules are created as independent objects, and how they are created is not reflected in the model.

In the s-modular knowledge base there exist two types of objects: named (`NamedObject`) and unnamed. Each named object takes exactly one value for the `hasName` attribute (axioms 1 and 2 in Table 1 below). To the named objects belong: basic s-modules (`NamedConglomeration`), named concepts (`NamedConcept`), named roles (`NamedRoleDef`), individuals (`Individual`) and rules (`Rule`) (axioms 3, 4, 5, 6, 7). Moreover, concepts (`Concept`) and properties (`PropertyDef`) are defined within named s-module (`NamedConglomeration`) (axioms 8 and 9), while rules use (`usesTermsFrom`) terms defined within s-modules (axioms 10 and 11).

Concepts has been divided into abstract ones (`AbstractConcept`) and concrete ones (`ConcreteConcept`) (those related to concrete domains) (they are not included in the presented framework due to space limitations). Within abstract concepts there have been defined named concepts (`NamedConcept`) and complex concepts (`ComplexConcept`).

Next, the formal model defines properties, including (as an analogy to OWL) attributes (similarly to concrete concepts, they are not included in the presented framework due to space limitations) and roles (axioms 18 and 19.) The model defines functional roles (axiom 20), transitive roles (axiom 21) and complex roles (axiom 22).

Complex concepts and roles are modeled with the use of `operator` and `operand` roles, that define respectively the operators used to create the given concept or role and components that can be subject of operator's action (axiom 23).

Among the complex roles there are defined complex roles in which the order of operands is important (for instance, such a role is a chain of roles) (axiom 24). For such roles an additional operand denoted by the role `operand1` must be defined (axiom 25).

Moreover, the axioms 26 to 28 define the inheritance hierarchy of roles and concepts. This hierarchy is defined with the roles `subsumer` and `subsumee`.

The last element of the model are instances of properties and concepts. For this reason there have been defined three concepts that represent instances of concepts (`Individual`), instances of roles (`PairOfIndividuals`) and instances of attributes (`PairIndividualValue`) (axioms 29 and 30). The concept `PairOfIndividual` consists of two individuals, while the `PairIndvidualValue` concept consists of a single individual and a single value (axioms 31 – 33).

Table 1. Metamodel framework

NamedObject ⊑ ∃hasName.*string* (1)	TransitiveRole ⊑ RoleDef (21)
NamedObject ⊑ =1hasName.*string* (2)	ComplexRoleDef ⊑ RoleDef (22)
NamedConglomeration ⊑ NamedObject (3)	ComplexConcept ⊔ ComplexRoleDef ⊑ ∃operator.⊤ ⊓ ∃operand.⊤ (23)
NamedConcept ⊑ NamedObject (4)	ComplexRoleOrderDef ⊑ ComplexRoleDef (24)
NamedRoleDef ⊑ NamedObject (5)	ComplexRoleOrderDef ⊑ =1operand1.RoleDef ⊓ =1operand.RoleDef (25)
Individual ⊑ NamedObject (6)	∃subsumer.⊤ ⊑ AbstractConcept ⊔ Role (26)
Rule ⊑ NamedObject (7)	⊤ ⊑ ∀subsumer.AbstractConcept ⊔ Role (27)
∃isDefinedIn.⊤ ⊑ Concept ⊔ PropertyDef (8)	subsumee ≡ subsumer^{-} (28)
⊤ ⊑ ∀isDefinedIn.NamedConglomeration (9)	PairOfIndividual ⊑ =2consistsOf.Individual (29)
∃usesTermsFrom.⊤ ⊑ Rule (10)	PairIndividualValue ⊑ =1consistsOf.Individual ⊓ =1value (30)
∀usesTermsFrom.NamedConglomeration ⊑ ⊤ (11)	Concept ⊑ ∀hasInstance.Individual (31)
AbstractConcept ⊑ Concept (12)	RoleDef ⊑ ∀hasInstance.PairOfIndividuals (32)
ConcreteConcept ⊑ Concept (13)	AttributeDef ⊑ ∀hasInstance.PairOfIndividuals (33)
NamedConcept ⊑ AbstractConcept (14)	
ComplexConcept ⊑ AbstractConcept (15)	
RoleDef ⊑ PropertyDef (18)	
AttributeDef ⊑ PropertyDef (19)	
FunctionalRole ⊑ RoleDef (20)	

4 Basic Assumptions for KQL

Development of new knowledge representation and development of new kwnowledge access language were guided by the same objective: to create „engineering" tools that could be used for creation knowledge based systems. Therefore it has been proposed a set of requirements (characteristics) that should be meet by a uniform language for accessing an s-modular knowledge base. The essential characteristics are: (1) existence of statements for creating knowledge bases, manipulating them and controlling access to them, (2) closure (query statements should return the same structures they operate upon), (3) ability of query nesting, (4) uniformity of definition and modification language (results of queries can be used in queries operating on the knowledge base and creating knowledge base), (5) ability to pose terminological queries, (6) support for ontology modularization.

These characteristics have been proposed in an analogy to characteristics of SQL. KQL, in analogy to SQL, provides statements for creating knowledge bases, manipulating them and controlling access to them.

Statements for creating a knowledge base include creation of: knowledge bases (ontologies); s-modules as conceptual parts of a knowledge base; and rules and mappings to external data sources.

The main statement for manipulating an s-module is the SELECT statement of the following structure:

```
SELECT concept_list, role_list, attributes_list
  ADD axiom_list
  FROM conglomeration_expression
  WHERE concept_expression
  HAVING concept_expression
```

With respect to the s-module algebra, the SELECT clause corresponds to the projection—the choice of terms for a newly created s-module, the ADD clause corresponds to the selection—the contraction of the set of allowed interpretations, the WHERE and HAVING clauses correspond to the projection with respect to individual names. The difference between the WHERE and HAVING clauses lies in the fact that the WHERE clause selects those individuals that belong to a specified concept of the original s-module (as defined in the FROM clause), while the HAVING clause selects those individuals that belong to a specified concept of the target s-module. The query is conceptually executed in the following steps: (1) Determining the basic s-module on the basis of the FROM clause. (2) Reducing the individual names on the basis of the WHERE clause. (3) Extending the alphabet with new concepts / roles / attributes / individuals that occur in the statements contained in the ADD clause (selection). (4) "Adding" (i.e. extending the ontology) to the s-module the statements from the ADD clause. (5) Projection of the alphabet only to the concepts / roles / attributes contained in the SELECT clause. (6) Reducing the individuals names basing on the HAVING clause.

The least matured commands in the current version of the KQL are commands of knowledge access control. The language, however, is designed in a way allowing for its further development also in this, very important from a practical point of view, direction.

KQL is **a closed language**. Utilizing the s-module algebra as a theoretical basis for the language allowed for developing a language that operates upon and returns s-modules. Results of queries that operate on s-modules are also s-modules. A direct consequence of KQL closure is ability of query nesting. Every time an s-module is needed, one can use named s-modules defined in knowledge base or s-modules created on the fly by KQL expressions with the use of s-module algebra operators (such as INTERSECT, JOIN, UJOIN) or the SELECT clause. As a consequence of KQL closure and the ability of **query nesting,** the **uniformity of modification and definition language** has been achieved: one can use query nesting also in s-module creation statements (in KQL, an s-module is called “conglomeration”):

```
CREATE CONGLOMERATION conglomeration_name
…
FROM conglomeration_expression
```

In KQL it is assumed that **terminological queries** are issued against a metaontology or a meta s-module. A metaontology is nothing but a single s-modular knowledge base that stores information about s-modules and rules. A meta s-module is also a single s-modular knowledge base that stores information about exactly one s-module. A metaontology and meta s-modules are built in accordance to the following assumptions:

– Any s-module, rule, concept, role, attribute, and individual of an s-modular knowledge base is reified to individuals that are instances of relevant concepts of the model;
– Description of world is infinite, because every concept and role are generating an infinite number of individuals: each concept and complex rule corresponds to a single individual, and every concept and rule may be defined in countless ways (e.g. A, $A \sqcup A$, $A \sqcup A \sqcup ... \sqcup A$). All named terms defined in a domain s-modular knowledge base are reified to individuals of names of terms of the domain knowledge base.

As a consequence of using the metaontology it is possible to issue terminological queries in a similar way how the assertional queries are issued; the only difference is that a query is directed to the metaontology rather than to the knowledge base itself. This follows from the fact that s-modules, concepts, roles, attributes, and individuals from a knowledge base are reified to individuals in the metaontology and the relationships between them are reified to the corresponding binary relationships between individuals.

4 Examples of KQL Usage

The examples below (1-3) have already been defined in terms of the s-module algebra in [7]. Here the same examples have been reformulated in KQL. Example 4 shows sample terminological queries.

Example 1 (simple import). We consider two s-modules: M_1 describes human resources, and M_2 describes a structure of a hospital.

```
CREATE CONGLOMERATION M1
    ADD CONCEPT HTBusinessUnit
    ADD CONCEPT Expert
    ADD CONCEPT Employee
    ADD ROLE isManagerIn

CONSTRAIN M1
   EXIST isManagerIn
     HTBusinessUnit ISSUB Expert
   Expert ISSUB Employee
```

```
CREATE CONGLOMERATION M2
   ADD CONCEPT Department
   ADD ROLE leadsDepartment
   ADD INDIVIDUAL johnSmith
   ADD INDIVIDUAL neurosurgery

CONSTRAIN M2
  (johnSmith, neurosurgery) IS
      leadsDepartment
  neurosurgery IS Department
```

In KQL each s-module is created in two steps. In the first step the s-module signature is created (`CREATE CONGLOMERATION` statement). In the second step the constraints (sets of statements that must be satisfied) are added (`CONSTRAIN` statement). In the example above, the first s-module defines an employee and an expert. We assume that each expert is an employee, and anyone who manages at least one business unit is an expert. The second s-module describes John Smith who leads the department of neurosurgery. We want to ask whether `johnSmith` is the expert.

```
SELECT Expert
   ADD leadsDepartment ISSUB isManagerIn
   ADD Department ISSUB HTBusisnessUnit
   FROM M1 INTERSECT M2
   WHERE {johnSmith}
```

To merge the information from the two s-modules in order to check whether `johnSmith` is an expert we first create an intersection of the s-modules (FROM statement), and then restrict the set of model by introducing additional “bridge” axioms (ADD statement). The result of the query is a new s-module whose signature consists of two terms: `johnSmith` and `Expert`. In our example `johnSmith` is an instance of `Expert` concept.

Example 2 (basic ontology alignment)
In the above example we did not encountered any name confilicts. In general, such a conflict may easily occur. In this example we show how to align two s-modules in which the same set of terms has been used to express different mappings. Let us assume that there exists two s-modules that define three concepts: `HSRoom`, `ASRoom` i `LSRoom` that denote rooms of high, average and low standard, respectively. The first s-module M_o defines the standard of rooms rented for a single night, while the s-module M_l defines those rented for more than one night. Our objective is to create a new s-module M that would define a different room standard in accordance to the rent time. Moreover, while creating the new s-module it has been assumed that rooms with bathrooms are of high standard in case of renting them for a single night, and rooms with no bathroom are of low standard if rented for more than one night.

```
CREATE CONGLOMERATION Mo            CREATE CONGLOMERATION Ml
       ADD CONCEPT HSRoom                  ADD CONCEPT HSRoom
       ADD CONCEPT ASRoom                  ADD CONCEPT ASRoom
       ADD CONCEPT LSRoom                  ADD CONCEPT LSRoom

            CREATE CONGLOMERATION M
            FROM(
               SELECT *
                 ADD RoomsWithBathroom ISSUB HSRoomON
                 ADD (NOT RoomsWithBathroom) ISSUB LSRoom
                 FROM(
                   SELECT HSRoomON, ASRoomON, LSRoomON
                     ADD HSRoomON EQ HSRoom
                     ADD ASRoomON EQ ASRoom
                     ADD LSRoomON EQ LSRoom
                     FROM (Mo INTERSECT Ml)
                   )
            )
```

To create M module firstly we gather information from both modules (intersection of M_1 module and M_o module with renamed concepts). In that way we create a temporary, unnamed module that defines six different classes of rooms. Secondly we have to establish a translation between “one night” and “longer” concepts. We extend the temporary module by the criteria which were used for the assessment in both cases (the two sentences about rooms with bathrooms).

Example 3 (different versions and what-if problems)
This example illustrates use of union and negation in KQL. Let us consider an s-module M:

```
CREATE CONGLOMERATION M          CONSTRAIN CONGLOMERATION M
   ADD CONCEPT TrustedWitness       Top ISSUB <= 1 INV (murdered)
   ADD CONCEPT CrimeScene                {victim}
   ADD ROLE murdered                EXIST INV(accuses) TrustedWitness
   ADD ROLE accuses                 ISSUB EXIST INV(murdered) {victim}
   ADD ROLE presentAt               TrustedWitness ISSUB EXIST
   ADD INDIVIDUAL victim                        presentAt CrimeScene
```

The M s-module describes a world that assumes the only one murderer who is accused by a trusted witness (who has been present at the crime scene). We consider two (mutually exclusive) versions of facts (e.g. collected by two investigating agents:

John Shady and Henry Brilliant). To achieve that we define two new s-modules (one for each agent).

```
CREATE CONGLOMERATION M1            CREATE CONGLOMERATION M2
FROM(                               FROM(
   SELECT *                            SELECT *
   ADD johnShady IS                    ADD henryBrillant IS TrustedWitness
     TrustedWitness                    ADD (henryBrillant, markGuilty) IS
   ADD (johnShady, tedInnocent)             accuses
     IS accuses                        FROM M
   FROM M                           )
)
```

Having such defined s-modules M_1 i M_2 we are able to analyze different scenarios. To perform such analyses we create a new s-module $M_0 = M_1$ UNION M_2.

1. We may assume that `henryBrillant` is not a trusted witness:

```
SELECT Murderer
   ADD NOT TrustedWitness(henryBrilliant)
   ADD Murderer EQ EXIST murdered {victim}
   FROM M1 UNION M2
```

We ask for a murderer. Therefore we define `Murderer` as a person who `murdered` a `victim`. The resulting s-module will return exactly one individual (`tedInnocent`) as an instance of `Murderer` concept.

2. We may assume that `markGuilty` is a murderer

```
SELECT TrustedWitness, NotTrustedWitness
   ADD (markGuilty, victim) IS murdered
   ADD NotTrustedWitness EQ NOT TrustedWitness
   FROM M1 UNION M2
```

In this case we can conclude that `henryBrillant` is a trusted witness (but this does not mean that `johnShady` is not a trusted witness). The s-module created within the query defines `NotTrustedWitness` concept (the complement of `TrustedWitness` concept defined in s-modules M_1 and M_2). The s-module which is the result for this query will return a single instance of `TrustedWitness` concept and will return no instance of `NotTrustedWitnesss` concept.

3. We may assume that `johnShady` was not present at the crime scene

```
SELECT TrustedWitness, Murderer
   ADD johnShady IS EXIST presentAt CrimeScene
   ADD Murderer EQ EXIST murdered {victim}
   FROM M1 UNION M2
```

In this case we can conclude that `johnShady`$\psi$is not a trusted witness and `markGuilty`$\psi$is the murderer. To do this we define `Murderer` concept similarly to the second example.

Example 4 (terminological queries)

Let us define the following s-module in which we have defined `Students`, `Teachers` and `WorkingPeople`.

```
CREATE CONGLOMERATION M          CONSTRAIN CONGLOMERATION M
   ADD CONCEPT Students             ADD Teachers ISSUB WorkingPeople
   ADD CONCEPT Teachers             ADD Students ISSUB WorkingPeople
   ADD CONCEPT WorkingPeople
```

We ask whether the concept `WorkingPeople` subsumes the concept `Teachers`:

```
SELECT CONCEPTS NamedConcept, ROLES subsumer
FROM META (CONGLOMERATION M)
WHERE { Teachers, WorkingPeople }
```

In KQL the terminological query is, as stated above, issued against a meta s-module that is created with the META tag. As a result we receive an s-module with a single concept defined named `NamedConcept` and a single role `subsumer`. The individuals of names `Teachers` and `WorkingPeople` are instances of `NamedConcept` concept and are in subsumption relation: the pair (`Teachers`, `WorkingPeople`) is instance of `subsumer` role. Otherwise (e.g. if we would ask for concepts `Students` and `Teachers`), the resulting s-module will not contain any instance of `subsumer` role. In this example the concept `Students` and the concept `Teachers` are named concepts. In the case of unnamed concepts a new s-module should be created in which names are assigned to the complex concepts. For instance, let us ask whether the complex concept being the intersection of `Students` and `Teachers` is subsumed by the concept `WorkingPeople`.

```
SELECT CONCEPTS NamedConcept, ROLES subsumer
  FROM META (
     SELECT CONCEPTS WorkingPeople, StudentsAndTeachers
     ADD (StudentsAndTeachers EQUALS Students AND Teachers)
     FROM CONGLOMERATION M
    )
  WHERE { Students, Teachers }
```

In the query above there is created a new conglomerate where the complex concept being an intersection of concepts `Students` and `Teachers` was named `StudentsAndTeachers`.

6 Summary

The paper presents the concept of a s-modular knowledge base along with a language for accessing the knowledge stored in such a base. Existing languages for accessing the knowledge bases focus either on defining knowledge bases, or on the querying a knowledge base [2], [4], [5]. On the contrary, the KQL language does cover all aspects of accessing a knowledge base, i.e. it offers both ability of creating and populating knowledge bases, as well as ability of querying them. The same language constructs are used in commands of writing to and reading from a knowledge base. In addition, due to the incorporation of s-module algebra which is the theoretical basis for KQL, ability of modularizing knowledge bases is smoothly integrated into the language. Different languages provide different ways for handling terminological queries. For KQL it has been chosen a very flexible and conceptually coherent method of asking terminological queries through a metaontology. Detailed comparison of KQL language with other languages can be found in [13].

A formal model for an s-modular knowledge base from the very beginning has been designed with a special attention for its use to handle terminological queries. This distinguishes it from other metamodels, such as Ontology Definition Metamodel published by OMG [14], created for the sake of generality and widest range of applications. KQL is also being actively and continuously developed, primarily for various practical extensions, such as operations on concrete domains. The language authors' goal is to provide knowledge engineers with a comfortable, but simultaneously based on solid theoretical foundations, set of tools to create and manipulate knowledge bases.

References

1. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member, Submission May 21 (2004), `http://www.w3.org/Submission/SWRL/`
2. Prud'hommeaux, E.: SPARQL Query Language for RDF, `http://www.w3.org/TR/rdf-sparqlquery/`
3. Fikes, R., Hayes, P., Horrocks, I.: OWL-QL - A Language for Deductive Query Answering on the Semantic Web. Knowledge Systems Laboratory. Stanford University, Stanford (2003), `http://ksl.stanford.edu/KSL_Abstracts/KSL-03-14.html`
4. Kubias, A., Schenk, S., Staab, S.: (Demonstration at ESWC), SAIQL Query Engine - Querying OWL Theories for Ontology Extraction, `http://www.eswc2007.org/pdf/demopdf/SaiqlPosterProposal.pdf`
5. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: Description Logic Handbook. Cambridge University Press, Cambridge (2002)
6. Goczyła, K., Waloszek, A., Waloszek, W.: S-Modules – An Approach to Capture Semantics for Modularized DL Knowledge Bases. In: KEOD 2009, Funchal-Madeira, Portugalia, pp. 117–122 (2009)
7. Goczyła, K., Waloszek, A., Waloszek, W.: Algebra of ontology modules for semantic agents. In: Nguyen, N.T., Kowalczyk, R., Chen, S.-M. (eds.) ICCCI 2009. LNCS (LNAI), vol. 5796, pp. 492–503. Springer, Heidelberg (2009)
8. Codd, E.F.: Relational Completeness of Data Base Sublanguages. Database Systems 6, 65–98 (1979)
9. Bouquet, P., Ghidini, C., Giunchiglia, F., Blanzieri, E.: Theories and uses of context in knowledge representation and reasoning. Journal of Pragmatics 35(3), 455–484 (2003); Intelligence, Third International Atlantic Web Intelligence Conference, pp. 163–169. Springer, Heidelberg (2005)
10. Ghidini, C., Giunchiglia, F.: Local Model Semantics, or Contextual Reasoning = Locality + Compatibility. Artificial Intelligence 127(2), 221–259 (2001)
11. OWL Web Ontology Language Guide, W3C Recommendation February 10 (2004), `http://www.w3.org/TR/owl-guide/`
12. Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission May 21 (2004), `http://www.w3.org/Submission/SWRL/`
13. Goczyła, K., Piotrowski, P., Waloszek, A., Waloszek, W., Zawadzka, T.: Język KQL jako realizacja idei języka SQL dla bazy wiedzy. Studia Informatica 31(2A(89)), 47–61 (2010)
14. Ontology Definition Metamodel (ODM) Version 1.0, Release date (May 2009), `http://www.omg.org/spec/ODM/1.0/PDF`