STUDIA INFORMATICA 2010
Volume 31 Number 2A (89)

Krzysztof GOCZYŁA, Aleksander WALOSZEK,
Wojciech WALOSZEK, Teresa ZAWADZKA
Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki

IDEOLOGICZNY I PRAKTYCZNY MODEL METAONTOLOGII

Streszczenie. Rozwój inicjatywy *Semantic Web* spowodował rozwój różnych języków definiowania wiedzy i manipulowania nią. W ramach tych języków istotnym elementem jest zaprojektowanie rozkazów terminologicznych. W artykule przedstawiono sposób realizacji zapytań terminologicznych w języku KQL (*Knowledge Query Language*) – języku dostępu do systemu zarządzania wiedzą RKaSeA.

Słowa kluczowe: bazy wiedzy, język zapytań, Semantic Web, system wnioskujący, wiedza, metaontologia, zapytania terminologiczne

IDEOLOGICAL AND PRACTICAL MODEL OF METAONTOLOGY

Summary. The development of *Semantic Web* initiative resulted in development of many languages for defining and manipulating knowledge. An important aspect of design of those languages is design of terminological statements. The paper presents the way the terminological statements are realized in KQL language (acronym for *Knowledge Query Language*) – an access language for RKaSeA knowledge management system.

Keywords: knowledge bases, query language, Semantic Web, inference system, knowledge, metaontology, terminological queries

1. Wstęp

Rozwój Internetu, a wraz z nim rozwój metod semantycznego zarządzania wiedzą, spowodował rozwój systemów baz wiedzy. Systemy te korzystają z różnych reprezentacji wiedzy. Wśród tych reprezentacji coraz bardziej popularne stają się reprezentacje łączące reguły z ontologiami wyrażonymi w logikach opisowych (ang. *Description Logics* [1]). Taka reprezentacja wiedzy została przyjęta również jako podstawa dla system RKaSeA, systemu zarzą-

dzania wiedzą opracowywanego w Politechnice Gdańskiej. W ramach prac nad tym systemem zauważono jednak, że głównym minusem tej reprezentacji jest brak wsparcia dla modularyzacji ontologii. Dlatego rozszerzono tę reprezentację o konglomeraty [2, 3], stanowiące pewne semantyczne fragmenty bazy wiedzy.

W niniejszym rozdziale opisano opracowany metamodel wiedzy dla systemu RKaSeA. Metamodel ten, dalej zwany modelem, stanowi semantyczną podstawę dla języka KQL (*Knowledge Query Language*) [4] dostępu do systemu RKaSeA. Podstawową cechą języka KQL jest jego domkniętość (język KQL operuje na konglomeratach i zwraca konglomeraty). Aby zachować tę cechę, twórcy języka KQL przyjęli, że zapytania terminologiczne zostaną zaprojektowane jako rozkazy do metaontologii, stanowiącej formalny zapis modelu.

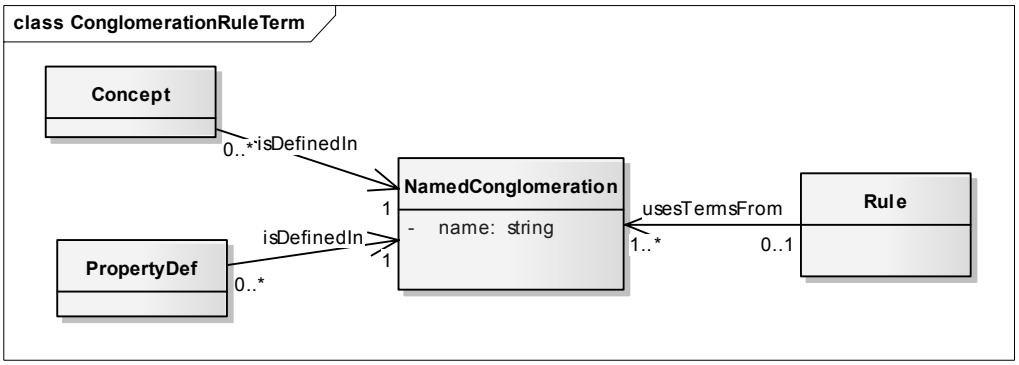
Podrozdział 2 przedstawia model wiedzy dla systemu RKaSeA ograniczony do elementtów, które są istotne z perspektywy języka porozumiewania się z systemem. Podrozdział 3 krótko wprowadza w budowę metaontologii, a podrozdział 4 w realizację zapytań metaontologicznych w systemie RKaSeA. Podrozdział 5 wprowadza podstawowe rozkazy języka KQL, które są wykorzystane do definiowania rozkazów terminologicznych w odniesieniu do interfejsu DIG 1.1 [5]. Podrozdział 6 stanowi podsumowanie niniejszych rozważań.

2. Model wiedzy

Obecne opracowanie modelu wiedzy dla systemu RKaSeA bazuje na modelu konglomeratowym. W modelu konglomeratowym przyjmuje się założenie, że baza wiedzy powinna być zorganizowana w moduły (fragmenty), które mogą być przez użytkownika przetwarzane i łączone w celu otrzymania nowych jednostek wiedzy. Uzyskuje się dzięki temu elastyczność przetwarzania wiedzy porównywalną z elastycznością przetwarzania danych oferowaną przez silniki relacyjne. Jako że twórca bazy wiedzy nie jest w stanie przewidzieć wszystkich przyszłych możliwości jej wykorzystania, użytkownik dysponuje zaawansowanym aparatem pozwalającym na dostosowanie wybranych fragmentów bazy wiedzy do wymaganej postaci. Aparat ten obejmuje operatory algebry konglomeratów [2], czyli zestaw działań na fragmentach bazy wiedzy wzorowany na zestawie operacji algebry relacyjnej Codda [6].

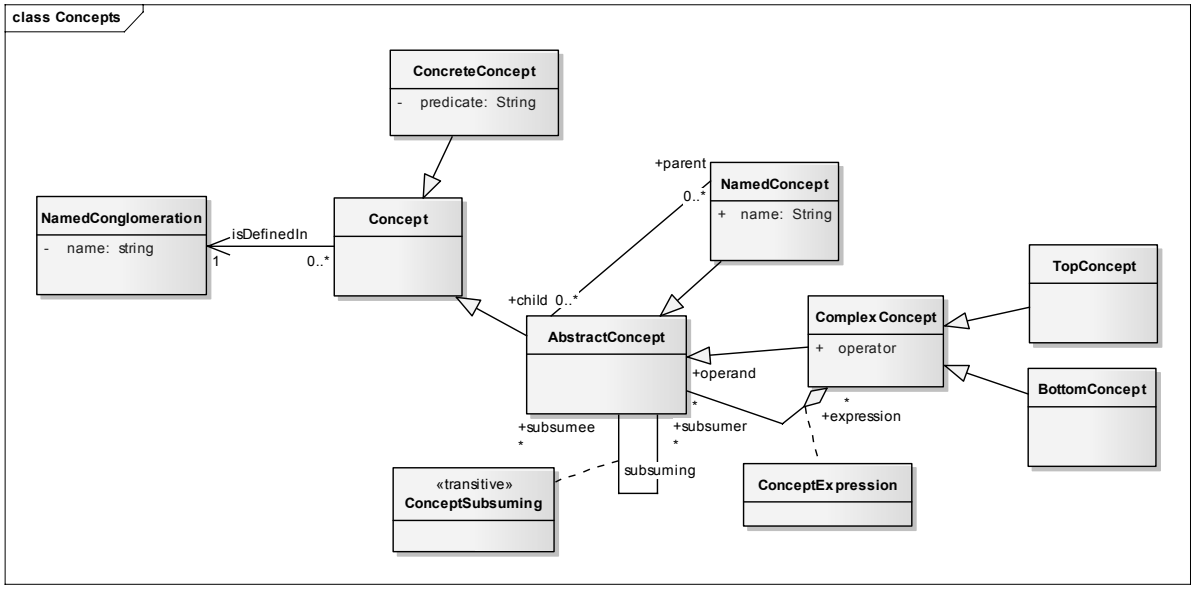
W koncepcji tej możliwe jest tworzenie konglomeratów i wykonywania na tych konglomeratach operacji algebry konglomeratów. W każdym konglomeracie mogą być zdefiniowane aksjomaty i asercje. System RKaSeA obsługuje również reguły. Ekspresywność reguł jest analogiczna do ekspresywności reguł w języku SWRL [7], ale ograniczona do pewnego podzbioru predykatów. Reguły nie są jednak elementem języka, który mógłby zostać podporządkowany konkretnemu konglomeratowi; reguły istnieją ponad podziałem na konglomeraty.

Na rysunku 1 przedstawiono powiązania między konglomeratami i regułami. Model nie opisuje możliwości budowania konglomeratów z wykorzystaniem algebry konglomeratów. W modelu przyjęto, że konglomeraty są tworzone jako niezależne obiekty i sposób ich tworzenia nie został odzwierciedlony w modelu.



Rys. 1. Powiązania między konglomeratami, regułami i terminami
Fig. 1. Relationships between conglomerations, rules and terms

Rysunek 2 przedstawia powiązania między konceptami w ramach terminologii konglomeratu. Klasa `Concept` reprezentuje dowolny koncept możliwy do wyrażenia w ramach konglomeratu. Dziedziczą od niej klasa `ConcreteConcept`, reprezentująca dziedziny i poddziedziny konkretne, oraz klasa `AbstractConcept` reprezentująca koncepty dziedziny abstrakcyjnej. W przypadku konceptów dziedziny abstrakcyjnej diagram opisuje możliwość tworzenia konceptów atomowych (klasa `NamedConcept`) oraz konceptów złożonych (`ComplexConcept`). Koncepty złożone są tworzone przy użyciu operatorów `and`, `or` i `not`. W przypadku konceptów dziedzin konkretnych tworzenie konceptów złożonych nie zostało zilustrowane, gdyż zależności między konceptami są tutaj wyrażane za pomocą predykatów.



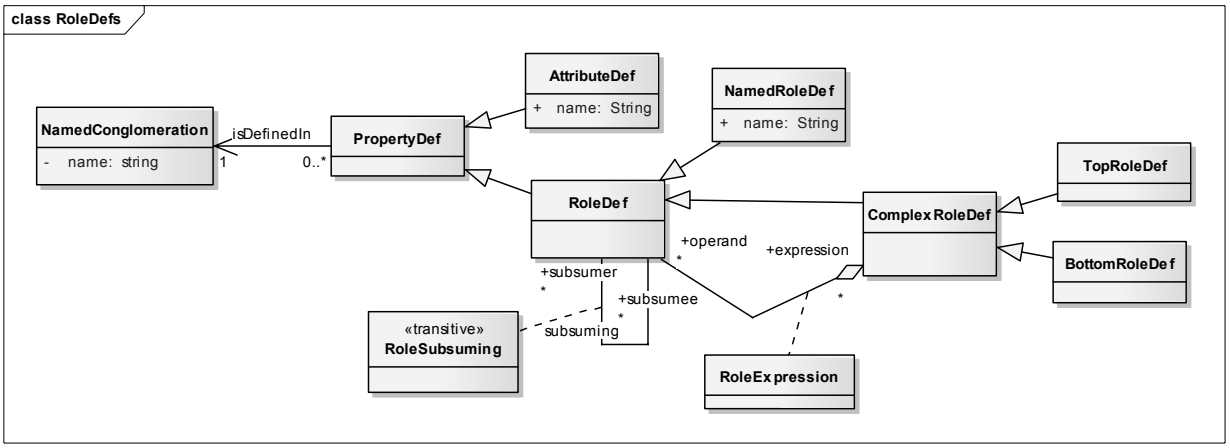
Rys. 2. Koncepty i powiązania między nimi
Fig. 2. Concepts and relationships between them

Koncepty `TOP` i `BOTTOM` zostały zaliczone do złożonych, ponieważ najczęściej są wyrażane jako:

```
TOP ≡ A OR ¬ A
BOTTOM ≡ A AND ¬ A,
```

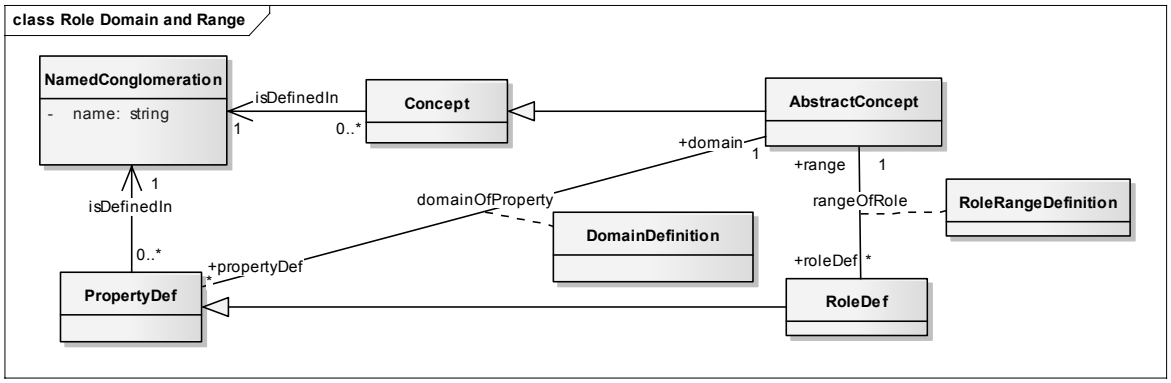
gdzie `A` oznacza dowolny koncept atomowy.

Na rysunku 3 przedstawione są powiązania między rolami. System RKaSeA obsługuje ontologie opisane językiem o ekspresywności $\mathcal{SROIQ}(\mathcal{D})$, co oznacza, że możliwe jest definiowanie dziedziczenia ról oraz ról dowolnie złożonych. Diagram ten w niewielkim stopniu różni się od przedstawionego na rysunku 2, dotyczącego konceptów.



Rys. 3. Role i powiązania między nimi
Fig. 3. Roles and relationships between them

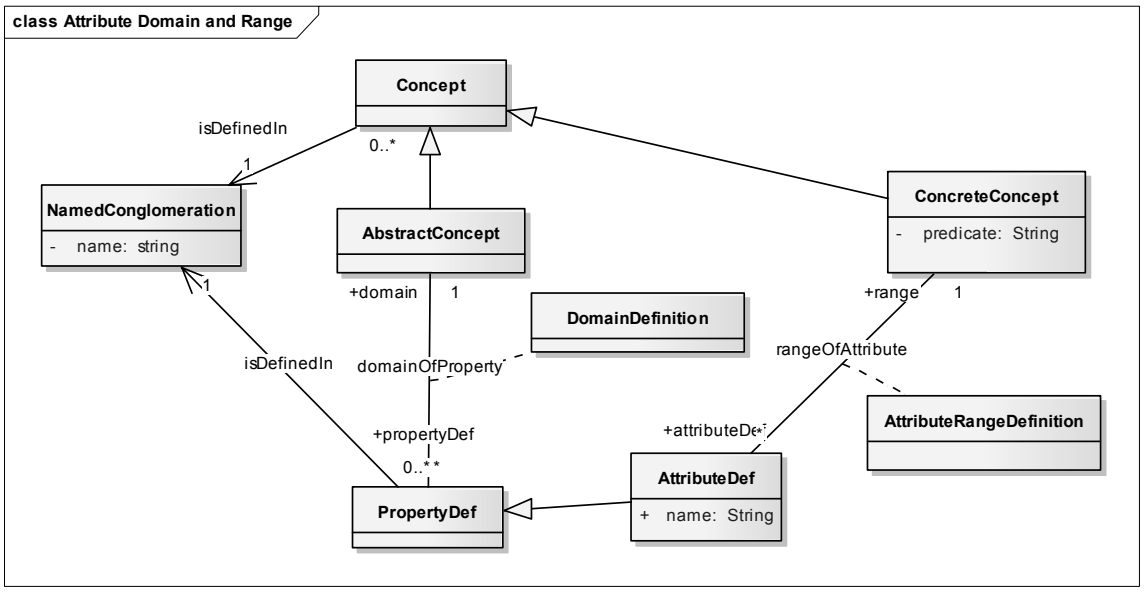
Zostały wyróżnione dwie klasy dotyczące właściwości: właściwości typu *DataProperty* (atrybuty) [8], reprezentowane za pomocą klasy `AttributeDef`, i właściwości typu *ObjectProperty* (role) [8], reprezentowane za pomocą klasy `RoleDef`.



Rys. 4. Zakres i dziedzina roli
Fig. 4. Domain and range of a role

Rysunki 4 i 5 ukazują powiązania między definicjami ról i atrybutów a konceptami stanowiącymi ich dziedzinę i zakres. W modelu przyjęto termin *RoleDef* (skrót od ang. *role defini-*

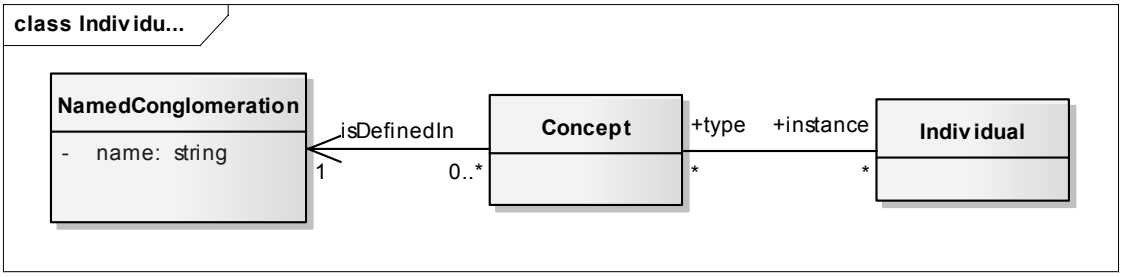
tion), aby odróżnić klasę reprezentującą rolę, która jest częścią terminologii, od wystąpienia roli rozumianej jako para dwóch osobników w opisie świata.



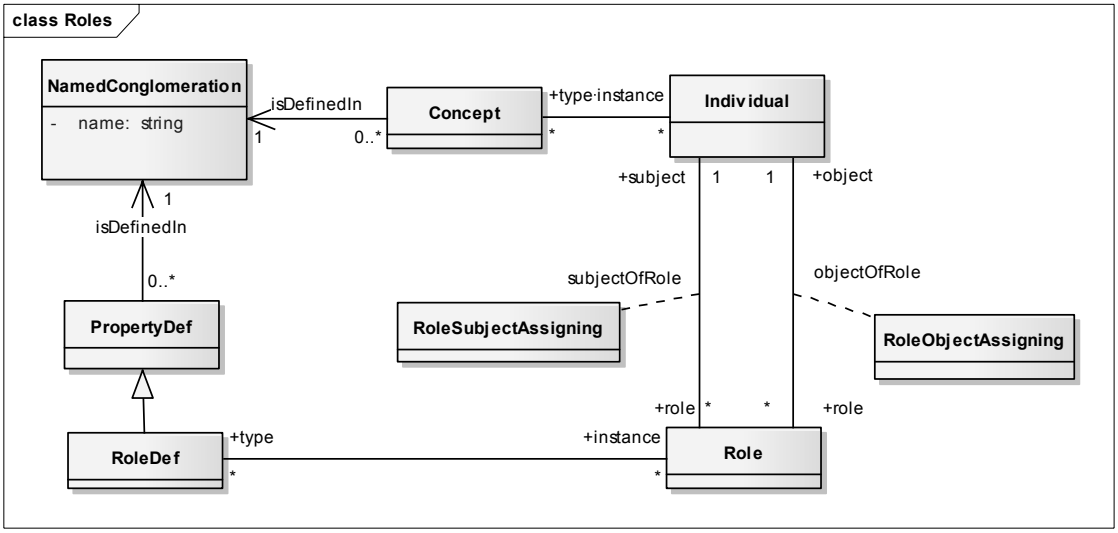
Rys. 5. Zakres i dziedzina atrybutu
Fig. 5. Domain and range of an attribute

Definicja roli tym różni się od definicji atrybutu, że w pierwszym przypadku zakres jest definiowany jako koncept w dziedzinie abstrakcyjnej, podczas gdy w drugim przypadku – jako podzbiór dziedziny konkretnej. Dlatego istnieją dwie oddzielne asocjacje `rangeOfRole` i `rangeOfAttribute`. W przypadku dziedziny roli i atrybutu takiej rozbieżności nie ma – w obu przypadkach jest nią koncept z dziedziny abstrakcyjnej. Dlatego asocjacja `domainOfProperty` jest zdefiniowana dla wspólnego przodka klas reprezentujących definicję roli i definicję atrybutu: `PropertyDef`.

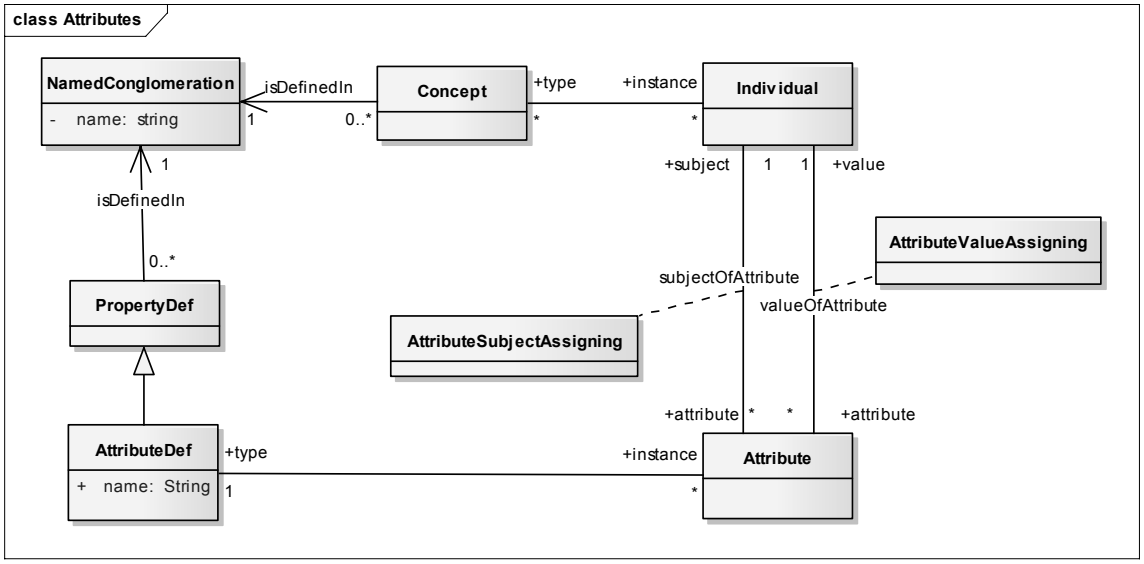
Następne trzy rysunki, o numerach 6, 7 i 8, ilustrują zależności wewnątrz opisu świata. Osobniki są przypisane do konglomeratów. Ten sam osobnik w ramach różnych konglomeratów jest identyfikowany tą samą nazwą (*Unique Name Assumption*). Może być tak, że danemu osobnikowi w różnych konglomeratach są przypisane różne koncepty. Analogicznie do osobników traktowane są wystąpienia ról i atrybutów.



Rys. 6. Osobniki
Fig. 6. Individuals



Rys. 7. Wystąpienia ról
Fig. 7. Roles instances



Rys. 8. Wystąpienia atrybutów
Fig. 8. Attributes instances

3. Metaontologia

Metaontologia jest formalnym zapisem modelu. Część terminologiczna odpowiada jednoznacznie obiektom zdefiniowanym w modelu i związkom między nimi. Część terminologiczna metaontologii została zdefiniowana w języku KQL i jest dostępna pod adresem [9]. Metaontologia została zaprojektowana jako jeden konglomerat. Każda klasa odpowiada konceptowi, właściwość atrybutowi, zaś związek roli. Liczebności związków zostały zaprojektowane jako ograniczenia liczebnościowe. Ponadto dla opisu świata przyjęto następujące założenia:

- każdy konglomerat, reguła, koncept, rola, atrybut i osobnik jest reifikowany do osobników w metaontologii, które są wystąpieniami odpowiednich konceptów;
- opis świata jest nieskończony, ponieważ każdy koncept i rola generują nieskończoną liczbę osobników: każdemu konceptowi i regule złożonej odpowiada jeden osobnik, a każdy koncept i reguła mogą być zdefiniowane na nieskończenie wiele sposobów (np.: `A`, `A OR A`, `A OR A OR … A`).

4. Realizacja zapytań metaontologicznych w systemie RKaSeA

Obsługa metaontologii została zaimplementowana w prototypowym systemie wnioskującym RKaSeA. System ten oferuje możliwości obsługi konglomeratowych baz wiedzy oraz przetwarzania ich za pomocą rozkazów KQL.

System RKaSeA oferuje możliwość wykorzystania dwóch rodzajów widoków systemowych (podobnych do widoków systemowych V_* w systemie Oracle), które przez użytkownika postrzegane są jako konglomeraty o terminologii zgodnej z prezentowaną metaontologią. Pierwszy rodzaj widoku to *metaschemat*, otrzymywany jest za pomocą wywołania funkcji `Meta` z nazwą schematu w parametrze. Ten rodzaj widoku pozwala użytkownikowi na określenie listy nazwanych konglomeratów w ramach schematu, wyznaczenie konglomeratów pomocniczych i tymczasowych oraz zależności pomiędzy tymi konglomeratami. Drugi rodzaj widoku to *metakonglomerat* otrzymywany za pomocą wywołania funkcji `Meta` z definicją konglomeratu w parametrze. Ten rodzaj widoku pozwala użytkownikowi na określenie zależności terminologicznych w ramach konglomeratu. Oba rodzaje konglomeratów są jedynie rodzajem interfejsu nałożonym na wewnętrzne struktury systemu i nie mają stałej reprezentacji w RKaSei, a faktyczne informacje są generowane na etapie materializacji metaschematu lub metakonglomeratu.

Metaschemat nie przenosi informacji na temat powiązań pomiędzy konglomeratami nazwanymi, gdyż w systemie przyjęto, że nawet jeżeli wstępna zawartość konglomeratu nazwanego jest generowana za pomocą wyrażenia algebry konglomeratów z innych konglomeratów, to i tak nowo utworzony konglomerat uzyskuje pełną autonomię (np. może być aktualizowany niezależnie) w stosunku do pozostałych.

Metakonglomeraty wykorzystują część metaontologii opisującą model konglomeratu. Stanowią one zapis zależności terminologicznych pomiędzy konceptami, rolami i atrybutami zdefiniowanymi w konglomeracie. Metakonglomerat stanowi *de facto* interfejs do wewnętrznej reprezentacji konglomeratu utrzymywanej w postaci rozszerzonego drzewa konceptów i ról. Drzewa te przechowują pełny zakres informacji terminologicznej pokrywany przez me-

taontologię (zależności podrzędności pomiędzy konceptami i rolami oraz informacje o dziedzinie i zakresie ról).

Do obsługi zależności pomiędzy terminologiczną i asercyjną częścią konglomeratu (koncepty `Individual`, `PairOfIndividual`, `PairIndividualValue`, role `hasIndividual`, `isInstanceOf`) wykorzystywane są mechanizmy materializacji konglomeratu podstawowego. RKaSeA próbuje tu dokonać ograniczenia liczby materializowanych konceptów, ale w skrajnych przypadkach (np. w odpowiedzi na zapytanie o typy osobnika) materializacja może objąć cały konglomerat.

Przyjęty sposób realizacji zapytań metaontologicznych zakłada, że użytkownik chcący określić zależność dotyczącą konceptu złożonego (np. zbadać problem podrzędności dwóch konceptów złożonych) powinien najpierw nazwać taki koncept nazwą atomową (patrz rozdział 5). Założenie to nie jest jednak bardzo ograniczające, gdyż analogiczne czynności należy zastosować również w zapytaniach nie korzystających z metaontologii (np. w celu uzyskania zbioru osobników będących wystąpieniami konceptu złożonego).

Należy zwrócić również szczególną uwagę na fakt, że realizacja zapytań terminologicznych wymaga rozpoznania, czy dane zapytanie zwraca skończony zbiór osobników. Dlatego w zapytaniach należy posługiwać się konceptami, których zbiór wystąpień jest zawsze skończony (np.: *NamedRole*, *NamedConcept*). W przypadku zapytań, w których skończony zbiór wystąpień nie może być wyznaczony, system RKaSeA zwraca odpowiedni komunikat błędu.

5. Realizacja zapytań terminologicznych w języku KQL

Głównym rozkazem manipulowania na konglomeratach jest rozkaz `SELECT`. Zapytanie `SELECT` ma następującą konstrukcję[1]:

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

W odniesieniu do algebry konglomeratów fraza `SELECT` odpowiada projekcji – wyborze terminów dla nowo tworzonego konglomeratu, fraza `ADD` odpowiada selekcji – ograniczeniu zbioru dopuszczalnych interpretacji, fraza `WHERE` i fraza `HAVING` odpowiada projekcji w odniesieniu do nazw osobników. Różnica pomiędzy frazą `WHERE` i frazą `HAVING` polega na tym, że fraza `WHERE` wybiera te osobniki, które należą do zadanego konceptu z konglomeratu oryginalnego (określonego we frazie `FROM`), zaś fraza `HAVING` wybiera te osobniki, które należą

[1] W celu zwiększenia czytelności w artykule zastosowano składnię wzorowaną na składni SQL, nie zaś składnię XML, którą posiada aktualna wersja języka KQL.

do zadanego konceptu z konglomeratu docelowego. Koncepcyjnie zapytanie wykonywane jest w następujących krokach: (1) Wyznaczenie konglomeratu podstawowego na podstawie zawartości klauzuli FROM. (2) Ograniczenie nazw osobników na podstawie zawartości klauzuli WHERE. (3) Rozszerzenie alfabetu o nowe koncepty/role/atrybuty/osobniki występujące w zdaniach zawartych w klauzuli ADD. (4) „Dodanie" do konglomeratu zdań z klauzuli ADD. (5) Projekcja alfabetu tylko do konceptów/ról/atrybutów zawartych w klauzuli SELECT. (6) Ograniczenie nazw osobników na podstawie zawartości klauzuli HAVING.

Do wykonania zapytań terminologicznych służy właśnie zapytanie SELECT, wykonywane na metakonglomeracie lub metaschemacie. Zapytanie to w odniesieniu do metaontologii ma jednak swoje ograniczenia. Przede wszystkim przyjmujemy, że zarówno opis świata, jak i terminologia dla metaontologii jest ustalona, choć nieskończona. Dlatego przyjęto, że w zapytaniach do metaontologii nie może wystąpić fraza ADD i fraza HAVING. Poniżej pokazano podstawowe zapytania terminologiczne w odniesieniu do interfejsu DIG 1.1.

Pierwszym z omówionych zapytań jest zapytanie o podrzędność dwóch konceptów. Przyjmijmy, że w konglomeracie *Test* są zdefiniowane koncepty *A* i *B*. Pytamy, czy koncept *A* dziedziczy od konceptu *B* (w interfejsie DIG zapytanie to odpowiada zapytaniu *AskSubsumesSubquery*).

```
SELECT CONCEPTS NamedConcept, ROLES subsumer
FROM META (CONGLOMERATION Test)
WHERE EXIST name.{"A","B"}
```

W systemie RKaSeA przyjęto, że nazwy osobników są tożsame z wartością atrybutu *name*, dlatego możliwa jest również prostsza wersja zapytania, w którym we frazie WHERE podajemy bezpośrednio koncept wyliczany $\{A, B\}$. Wynikiem zapytania będzie konglomerat ze zdefiniowanym jednym konceptem i jedną rolą. Osobniki o nazwach *A* i *B* będą wystąpieniami konceptu *NamedConcept* i zostaną w przypadku zaistnienia relacji podrzędności odpowiednio powiązane rolą *subsumer*. W przeciwnym przypadku w konglomeracie wynikowym rola *subsumer* nie będzie miała żadnego wystąpienia. W przedstawionym przykładzie koncept *A* i koncept *B* są konceptami nazwanymi. Tak jak zostało wspomniane w sekcji poprzedniej, w przypadku konceptów nienazwanych należy stworzyć konglomerat, w którym konceptom złożonym nadajemy nazwy. Przykładowo, przyjmijmy, że w konglomeracie *Test* dodatkowo zdefiniowaliśmy jeszcze jeden koncept nazwany *C*. Pytamy, czy koncept złożony będący przecięciem konceptów *A* i *B* dziedziczy od konceptu *C*.

```
SELECT CONCEPTS NamedConcept, ROLES subsumer
FROM META (
CREATE CONGLOMERATION FROM(
    SELECT CONCEPTS C, D
    ADD (D EQUALS A AND B)
    FROM CONGLOMERATION Test)
)
WHERE {C, D}
```

W zapytaniu tym utworzyliśmy nowy konglomerat, w którym konceptowi złożonemu będącemu przecięciem konceptów *A* i *B* nadaliśmy nową nazwę *D*. Zastosowanie takiego rozwiązania umożliwia łatwe sprawdzenie dziedziczenia konceptów w metakonglomeracie. Zgodnie z przyjętym założeniem, że każdy koncept jest reifikowany do pewnego osobnika w metakonglomeracie, możliwe jest wykonanie powyższego zapytania również bez tworzenia nowego konglomeratu.

```
SELECT CONCEPTS NamedConcept, ComplexConcept, ROLES subsumer
FROM META (CONGLOMERATION Test)
WHERE ((EXIST operator.{”AND”}) AND EXIST operand.{A} AND EXIST operand.{B} AND
=2operand.TOP) OR {C}
```

Stosując takie rozwiązanie, nie znamy jednak nazwy konceptu złożonego, a jest on definiowany jako wystąpienie konceptu *ComplexConcept*, który ma zdefiniowany operator przecięcia konceptów, a operandami są koncepty *A* i *B*. Ponadto osobnik reprezentujący ten koncept złożony ma dokładnie dwa operandy.

Poniżej pokazano jeszcze kilka przykładowych zapytań odpowiadających zapytaniom terminologicznym zdefiniowanym w interfejsie DIG 1.1 (pełna lista zapytań jest dostępna pod adresem [10]):

Zapytanie o typy osobnika *I* (*AskTypesSubquery*)

```
SELECT CONCEPTS NamedConcept
FROM CONGLOMERATION Test
WHERE EXIST instance.{I};
```

Zapytanie o koncepty bezpośrednio dziedziczące od konceptu *A* (*AskChildrenSubquery*)

```
SELECT CONCEPTS NamedConcepts
FROM META (CONGLOMERATION Test)
WHERE (EXIST (INVERSE child).{A})
```

W metaontologii dla prostoty wyrażenia zapytań *AskChildrenSubquery* i *AskParentSubquery* zostały wprowadzone role *parent* i *child*. Możliwe jest wykonanie tych zapytań również używając roli *subsumer*, ale zapytanie jest wtedy nieco bardziej skomplikowane:

```
SELECT CONCEPTS NamedConcepts
FROM META (CONGLOMERATION Test)
WHERE (EXIST subsumer.{A}) AND NOT (EXIST subsumer.(NOT {A} AND EXIST
subsumer.{A})) AND NOT {A}
```

Zapytanie to wybiera wszystkie osobniki odpowiadające konceptom nazwanym, które dziedziczą od konceptu *A* (i niebędące *A*), dla których nie istnieje taki koncept nadrzędny, który dziedziczy od *A*, ale nie jest *A*. Pozostałe zapytania mogą być wyrażone w bardzo podobny sposób.

6. Podsumowanie

W artykule przedstawiono metamodel ontologii dla systemu zarządzania wiedzą RKaSeA, na którym bazuje język KQL, a który został wykorzystany do wykonywania zapytań terminologicznych. W literaturze można znaleźć również inne specyfikacje metamodelu ontologii, m.in. *Networked Ontology Metamodel* [11], zaprojektowany w ramach projektu NeOn, czy opublikowany w ostatnim roku przez grupę OMG *Ontology Defininition Metamodel* [12]. Główną cechą tych modeli jest jednak niezależność modelu od jego zastosowania. *Networked Ontology Metamodel* definiuje metamodel nie tylko dla ontologii OWL, ale również dla reguł, odwzorowań ontologii i ontologii zmodularyzowanych. Odpowiednie części tego metamodelu mogą być wykorzystane w różnych zastosowaniach. Podobnie jest w przypadku specyfikacji OMG, w której metamodel jest definiowany dla różnych formalizmów zapisu ontologii (RDF, OWL, Topic Maps, …). Przedstawiony w artykule metamodel ma oczywiście pewną część wspólną z innymi modelami. Dotyczy to w szczególności konceptów, własności, osobników i powiązań między nimi (które we wszystkich modelach bazują na standardzie języka OWL DL). Metamodel ten był budowany jednak w celu przekształcenia go na metaontologię: wygodne narzędzie do zadawania zapytań terminologicznych. Stąd też pewne uproszczenia i dodatkowe role służące wygodzie i prostocie użytkowania. Ponadto przedstawiony metamodel zakłada modularyzację ontologii opartą na konglomeratach, która nie jest reprezentowana w innych metamodelach.

Zaprezentowany metamodel jest oczywiście tylko jednym z wielu możliwych widoków na ontologię i może być w przyszłości rozszerzany w celu zwiększenia zakresu możliwych zapytań. Jednak ze względu na podstawowe założenie, jakim jest prostota wykonywania zapytań przez inżyniera ontologii, rozszerzenia te będą dodawane w miarę potrzeb.

BIBLIOGRAFIA

1. Baader F. A., McGuiness D. L., Nardi D., Patel-Schneider P. F. (red.): The Description Logic Handbook: Theory, implementation, and applications. Cambridge University Press, 2003.
2. Goczyła K., Waloszek A., Waloszek W.: Algebra konglomeratów jako narzędzie opisu problemów przetwarzania ontologii. Studia Informatica, Vol. 30, No. 2A (83), Silesian University of Technology Press, Gliwice, 2009, s. 141÷156.

3. Goczyła K., Waloszek A., Waloszek W.: A Semantic Algebra for Modularized Description Logics Knowledge Bases. Proc. of the 22nd International Workshop on Description Logics DL'2009, Oxford, UK, July 27-30, 2009. Eds.: B. C. Grau, I. Horrocks, B. Motik, U. Sattler, 2009, s. 1÷12.
4. Goczyła K., Piotrowski P., Waloszek A., Waloszek W., Zawadzka T.: Specyfikacja języka KQL XML, raport projektu „Metody zarządzania ontologiami i regułami w bazach wiedzy zgodnych z Semantic Web". http://knot221.eti.pg.gda.pl/kmg/KQLSpecyfikacja .
5. Bechhofer S.: The DIG Description Logic Interface: DIG/1.1. University of Manchester, 2003.
6. Codd E. F.: Relational Completeness of Data Base Sublanguages. Database Systems, Vol. 6, 1979, s. 65÷98.
7. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004. http://www.w3.org/Submission/SWRL/ .
8. B. Motik, P. F. Patel-Schneider, B. Parsia, eds.: Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation, 27 October 2009. http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/ .
9. Goczyła K., Piotrowski P., Waloszek A., Waloszek W., Zawadzka T.: Definicja metaontologii w języku KQL. http://knot221.eti.pg.gda.pl/kmg/KQLMetaontology.kql .
10. Goczyła K., Piotrowski P., Waloszek A., Waloszek W., Zawadzka T.: Realizacja zapytań terminologicznych w języku KQL. http://knot221.eti.pg.gda.pl/kmg/KQLTerminologicalQueries.kql .
11. D1.1.1 Networked Ontology Model. NEON project deliverable. Raport projektu NEON, 2006. http://www.neon-project.org .
12. Ontology Definition Metamodel (ODM) Version 1.0. http://www.omg.org/spec/ODM/1.0/PDF .

Recenzenci: Dr hab. Tadeusz Pankowski, prof. Uniwersytetu im. Adama Mickiewicza
Prof. dr hab. inż. Andrzej Świerniak

Wpłynęło do Redakcji 30 stycznia 2010 r.

Abstract

Development of the Internet, along with rapid development of semantic knowledge management resulted in development of knowledge bases. These systems incorporate

different ways of knowledge representation. Among these ways the ones combining rules with ontologies expressed in description logics are becoming more and more popular. Such a representation is also taken as a basis for RKaSeA system—a knowledge management system built at Gdańsk University of Technology. During the work on this system it has been noticed that the main drawback of this representation is the lack of support for ontology modularization. Therefore the representation was extended with conglomerates, which are semantic parts (modules) of the whole ontology.

The paper presents the knowledge metamodel developed for RKaSeA system. The metamodel, hereafter referred to as "model", is a semantic basis for KQL language (acronym for *Knowledge Query Language*), an access language for RKaSeA. The main feature of KQL is its closure (the KQL operates on and returns conglomerates). To preserve this feature, the creators of KQL assumed that terminological statements will be designed as statements against the metaontology, which is a formal notation of the model.

The paper describes:

- knowledge model for RKaSeA, limited to the elements taken into consideration from perspective of language for communication with the system,
- the basic KQL commands used to provide a formal definition of a metaontology,
- the way of defining terminological statements in relation to the DIG interface,
- the internal implementation of terminological queries in RKaSeA,
- a brief examples of using terminological queries in KQL.

Adresy

Krzysztof GOCZYŁA: Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki, Katedra Inżynierii Oprogramowania, ul. G. Narutowicza 11/12, 80-233 Gdańsk, Polska, kris@eti.pg.gda.pl .

Aleksander WALOSZEK: Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki, Katedra Inżynierii Oprogramowania, ul. G. Narutowicza 11/12, 80-233 Gdańsk, Polska, alwal@eti.pg.gda.pl .

Wojciech WALOSZEK: Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki, Katedra Inżynierii Oprogramowania, ul. G. Narutowicza 11/12, 80-233 Gdańsk, Polska, piopio@eti.pg.gda.pl .

Teresa ZAWADZKA: Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki, Katedra Inżynierii Oprogramowania, ul. G. Narutowicza 11/12, 80-233 Gdańsk, Polska, tegra@eti.pg.gda.pl .