STUDIA INFORMATICA 2010
Volume 31 Number 2A (89)

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

JĘZYK KQL JAKO REALIZACJA IDEI JĘZYKA SQL DLA BAZY WIEDZY

Streszczenie. W ramach inicjatywy *Semantic Web* rozwijane są systemy wnioskowania z wiedzy. Ciągle otwartym problemem są również języki dostępu do takich systemów. W artykule zaproponowano nowy język dostępu do zmodularyzowanych baz wiedzy, o cechach umożliwiających jego kompleksowe wykorzystanie w systemach zarządzania wiedzą, w sposób analogiczny do wykorzystania języka SQL w systemach relacyjnych baz danych.

Słowa kluczowe: bazy wiedzy, język zapytań, *Semantic Web*, system wnioskujący, wiedza

KQL AS AN APPLICATION OF SQL RATIONALE FOR KNOWLEDGE BASES

Summary. As a part of the *Semantic Web* initiative there are being developed knowledge inference systems. Moreover, the languages for accessing such systems are also open issues. The paper proposes a new language for accessing modularized knowledge bases, whose features allows for its comprehensive usage in knowledge management systems in an analogous way as SQL language in relational database systems.

Keywords: knowledge bases, query language, Semantic Web, inference system, knowledge

1. Wstęp

Powstała około roku 2000 inicjatywa *Semantic Web* wprowadza standardy pozwalające na semantyczne opisanie zasobów internetowych w sposób, który umożliwia łatwe odszukiwanie i inteligentne kojarzenie informacji przez programy komputerowe. Inicjatywa dowiod-

ła poprawności obranej drogi, jednak pozwoliła też odkryć jej słabości. Największą wadą powstających w Internecie ontologii jest wysoki poziom ich złożoności. Wysoka złożoność z kolei pociąga za sobą wiele negatywnych skutków. Pierwszą ich grupą jest obniżenie wydajności programów wnioskujących i innych systemów korzystających z ontologii. Drugą jest utrudnienie prac związanych z wytwarzaniem i korzystaniem z tego typu systemów. Aby usunąć te wady, potrzebna jest nowa inżynieria wiedzy, która pod względem dojrzałości i możliwości praktycznego wykorzystania dorówna współczesnej inżynierii baz danych. Aby tak się stało, musi zostać zweryfikowany sposób tworzenia i posługiwania się ontologiami. Muszą w tym celu powstać nowe narzędzia i towarzyszące im nowe metody pozwalające zoptymalizować wydajność pracy zespołów ludzkich zaangażowanych w te przedsięwzięcia. Jednym z elementów tej nowej inżynierii musi być nowy język, pozwalający tworzyć, modyfikować oraz odpytywać ontologie. W niniejszym artykule opisany zostanie język KQL (*Knowledge Query Language*), opracowany w grupie KMG na Wydziale ETI Politechniki Gdańskiej.

W rozdziale 2 przestawiono motywacje, którymi kierują się autorzy w swojej pracy, w rozdziale 3 naszkicowano algebrę konglomeratów, będącą teoretyczną podstawą języka KQL, w rozdziale 4 przybliżono ideę języka oraz zademonstrowano jego właściwości, w rozdziale 5 przedstawiono narzędzie do tworzenia rozkazów KQL, w rozdziale 6 dokonano porównania języka KQL z innymi językami przeznaczonymi do operowania na ontologiach, zaś w rozdziale 7 przedstawiono krótkie studium przypadku użycia języka KQL. Artykuł podsumowano w rozdziale 8.

2. Motywacje

Główną ideą motywującą autorów w pracach, które są tematem niniejszego artykułu, jest udostępnienie w ramach modelu ontologicznego podobnych możliwości, jakie są dostępne w ramach modelu relacyjnego. Model relacyjny dysponuje narzędziami, które umożliwiają elastyczne operowanie relacjami i ich składnikami. Są to: algebra relacji Codda i oparty na niej język SQL [1]. Narzędzia te pozwalają w każdej chwili, korzystając z relacji trwale zapisanych w bazie danych, utworzyć nowe relacje, które przedstawią istniejące dane z nowego, dostosowanego do aktualnych potrzeb użytkownika, punktu widzenia. Operatory algebry relacji wykonują działania na nagłówkach (schematach) relacji oraz na zbiorach krotek. Daje to możliwość wpływania zarówno na elementy schematu bazy danych, jak i na zgromadzone w niej dane. Aby te zalety przenieść do modelu ontologicznego, potrzebne jest przede wszystkim poprawne przeprowadzenie analogii między obydwoma modelami.

W proponowanym rozwiązaniu ontologia jest traktowana jako analogia do nagłówka pojedynczej relacji, a nie całej bazy danych. To wymaga założenia, że baza wiedzy musi składać się z wielu ontologii, tak jak baza danych składa się z wielu relacji. Taki właśnie jest kierunek prac nad modularyzacją baz wiedzy [2, 3, 4] — w proponowanych rozwiązaniach moduł ontologii jest również ontologią. Również w takim kierunku idzie opracowana w grupie KMG na Wydziale ETI Politechniki Gdańskiej metoda modularyzacji ontologii [5, 6]. Kolejną przyjętą analogią jest potraktowanie zbioru modeli (w sensie logiki pierwszego rzędu) ontologii jako odpowiednika krotek. Tak więc działania na nagłówkach relacji i ich krotkach w obrębie modelu ontologicznego powinny mieć swój odpowiednik w działaniach na ontologiach i ich modelach.

Przedstawione powyżej rozumowanie pozwoliło wprowadzić pojęcie *konglomeratu*, który jest odpowiednikiem relacji bazodanowej. Konglomerat składa się ze słownika, którym jest sygnatura danej ontologii, oraz ze zbioru modeli tej ontologii. Dokładna definicja konglomeratu zostanie podana w następnym rozdziale. Tutaj warto jedynie zauważyć, że nie jest to twór syntaktyczny, gdyż nie zawsze zawartość zbioru modeli da się opisać za pomocą zdań. W analogii do relacyjnych baz danych, które są zbiorem relacji, baza wiedzy jest zbiorem konglomeratów. Konglomeraty w obrębie bazy wiedzy są powiązane ze sobą za pomocą operatorów algebry konglomeratów, tak jak w bazie danych relacje są powiązane kluczami obcymi.

Na podstawie powyższych analogii powstał język KQL, który jest odpowiednikiem języka SQL. Język KQL opiera się na algebrze konglomeratów i dzięki temu posiada cechę domknięt ości względem konglomeratów. Dzięki temu daje możliwość zagnieżdżania zapytań i pozwala na modyfikowanie baz wiedzy. Podobnie jak SQL, który ma rozkazy definiowania danych, manipulowania na danych i kontroli dostępu do danych, KQL jest również zaprojektowany w taki sposób, aby umożliwiać definiowanie wiedzy, manipulowanie wiedzą i kontrolę dostępu do wiedzy. Podstawowymi rozkazami w KQL-u są CREATE, DROP, SELECT, INSERT, DELETE, a zatem jasne jest ich przeznaczenie.

3. Algebra konglomeratów

Algebra konglomeratów jest formalizmem opisującym pojęcie konglomeratu i działania, jakie można wykonać na konglomeratach. Jak wspomniano powyżej, jedną z zasadniczych części konglomeratu jest słownik. Pojęcie słownika formalizujemy następująco: *sygnaturą* (*słownikiem*) **S** nazywamy sumę rozłączną zbiorów *nazw konceptów* (**C**), *nazw ról* (**R**) oraz *nazw osobników* (**I**), co zapisujemy $\mathbf{S} = \mathbf{C} \uplus \mathbf{R} \uplus \mathbf{I}$, gdzie $\uplus$ oznacza operację sumy rozłącz-

nej. Przez nazwy rozumiemy tu nazwy atomowe, zakładamy przy tym, że pochodzą one z pewnego ustalonego zbioru wszystkich możliwych nazw $\mathcal{N}$ (**C**, **R**, **I** $\subseteq \mathcal{N}$).

Interpretacją $\mathcal{I}$ nazywamy parę $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, gdzie $\Delta^{\mathcal{I}}$ jest niepustym zbiorem nazywanym *dziedziną* interpretacji, a $\cdot^{\mathcal{I}}$ jest funkcją przypisującą każdemu konceptowi atomowemu $A \in \mathbf{C}$ podzbiór dziedziny $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, każdej roli atomowej $R \in \mathbf{R}$ binarną relację $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, a każdemu osobnikowi $a \in \mathbf{I}$ element dziedziny $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

Przedstawione uwagi pozwalają nam na formalne zdefiniowane pojęcia konglomeratu (patrz także obszerniejsze informacje na ten temat w [5]):

Definicja 1. *Konglomeratem K* nazywamy parę (S, W) składającą się z sygnatury S (*sygnatury* konglomeratu) i zbioru W zawierającego interpretacje (*modele* konglomeratu). Kiedy chcemy się odwołać do jednego z elementów pary używamy notacji odpowiednio S(*M*) oraz W(*M*).

Na konglomeratach zdefiniowano zestaw działań wzorowany na algebrze relacyjnej Codda [7]:

- operacje rozszerzenia słownika (ε), projekcji (π) i przemianowania (ρ) operujące na słownikowej części konglomeratu,
- operacje na interpretacjach, czyli (σ), suma (∪), przecięcie (∩) i różnica (–) konglomeratów (działające na zbiorach modeli),
- operacja selekcji (σ) zawężająca zbiór modeli konglomeratu do tych, które spełniają zadane zdanie α.

Algebra konglomeratów daje bardzo dużą elastyczność w manipulowaniu ich zawartością. Można za jej pomocą np. wyekstrahować fragment konglomeratu i dołączyć go do fragmentu innego konglomeratu. Ekstrakcja fragmentu konglomeratu odbywa się za pomocą zawężającej słownik operacji projekcji. Dołączenia można dokonać za pomocą operacji przecięcia (∩). W przypadkach, w których następuje konflikt nazw, można wykorzystać operacje przemianowania (ρ) w sposób, który zapewni, że nazwy z poszczególnych konglomeratów nie powtórzą się. Bardziej zaawansowane zastosowania konglomeratów obejmują przechowywanie kilku wersji opisu świata w ramach jednego konglomeratu (co można osiągnąć dzięki użyciu operacji sumy).

Algebra konglomeratów przejawia szereg cech, które czynią ją ciekawym narzędziem opisu zależności między fragmentami ontologii. Pozwala ona na *sprawne* (w znaczeniu opisanym w [8]; ang. *robust*) wykonanie czynności wskazanych w [8] jako podstawowe: zmiany słownika modułu, rozszerzenia słownika i złączenia modułów, a w rzeczywistości stanowi naturalne teoriozbiorowe domknięcie minimalnego zestawu operacji pozwalającego na wykonanie tych działań. Ponadto można dowieść [9], że algebra konglomeratów jest wariantem bardziej ogólnej struktury matematycznej, zwanej algebrą cylindryczną [10]. To ostatnie

stwierdzenie jest bardzo ważne, gdyż na jego podstawie można stosować do algebry konglomeratów również prawa algebry relacyjnej [9].

4. Język KQL

Korzystając z wcześniej opisanej analogii między językiem SQL i KQL, zaproponowano zbiór wymagań, który powinien spełniać jednolity język dostępu do bazy wiedzy. Taki język powinien charakteryzować się: (1) dostępnością rozkazów tworzenia baz wiedzy, manipulowania bazami wiedzy i kontroli dostępu do baz wiedzy, (2) domkniętością (instrukcje odpytywania powinny zwracać te same konstrukcje, na których działają), (3) możliwością zagnieżdżania zapytań, (4) jednolitością języka modyfikacji i definiowania (wyniki zapytań mogą być użyte w zapytaniach operujących na bazie wiedzy i tworzących bazę wiedzy), (5) możliwością zadawania zapytań terminologicznych oraz (6) wsparciem dla modularyzacji ontologii.

Język KQL, w analogii do języka SQL, **udostępnia rozkazy tworzenia bazy wiedzy, manipulowania bazami wiedzy i kontroli dostępu do bazy wiedzy**. Rozkazy tworzenia bazy wiedzy obejmują rozkazy tworzenia: (1) baz wiedzy (ontologii), (2) konglomeratów będących modułami bazy wiedzy, (3) reguł oraz (4) odwzorowań do zewnętrznych źródeł danych.

W aktualnej implementacji ekspresywność języka definiowania konglomeratów odpowiada ekspresywności języka OWL DL w wersji 1.0. Ekspresywność języka definiowania reguł odpowiada ekspresywności języka SWRL przy ograniczonym zbiorze predykatów.

Głównym rozkazem manipulowania konglomeratami 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 do zadanego konceptu z konglomeratu docelowego. Koncepcyjnie zapytanie wykonywane jest

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

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.

W aktualnej wersji języka najmniej rozwinięte są rozkazy kontroli dostępu do wiedzy. Język jednak jest skonstruowany w sposób umożliwiający jego dalszy rozwój także w tym, niezmiernie ważnym, również z praktycznego punktu widzenia, kierunku.

Język KQL jest językiem **domkniętym**. Wykorzystanie algebry konglomeratów jako teoretycznej podstawy dla języka umożliwiło skonstruowanie języka w taki sposób, że operuje on na konglomeratach i zwraca konglomeraty. Wynikiem zapytań operujących na konglomeratach są również konglomeraty.

Bezpośrednią konsekwencją domkniętości języka KQL jest **możliwość zagnieżdżania zapytań**. Wszędzie tam, gdzie jest potrzeba użycia konglomeratu, możliwe jest wykorzystanie konglomeratów nazwanych i zdefiniowanych w bazie wiedzy lub konglomeratów utworzonych na życzenie, z wykorzystaniem możliwości języka KQL do tworzenia wyrażeń konglomeratowych z użyciem operatorów algebry konglomeratów (np.: INTERSECT, JOIN, UJOIN) lub rozkazu SELECT.

W konsekwencji domkniętości języka KQL i możliwości zagnieżdżania zapytań uzyskano **jednolitość języka modyfikacji i definiowania**. W ramach rozkazów tworzenia konglomeratów można wykorzystywać zagnieżdżanie zapytań:

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

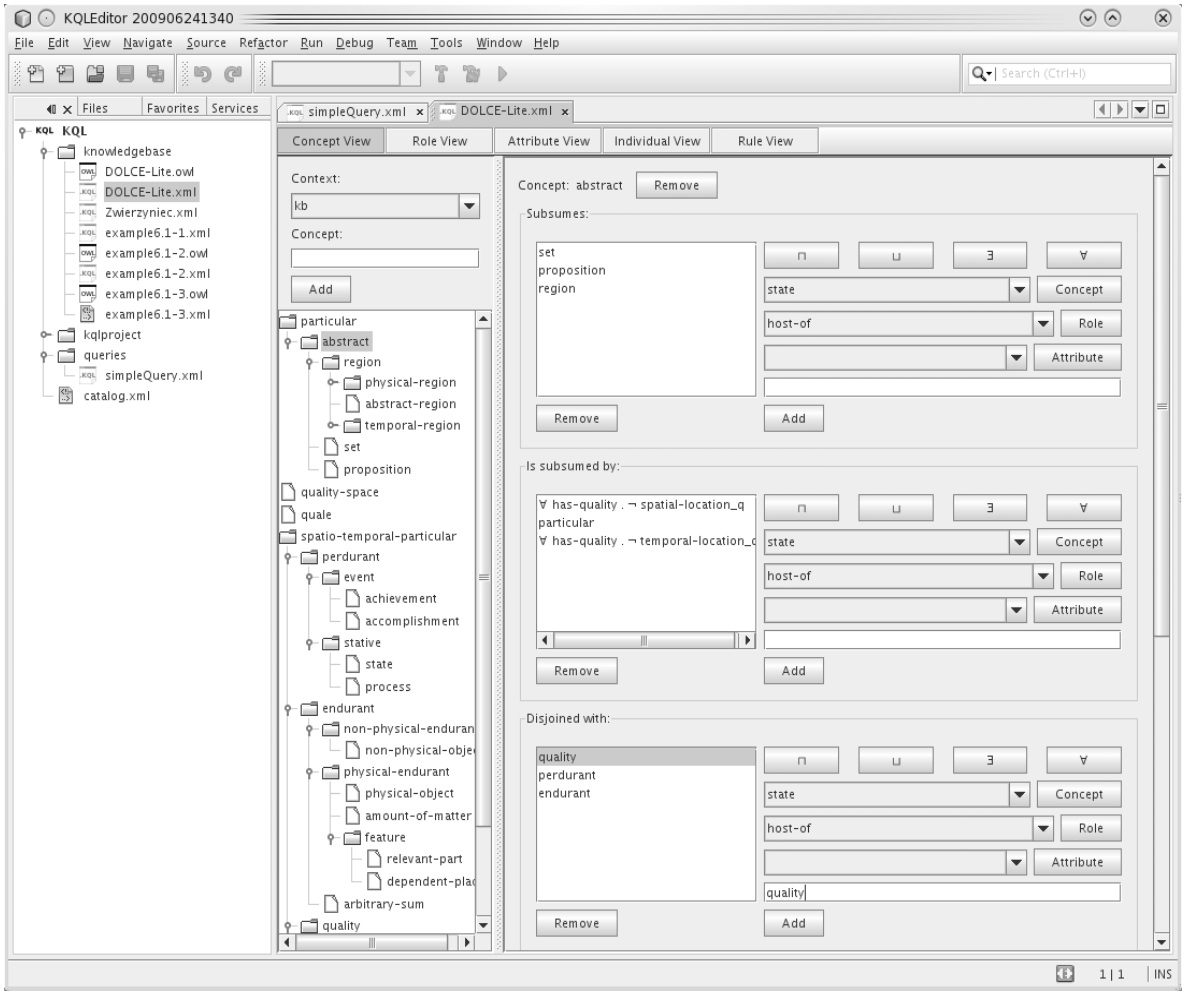
W języku KQL przyjęto, że **zapytania terminologiczne** są zadawane do metaontologii tworzonej dla każdej bazy wiedzy.

Metaontologia jest zbudowana na bazie metamodelu konglomeratowej bazy wiedzy i jest dostępna pod adresem *http://knot221.eti.pg.gda.pl:8080/RKaseaPortal-war/*. Dzięki zastosowaniu metaontologii możliwe jest zadawanie zapytań terminologicznych analogicznie do zapytań o osobniki, z tą różnicą, że zapytanie jest kierowane do metaontologii, a nie bezpośrednio do bazy wiedzy. Wynika to z faktu, że w metaontologii konglomeraty, koncepty, role, atrybuty i osobniki z bazy wiedzy są reifikowane do osobników, a zależności między nimi do odpowiednich relacji binarnych między osobnikami.

W przypadku języka KQL właściwie trudno mówić o wsparciu dla modularyzacji ontologii. Tak naprawdę to właśnie modularyzacja ontologii do postaci konglomeratów jest podstawą języka KQL, dlatego też modularyzacja ontologii jest podstawową cechą, na której bazuje język KQL.

5. Edytor języka KQL

Ze względu na różnice języka KQL w porównaniu z istniejącymi rozwiązaniami, niezbędne było stworzenie nowego edytora dedykowanego dla tego języka i kryjącego się za nim modelu. Edytor pozwala na modyfikowanie ontologii poprzez graficzny interfejs wizualizujący wybrane elementy ontologii oraz pozwala na modyfikowanie bezpośrednio kodu źródłowego ontologii. Pierwsza metoda edycji ułatwia skupienie się na konkretnych fragmentach ontologii, druga natomiast daje dostęp do pełnych możliwości języka. Podobnie jak w edytorze Protégé [11] ontologia widziana jest poprzez następujące widoki: widok konceptów, widok ról, widok atrybutów, widok osobników i widok reguł.



Rys. 1. Ekran widoku konceptów
Fig. 1. Concept view screen

W każdym widoku prezentowane są odpowiednie elementy ontologii, np. widok konceptów (patrz rys. 1) zawiera drzewo konceptów, listę konceptów, po których dziedziczy wybrany koncept, listę konceptów dziedziczących po wybranym koncepcie, listę konceptów rozłącznych z wybranym konceptem oraz listę instancji wybranego konceptu. Widoki ról i atrybutów zawierają między innymi dziedziny, zakresy itd.

6. Porównanie języka KQL z innymi językami

W celu porównania języków definiowania ontologii, zadawania zapytań, czy komunikowania się między agentami zdefiniowano następujące metryki:

1. Cel języka określający, jakie założenia stoją u podstaw danego języka, np.: OWL jest językiem definiowania ontologii, a nie językiem tworzenia bazy wiedzy. Jest to miara opisowa.
2. Typ języka określający, jakiego typu jest dany język: język tworzenia czy język manipulacji. Miara może przyjmować wartość *T* (tworzenie) lub *M* (manipulacja).
3. Domkniętość, czyli cechę, zgodnie z którą instrukcje odpytywania zwracają te same konstrukcje, na których działają. Miara przyjmuje wartość logiczną tak (*T*) albo nie (*N*).
4. Zagnieżdżanie zapytań: określa, czy odpowiedź na zapytanie może być argumentem innego zapytania. Miara przyjmuje wartość logiczną tak (*T*) albo nie (*N*).
5. Jednolitość języka modyfikacji i definiowania: określa, czy wyniki zapytania mogą być użyte w zapytaniu operującym na bazie wiedzy i tworzącym bazę wiedzy. Miara przyjmuje wartość logiczną tak (*T*) albo nie (*N*).
6. Ujęcie problemu zapytań terminologicznych będące miarą określającą możliwe sposoby ujęcia problemu zapytań terminologicznych: różne rodzaje zapytań (wartość *R*), zapytania działają jednolicie i jednocześnie z częścią asercjonalną (*Abox* i *Tbox* traktowane jako jeden homogeniczny obiekt) (wartość *J*), poprzez traktowanie terminologii jako opisu świata metaontologii (wartość *M*).
7. Wsparcie dla modularyzacji ontologii, czyli jaki typ wsparcia dla modularyzacji ontologii dostarcza dany język, np. OWL dostarcza mechanizmu importów. Jest to miara opisowa.

Tabela 1

Podstawowe założenia i cel utworzenia danego języka

Język	Cel
OWL	Definiowanie ontologii DL
SWRL	Definiowanie ontologii DL wzbogaconych o reguły Horna
SPARQL	Język zapytań wraz z interfejsem do repozytoriów RDF
DIG 1.1	Interfejs dostępu do bazy wiedzy
DIGUT	Interfejs dostępu do bazy wiedzy
DIG 2.0	Interfejs dostępu do bazy wiedzy
OWL-QL	Język ma być standardowym środkiem porozumiewania się agentów w sieci *Semantic Web*.
*n*RQL	Język zapytań dla programu RacerPro.
SAIQL	Język służący do ekstrakcji małych fragmentów z dużych ontologii OWL.
KQL	Jednolity język tworzenia, manipulacji i kontroli dostępu do wiedzy.

Dalej przedstawiono najbardziej popularny zbiór aktualnie dostępnych języków opisu ontologii: OWL [12], [13], SWRL [14], SPARQL [15], DIG 1.1 [16], DIGUT [17], DIG 2.0 [18], OWL-QL [19], *n*RQL [20] i SAIQL [21]. W tabeli 1 znajduje się opis podstawowych

celów utworzenia danego języka. W kolejnych tabelach porównano wartości miar dla poszczególnych ję□yków.

Tabela 2

Nieopisowe miary języków

Język	Typ języka	Domkniętość	Zagnieżdżanie zapytań	Jednolitość	Zapytania termininologiczne
OWL	*T*	*n/d*	*n/d*	*n/d*	*n/d*
SWRL	*T*	*n/d*	*n/d*	*n/d*	*n/d*
SPARQL	*M*	*T*	*N*	*n/d*	*J*
DIG 1.1	*T, M*	*N*	*N*	*N*	*R*
DIGUT	*T, M*	*N*	*N*	*N*	*R*
DIG 2.0	*T, M*	*N*	*N*	*N*	*R*
OWL-QL	*M*	*N*	*N*	*N*	*J*
nRQL	*T, M* (tylko *Abox*)	*N*	*N*	*N*	*M*
SAIQL	*M*	*T*	*N*	*N*	*J*
KQL	M	T	T	T	M

Tabela 3

Zależności między zapytaniami terminologicznymi i asercjonalnymi

Język	Zależność
OWL	*n/d*
SWRL	*n/d*
SPARQL	Terminologia traktowana jest jako zestaw trójek, zatem całość traktowana jest jako jeden Abox.
DIG 1.1	Po stronie aplikacji
DIGUT	Po stronie aplikacji
DIG 2.0	Po stronie aplikacji
OWL-QL	Brak podziału na zapytania terminologiczne i asercjonalne
nRQL	Brak możliwości zagnieżdżania
SAIQL	Brak podziału na zapytania terminologiczne i asercjonalne
KQL	W ramach list selekcji możliwe jest odpytanie metaontologii o listę konceptów, ról lub atrybutów spełniających pewne warunki.

Tabela 4

Wsparcie dla modularyzacji ontologii

Język	Modularyzacja
OWL	Importowanie ontologii
SWRL	Importowanie ontologii
SPARQL	Importowanie ontologii
DIG 1.1	Brak wsparcia dla modularyzacji ontologii
DIGUT	Wsparcie dla kontekstualizacji opartej na paradygmacie obiektowym
DIG 2.0	Brak wsparcia dla modularyzacji ontologii
OWL-QL	Brak wsparcia dla modularyzacji ontologii
*n*RQL	Brak wsparcia dla modularyzacji ontologii
SAIQL	Brak wsparcia dla modularyzacji ontologii
KQL	Wsparcie dla ontologii modułowych zbudowanych z konglomeratów

Tabela 2 zawiera porównanie miar nieopisowych, zaś kolejno, tabela 3 zawiera opis rozwiązania (jeśli istnieje) problemu wzajemnych zależności między zapytaniami terminologicznymi i asercjonalnymi w danym języku, zaś tabela 4 wsparcie dla modularyzacji ontologii.

7. Studium przypadku użycia języka KQL

Poniżej pokazano prosty przykład wykorzystania języka KQL do utworzenia ontologii składającej się z dwóch konglomeratów: konglomeratu Zwierzyniec i konglomeratu SystematykaSsaków. W konglomeracie Zwierzyniec zdefiniowana jest zarówno część terminologiczna (terminy 1.1 – 1.6, aksjomaty 2.1 – 2.11), jak i asercjonalna bazy wiedzy (terminy 1.7 – 1.9, asercje 2.12 – 2.16). Aksjomaty dzielą świat na ludzi i zwierzęta (2.1). Ludzie to dziewczęta i chłopcy (2.2, 2.3), a zwierzęta to psy i koty (2.4, 2.5). Człowiek może mieć zwierzę (2.6, 2.7). Ponadto wiadomo, że między kotami i psami trwają odwieczne walki: pies nie może lubić kota (2.8) ani właściciela kota (2.9), a kot nie może lubić psa (2.10) ani właściciela psa (2.11). Formalna definicja konglomeratu Zwierzyniec została pokazana poniżej. Rozkaz pierwszy definiuje słownik konglomeratu, zaś rozkaz drugi definiuje zbiór aksjomatów i asercji. Cześć asercjonalna określa, że Ala jest człowiekiem (2.12) i ma Maxa i Asa (2.13, 2.14). As jest psem (2.15) i Max lubi Asa (2.16).

```
(1) CREATE CONGLOMERATION Zwierzyniec
     ADD CONCEPT Człowiek                                          (1.1)
     ADD CONCEPT Dziewczyna                                        (1.2)
     ADD CONCEPT Chłopak                                           (1.3)
     ADD CONCEPT Zwierzę                                           (1.4)
     ADD CONCEPT Pies                                              (1.5)
     ADD CONCEPT Kot                                               (1.6)
     ADD INDIVIDUAL Ala                                            (1.7)
     ADD INDIVIDUAL As                                             (1.8)
     ADD INDIVIDUAL Max                                            (1.9)

(2) CONSTRAIN
     Top EQ Człowiek OR Zwierzę                                    (2.1)
     Człowiek EQ (Dziewczyna OR Chłopak)                           (2.2)
     Bottom EQ (Dziewczyna AND Chłopak)                            (2.3)
     Zwierzę EQ (Pies OR Kot)                                      (2.4)
     Bottom EQ (Pies AND Kot)                                      (2.5)
     (EXIST ma Top) ISSUB Człowiek                                 (2.6)
     Top ISSUB ALL ma Zwierzę                                      (2.7)
     Pies ISSUB (NOT EXIST lubi Kot)                               (2.8)
     Pies ISSUB (NOT EXIST lubi (EXIST ma Kot))                    (2.9)
     Kot ISSUB (NOT EXIST lubi Pies)                               (2.10)
     Kot ISSUB (NOT EXIST lubi (EXIST ma Pies))                    (2.11)
     Ala IS Człowiek                                               (2.12)
     As IS Pies                                                    (2.13)
     (Ala, As) IS ma                                               (2.14)
     (Ala, Max) IS ma                                              (2.15)
     (Max, As) IS lubi                                             (2.16)
```

Konglomerat SystematykaSsaków definiuje tylko część terminologiczną. W terminologii wyróżnione są dwie rodziny ssaków drapieżnych – psowate i kotowate (4.1, 4.2, 4.3). Ponadto wiadomo, że kotowate są pazurzaste (4.4), a psowate nie (4.5), psowate są kudłate (4.6), a kotowate nie (4.7), zaś zarówno kotowate jak i psowate są ogoniaste (4.8, 4.9). Definicja konglomeratu Systematyka Ssaków została pokazana poniżej.

```
(3) CREATE CONGLOMERATION SystematykaSsaków
     ADD CONCEPT SsakiDrapieżne                                    (3.1)
     ADD CONCEPT Psowate                                           (3.2)
     ADD CONCEPT Kotowate                                          (3.3)
     ADD CONCEPT Pazurzaste                                        (3.4)
     ADD CONCEPT Kudłate                                           (3.5)
     ADD CONCEPT Ogoniaste                                         (3.6)

(4) CONSTRAIN
     Top EQ SSakiDrapieżne                                         (4.1)
     SsakiDrapieżne EQ (Psowate OR Kotowate)                       (4.2)
     Bottom EQ (Psowate AND Kotowate)                              (4.3)
     Kotowate ISSUB Pazurzaste                                     (4.4)
     Psowate ISSUB (NOT Pazurzaste)                                (4.5)
     Psowate ISSUB Kudłate                                         (4.6)
     Kotowate ISSUB (NOT Kudłate)                                  (4.7)
     Psowate ISSUB Ogoniaste                                       (4.8)
     Kotowate ISSUB Ogoniaste                                      (4.9)
```

Interesuje nas, czy Max jest ogoniasty, kudłaty, czy pazurzasty. Zgodnie z zasadą domkniętości języka KQL należy stworzyć konglomerat składający się z trzech konceptów Pazurzaste, Kudłate, Ogoniaste i jednego osobnika Max. W tym celu należy wykonać zapytanie:

```
SELECT Ogoniaste, Kudłate, Pazurzaste
ADD Psy ISSUB Psowate
ADD Koty ISSUB Kotowate
FROM SystematykaSsaków INTERSECT Zwierzyniec
WHERE {Max}
```

W zapytaniu konglomerat podstawowy wyznaczany jest jako przecięcie konglomeratu SystematykaSsaków i Zwierzyniec, a lista osobników jest ograniczona do Maksa. Do tak utworzonego konglomeratu dodawane są aksjomaty definiujący koncepty Psy i Koty jako podkoncepty odpowiednio konceptów Psowate i Kotowate. W ostatnim kroku jest wykonywana projekcja do trzech nazw konceptów. Poprawną odpowiedzią systemu wnioskującego jest konglomerat, w którym Max jest wystąpieniem konceptu Ogoniaste i Kudłate i nie jest wystąpieniem konceptu Pazurzaste. Taki konglomerat dostajemy w wyniku następującego wnioskowania: Max jest psem, ponieważ As go lubi; pies jest psowaty, a więc jest ogoniasty i kudłaty, a nie jest pazurzasty.

8. Podsumowanie

W artykule został przedstawiony język KQL dostępu do systemu RKaSeA, który charakteryzuje się na tle innych dostępnych języków przede wszystkim: jednolitością w zakresie tworzenia bazy wiedzy i manipulowania wiedzą; domkniętością osiągniętą w wyniku oparcia języka na algebrze konglomeratów; możliwością dowolnego zagnieżdżania zapytań, również wywoływania zapytań manipulujących bazą wiedzy w ramach zapytań tworzących tę bazę; prostą składnią, analogiczną do składni języka SQL; możliwością odpytywania metaontologii (wyszukiwania konceptów, ról i atrybutów spełniających pewne warunki) w ramach listy selekcji.

Język KQL jest językiem ciągle rozwijanym. W kolejnych wersjach zostanie przede wszystkim wzbogacony o rozkazy dotyczące kontroli dostępu do wiedzy oraz rozkazy dotyczące wykonywania operacji liczbowych.

BIBLIOGRAFIA

1. Date C.J.: Wprowadzenie do systemów baz danych. WNT, Warszawa, 2000.
2. Serafini L., Tamilin A.: Reasoning with Instances in Distributed Description Logics. Raport badawczy, Fondazione Bruno Kessler – IRST, 2007. http://sra.itc.it/people/tamilin/publications/2007/iswc/tr.pdf .
3. D1.1.1 Networked Ontology Model. NEON project deliverable. Raport projektu NEON, 2006. http://www.neon-project.org .
4. Cuenca Grau B., Horroks I., Kazakov Y., Sattler U.: Extracting modules from ontologies: Theory and practice. Raport badawczy, University of Manchester, 2007. http://www.cs.man.ac.uk/~ykazakov/publications/reports/CueHorKazSat06Modul_TR.pdf .
5. Goczyła K., Waloszek A., Waloszek W.: Algebra konglomeratów jako narzędzie opisu problemów przetwarzania ontologii. Studia Informatica, Vol. 30, No. 2A, 2009, s. 141÷156.
6. Goczyła K., Waloszek A., Waloszek W.: S-Modules – An Approach to Capture Semantics for Modularized DL Knowledge Bases, KEOD 2009. Funchal-Madeira, Portugalia, 2009, s. 117÷122.
7. Codd E.F.: Relational Completeness of Data Base Sublanguages. Database Systems, Vol. 6, 1979, s. 65÷98.

8. Konev B., Lutz C., Walther D., Wolter F.: Formal Properties of Modularisation. Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization, red. C. Parent, S. Spaccapietra, H. Stuckenschmidt, Springer, 2009.
9. Goczyła K., Waloszek A., Waloszek W.: Algebra of ontology modules for semantic agents. Springer-Verlag, 2009, s. 492÷503.
10. Henkin L., Monk J.D., Tarski A.: Cylindric Algebras pt. 1. Studies in Logic and the Foundations of Mathematics, Vol. 64, Amsterdam, 1971.
11. The Protégé Ontology Editor and Knowledge Acquisition System http://protege.stanford.edu/ .
12. Patel-Schneider P.F., Horrocks I.: OWL 1.1 Web Ontology Language Overview. http://www.w3.org/Submission/2006/SUBM-owl11-overview-20061219/, 2006.
13. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/.
14. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004, http://www.w3.org/Submission/SWRL/.
15. Prud'hommeaux E., SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/.
16. Bechhofer S.: The DIG Description Logic Interface: DIG/1.1. University of Manchester, 2003.
17. DIGUT Interface Version 1.3. Raport grupy KMG@GUT. 2005, http://km.pg.gda.pl/km/digut/1.3/DIGUT_Interface_1.3.pdf.
18. DIG 2.0 Proposal for a Query Interface. http://www.sts.tu-harburg.de/~al.kaplunova/dig-query-interface.html
19. Fikes R., Hayes P., Horrocks I.: OWL-QL – A Language for Deductive Query Answering on the Semantic Web. Knowledge Systems Laboratory, Stanford University, Stanford, CA, 2003. http://ksl.stanford.edu/KSL_Abstracts/KSL-03-14.html .
20. RacerPro User's Guide: http://www.racer-systems.com/products/racerpro/manual.phtml .
21. 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 .

Recenzenci: Dr inż. Sławomir Niedbała
Dr inż. Michał Świderski

Wpłynęło do Redakcji 30 stycznia 2010 r.

Abstract

As a part of the Semantic Web initiative there have been developed many languages for defining ontologies, issuing queries and communication between agents. Although these languages are useful for many applications, none of them meets requirements for a uniform language for accessing the knowledge bases. The main features that characterize a uniform language are:

- availability of statements to create and manipulate knowledge bases as well as to control the access to them,
- language closure (result of a query should be of the same type as an argument of queries),
- ability of embedding queries,
- uniformity of modification and definition queries (query results can be used in queries that update as well as create the knowledge base),
- ability of asking terminological queries,
- support for ontology modularization.

It has been noticed that most of these requirements are met by SQL (however in terms of relational data and not ontological ones), thus the natural step ahead was to introduce the language for accessing ontological data analogous to SQL for relational data. The SQL, basing on relationships stored in a database, allows for creating new relations presenting existing data from perspective currently required by the user. The relational algebra operators perform actions both on the relations headers as well as on entity sets. This allows for manipulating both on database schemas and the data stored in it.

Similar capabilities as relational algebra in relational databases are provided by conglomerate algebra for ontological data. Therefore, a new language based on conglomerate algebra for accessing knowledge bases named KQL (acronym for *Knowledge Query Language*) has been proposed and described in this paper.

The paper presents:

- the authors' motivation,
- the conglomerate algebra,
- the overall idea of the language as well as its features,
- the tool for creating KQL statements,
- the comparison of KQL language with other ontology-oriented languages,
- a short case study of KQL usage.

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 .

Piotr PIOTROWSKI: Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki, Katedra Inżynierii Oprogramowania, ul. G. Narutowicza 11/12, 80-233 Gdańsk, Polska, piopio@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 .