Implementacja widoków danych na bazę wiedzy

Piotr Piotrowski[1]

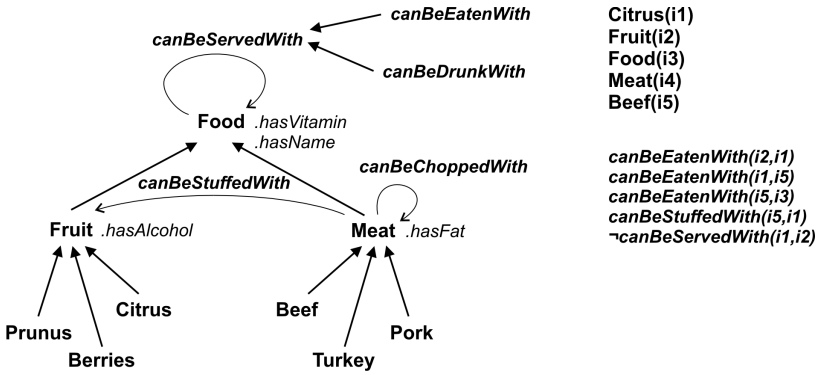
Streszczenie: Niniejszy artykuł opisuje koncepcję i implementację widoków danych na bazę wiedzy. Widoki danych przesłaniają interfejs bazy wiedzy, w zamian udostępniając interfejs pozwalający na zadawanie zapytań do bazy wiedzy w języku SQL. Głównym przeznaczeniem takich widoków jest ułatwienie integracji komponentów opartych na wiedzy z komponentami opartymi na danych, co może uprościć budowę systemów korzystających z bazy wiedzy. W artykule zaprezentowana jest koncepcja widoków danych, podstawowe algorytmy wykorzystane w implementacji, napotkane problemy oraz plany dalszego rozwoju.

Słowa kluczowe: widok danych, baza wiedzy, ontologia

1. Wstęp

We współczesnej informatyce coraz większą popularnością cieszą się technologie Semantic Web, które operują na wiedzy. Jednakże w praktyce częściej mamy do czynienia z „tradycyjnymi" komponentami, które działają na danych uzyskanych z klasycznych źródeł danych, takich jak np. relacyjne bazy danych. W wielu przypadkach może też nie być celowe rezygnowanie z dotychczasowych rozwiązań ze względów wydajnościowych lub z uwagi na nadmierną komplikację systemu. Pojawia się zatem potrzeba integracji w jednym systemie komponentów z tych dwóch światów, a do tego niezbędne jest, aby komunikowały się one we wspólnym języku. Rozwiązaniem zaproponowanym w Goczyła et al. (2005) jest danie możliwości tradycyjnym bazodanowym komponentom komunikowania się w ich języku macierzystym z komponentami przetwarzającymi wiedzę. Powstała koncepcja stworzenia warstwy pośredniczącej, która umożliwiałaby zadawanie zapytań do bazy wiedzy w języku SQL, tworząc tzw. widok danych (ang. Data View, DV) na bazę wiedzy. Tworząc relacyjny widok na bazę wiedzy, dokonujemy „domknięcia świata". Dla przykładu baza wiedzy nie może stwierdzić, czy owoc jest jadalny, czy też nie, dopóki nie może dowieść zgodnie z zasadami logiki jednej z tych możliwości. W przypadku baz danych zakłada się, że jeśli owoc nie wystąpi w tabeli „produkty jadalne", to znaczy, że nie jest jadalny, co może nie być zgodne z prawdą. W świecie zamkniętym (a takim jest świat baz danych) obowiązuje bowiem zasada domniemania zaprzeczenia (*negation as failure*). Kwestia domykania świata otwartego jest szerzej opisana w Goczyła et al. (2005). Dodatkowo w tej samej pracy zostało wprowadzone następujące ograniczenie: klient komunikujący się z bazą wiedzy poprzez Data View nie może dokonywać modyfikacji. Wynika to stąd, że jeśli klient ogranicza się do widoku danych, to może nie być świadomy konsekwencji, które

[1] Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, ul. Gabriela Narutowicza 11/12, 80-952 Gdańsk Wrzeszcz
e-mail: piopio@eti.pg.gda.pl



Rysunek 1. Przykładowa ontologia (Goczyła et al., 2005)

wynikałyby z wprowadzonej modyfikacji i w celu modyfikacji danych powinien skorzystać z interfejsu bazy wiedzy.

Niniejszy artykuł opisuje pierwszą wersję implementacji koncepcji Data Views oraz plany dalszego rozwoju. Przedstawione są algorytmy reprezentowania wiedzy zapisanej w bazie wiedzy w formie relacyjnej bazy danych. Następnie omówione są dwa podejścia do tworzenia widoków: widoki zmaterializowane i widoki niezmaterializowane. Zaprezentowane są główne problemy wykonanej implementacji oraz propozycje dalszego rozwoju. W naszym wywodzie zakładamy, że Czytelnik zaznajomiony jest z podstawowymi zagadnieniami dotyczącymi baz wiedzy, a w szczególności z pojęciami dotyczącymi ontologii opartych na logice opisowej (ang. *Description Logics*), takimi jak koncepty, role i atrybuty.

2. Język NeeK

Klient bazy wiedzy często nie jest zainteresowany całą ontologią, lecz tylko jej fragmentem. W tym aspekcie Data Views przypominają bazodanowe widoki, które pozwalają ograniczyć ilość prezentowanych klientowi informacji. Na potrzeby określenia fragmentu ontologii, którym zainteresowany jest klient Data View, stworzony został język NeeK (patrz Knowledge Management Group, 2005). Poza funkcją swojego rodzaju filtra znanego z widoków bazodanowych (`SELECT`) NeeK jest też środkiem do wyrażenia domknięcia świata.

Język NeeK składa się z kilku podstawowych zdań, które zostaną poniżej przedstawione, oraz wyrażeń zapożyczonych z języka DIGUT (patrz Grabowska et al., 2005), pozwalających określić koncepty, role i atrybuty zdefiniowane w ontologii. Do podstawowych zdań NeeK-a należą:

- InterestedInConceptStatement,
- NotInterestedInConceptStatement,
- InterestedInRoleStatement,

```
<manifest>
    <interestedInConcept>
        <catom name="http://xyz.pl/#Food" hierarchy="true"/>
    </interestedInConcept>
    <not>
        <interestedInConcept>
            <catom name="http://xyz.pl/#Fruit" hierarchy="true"/>
        </interestedInConcept>
    </not>
    <interestedInConcept>
        <catom name="http://xyz.pl/#Fruit" hierarchy="false"/>
    </interestedInConcept>
    <interestedInRole>
        <ratom name="http://xyz.pl/#canBeEatenWith"/>
    </interestedInRole >
</manifest>
```

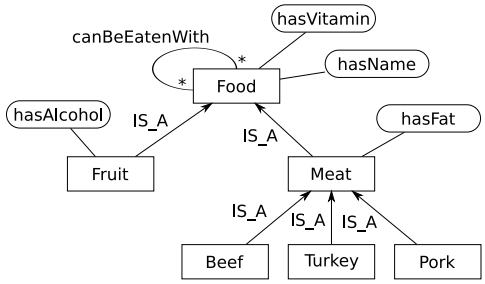
Rysunek 2. Przykładowy manifest zapisany w języku NeeK

- NotInterestedInRoleStatement,
- InterestedInAttributeStatement,
- NotInterestedInAttributeStatement.

Zgodnie z ich nazwami wyrażają one zainteresowanie danym konceptem lub brak takiego zainteresowania. Znaczenie tych zdań rozpatrzmy na przykładzie, który będzie opierał się na ontologii zaprezentowanej w Goczyła et al. (2005) (patrz rys. 1).

Na rys. 2 przedstawiony jest manifest definiujący fragment ontologii z rys. 1. Pierwsze zdanie tego manifestu określa, że jesteśmy zainteresowani konceptem Food wraz z wszystkimi konceptami od niego dziedziczącymi. W tym przypadku są to wszystkie nazwane koncepty. Drugie zdanie stwierdza, że nie jesteśmy zainteresowani konceptem Fruit i jego hierarchią (Prunus, Berries, Citrus). Następnie określamy, że jesteśmy zainteresowani konceptem Fruit, tym razem bez hierarchii. W efekcie w dziedzinie naszych zainteresowań znajdują się następujące koncepty: Food, Fruit, Meat, Beef, Turkey, Pork. W ostatnim zdaniu dodajemy do dziedziny zainteresowania rolę canBeEatenWith. W efekcie w widoku określonym danym manifestem znajdą się instancje wszystkich wymienionych konceptów oraz instancje wskazanej roli. Ponadto, jeśli w manifeście nie jest określone inaczej, wszystkie atrybuty wymienionych konceptów też dołączane są do dziedziny zainteresowania.

Szczegółowe algorytmy tworzenia Data View przedstawione są w kolejnych punktach.



Rysunek 3. Diagram ERD przykładowego Data View

3. Algorytmy tworzenia Data View

Kluczowym elementem opisywanego systemu jest przekształcenie ontologii w schemat relacyjnej bazy danych. Dokonywane jest to przy użyciu kilku algorytmów, z których najważniejsze zostaną pokrótce przedstawione poniżej.

3.1. Algorytm tworzenia widoku

Tworzenie widoku przebiega w kilku etapach. Pierwszy etap polega na stworzeniu ontologii, która jest okrojona w stosunku do oryginalnej na podstawie manifestu. Następnie wykonywane jest odwzorowanie tej ontologii na schemat relacyjnej bazy danych. Odwzorowanie ontologii na diagram związków encji jest realizowane według kilku reguł (patrz Goczyła et al., 2005):

- każdy koncept reprezentowany jest przez jeden zbiór encji,
- każdy zbiór encji ma atrybuty konceptu, który jest reprezentowany przez ten zbiór,
- każda rola reprezentowana jest przez związek typu wiele do wielu pomiędzy odpowiednimi zbiorami encji,
- hierarchia konceptów reprezentowana jest przez związek IS_A.

W docelowym schemacie bazy danych związek IS_A jest implementowany w następujący sposób. Każdy zbiór encji reprezentowany jest przez osobną tabelę. Każda tabela ma kolumny odpowiadające atrybutom, o ile dany atrybut nie należy również do zbioru encji położonego wyżej w hierarchii dziedziczenia. Instancja konceptu występuje jako wiersz w każdej tabeli reprezentującej koncept, do którego dana instancja należy.

Kontynuując powyższy przykład, interpretacja manifestu NeeK zaprezentowanego na rys. 2 zaowocuje utworzeniem Data View, którego diagram związków encji jest przedstawiony na rys. 3. Wynikowa relacyjna baza danych Data View przedstawiona jest na rys. 4. Dla utworzonych tabel zakładane są indeksy na kluczach głównych.

Food		
Id	*hasVitamin*	*hasName*
i1		
i2		
i3		
i4		
i5		

Fruit	
Id	*hasAlcohol*
i1	
i2	

Meat	
Id	*hasFat*
i4	
i5	

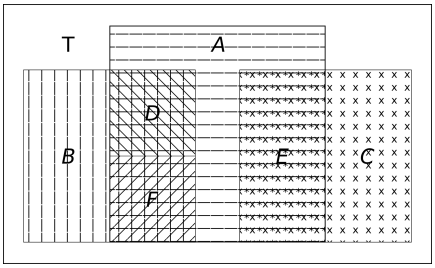
canBeEatenWith	
subject	*object*
i2	i1
i1	i5
i5	i3

Beef
Id
i5

Turkey
Id

Pork
Id

Rysunek 4. Tabele przykładowego Data View



Rysunek 5. Przykładowa prosta ontologia

3.2. Przypisanie atrybutów do zbiorów encji

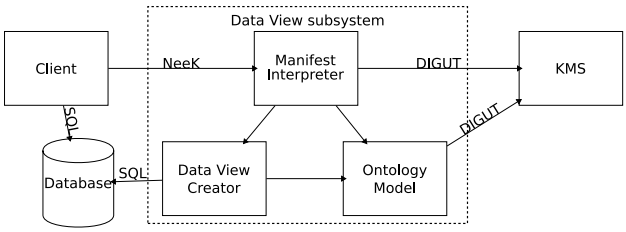
Atrybuty są przypisywane zbiorom encji tak, aby każda encja, która należy do dziedziny atrybutu, należała przynajmniej do jednego zbioru encji, któremu ten atrybut przypisano. Aby nie doprowadzić do sytuacji, w której atrybut powtarza się wielokrotnie, jest on przypisany zbiorom encji położonym jak najwyżej w hierarchii. Jednakże, aby uniknąć zdegenerowanego przypadku, w którym wszystkie atrybuty znajdują się w jednym nadrzędnym zbiorze encji w hierarchii dziedziczenia, atrybut przypisywany jest tak, by zminimalizować liczbę zbiorów encji, do których należą encje spoza dziedziny rozpatrywanego atrybutu.

Dla przykładu rozważmy ontologię, do której należą koncepty: A, B, C, D, E, F. W terminologii występują następujące aksjomaty:

$$\begin{aligned} B \sqcap C &\sqsubseteq \bot \\ D &\sqsubseteq A \sqcap B \\ E &\equiv A \sqcap C \\ F &\equiv A \sqcap B \sqcap \neg D \end{aligned} \tag{1}$$

Powyższa terminologia zaprezentowana jest w formie diagramu Venna na rys. 5.

Ponadto w tej ontologii występuje atrybut a, którego dziedziną jest koncept A. Załóżmy, że do Data View zostały dodane koncepty B, C, D, E oraz atrybut a. W bazie danych atrybut zostanie zaprezentowany jako kolumna w tabeli B oraz tabeli E. Gdyby było inaczej i zamiast tabeli B tę kolumnę miałaby tabela D, to w tabeli B mogłyby znaleźć się instancje konceptu F, które wprawdzie mają wartość tego atrybutu, ale nie jest to odzwierciedlone w wynikowym widoku. Natomiast gdyby zamiast tabeli E kolumnę a miała tabela C, to niepotrzebnie powiększony zostałby zbiór wierszy z pustą kolumną a.



Rysunek 6. Architektura podsystemu Data View

4. Materialized Data Views

Materialized Data Views są widokami na bazę wiedzy, które w chwili tworzenia przepisują opisany w manifeście fragment ontologii (i bazy wiedzy) do relacyjnej bazy danych. Po tym zabiegu klient odpytuje bezpośrednio bazę danych. Ponieważ tak stworzony widok „starzeje się" i po pewnym czasie może zawierać nieaktualne lub błędne dane, możliwe jest odświeżenie go na żądanie lub odświeżanie w regularnych odstępach czasu określonych w manifeście. Podstawową zaletą tego rozwiązania jest szybkość dostępu do raz wyeksportowanych danych.

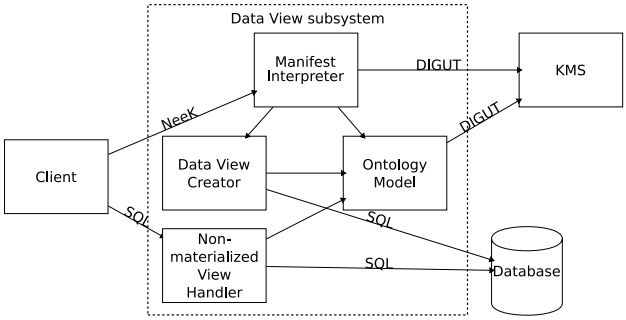
System tworzący Data Views przypomina architekturą (patrz rys. 6) wzorzec Model-View-Controller (MVC), gdzie *Manifest Interpreter* odpowiada kontrolerowi, *Data View Creator* odpowiada widokowi, a *Ontology Model* modelowi. Wzorzec ten został wzbogacony o dodatkowe bezpośrednie połączenie pomiędzy kontrolerem a źródłem danych, aby uprościć implementację.

Manifest Interpreter jako wejście przyjmuje manifest sformułowany w języku NeeK. Na jego podstawie tworzy uproszczony model wybranego fragmentu ontologii, którym zainteresowany jest klient. Po utworzeniu modelu ontologii wywoływany jest *Data View Creator*, który na podstawie tego modelu i zgodnie z przedstawionym algorytmem tworzy schemat bazy danych i wypełnia utworzone tabele danymi pobranymi z bazy wiedzy będącej elementem systemu zarządzania wiedzą (*Knowledge Management System*, *KMS*). Dane te pobierane są za pośrednictwem *Ontology Model*. Jako rezultat, klient otrzymuje URI bazy danych, którą może odpytywać oraz schemat utworzonej bazy danych.

Cała komunikacja pomiędzy podsystemem DV a bazą wiedzy odbywa się poprzez interfejs DIGUT (patrz Grabowska et al., 2005). Dzięki temu podsystem DV abstrahuje od sposobu reprezentacji wiedzy przechowywanej przez KMS.

Stworzony przez automat schemat bazy danych może nie zawsze odpowiadać potrzebom użytkownika. Dzięki temu, że użytkownik uzyskuje pełen dostęp do utworzonej bazy danych, może on utworzyć zwykłe widoki bazodanowe, które przystosowują strukturę danych do konkretnej aplikacji.

W bazie danych Oracle 10g istnieje możliwość przechowywania danych RDF (patrz Oracle, 2005), na które można nakładać widoki. Pozwala to na odpytywanie danych RDF za pomocą z zapytań SQL. Wydaje się to być alternatywą dla DV, lecz w obecnej wersji Oracle'a możliwości wnioskowania nie sięgają logiki opisowej. Ponadto nie ma możliwości podłączenia istniejących silników wniosku-



Rysunek 7. Architektura podsystemu Non-materialized Data View

jących w sposób pozwalający skorzystać z oferowanych przez Oracle'a możliwości tworzenia widoków.

5. Non-materialized Data Views

Widok, który nie został zmaterializowany w bazie danych, wymaga zinterpretowania zapytania SQL na bieżąco i wystosowania odpowiednich zapytań do bazy wiedzy. Implementacja wydajnej interpretacji wszystkich konstrukcji języka SQL wydaje się bardzo kosztowna, stąd powstał pomysł wykorzystania istniejącego DBMS-a do przetwarzania zapytań użytkownika. Jest to w dużym stopniu zbliżone do rozwiązań wykorzystywanych w przypadku widoków zmaterializowanych. Zaprezentowane rozwiązania dotyczące widoków niezmaterializowanych nie zostały jeszcze zaimplementowane w aktualnej wersji systemu, lecz są w trakcie realizacji.

Architektura systemu tworzącego widoki niezmaterializowane różni się od poprzedniej architektury dodatkowym modułem, który pośredniczy w zapytaniach SQL-owych pomiędzy klientem a bazą danych (patrz rys. 7).

Scenariusz tworzenia niezmaterializowanego widoku nie różni się znacznie od tworzenia widoku zmaterializowanego. Klient podaje manifest w języku NeeK, na podstawie którego tworzony jest model ontologii. Następnie *Data View Creator* tworzy schemat bazy danych. Jednakże zamiast stworzyć i wypełnić tabele reprezentujące ontologię, tworzone są tylko puste tabele tymczasowe. Jako wynik zwracane jest URI, które tym razem nie wskazuje bezpośrednio bazy danych. Tak jak w przypadku widoków zmaterializowanych, zwracany jest również opis schematu bazy danych.

Ponieważ URI, którym dysponuje klient, nie wskazuje bezpośrednio na bazę danych, wszelkie zapytania SQL-owe kierowane są do *Non-materialized View Handler*, który je interpretuje. Na podstawie treści zapytania z modelu ontologii pobierana jest tylko pewna część całego widoku i umieszczana w tymczasowych tabelach bazy danych. Następnie na tych tabelach wykonywane jest oryginalne zapytanie pochodzące od użytkownika.

Takie podejście daje możliwość szybkiej realizacji działającego systemu. Najprostszą implementacją jest wykorzystanie kodu widoków zmaterializowanych i po-

bieranie przy każdym zapytaniu wszystkich danych, umieszczeniu ich w tymczasowych tabelach i wykonaniu zapytania. To rozwiązanie może być nieakceptowalne z punktu widzenia wydajności, lecz pozwala na sukcesywne wprowadzanie kolejnych optymalizacji bez modyfikacji interfejsu dostarczanego użytkownikowi.

Jedną grupą optymalizacji, które można wprowadzić, jest specjalne traktowanie typowych zapytań, które będą kierowane do niezmaterializowanego widoku. Dla przykładu najprostsze zapytanie

```
SELECT * FROM koncept
```

może zostać wykonane bez udziału zewnętrznej bazy danych przez bezpośrednie odpytanie *Ontology Model* przez *Non-materialized View Handler*. Zapytanie to można uogólnić do postaci

```
SELECT atrybut1, atrybut2... FROM koncept
    WHERE atrybut1 = 'abc' AND atrybut2 > 3...
```

Takie zapytanie można wykonać poprzez odpytanie bazy wiedzy o instancje konceptu:

$$\texttt{koncept} \sqcap \exists \texttt{atrybut1}.= \text{'abc'} \sqcap \exists \texttt{atrybut2}.> 3 \ldots \tag{2}$$

Inny typ optymalizacji można zastosować do zapytań ze złączeniami. Na podstawie prostej analizy zapytania SQL można uzyskać informacje, jakie tabele są wykorzystywane w złączeniu, więc można pominąć zapytania do bazy wiedzy o koncepty, które nie są wymagane do uzyskania odpowiedzi. Tak uzyskane dane umieszczane są w tymczasowych tabelach i wykonywane jest na nich oryginalne zapytanie SQL.

W przypadkach nieobsługiwanych w szczególny sposób następuje wykorzystanie wspomnianego prymitywnego mechanizmu pobierającego wszystkie dane z bazy wiedzy wynikające z manifestu i umieszczającego je w tymczasowych tabelach.

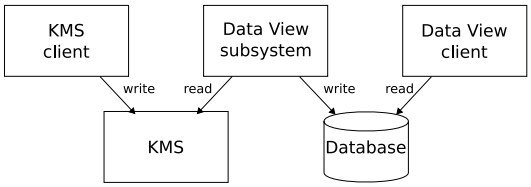
Z zaprezentowanych przykładów widać, że pomocnicza baza danych może być wykorzystywana do wykonywania bardziej złożonych operacji, lecz w niektórych przypadkach może zostać pominięta.

6. Problemy

Dwoma głównymi problemami zaproponowanej implementacji Data View jest wydajność oraz spójność udostępnianych danych. Te dwie kwestie zostaną omówione w następnych punktach.

6.1. Wydajność

Podsystem Data View komunikuje się z bazą wiedzy i z bazą danych. Są to zazwyczaj dość kosztowne operacje, których wykonanie pochłania większość czasu tworzenia widoków. Aby przyspieszyć działanie systemu, kluczowe jest zmniejszenie liczby zapytań kierowanych do bazy wiedzy.



Rysunek 8. Zależności pomiędzy elementami widoku zmaterializowanego

Niejednokrotnie do optymalizacji algorytmów komunikacji z bazą wiedzy można wykorzystać właściwości struktury ontologii. Dla przykładu podsystem Data View musi wiedzieć, które z konceptów występujących w manifeście NeeK są różne, a które są równoważne. W manifeście koncepty mogą być zapisywane jako złożone wyrażenia lub też nie być jawnie wymienione (co może mieć miejsce w przypadku zainteresowania hierarchią konceptów). Jedynym sposobem, aby rozstrzygnąć równoważność dwóch konceptów, jest odpytanie bazy wiedzy. Porównując koncepty parami, otrzymujemy kwadratową liczbę zapytań w funkcji liczby konceptów. Jeśli ontologia, której widok tworzymy, ma strukturę hierarchiczną, to można wykorzystać ten fakt i wyeliminować cały zbiór konceptów, które zawierają się w innym koncepcie rozłącznym z konceptem, którego równoważność z pozostałymi chcemy ustalić. Taki algorytm może dać duże przyspieszenie. Jeśli jednak ontologia składa się z konceptów rozłącznych (co jest jednak w praktyce sytuacją rzadką), to nadal występuje ów pesymistyczny przypadek, dla którego ten algorytm ma złożoność kwadratową.

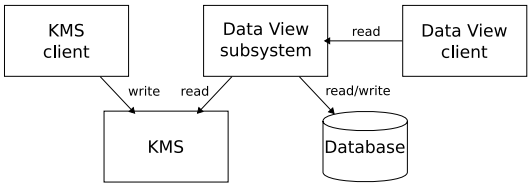
Możliwe do wymienienia są również inne optymalizacje, które dają przyspieszenie tylko w niektórych przypadkach, natomiast w innych powodują degradację wydajności. Ważne jest zatem, aby optymalizacji dokonywać na podstawie zbioru przypadków testowych, które odpowiadają rzeczywistemu wykorzystaniu systemu.

6.2. Transakcje

Przy tworzeniu zmaterializowanych widoków pobierane są wszystkie potrzebne dane umieszczane w bazie danych i udostępniane użytkownikowi tylko do odczytu. Scenariusz ten wydaje się być bardzo prosty, lecz i tu może dojść do sytuacji, w której użytkownik otrzymuje niespójne dane.

Na rys. 8 zaprezentowano zależności pomiędzy elementami systemu z zaznaczonym wykonywanym typem operacji. Operacje zapisu do bazy danych przez podsystem Data View są wykonywane w transakcji. Klient Data View może odczytywać dane również w transakcji, otrzymując dane spójne z punktu widzenia bazy danych, jeśli zastosowany jest odpowiedni poziom izolacji. Jednakże, aby w bazie danych znajdowały się spójne dane z punktu widzenia KMS, niezbędne jest, aby KMS udostępniał mechanizm transakcji z odpowiednim poziomem izolacji.

W przypadku widoków niezmaterializowanych klient nie ma bezpośredniego dostępu do bazy danych, lecz wszystkie zapytania kierowane są do podsystemu Data View. Jak widać na rys. 9, jedyne operacje zapisu, które mogą potencjalnie spo-



Rysunek 9. Zależności pomiędzy elementami widoku niezmaterializowanego

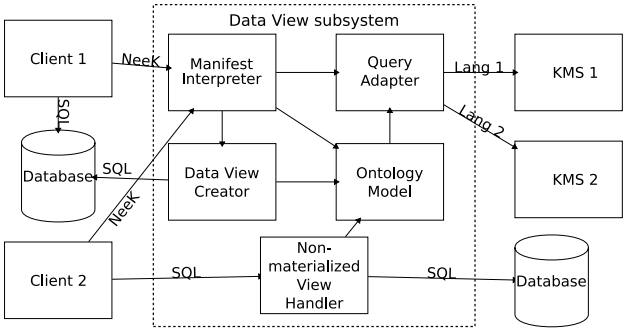
wodować niespójność danych, są wykonywane w stosunku do KMS. Zatem implementacja transakcji w podsystemie Data View polegałaby na przekazywaniu żądań rozpoczęcia i zakończenia transakcji od klienta Data View do KMS.

7. Możliwości rozwoju

Poza implementacją widoków niezmaterializowanych oraz sukcesywnym rozszerzaniem funkcjonalności możliwe jest dodanie specyficznych możliwości, co jednak w dużym stopniu zależy od sposobów wykorzystywania systemu i wynikających z nich potrzeb.

Jedną z tych możliwości jest dostęp do różnych baz wiedzy. Podsystem Data View został stworzony z myślą o bazach wiedzy udostępniających interfejs DIGUT (patrz rys. 6 i 7 oraz Grabowska et al., 2005). Istnieje jednak możliwość parametryzacji tego podsystemu tak, aby mógł obsługiwać również inne bazy wiedzy, np. wykorzystujące interfejs DIG (patrz Bechhofer, 2003): CEL (patrz Suntisrivaraporn, 2006), FaCT++ (patrz Tsarkov), Pellet (patrz Sirin et al.), RacerPro (patrz Racer Systems, 2005) lub jakikolwiek inny interfejs pozwalający na wydobywanie potrzebnej wiedzy, np. programistyczny: Jena (patrz Jena). Można to osiągnąć poprzez wprowadzenie dodatkowego modułu *Query Adapter* (patrz rys. 10), który wybierałby odpowiedni język do komunikacji z konkretną bazą wiedzy. Istnieją jednak pewne wymagania, które wynikają z języka NeeK i dotyczą zapytań, które kierowane są do bazy wiedzy podczas tworzenia Data View. W obecnej wersji używane są następujące zapytania:

- podaj koncepty, które dziedziczą po danym koncepcie;
- podaj koncepty, po których dziedziczy dany koncept;
- sprawdź, czy dwa koncepty są równoważne;
- sprawdź, czy jeden koncept dziedziczy po innym;
- sprawdź, czy koncept jest spełnialny;
- podaj instancje konceptu;
- sprawdź, czy dana instancja jest instancją danego konceptu;
- podaj wszystkie role;



Rysunek 10. Architektura podsystemu Data View dostosowana do różnych baz wiedzy

- podaj dziedzinę i zakres roli;
- podaj instancje ról;
- podaj wszystkie atrybuty;
- podaj atrybuty, których dziedziną jest dany koncept;
- podaj dziedzinę i typ atrybutu;
- podaj wartość atrybutu dla danej instancji.

Pod pojęciem koncept rozumiany jest nazwany koncept lub wyrażenie składające się z sumy, przecięcia oraz dopełnienia konceptu (patrz Knowledge Management Group, 2005).

Nie wszystkie z podanych zapytań są niezbędne, ponieważ niektóre z nich można emulować po stronie podsystemu Data View. Dla przykładu wynik zapytania o atrybuty wskazanego konceptu można uzyskać poprzez pobranie wszystkich atrybutów, dziedziny dla każdego z nich i porównanie tych dziedzin z danym konceptem. Emulacja taka może jednak negatywnie wpłynąć na wydajność systemu.

8. Wykorzystanie

Podsystem Data View jest jednym z modułów systemu KMS powstającego na potrzeby projektu PIPS (*Personalised Information Platform for Life & Health Services*) fundowanego przez Komisję Europejską w ramach 6. Programu Ramowego. Przeznaczeniem systemu KMS w projekcie jest zarządzanie wiedzą służącą wspomaganiu podejmowania decyzji w zakresie ochrony zdrowia obywateli i utrzymania przez nich zdrowego trybu życia.

Literatura

Bechhofer S. (2003) *The DIG Description Logic Interface: DIG/1.1.* University of Manchester

Goczyła K., Waloszek W., Zawadzka T. and Zawadzki M. (2005) Designing World Closures for Knowledge-based System Engineering. *Software engineering: evolution and emerging technologies*, IOS Press, 271–282

Grabowska T., Goczyła K., Waloszek W. and Zawadzki M. (2005) *DIGUT Interface Specification, Version 1.3, Technical Report.* Gdańsk University of Technology, http://km.pg.gda.pl//km/digut/1.3/DIGUT_Interface_1.3.pdf

Sirin E., Parsia B., Grau B. C., Kalyanpur A. and Katz Y. *Pellet: A Practical OWL-DL Reasoner.* http://www.mindswap.org/papers/PelletJWS.pdf

Waloszek W. (2005) Kartograficzna metoda reprezentacji wiedzy w systemie KaSeA. *Technologie Przetwarzania Danych*, Wydawnictwo Politechniki Poznańskiej, 14–25

Knowledge Management Group (2005) *World Closures for KaSeA system: Specification of NeeK ver. 1.0.* Gdańsk University of Technology http://km.pg.gda.pl/km/NeeK/1.0/Specification%20of%20NeeK%20Language.pdf

Suntisrivaraporn B. (2006) *The CEL Manual (v0.9b).* TU Dresden http://lat.inf.tu-dresden.de/systems/cel/manual-0.9.pdf

Tsarkov D. *OWL: FaCT++.* http://owl.man.ac.uk/factplusplus/

Racer Systems GmbH & Co. KG (2005) *RacerPro User's Guide Version 1.9.* http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf

Jena *A Semantic Web Framework for Java.* http://jena.sourceforge.net/

Oracle (2005) *Oracle Spatial Resource Description Framework (RDF).* http://download-uk.oracle.com/docs/cd/B19306_01/appdev.102/b19307/toc.htm

Knowledge Base Data Views implementation

This article describes the concept and implementation of Data Views for knowledge base. Data Views hide the knowledge base interface and provide a new interface that allows using SQL for querying the knowledge base. Their main purpose is to ease the integration of knowledge-based components with data-based components, that can simplify the design of systems using knowledge bases. This article presents the concept of the Data Views, basic algorithms, encountered problems as well as plans of future development.