

Wnioskowanie hybrydowe w relacyjnej bazie danych, wykorzystujące ontologię OWL wzbogaconą regułami języka SWRL

Jarosław Bąk, Czesław Jędrzejek
Instytut Automatyki i Inżynierii Informatycznej, Politechnika Poznańska

jjkbak@poczta.wp.pl czeslaw.jedrzejek@put.poznan.pl

Abstrakt. Niniejszy artykuł przedstawia środowisko semantycznego przetwarzania danych z wykorzystaniem relacyjnej bazy danych, ontologii i reguł stanowiących bazę wiedzy oraz zaproponowanego przez autorów hybrydowego narzędzia wnioskującego. Hybryda składa się z silników wnioskujących: wstecz oraz w przód zadających zapytania na bazie ontologii. Oba silniki zostały zaimplementowane w języku skryptowym Jess Language. Mechanizm wnioskowania jest realizowany na bazie pojęć i właściwości zdefiniowanych w ontologii zapisanej w języku OWL oraz na bazie reguł operujących na pojęciach, wyrażonych w języku regułowym SWRL. Wnioskowanie wstecz służy do pobierania informacji z relacyjnej bazy danych. Pobrane dane odwzorowywane są w ustrukturalizowanej bazie wiedzy. Mechanizm wnioskowania w przód służy natomiast wykryciu zależności pomiędzy wczytanymi danymi, uwzględniając ograniczenia zdefiniowane w regułach ontologicznych, które nie mogły być brane pod uwagę w mechanizmie wnioskowania wstecz. Nie posiada on żadnej interakcji z relacyjną bazą danych. Operuje natomiast na pojęciach i regułach znajdujących się w bazie wiedzy, zawierającej również wczytane wcześniej dane. Hybryda podejmuje decyzję dotyczącą mechanizmu wnioskowania. Jeśli silnik wnioskujący w przód nie potrafi udzielić odpowiedzi, zapytanie jest kierowane do silnika wnioskującego wstecz, które stara się ją znaleźć bazując na danych relacyjnych oraz regułach i pojęciach ontologicznych. Zaproponowana analiza bazująca na formalnie zdefiniowanej semantyce danych została zaimplementowana w bibliotece narzędziowej SDL (ang. *Semantic Data Library*). Biblioteka ta odpowiada za automatyczną integrację relacyjnych baz danych, ontologii i reguł silnika Jess, oraz pozwala na zastosowanie wyżej wspomnianej metody wnioskowania hybrydowego. Ze względu na wstępny charakter prac nad biblioteką, przedstawione zostaną również ograniczenia przyjęte w procesie transformacji, oraz zagadnienia wydajności wnioskowania. Zaprezentowany zostanie przykład zastosowania analizy semantycznej realizowanej za pomocą ontologii opisującej relacje rodzinne, zawierającej zarówno pojęcia jak i reguły.

Informacja o autorach. Prof. dr hab. inż. Czesław Jędrzejek – w początkowym okresie pracy związany z AGH i UJ w Krakowie. Przez okres 10 lat odbywał staże naukowe i pracował jako Visiting Professor kolejno na kilku uczelniach w USA. W roku 1994 związał się Francusko-Polską Wyższą Szkołą Nowych Technik Informatyczno-Komunikacyjnych (EFP) w Poznaniu. W latach 1999-2005 zajmował stanowisko Wiceprezesa Zarządu firmy ITTI w Poznaniu. Jest autorem lub współautorem ponad 150 publikacji. Kierował kilkudziesięcioma projektami dla wiodących operatorów oraz dostawców sprzętu telekomunikacyjnego w Polsce w zakresie ewolucji sieci i usług, inżynierii ruchu w sieciach teleinformatycznych oraz wykonania, integracji i wdrożenia systemów informatycznych. Od 2003 r. zajmuje stanowisko profesora w Instytucie Automatyki i Inżynierii Informatycznej Politechniki Poznańskiej. Realizował kilka projektów europejskich dotyczących aplikacji informatycznych. Mgr inż. Jarosław Bąk jest pracownikiem Instytutu Automatyki i Inżynierii Informatycznej Politechniki Poznańskiej. W kręgu jego zainteresowań znajduje się szereg zagadnień związanych z semantyką danych, technologiami semantycznymi oraz przetwarzaniem metadanych.

1. Założenia projektowe środowiska semantycznego przetwarzania danych

Współczesne aplikacje wymagają zarządzania danymi we wszystkich warstwach. Relacyjne bazy danych są standardem dla systemów korporacyjnych – najbardziej zestrukturalizowanych.

Nawet tu jednak występuje bariera semantyczna pomiędzy modelem danych, w oparciu o które działają obiektowe aplikacje a magazynem danych. Dla metadanych opisujących dane o ubogiej strukturze nie wystarczają usługi odpowiedzi na zapytania, wykonania aktualizacji czy zapewnienie mechanizmów wykonywania transakcji. Pozyskana informacja, żeby mogła być przetwarzana i analizowana wymaga bogatszych możliwości w warstwie konceptualnej (stworzenia ontologii) i wnioskowania. Konieczna jest budowa środowiska obsługi baz wiedzy o złożonej architekturze, bo takie jest niezbędne do działania systemów decyzyjnych czy regułowych.

1.1. Cel pracy i architektura środowiska narzędziowego

Celem pracy jest zaproponowanie metody i narzędzi integrowania różnych technologii reprezentowania i przetwarzania danych, które umożliwią efektywną ich analizę. Tworzona biblioteka SDL (ang. *Semantic Data Library*) umożliwi przeprowadzanie semantycznej analizy danych pochodzących z relacyjnych baz danych. Pojęcie semantycznej analizy należy rozumieć jako analiza bazująca na formalnie zdefiniowanej semantyce danych (tzw. metadane). Wspomniana biblioteka umożliwi integrację modeli konceptualnych wiedzy reprezentowanych w postaci ontologii wraz z regułami, maszynami wnioskującymi i danymi zawartymi w relacyjnej bazie danych. Potrzeba realizacji systemu wyniknęła w pracach przygotowawczych Systemu Analizatora Faktów i Związków. Dotychczas istniejące systemy (SweetRules [SwRu], OWLJessKB [OWLJK]) integrują lub próbują integrować tylko niektóre elementy spośród wyżej wymienionych. Obecnie istniejące systemy wnioskujące posiadające możliwość interakcji z bazą danych (np. KAON2 [KAON2]), wczytują całą zawartość bazy danych (partiami lub całościowo) do pamięci operacyjnej, co w znacznym stopniu spowalnia działanie mechanizmu wnioskującego. Stąd wynikła potrzeba zbudowania uniwersalnego narzędzia, łączącego zintegrowane funkcjonalności ontologii, wnioskowania oraz baz danych. Zastosowanie reguł wraz z ontologiami powinno umożliwić odkrywanie wiedzy, którą bardzo trudno uzyskać w przypadku samych zapytań SQL. Hybryda wnioskująca pozwala na pobranie tylko i wyłącznie tych danych, które mają znaczenie w procesie analizy. Zaproponowane podejście znacznie zwiększa wydajność wnioskowania wykorzystującego relacyjne bazy danych. Ontologiczne podejście do danych pozwala na ich strukturalizację oraz odnalezienie relacji pomiędzy nimi. Dzięki dołączeniu wiedzy ontologicznej do danych relacyjnych jest możliwe uzyskanie tzw. metadanych semantycznych. Takie podejście pozwala na semantyczną analizę danych, w której można wykorzystywać silniki wnioskujące oparte wyłącznie na ontologiach lub na ontologiach wzbogaconych o reguły, w znaczny sposób zwiększające możliwości analizy właściwości semantycznych przetwarzanych danych.

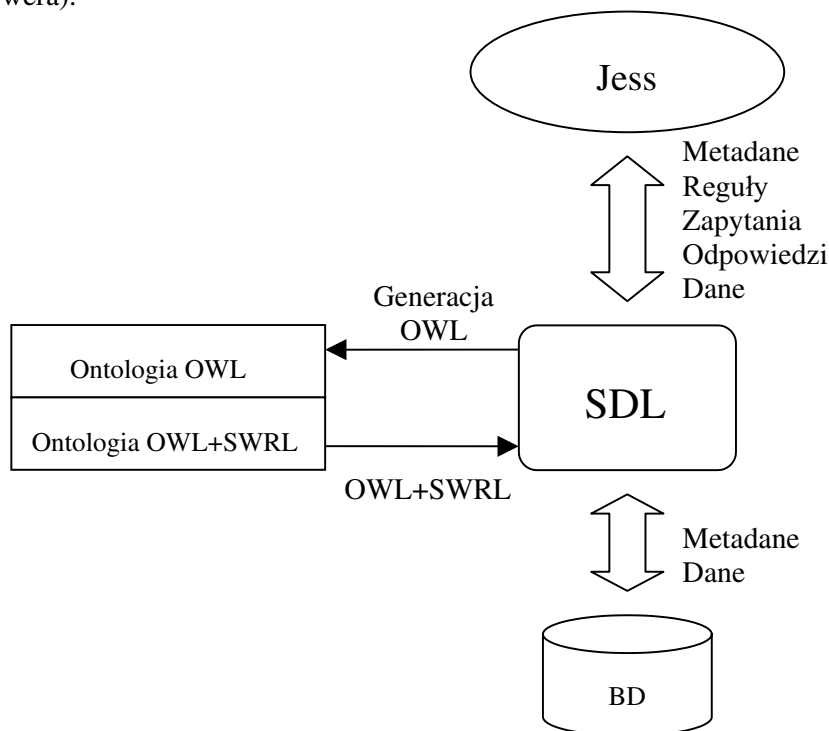
Głównymi zadaniami stawianymi przed środowiskiem semantycznego przetwarzania danych są: zcentralizowany dostęp do różnych technologii oraz semantyczna analiza danych wykorzystująca wnioskowanie hybrydowe. W ramach zadań szczegółowych są realizowane następujące wymagania:

- odczytywanie schematu relacyjnej bazy danych,
- generowanie ontologii OWL [OWL] na podstawie schematu relacyjnej bazy danych,
- generowanie skryptu w języku Jess [Jess], zawierającego meta opis struktury bazy danych,
- przetwarzanie plików w formacie OWL wraz z regułami SWRL [SWRL],

- odwzorowywanie wszystkich konceptów ontologicznych, właściwości oraz reguł wraz z niektórymi więzami na zmienne zapisane w język SWRLB na skrypcie w języku Jess Language,
- generowanie zapytań Jess na podstawie reguł SWRL wraz z funkcjami SWRLB,
- manipulowanie regułami Jess z poziomu języka Java,
- dodawanie danych z relacyjnej bazy danych, odpowiadających struktutom szablonów w języku Jess z poziomu języka Java,
- zadawanie zapytań do relacyjnej bazy danych,
- zadawanie zapytań do silnika reguł Jess,
- wnioskowanie w przód,
- wnioskowanie wstecz,
- moduł wnioskowania hybrydowego.

Architektura całego systemu została przedstawiona na rysunku Rys. 1, na którym kierunek strzałek określa przepływ informacji pomiędzy modułami, natomiast poszczególne skróty mają następujące znaczenie:

- Jess – Java Expert System Shell,
- Ontologia OWL – ontologia zawierająca metadane semantyczne opisujące bazę danych, zapisana w języku OWL,
- Ontologia SWRL – ontologia wzbogacona nowymi pojęciami oraz regułami w języku SWRL,
- DB – baza danych MS SQL Server 2000 lub 2005 (biblioteka SDL obsługuje obie wersje serwera).



Rys. 1. Architektura narzędzia semantycznego przetwarzania – SDL

1.2. Narzędzia i standardy wykorzystywane przy realizacji systemu

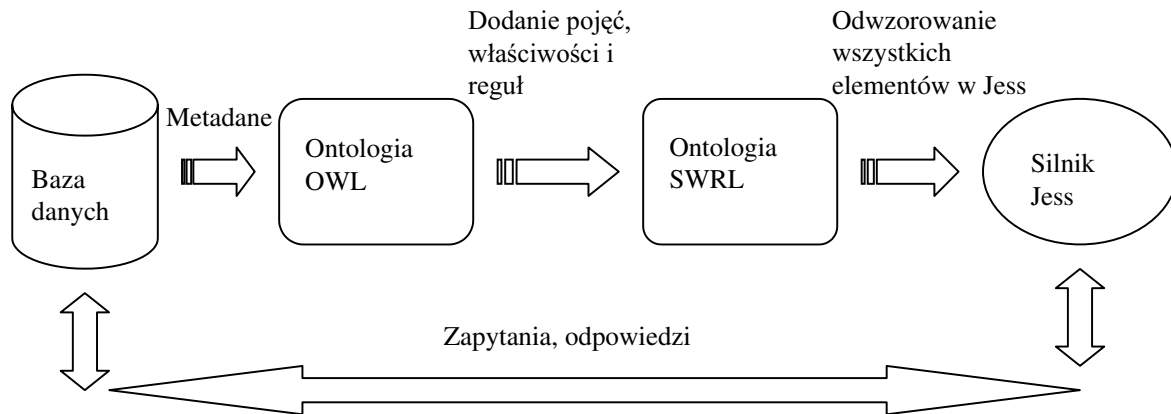
Metody i narzędzia do specyfikowania i przetwarzania semantyki danych to jedna z najbardziej dynamicznie rozwijających się gałęzi systemów informatycznych w ostatnich latach. Dlatego dobór narzędzi do realizacji systemu został dokonany na podstawie wymienionych założeń.

Bibliotekę SDL zaimplementowano w języku Java ze względu na jego powszechną dostępność i szerokie stosowanie w informatyce. Serwer baz danych stanowi produkt firmy Microsoft – MS SQL Serwer 2000 (biblioteka obsługuje również serwer w wersji 2005). Silnik wnioskujący stanowi narzędzie Jess, zaimplementowane i rozwijane w laboratoriach rządowych Sandia w Stanach Zjednoczonych. Jess, jako silnik reguł, pozwala na konstruowanie oprogramowania, które posiada elementy sztucznej inteligencji, zapisane w faktach i regułach stanowiących bazę wiedzy. Silnik ten działa w oparciu o algorytm wnioskowania w przód – Rete [Rete]. Wnioskowanie wstecz możliwe jest poprzez zastosowanie mechanizmu aktywującego dodanie faktu do bazy wiedzy wtedy, gdy jest on potrzebny (prefiks *need-(x)*). Narzędzie posiada swój własny język skryptowy – Jess Language. Wszystkie fakty oraz reguły przechowywane są w pamięci roboczej, co pozwala na szybkie działanie programu. Fakt ten ogranicza jednak ilość możliwych do wczytania danych. Narzędzie zostało zaimplementowane w języku Java.

Do definiowania ontologii wybrany został język OWL. Jest to obecnie najpowszechniej używany standard będący rekomendacją konsorcjum W3C. Język OWL można wzbogacać o reguły, stanowiące największą siłę analizy semantycznej. Reguły zapisywane są w standardzie SWRL, stanowiącego rekomendację konsorcjum W3C. Modyfikacja ontologii możliwa jest w dowolnym narzędziu obsługującym standardy OWL oraz SWRL. W proponowanym systemie wykorzystano edytor Protégé [Protégé].

2. Opis procesu integracji relacyjnej bazy danych z silnikiem wnioskującym przy wykorzystaniu ontologii

Celem przedstawionej integracji jest możliwość przekształcania stanowiących powiązaną całość danych relacyjnych, ontologii OWL oraz reguł SWRL do formatu akceptowalnego przez system wnioskujący Jess. Możliwości biblioteki Jess sprawiły, że jest ona punktem końcowym całej integracji, w którym dokonywane jest wnioskowanie oraz uruchamiane są zapytania. W trybie wnioskowania w przód przestrzeń robocza Jess zawiera dane, pochodzące z relacyjnej bazy danych, odwzorowane na format szablonów języka Jess Language. Warstwa ontologiczna systemu, którą stanowi dowolna ontologia OWL wraz z regułami SWRL, jest również odwzorowywana na formę szablonów oraz reguł i zapytań Jess. Prezentacja wszystkich danych w jednym formacie języka Jess pozwala na wykorzystanie wszystkich możliwości systemu wnioskującego. Całkowity proces integracji został przedstawiony na Rys. 2.



Rys. 2. Proces integracji realizowany przez bibliotekę SDL

Możliwość wnioskowania na danych, pochodzących z relacyjnej bazy danych, przy użyciu silnika Jess, wymaga przeprowadzenia odpowiedniej transformacji. Proces ten składa się z trzech etapów oraz wiąże się z koniecznością przyjęcia pewnych założeń, których spełnienie jest koniecznym warunkiem dla przynajmniej częściowego zautomatyzowania całego procesu. Kryteria te muszą obowiązywać od momentu projektowania bazy danych, aż po otrzymanie wyników wnioskowania. Przyjęte założenia dotyczą głównie form zapisu, których należy przestrzegać w zapisie metadanych. Należą do nich następujące wymagania:

- zarówno nazwa tabeli, jak i nazwy kolumn w relacyjnej bazie danych nie mogą zawierać znaku „-”,
- nazwa kolumny powinna być czytelna dla użytkownika i przystosowana do traktowania, jako właściwość w generowanej ontologii; oznacza to, że zamiast nazwy kolumny *IdOjca* najlepiej by było użyć nazwy *maOjca*, która wiąże identyfikatory dwóch osób będące w relacji syn-ojciec; nie jest to jednak wymóg niezbędny,
- właściwości w ontologii są rozumiane w następujący sposób: zapis *maOjca(?x, ?y)* oznacza, iż *?y* jest ojcem *?x* (inaczej: *?x maOjca ?y*); pierwsza zmienna jest odpowiednikiem *rdfs:domain*, natomiast druga *rdfs:range*,
- specyfikacja zmiennych, występujących w regułach SWRL, pochodzących z bazy danych, musi identyfikować kolumnę w tabeli i być rozdzielona znakiem „-” z identyfikatorem zmiennej (przykładowo: jeśli zmienna *?x* ma reprezentować wartość w kolumnie *Osoby*, jej zapis powinien być następujący: *?Osoba-x*),
- ograniczenia, nałożone na zmienne, muszą być zapisywane po wcześniejszej deklaracji zmiennej (czyli określenia jej przynależności, np. *Osoba(?ID-x) ^ swrlb:equal(?ID-x,4) → ...*),
- zalety ontologii są wykorzystywane w bardzo wąskim zakresie; ograniczono się wyłącznie do zastosowania jej w charakterze siatki pojęć i właściwości, jednak bez określania warunków relacji między nimi; opis relacji oraz ograniczenia zapisać można za pomocą reguł,
- jeśli chcemy wprowadzić warunki na przynależność jakiejś danej do klasy, należy zapisać odpowiednią regułę w języku SWRL (przykładowo reguła określająca przynależność do klasy *Kobieta* może wyglądać tak: *Osoby-Płeć(?ID-x, "K") → Kobieta(?ID-x)*),
- w zapisie ontologii nie specyfikuje się instancji – użyte mogą być tylko klasy, właściwości i reguły (OWL i SWRL),

- w zapisie klas nie precyzuje się warunków, jakie klasa musi spełniać, a także właściwości dotyczących tej klasy,
- w zapisie właściwości nie rozróżnia się zakresów domain i range,
- właściwości `DatatypeProperty` pochodzą wyłącznie z bazy danych i są poprzedzone prefiksem, informującym z jakiej tabeli pochodzą (przykładowo właściwość `Płeć`, pochodząca z bazy danych `Osoby` zostanie zaprezentowana następująco: `Osoby-Płeć`),
- istnieje możliwość dodania własnych właściwości `DatatypeProperty` w etapie drugim transformacji, pamiętając o przyjętych założeniach,
- właściwości `ObjectProperty` są tworzone podczas pracy nad ontologią w edytorze Protégé,
- dziedziczenie klas lub właściwości nie jest uwzględniane w transformacji ontologii OWL do formatu języka Jess Language,
- każda klasa, nie pochodząca z bazy danych, ale stworzona w narzędziu Protégé, posiada atrybut `Name`, który pozwala zidentyfikować instancję klasy, pod warunkiem, że jest unikalny (np. klucz główny pochodzący z tabeli relacyjnej);
- typy danych przechowywane w kolumnach nie mogą być postaci: `Image` i `BLOB`, gdyż późniejsza próba ich odwzorowania w silniku Jess jest niemożliwa,
- typ danych, przechowywanych w danej kolumnie, nie jest odwzorowywany, ponieważ obecna wersja biblioteki Jess 7.0p1 nie wymaga podania typu danych; dlatego też wszystkie dane traktuje się jako łańcuchy znaków, dla których można wykonywać wszystkie funkcje języka SWRLB zawarte w implementacji,
- wynikiem reguł SWRL jest dodanie nowej informacji – brak możliwości innych manipulacji, takich jak usunięcie czy modyfikacja,
- reguły SWRL są rozszerzone o funkcje wbudowane tzw. butli-ins, które zawierają ograniczenia nakładane na zmienne; obecnie, spośród 122 funkcji wbudowanych obsługiwane są wyłącznie następujące ograniczenia na zmienne:
 1. `swrlb:equal` – określa równość pomiędzy zmiennymi lub wartością i zmienną,
 2. `swrlb:notEqual` – brak równości,
 3. `swrlb:greaterThan` – większa,
 4. `swrlb:greaterThanOrEqual` – większa lub równa,
 5. `swrlb:lessThan` – mniejsza,
 6. `swrlb:lessThanOrEqual` – mniejsza lub równa,
- ograniczenia SWRLB należy podawać na końcu reguły zgodnie z kolejnością wystąpienia zmiennej w regule (nie można nadać więzów na zmienną, która nie jest wcześniej zdefiniowana),
- wnioskowanie dokonywane na danych to wnioskowanie w przód (sterowane danymi zgodnie z algorytmem Rete).

Etap pierwszy to automatyczna transformacja schematu relacyjnej bazy danych na klasy i właściwości, wyrażone w języku OWL. Transformacja jest możliwa poprzez podanie nazwy bazy danych. Wszystkie tabele wraz z kolumnami zostają odwzorowane w następujący sposób:

- nazwa tabeli jest odwzorowywana jako klasa,
- kolumny tabeli są odwzorowywane, jako atrybuty klasy w następujący schemat: klasa- atrybut, przykładowo: `Osoby-Płeć`,

- pomija się typy kolumn, ze względu na późniejsze odwzorowanie w silniku Jess; jest to jednak możliwe, lecz wymaga w etapie trzecim synchronizacji pomiędzy bazą danych, ontologią i silnikiem Jess.

Tak skonstruowane klasy i właściwości są zapisywane w pliku, w formacie języka OWL. Użytkownik musi jedynie utworzyć nazwę swojej ontologii przez podanie adresu URI oraz nazwę pliku docelowego. Przykładowa struktura przedstawiona została w Tabeli 1. Jest ona odwzorowywana na klasę *Osoby* oraz właściwości:

- Osoby-ID,
- Osoby-Imię,
- Osoby-Nazwisko,
- Osoby-Płeć,
- Osoby-Wiek,
- Osoby-IdOjca,
- Osoby-IdMatki,
- Osoby-PESEL.

Tabela 1. Schemat tabeli Osoby

ID	Imię	Nazwisko	Płeć	Wiek	IdOjca	IdMatki	PESEL
----	------	----------	------	------	--------	---------	-------

Etap drugi transformacji, nie może być zautomatyzowany. Polega on na tym, że inżynier wiedzy wczytuje wygenerowany plik OWL do edytora Protégé (lub dowolnego innego obsługującego formaty OWL i SWRL), a następnie tworzy własne klasy, właściwości oraz reguły w języku SWRL. Użytkownik może również tworzyć klasy złożone (na wzór klas pochodzących z bazy danych). W tym celu wystarczy podać nazwę klasy, a podczas definiowania właściwości typu *DatatypeProperty* poprzedzić jej nazwę, nazwą klasy ze znakiem „-”, przykładowo: klasa *Wujek* i atrybut *Wujek-mieszka*. Aby móc przejść do etapu trzeciego transformacji, należy przestrzegać, wcześniej przyjętych założeń, dotyczących tworzenia reguł SWRL i właściwości. Konkretnie wartości występujące w definicji reguł, muszą być ujęte w cudzysłów. Przykładowo, chcąc sprawdzić wartość właściwości *Osoby-Płeć*, zdefiniować można następującą regułę: *Osoby-Płeć(?ID-x, "K") → Kobieta(?ID-x)*.

Etap trzeci stanowi ostatni krok transformacji. Na tym etapie, pojęcia, właściwości oraz reguły, są zamieniane na format języka Jess. W pierwszej kolejności proces ten dotyczy klas oraz właściwości. Każda klasa jest zamieniana według następującego algorytmu.

1. Zapamiętaj nazwę klasy.
2. Jeśli istnieją jakiegokolwiek właściwości, które są powiązane z zapamiętaną nazwą klasy, to zapamiętaj je jako sloty (ang. *slots*); jeśli nie istnieją – utwórz jeden slot *name*.
3. Utwórz *deftemplate* o nazwie klasy wraz ze wszystkimi odpowiadającymi slotami (jednym *name* lub wieloma).

W ten sposób jest odzwierciedlona struktura bazy danych, w której *deftemplate* oznacza nazwę tabeli, a nazwa kolumny to odpowiedni slot. Podczas zamiany brane są wyłącznie właściwości typu *DatatypeProperty* (zgodnie z założeniami).

Następną częścią transformacji na etapie trzecim jest zamiana wszystkich właściwości niepowiązanych z bazą danych. Są to właściwości typu *ObjectProperty*, utworzone podczas rozwijania, wygenerowanej przez człowieka ontologii. Każda właściwość jest traktowana w ten sam sposób. Na początku tworzony jest *deftemplate* o nazwie właściwości, a następnie dwa sloty: *domain* i *range*, które odpowiadają *rdfs:domain* i *rdfs:range*. Ich ontologiczne znaczenie nie jest jednak

brane pod uwagę (wymagałoby to stworzenia nowych reguł, które dokładnie rozróżniają, co jest dziedziną, a co zakresem danej właściwości). Przykładowo właściwość *maDziadka* zostałaby zapisana następująco:

```
(deftemplate maDziadka
  (slot domain)
  (slot range))
```

Ostatnią, a zarazem najtrudniejszą część samej transformacji stanowi zamiana reguł zapisanych w języku SWRL na postać reguł języka Jess. Pomimo podobieństwa, związanego z ogólną budową reguł (warunki → konkluzja), ich zapis jest różny. Reguła, określająca relację posiadania córki, w języku SWRL wygląda następująco:

```
PESEL-IdOjca(?ID-x, ?IdOjca-z),
Kobieta(?ID-x)
→ maCorke(?IdOjca-z, ?ID-x)
```

natomiast w języku Jess Language prezentuje się ona jak poniżej:

```
(defrule MAIN::Def-maCorke
  (PESEL (IdOjca ?z) (ID ?x))
  (Kobieta (name ?x))
  =>
  (assert (MAIN::maCorke (domain ?z) (range ?x)))
)
```

Powyższe reguły zostały zapisane w konwencji, która została przyjęta w założeniach, dlatego można je odwzorowywać w prostszy sposób. Podczas zamiany reguł SWRL na Jess użyto bibliotek, dostarczonych z edytorem Protégé, z których wykorzystano głównie SWRLJessTab (służy do częściowej zamiany elementów języka OWL na Jess) oraz narzędzie Jena2 (służące do operacji dokonywanych na plikach w formacie OWL). Narzędzie SWRLJessTab przekształca reguły SWRL na swój własny wewnętrzny format. W związku z tym wystąpiła potrzeba zaimplementowania algorytmu, umożliwiającego dostosowanie formatu reguł SWRLJessTab do formatu Jess. Algorytm zamiany każdej reguły prezentuje się następująco.

1. Wczytaj regułę SWRL, zawartą w ontologii.
4. Transformuj jej postać na format i przy użyciu SWRLJessTab.
5. Transformuj regułę w formacie SWRLJessTab na format języka Jess.
6. Jeśli w zapisie reguły wystąpiły ograniczenia języka SWRLB, to dodaj je do utworzonej reguły Jess.
7. Na podstawie ciała reguły, wygeneruj zapytanie Jess, poprzedzone w nazwie literą „q” (od ang. *query*).
8. Dodaj regułę i zapytanie do przestrzeni roboczej silnika Jess.

Przykład działania algorytmu, funkcjonującego na regułach z ograniczeniami, można przedstawić na podstawie następującej reguły, zapisanej w języku SWRL:

```
Osoby-IdOjca(?ID-x, ?IdOjca-y),
Osoby-IdOjca(?ID-y, ?IdOjca-z),
Osoby-Wiek(?ID-z, ?Wiek-w),
swrlb:greaterThan(?Wiek-w, "50")
→ maDziadka(?ID-x, ?IdOjca-z)
```

Powyższa reguła dodaje do bazy wiedzy wszystkie pary osób typu „wnuk, dziadek”, gdzie hasło: dziadek określa osobę, która ma więcej niż 50 lat i posiada wnuka. Po zamianie tej reguły przez narzędzie SWRLJessTab wygląda ona następująco:

```
(defrule Def-Dziadek (Osoby-IdOjca ?ID-x ?IdOjca-y) (Osoby-IdOjca ?ID-y ?IdOjca-z) (Osoby-Wiek ?ID-z ?Wiek-w) (forceCall ?f:0&:(invokeSWRLBuiltIn Def-Dziadek swrlb:greaterThan ?Wiek-w "50")) => (assert (maDziadka ?ID-x ?IdOjca-z)) (assertOWLProperty "maDziadka" ?ID-x ?IdOjca-z) )
```

Elementy powyższej reguły są zamieniane na odpowiedniki w języku Jess, np. (*Osoby-IdOjca ?ID-x ?IdOjca-y*) zamieniane jest na (*Osoby (IdOjca ?y) (ID ?x)*). Należy to interpretować, jako fakt, iż osoba o identyfikatorze ?x ma ojca o identyfikatorze ?y. Elementy rozpoczynające się od wyrażenia *forceCall*, oznaczają wystąpienie ograniczenia SWRLB. Element ten jest usuwany i ulega transformacji na ograniczenie, wyrażone w języku Jess, czyli w tym przypadku: (*Wiek ?w&:(> ?w "50")*). Reszta reguły jest przepisywana do końca wraz z uwzględnieniem składni Jess'a, natomiast rezygnuje się z końcowej części reguły, czyli predykatów: *assertOWLProperty* lub *assertOWLIndividual*. Ostateczna postać reguły w formacie Jess wygląda następująco:

```
(defrule MAIN::Def-Dziadek
  (Osoby (IdOjca ?y) (ID ?x))
  (Osoby (IdOjca ?z) (ID ?y))
  (Osoby (Wiek ?w&:(> ?w "50")) (ID ?z))
  =>
  (assert (MAIN::maDziadka (domain ?x) (range ?z)))
)
```

Obecnie możliwa jest tylko operacja dodawania informacji do bazy wiedzy (*assert*), silnik Jess pozwala jednak dodawać, usuwać i modyfikować dane w pamięci roboczej. Wykorzystanie wszystkich dostępnych funkcjonalności języka Jess wymaga stworzenia własnego edytora reguł, w którym zwykły użytkownik mógłby tworzyć reguły i wnioskować, bez potrzeby znajomości języka Jess. Wystarczyłaby znajomość języków OWL i SWRL. Wskazany jest również stworzenie rozszerzenia dla narzędzia Protégé, które posiadałoby wspomnianą funkcjonalność.

Wszystkie etapy odwzorowania bazy relacyjnej oraz ontologii realizowane są za pomocą metod i funkcji, zaimplementowanych w bibliotece SDL. Biblioteka daje możliwość zastosowania jej nie tylko w proponowanej metodzie, ale również w innych podejściach, wymagających zamiany bazy danych na ontologię lub ontologii na język Jess. Aby transformacja przebiegała bez zakłóceń, należy przestrzegać, przedstawionych wcześniej, ograniczeń i zasad.

3. Charakterystyka wnioskowania hybrydowego

Zaproponowana metoda wnioskowania hybrydowego składa się z dwóch mechanizmów wnioskowania: wstecz i w przód. Wnioskowanie wstecz ma za zadanie zwiększyć wydajność działania biblioteki SDL. Pozwala pobierać z bazy danych informacje, które są istotne w trakcie analizy, przez co eliminuje potrzebę wczytywania wszystkich danych znajdujących się w bazie.

Hybrydę wnioskującą stanowią dwa silniki Jess, zawierające skrypty dotyczące wnioskowania w przód oraz wstecz. Ze względu na implementację narzędzia Jess oraz stosowane w nim algorytmy, niemożliwe jest uwzględnienie dodatkowych ograniczeń nakładanych na zmienne występujące w regułach. Stąd nastąpiła potrzeba dokonania podziału na silniki wnioskujące w przód oraz wstecz. Pierwszy z nich tworzony jest we wspomnianej wcześniej integracji. Proces tworzenia drugiego wykorzystuje ontologię wygenerowaną na etapie integracji. Etap zamiany ontologii na skrypt języka Jess dla procesu wnioskowania wstecz przebiega podobnie jak na etapie opisanym w integracji. Przeprowadzane są następujące modyfikacje:

- dla każdej definicji właściwości, która wymaga pobrania danej z bazy danych, tworzone jest proste zapytanie SQL; przykładowo dla reguły tworzącej właściwość *maOjca(?x, ?y)*, tworzone jest zapytanie SQL: *select * from PESEL where IdOjca = ?x;*
- generowane zapytania SQL mogą zawierać deklaracje zmiennych; ilość definiowanych zmiennych zależy od liczby zmiennych zawartych w regule (liczba zmiennych – 1);
- dla każdej takiej właściwości tworzone są 3 reguły: dwie odpowiedzialne za pobrane danych z bazy (odpowiednio dla *domain* i *range*) i ich dodanie do przestrzeni roboczej Jess oraz reguła definiująca samą właściwość; przykładowo dla właściwości *maOjca(?x, ?y)* zostaną utworzone następujące reguły:

```

(defrule MAIN::Def-maOjcaD
  (need-maOjca (domain ?y))
  =>
  (if (neq ?y nil) then (bind ?query (str-cat "select * from PESEL where
  IdOjca = " ' ?y ' ";""))
  (bind ?statement (?conn createStatement))
  (bind ?rs (?statement executeQuery ?query))
  (while (?rs next) do
    (assert (PESEL (ID (?rs getString ID))
      (Nazwisko (?rs getString Nazwisko))
      (Imie (?rs getString Imie))
      (Plec (?rs getString Plec))
      (PESEL (?rs getString PESEL))
      (IdOjca (?rs getString IdOjca))
      (IdMatki (?rs getString IdMatki))))
    )
  (?statement close))
  )

(defrule MAIN::Def-maOjcaR
  (need-maOjca (range ?x))
  =>
  (if (neq ?y nil) then (bind ?query (str-cat "select * from PESEL where
  ID = " ' ?x ' ";""))
  (bind ?statement (?conn createStatement))
  (bind ?rs (?statement executeQuery ?query))
  (while (?rs next) do
    (assert (PESEL (ID (?rs getString ID))
      (Nazwisko (?rs getString Nazwisko))
      (Imie (?rs getString Imie))
      (Plec (?rs getString Plec))
      (PESEL (?rs getString PESEL))
      (IdOjca (?rs getString IdOjca))
      (IdMatki (?rs getString IdMatki))))
    )
  (?statement close))
  )

(defrule MAIN::Def-maOjca
  (PESEL (IdOjca ?y) (ID ?x))
  =>
  (assert (MAIN::maDziecko (domain ?y) (range ?x)))
  )

```

- dla każdej definicji szablonu Jess (ang. *deftemplate*) dodaje się informację, która specyfikuje, że dany szablon uczestniczy tylko i wyłącznie w procesie wnioskowanie wstecz (polecenie *do-backward-chaining*).

Proces tworzenia skryptu silnika wnioskującego wstecz odbywa się automatycznie przy wykorzystaniu biblioteki SDL. Funkcje automatycznego utworzenia hybrydy wnioskującej dostępne są z poziomu języka Java i nie wymagają znajomości języka Jess.

Proces wnioskowania hybrydowego składa się z następujących etapów:

1. Uruchomienie zapytania (będącego w istocie ciałem reguły) komendą `run-query*` – użytkownik otrzyma odpowiedź (o ile znajduje się ona w bazie wiedzy znajdującej się w pamięci roboczej).
9. W przypadku braku odpowiedzi na zapytanie, zaktywuje ono odpowiednią regułę (lub listę reguł) znajdującą się w silniku wnioskującym wstecz. Następuje uruchomienie wnioskowania poprzez wydanie komendy `run` – uruchomienie reguł wnioskujących (pobranie potrzebnych krotek z bazy danych).
10. Ponowne uruchomienie zapytania komendą `run-query*` – w tym momencie użytkownik otrzyma odpowiedź na zadane zapytanie.

Etap 1 oraz 3 to etapy wnioskowania w przód, natomiast etap drugi to etap wnioskowania wstecz. Zadawane zapytania mają postać trójek RDF: obiekt, predykat, wartość. Przy czym należy pamiętać, iż wartością może być również inny obiekt lub zmienna.

Powyższa metoda posiada jednak poważne ograniczenie: proces wnioskowania wymaga podania konkretnej wartości dla konkretnej właściwości bądź pojęcia. Tylko na tej podstawie można rozpocząć wnioskowanie. W praktyce oznacza to, że niemożliwe jest zadanie zapytania dotyczącego spełnienia właściwości przez dowolne obiekty, a jedynie zapytanie czy dany obiekt spełnia właściwość. Przykładowo: nie można się zapytać o relację *maKuzyna*(?x, ?y). Można się jednak zapytać o spełnienia relacji kuzynostwa przez konkretną osobę: *maKuzyna*(„100800”, ?y) (gdzie 100800 jest identyfikatorem osoby w bazie danych).

4. Przykład zastosowania analizy semantycznej

W celu zademonstrowania metody wnioskowania hybrydowego została utworzona tabela relacyjna, której schemat został przedstawiony w Tabeli 1. Schemat relacyjny został przekształcony na ontologię OWL. Następnie ontologię wzbogacono o reguły i pojęcia dodatkowe. Całość przekształcono odpowiednio na dwa skrypty: dla silników wnioskujących w przód oraz wstecz. Tak skonstruowana hybryda pozwala na zadawanie dowolnych zapytań. Testy były przeprowadzane na bazach zawierającej 1000 i 300000 wierszy. Czasy odpowiedzi na zapytania zostały przedstawione w Tabeli 2. Wyniki prezentowane w odpowiedziach pochodzą z bazy zawierającej 1000 wierszy.

Tabela 2. Szybkość odpowiedzi na zapytanie

Baza	Zapyt.	maDziadka	maKuzyna
1000		120ms	5800ms
300000		150ms	68000ms

Pierwszy z dwóch zaprezentowanych przykładów pozwala znaleźć dziadków danej osoby. W pierwszej kolejności zostały zaprezentowane reguły specyfikujące relację *maDziadka*. W relacji tej została wykorzystana również relacja *maOjca* przedstawiona w punkcie 2. Występująca w regułach końcówka „B” oznacza silnik wnioskujący wstecz, końcówka „F” – silnik wnioskujący w przód. Ze względów objętościowych zostaną pominięte reguły wnioskowania wstecz, których aktywatorem jest wartość atrybutu *range*. W większości przypadków pominięte zostały również reguły, których aktywatorem jest wartość atrybutu *domain*. Wynika to z podobieństwa reguł, które jedyną różnicą są drobne modyfikacje zapytań SQL. Reguły definiujące relację *maDziadka* prezentują się następująco:

```
(defrule MAIN::Def-maDziadka1
  (PESEL (IdOjca ?y) (ID ?x))
  (PESEL (IdOjca ?z) (ID ?y))
  =>
  (assert (MAIN::maDziadka (domain ?x) (range ?z)))
)
(defrule MAIN::Def-maDziadka2
  (PESEL (IdMatki ?y) (ID ?x))
  (PESEL (IdOjca ?z) (ID ?y))
  =>
  (assert (MAIN::maDziadka (domain ?x) (range ?z)))
)
(defrule MAIN::Def-maDziadka1B
  (need-maDziadka (domain ?x))
  =>
  (if (neq ?x nil) then (bind ?query (str-cat
    "declare @z varchar(100)"
```

```

        "select @z=IdOjca from PESEL where ID=" ' ?x ' ";"
        "select * from PESEL where ID = @z;")
(bind ?statement (?conn createStatement))
  (bind ?rs (?statement executeQuery ?query))
  (while (?rs next) do
    (assert (PESEL (ID (?rs getString ID))
                  (Nazwisko (?rs getString Nazwisko))
                  (Imie (?rs getString Imie))
                  (Plec (?rs getString Plec))
                  (PESEL (?rs getString PESEL))
                  (IdOjca (?rs getString IdOjca))
                  (IdMatki (?rs getString IdMatki))))
    )
  )
  (?statement close))
)

```

Definicja specyfikująca zapytanie prezentują się następująco:

```

(defquery MAIN::qDef-maDziadka
  (declare (variables ?x))
  (maDziadka (domain ?x) (range ?y))
)

```

Zapytanie o dziadków osoby o ID = „100831” zadane z poziomu języka Java:

```

ValueVector vector = new ValueVector();
vector.add(new Value("100831", 2));
Hybrid.runQuery("qDef-maDziadka", vector);

```

Oraz odpowiedzi:

```

maDziadka: domain 100831 range 100247
maDziadka: domain 100831 range 100236

```

Drugi przykład pozwala na znalezienie osób będących w relacji kuzynostwa, przy czym relacja ta jest spełniona tylko i wyłącznie przez osoby, których rodzice są rodzeństwem. Takie założenie wymagało wprowadzenia nowych pojęć: *maDziecko* oraz *maRodzenstwo*. Poniżej znajdują się reguły definiujące wszystkie relacji wykorzystywane w procesie wnioskowania:

```

(defrule MAIN::Def-maDziecko1
  (PESEL (IdOjca ?y) (ID ?x))
  =>
  (assert (MAIN::maDziecko (domain ?y) (range ?x)))
)
(defrule MAIN::Def-maDziecko2
  (PESEL (IdMatki ?y) (ID ?x))
  =>
  (assert (MAIN::maDziecko (domain ?y) (range ?x)))
)
(defrule MAIN::Def-maDziecko1B
  (need-maDziecko (domain ?y))
  =>
  (if (neq ?y nil) then (bind ?query (str-cat "select * from PESEL where IdO-
jca = " ' ?y ' ";"))
  (bind ?statement (?conn createStatement))
  (bind ?rs (?statement executeQuery ?query))
  (while (?rs next) do
    (assert (PESEL (ID (?rs getString ID))
                  (Nazwisko (?rs getString Nazwisko))
                  (Imie (?rs getString Imie))
                  (Plec (?rs getString Plec))
                  (PESEL (?rs getString PESEL))
                  (IdOjca (?rs getString IdOjca))
                  (IdMatki (?rs getString IdMatki))))
    )
  )
  (?statement close))
)
(defrule MAIN::Def-maRodzenstwo1
  (PESEL (ID ?x) (IdOjca ?z))
  (PESEL (ID ?y) (IdOjca ?z))
)

```

```

=>
(assert (MAIN::maRodzenstwo (domain ?x) (range ?y)))
)

(defrule MAIN::Def-maRodzenstwo2
  (PESEL (ID ?x) (IdMatki ?z))
  (PESEL (ID ?y) (IdMatki ?z))
  =>
  (assert (MAIN::maRodzenstwo (domain ?x) (range ?y)))
  )
(defrule MAIN::Def-maRodzenstwo1
  (need-maRodzenstwo (domain ?x))
  =>
  (if (neq ?x nil) then (bind ?query (str-cat
    "declare @z varchar(100), @y varchar(100)"
    "select @z=IdOjca from PESEL where ID=" ' ' ?x ' ' ";"
    "select * from PESEL where IdOjca = @z;"))
  (bind ?statement (?conn createStatement))
  (bind ?rs (?statement executeQuery ?query))
  (while (?rs next) do
    (assert (PESEL (ID (?rs getString ID))
      (Nazwisko (?rs getString Nazwisko))
      (Imie (?rs getString Imie))
      (Plec (?rs getString Plec))
      (PESEL (?rs getString PESEL))
      (IdOjca (?rs getString IdOjca))
      (IdMatki (?rs getString IdMatki))))
    )
  (?statement close))
  )
(defrule MAIN::Def-maRodzenstwo1F
  (PESEL (IdOjca ?z) (ID ?x))
  (PESEL (IdOjca ?z) (ID ?y&:(neq ?y ?x)))
  =>
  (assert (MAIN::maRodzenstwo (domain ?x) (range ?y)))
  )
(defrule MAIN::Def-maKuzyna1
  (maDziecko (domain ?z) (range ?x))
  (maRodzenstwo (domain ?z) (range ?w))
  (maDziecko (domain ?w) (range ?y))
  =>
  (assert (MAIN::maKuzyna (domain ?x) (range ?y)))
  )
(defrule MAIN::Def-maKuzyna1B
  (need-maKuzyna (domain ?x) )
  (maDziecko (domain ?z) (range ?x))
  (maRodzenstwo (domain ?z) (range ?w))
  (maDziecko (domain ?w) (range ?y))
  =>
  (assert (MAIN::maKuzyna (domain ?x) (range ?y)))
  )
(defrule MAIN::Def-maKuzyna1F
  (maRodzenstwo (domain ?z) (range ?w))
  (maDziecko (domain ?z) (range ?x))
  (maDziecko (domain ?w) (range ?y&:(neq ?y ?x)))
  =>
  (assert (MAIN::maKuzyna (domain ?x) (range ?y)))
  )

```

Przykładowe zapytanie o kuzynów osoby o ID = „100831”:

```

ValueVector vector = new ValueVector();
vector.add(new Value("100831", 2));
Hybrid.runQuery("qDef-maKuzyna", vector);

```

Odpowiedzi:

```

maKuzyna : domain 100831 range: 100834
maKuzyna : domain 100831 range: 100832
maKuzyna : domain 100831 range: 100833

```

```
maKuzyna : domain 100831 range: 100826
maKuzyna : domain 100831 range: 100560
maKuzyna : domain 100831 range: 100559
maKuzyna : domain 100831 range: 100561
maKuzyna : domain 100831 range: 100580
maKuzyna : domain 100831 range: 100579
maKuzyna : domain 100831 range: 100578
maKuzyna : domain 100831 range: 100581
maKuzyna : domain 100831 range: 100577
maKuzyna : domain 100831 range: 100802
maKuzyna : domain 100831 range: 100803
maKuzyna : domain 100831 range: 100755
maKuzyna : domain 100831 range: 100754
maKuzyna : domain 100831 range: 100753
maKuzyna : domain 100831 range: 100756
maKuzyna : domain 100831 range: 100752
maKuzyna : domain 100831 range: 100498
maKuzyna : domain 100831 range: 100497
```

W tym miejscu należy podkreślić zasadniczą różnicę pomiędzy zapytaniem realizowanym w opisany powyżej sposób, a standardowym zapytaniem SQL. W przypadku SQL zapytanie może dotyczyć jedynie relacji, które mają swoje odbicie w tabelach w bazie danych. Natomiast przy użyciu silnika wnioskującego i języka reguł możemy zadawać zapytania o relacje, które nie są bezpośrednio odzwierciedlone w bazie danych.

5. Podsumowanie

Wnioskowanie hybrydowe ma za zadanie zwiększyć wydajność procesu wnioskowania silnika Jess wykorzystującego relacyjną bazę danych. Zastąpienie potrzeby wczytania wszystkich danych z bazy, trybem „na żądanie”, pozwala na kilkunastokrotne zmniejszenie czasu potrzebnego na uzyskanie odpowiedzi. Zaletą takiego podejścia jest skalowalność. Wynika ona z faktu, iż nie jest wczytywana cała baza danych, a jedynie jej fragmenty istotne w procesie wnioskowania.

Pierwsza implementacja przygotowana przez nas oraz przeprowadzone testy wydajnościowe wskazują, że połączenie technologii relacyjnych baz danych z silnikiem wnioskującym przy użyciu ontologii tworzy nową jakość i dostarcza narzędzia, które pozwala na efektywne zadawanie bardzo złożonych zapytań do bazy danych. Wnioskowanie hybrydowe pozwala na usunięcie wad istniejących silników wnioskujących takich jak np. KAON, KAON2 czy JENA, których szybkość w wielu przypadkach jest niezadowalająca. Wykorzystywany w pracy Jess jest uznawany za jeden z najszybszych silników wnioskujących. Ze względu na wykorzystywany algorytm RETE fakty poddawane procesowi wnioskowania muszą się znajdować w pamięci RAM. Powoduje to, że silnik ten może działać na relatywnie małej porcji danych. Zastosowanie wnioskowania hybrydowego usuwa wspomnianą niedogodność.

Zaprezentowany silnik wnioskujący znajdzie zastosowanie w problemach wymagających stosowania reguł. W regułach występują zmienne, które należy zwartościować w celu uzyskania odpowiedzi. Zmienne te są niezależne od siebie. Niektóre problemy wymagają dużej liczby zmiennych wzajemnie od siebie zależnych. Przykładem może być tzw. Zagadka Einsteina [Einst], której rozwiązania prezentowane są w językach regułowych (np. Prolog). Silnik Jess również pozwala na rozwiązanie wspomnianej zagadki. W tym przypadku zbiór danych stanowią wszystkie możliwe kombinacje trójek typu: relacja (numer_domu, wartość relacji), np. zamieszkuje(„1”, „Norweg”). Dotychczasowe prace pozwoliły na uzyskanie skryptu silnika umożliwiającego rozwiązanie zagadki w trybie wnioskowania w przód. Odpowiedź generowana jest za pomocą jednej reguły silnika Jess. Trwają intensywne prace umożliwiające rozwiązanie problemu przy wykorzystaniu relacyjnej bazy danych oraz hybrydy wnioskującej.

Obecnie autorzy pracują nad praktycznym zastosowaniem opisanej metody w projekcie systemu eksperckiego wspomagającego analizę dowodów w postępowaniu przygotowawczym prowa-

dzonym w sprawach gospodarczych. Kolejnym etapem prac nad wnioskowaniem hybrydowym będzie dalsza poprawa jego efektywności ze szczególnym naciskiem na poprawę mechanizmu wnioskowania wstecz i ściągnięcia danych do pamięci w trybie „na żądanie”.

Praca ta została sfinansowana ze środków na naukę w latach 2006-2009 jako projekt badawczy rozwojowy "Narzędzie wspomagające procedury śledcze wykorzystujące automatyczne wnioskowanie" oraz przez grant Politechniki Poznańskiej 45-083/07/BS.

Bibliografia

- [Einst] http://pl.wikipedia.org/wiki/Zagadka_Einsteina
- [Frie03] Friedman-Hill E., „Jess in Action”, 2003, Manning Publications Co.
- [Jess] Java Expert System Shell, <http://www.jessrules.com/>
- [KAON2] <http://kaon2.semanticweb.org/>
- [OWL] Web Ontology Language (OWL) Guide Version 1.0, <http://www.w3.org/TR/owl-features/>
- [OWLJK] <http://edge.cs.drexel.edu/assemblies/software/owljesskb/>
- [Protégé] Protégé Editor, <http://protege.stanford.edu/>
- [RDF] RDF, <http://www.w3.org/RDF/>
- [Rete] Algorytm Rete w narzędziu Jess, <http://herzberg.ca.sandia.gov/jess/docs/70/rete.html>
- [SwRu] SweetRules, <http://sweetrules.projects.semwebcentral.org/>
- [SWRL] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, <http://www.w3.org/Submission/SWRL/>