Opracowanie i implementacja podstaw optymalnego przechowywania i transferowania danych medycznych z użyciem ontologii

Łukasz Czyżycki
ciziu@student.agh.edu.pl

Abstrakt – Artykuł opisuje projekt i implementację podstaw przykładowego systemu wykorzystującego ontologie do przechowywania danych medycznych i ich transferu pomiędzy zainteresowanymi podmiotami. Na potrzeby systemu zostały zaprojektowane ontologie zgodne ze standardem *OWL* oraz wykonano aplikacje w języku *Java* korzystające z API udostępnianego przez bibliotekę *Apache Jena*. Do przechowywania danych wykorzystano relacyjną bazę danych *MySQL*, za komunikację z którą odpowiada część biblioteki Apache Jena o nazwie *SDB*, oraz uniwersalny sterownik *JDBC*.

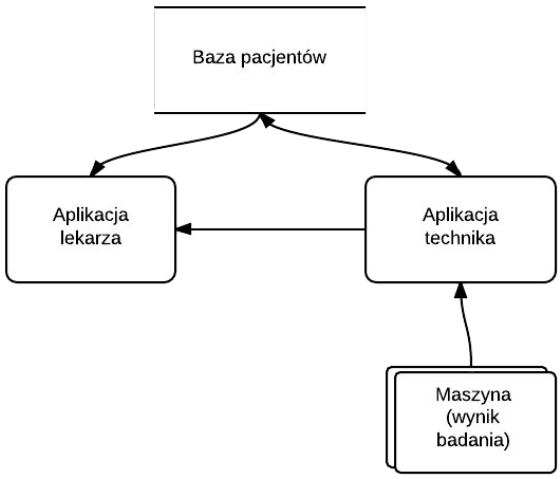
Słowa kluczowe: ontologie, OWL, Apache Jena, onotologie medyczne, relacyjne bazy danych

1. Wstęp

Ontologie, czyli specyfikacja konceptualizacji, reprezentująca domenę wiedzy i dająca możliwośc wykorzystania jej później [1] to interesujący i dynamicznie rozwijający się sposób przechowywania wiedzy w formie elektronicznej. Dzięki mocnej standaryzacji i elastycznej strukturze definicji i powiązań danych, mają do zaoferowania szereg obiecujących możliwości, nieosiągalnych dla danych zamkniętych w bardziej hermetycznych strukturach, takich jak choćby bazy relacyjne. Choć wydajność ontologii wciąż mocno ustępuje konkurencyjnym rozwiązaniom [2], to rosnące możliwości obliczeniowe komputerów, w połączeniu z intensywnym rozwojem bibliotek i technologii pozwalających na tworzenie systemu korzystających z tego typu baz wiedzy, stanowi wystarczającą zachętę to zagłębienia się w ten niewątpliwie przyszłościowy temat.

2. Architektura systemu

W ramach opisywanego projektu podjęto próbę implementacji systemu przedstawionego na **Obrazie 1**.



***Obraz 1**. Architektura systemu*

Stworzono dwie aplikacje. Pierwszą z przeznaczeniem dla technika dokonującego badania medycznego, który chce opatrzyć je informacjami na temat pacjenta na którym dokonano badania oraz własnymi danymi osobowymi, by umożliwić bezproblemowy kontakt w wypadku pojawienia się jakichkolwiek niejasności związanych z badaniem. Druga aplikacja ma stanowić narzędzie dla lekarza medycyny, który z jej pomocą może odebrać dane od technika dokonującego badania, zapoznać się z jego wynikami oraz na podstawie swoich obserwacji wystawić diagnozę, opatrzoną oczywiście jego danymi osobowymi. Obie aplikacje mają możliwość kontaktu z bazą pacjentów, jednak jedynie aplikacja technika ma możliwość jej trwałej modyfikacji, poprzez dodawanie do niej nowych, wcześniej nie badanych w ramach systemu, osób. Dane transportowane w systemie ewoluują w kolejnych krokach przebywanej ścieżki. Badanie z maszyny oczekiwane jest w prostym formacie binarnym, właściwym dla danego urządzenia. Na ścieżce wymiany informacji między technikiem a lekarzem oczekiwana już jest jednak w pełni ustrukturyzowana informacja,

zapisana w podlegającym standardom formacie i zawierająca nie tylko wynik badania, ale moment jego wykonania, identyfikator umożliwiający jednoznaczne wskazanie badanego pacjenta oraz komplet danych osobowych technika dokonującego badania.

3. Struktura ontologii

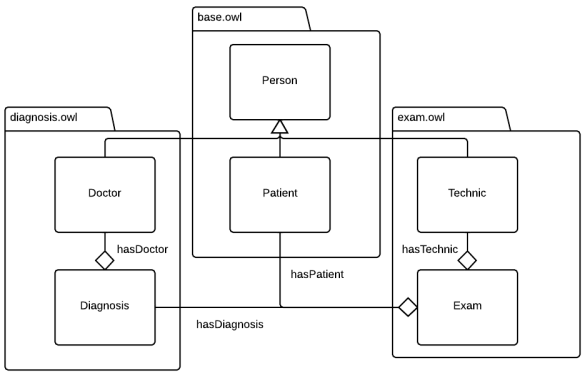
Kluczowym elementem potrzebnym do uzyskania efektywnego systemu przechowywania i transferu informacji było zaprojektowanie odpowiedniej struktury budowanych ontologii, tak by oferowały równocześnie prostotę pozwalającą na stworzenie przejrzystego i wydajnego rozwiązania, jak i złożoność konieczną do zawarcia wszystkich istotnych danych i umożliwienia zwartej selekcji istotnych dla danego podmiotu informacji, na danym etapie drogi przez nią przebywanej.

Istotne z punktu widzenia wydajności i przejrzystości, była też taka strukturyzacja bazy wiedzy, żeby zarówno informacje o jednostkach, jak i strukturze części ontologii były dodawane dopiero w momencie, w którym faktycznie są potrzebne. W ten sposób można było mocno ograniczyć ilość przesyłanych danych, zachowując przy tym wszystkie istotne informacje.

Ostatecznie, z użyciem aplikacji *Protege*, zaprojektowane zostały trzy wzajemnie powiązane ontologie *OWL*, pozwalające zawrzeć kolejno:

- informacje przechowywane w bazie pacjentów *(base.owl)*
- informacje przesyłane przez technika lekarzowi *(exam.owl)*
- całość informacji zapisanych przez lekarza *(diagnosis.owl)*

Strukturę klas w tych ontologiach, wraz z podziałem między poszczególne pliki, pokazuje **Obraz 2**.



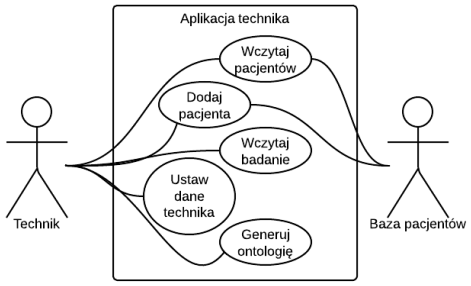
Obraz 2. *Struktura klas w ramach ontologii*

Przygotowanie ontologii w aplikacji takiej jak *Protege*, umożliwiającej łatwą kontrolę nad powiązaniami danych, ich typem i nazewnictwem, z poziomu przejrzyście zaprojektowanego interefejsu graficznego, pozwoliło w trakcie prac nad projektem na szybkie wprowadzanie potrzebnych korekt w ontologiach, co byłoby zadaniem dalece bardziej uciążliwym jeżeli pliki *OWL* byłyby pisane od podstaw w języku *XML* lub tworzone dynamicznie z użyciem programistycznego API.

4. Aplikacja technika

W wyniku przeglądu dostępnych otwartych bibliotek do odczytu, manipulacji i budowy ontologii, zidentyfikowano *Apache Jena* jako najbardziej kompletne, rozbudowane i przejrzyste z rozwiązań. Jako że sama biblioteka została napisana w języku *Java*, zdecydowano się na użycie tego właśnie języka programowania do wykonania aplikacji technika. Jako środowisko programistyczne, jak i również narzędzie do budowy interfejsu graficznego, wybrany został popularny pakiet *Netbeans*.

Praca z programem opiera się na wyborze pacjenta z bazy lub dodaniu nowego, wyborze pliku badania oraz uzupełnieniu danych technika, a następnie generacji ontologii *OWL* zawierającej komplet informacji o badaniu. Komplet funkcjonalności programu prezentuje diagram *UML* na **Obrazie 3**.



Obraz 3. *Przypadki użycia dla aplikacji technika*

4.1. Przechowywanie danych pacjentów

Do realizacji trwałego przechowywania bazy pacjentów został użyty moduł *SDB* wchodzący w skład *Jeny*. Pozwala on na stworzenie z poziomu API biblioteki modelu ontologii, którego zawartość jest automatycznie, obustronnie synchronizowana z relacyjną bazą danych. *SDB* pozwala na wykorzystanie wielu popularnych implementacji bazodanowych, dla których istnieje sterownik *JDBC*. W opisywanym projekcie zdecydowano się na użycie bazy *MySQL* ze względu na jej dużą popularność i doświadczenie zespołu w pracy z nią.

Aplikacja przy starcie sprawdza czy baza jest poprawnie sformatowana i znajduje się w niej ontologia pacjentów, a następnie wczytuje tę ontologię do programu. W przeciwnym wypadku baza jest formatowana z użyciem zapisanej w pliku ontologii *base.owl*. Proces ten obrazuje poniższy fragment kodu źródłowego:

```
dbStore =
SDBFactory.connectStore("res/dbconf.ttl");
```

```
// Jeżeli baza jest niesformatowana, wczytujemy
ontologię z pliku

if (!StoreUtils.isFormatted(dbStore))

{

	dbStore.getTableFormatter().format();

	baseModel =
SDBFactory.connectDefaultModel(dbStore);

	ontModel =
ModelFactory.createOntologyModel(OntModelSpec.OW
L_DL_MEM, baseModel);

	ontModel.read("res/base.owl");

}

// W przeciwnym wypadku czytamy model z bazy

else

{

	baseModel =
SDBFactory.connectDefaultModel(dbStore);

	ontModel =
ModelFactory.createOntologyModel(OntModelSpec.OW
L_DL_MEM, baseModel);

}
```

4.2. Dodawanie pacjentów do bazy

Aplikacja umożliwia technikowi dodanie pacjenta do bazy poprzez proste okno programu zawierające pola do uzupełnienia wszelkimi danymi przechowywanymi w ontologii. Wpisane dane są opakowywane w klasie *PatientData*, której struktura pól odzwierciedla dane klasy pacjenta w ontologii, a następnie przesyłane do klasy *PatientStorage*, gdzie dochodzi do dodania pacjeta do ontologii poprzez następujący kod programu:

```
public void AddPatient(PatientData newPatient)

{

	Individual ontPatient =


	ontModel.createIndividual(NS +
newPatient.peselNumber, patientClass);

	ontPatient.addLiteral(firstNameProp,
newPatient.firstName);

	ontPatient.addLiteral(surnameProp,
newPatient.surname);

	ontPatient.addLiteral(emailProp,
newPatient.email);

	ontPatient.addLiteral(phoneNumberProp,
newPatient.phoneNumber);

	ontPatient.addLiteral(peselNumberProp,
newPatient.peselNumber);

}
```

4.3. Dane technika i wynik badania

Aplikacja posiada osobne okno pozwalające technikowi wprowadzić swoje dane osobowe, które później są oczywiście zawierane w generowanej, finalnej wersji ontologii. Dla wygody użytkownika, informacje te są zapamiętywane pomiędzy uruchomieniami aplikacji poprzez zawarty w bibliotece standardowej języka *Java* mechanizm preferencji.

Wybierając odpowiednią opcję w aplikacji, technik ma możliwość wczytania z dysku twardego pliku badania. Wyświetlane jest standardowe, systemowe okno wyboru pliku, umożliwiające przeglądanie katalogów i wybór dokładnej lokalizacji zapisu badania, który jest oczywiście potem zawierany w całości w generowanej ontlogii.

4.4. Generacja ontologii wynikowej

Po wprowadzeniu kompletu danych, użytkownik ma możliwość wygenerowanie wynikowej ontologii, zawierającej komplet danych potrzebnych do wysłania lekarzowi.

Całość generacji ontologii została zawarta w metodzie statycznej *GenerateExamOntology* klasy *ExamOntologyGenerator*. Metoda ta rozpoczyna od wczytania struktury ontologii z przygotowanego pliku *exam.owl*. Następnie generowany jest egzemplarz klasy technika i dodawane są wszelkie jego dane osobowe:

```
Individual technic =
examModel.createIndividual(NS +
techData.idNumber, examModel.getOntClass(NS +
"Technic"));

technic.addLiteral(examModel.getDatatypeProperty
(baseNS + "first_name"), techData.firstName);

technic.addLiteral(examModel.getDatatypeProperty
(baseNS + "surname"), techData.surname);

technic.addLiteral(examModel.getDatatypeProperty
(baseNS + "email"), techData.email);

technic.addLiteral(examModel.getDatatypeProperty
(baseNS + "phone_number"),
techData.phoneNumber);

technic.addLiteral(examModel.getDatatypeProperty
(NS + "id_number"), techData.idNumber);
```

Po dodaniu do ontologii informacji o techniku, dodawany jest egzemplarz klasy badania, wraz z wszystkimi informacjami o nim:

```
Individual exam =
examModel.createIndividual(examModel.getOntClass
(NS + "Exam"));

// Ustawiamy relację z technikiem

exam.addProperty(examModel.getObjectProperty(NS
+ "hasTechnic"), technic);

// Relację z wybranym pacjentem
```

```
exam.addProperty(examModel.getObjectProperty(NS
+ "hasPatient"),

               examModel.getIndividual(baseNS +
selPatient.peselNumber));

// Zapisujemy aktualną datę i czas, w formacie
zgodnym z XMLSchema

exam.addProperty(examModel.getDatatypeProperty(N
S + "timestamp"),

               new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss").format(new Date()));


// Wczytujemy bajtową zawartość pliku, kodujemy
ją za pomocą base64 i dodajemy do badania

try {

       byte[] bytes =
Files.readAllBytes(examFile.toPath());


exam.addProperty(examModel.getDatatypeProperty(N
S + "data"), Base64.encode(bytes));

} catch (IOException ex) {


Logger.getLogger(ExamOntologyGenerator.class.get
Name()).log(Level.SEVERE, null, ex);

}
```

Jak widać w załączonym powyżej fragmencie kodu, binarne dane pliku badania są kodowane z użyciem popularnego szyfrowania *Base64*, w efekcie czego otrzymywany jest łańcuch znakowy, który można zawrzeć w poprawnym pliku *XML* nie naruszając jego wewnętrznej struktury, i bezproblemowo zapisać jako plik tekstowy.

Do ontologii dodawany jest również aktualny czas, stanowiący informację dla lekarza o momencie skompletowania ogółu informacji o badaniu. Upewniono się by format tego zapisu był w pełni zgodny ze standardami zapisu czasu w plikach zgodnych z *XMLSchema* [3]. Oczywiście zapisanie tej informacji w żaden sposób nie uniemożliwia zawarcia w binarnym pliku badania czasu sporządzenia samego pliku binarnego.

4.5. Testy wydajności aplikacji

W celach informacyjnych, przeprowadzono proste testy wydajności fragmentów stworzonej aplikacji. Środowisko testów charakteryzują następujące parametry:

- System *Windows 7 x64 Professional*
- Maszyna wirtualna *Javy* w wersji *JRE7*
- Procesor *Intel Core i5 M450* 2,4 GHz (2 rdzenie)
- Pamięć operacyjna RAM – 4 GB

Prosta baza zawierająca informacje o 5 pacjentach została wczytana z ontologii do prostej struktury danych języka *Java* w czasie 73061493 ns (ok. 73 ms).

Dodanie nowego pacjenta do ontologii zostało wykonane w czasie 350722024 ns (ok. 351 ms).

Wygenerowanie przykładowej, zawierającej komplet danych pojedynczego badania, ontologii wynikowej zajęło 134087196 ns (ok. 134 ms), z czego 69478211 ns (ok. 69 ms) zajęło zakodowanie z użyciem *Base64* i dodanie do ontologii pliku binarnego. Oczywiście wartość ta jest ściśle uzależniona od rozmiaru wspomnianego pliku (w tym teście został użyty niewielki plik o rozmiarze 6 KB).

Z wyników testów jasno wynika, że wąskie gardło takiego systemu stanowić będą zawsze operacje modyfikacji na bazie danych, gdyż obarczone są one zakładaniem tymczasowych blokad w bazie w celu zagwarantowania jej pełnej transakcyjności.

5. Podsumowanie

Użycie ontologii do przechowywania i transferu danych medycznych okazało się być wartym uważnego zainteresowania pomysłem. Mimo ich strukturze, nieprzekładającej się bezpośrednio na możliwości zapisu w relacyjnych bazach danych, wydajność uzyskana w tym prostym projekcie była w pełni satysfakcjonująca. Ogrom możliwości, przejrzyste API i wyczerpująca dokumentacja oferowana przez *Apache Jena* sprawia, że implementacja w tworzonej aplikacji nawet złożonej ontologii nie stanowi olbrzymiego przedsięwzięcia w skali programistycznej. Interesujące możliwości przetwarzania które na ontologiach *OWL*-owych oferuje szereg dostępnych publicznie programów, czyni z nich opcje wartą rozważenia przy wyborze sposobu implementacji dużego systemu.

Bibliografia

1. Robal, T.; Kann, T.; Kalja, A., "An ontology-based intelligent learning object for teaching the basics of digital logic," *Microelectronic Systems Education (MSE), 2011 IEEE International Conference on* , vol., no., pp.106,107, 5-6 June 2011
2. Agostini, A.; Bettini, C.; Riboni, D., "A Performance Evaluation of Ontology-Based Context Reasoning," *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07. Fifth Annual IEEE International Conference on* , vol., no., pp.3,8, 19-23 March 2007
3. *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes* http://www.w3.org/TR/xmlschema11-2/ (stan na dzień 06.12.2013)