Bartosz Kukawka 75911
Marcin Mikołajczak 75922

Diagnostyka obrazowa przez WWW

Spis treści

1 Zarys projektu

1.1 Opis problemu

Pliki graficzne powstające podczas eksploatacji urządzeń medycznych mogą mieć rozmiary dochodzące do kilkudziesięciu tysięcy pikseli w obu wymiarach, co jest równoznaczne z zajęciem przez jedno skompresowane zdjęcie nawet do 1GB przestrzeni dyskowej. Podstawową trudnością w przetwarzaniu obrazów o takich rozmiarach jest brak możliwości ich wczytania do pamięci operacyjnej komputera w całości. Ponadto, nawet przy wykorzystaniu współczesnych, szybkich sieci komputerowych, dostarczanie takich ilości danych do specjalisty pragnącego dokonać na ich podstawie diagnozy jest co najmniej kłopotliwe. Tradycyjne podejście, polegające na kopiowaniu obrazu na stację roboczą, wprowadza też niepotrzebne, duże wymagania na jej przestrzeń dyskową, jak również problemy związane z ochroną danych. Dobrym rozwiązaniem wymienionych problemów wydaje się stosowanie w jednostkach medycznych centralnych repozytoriów obrazów, obsługiwanych przez serwery o znacznej powierzchni dyskowej i odpowiednich zabezpieczeniach. Dalsze usprawnienie interakcji z obrazami medycznymi może polegać na udostępnieniu ich przy pomocy interfejsu WWW, co przynosi dodatkowe korzyści, takie jak niższe koszty administracji stacjami roboczymi w jednostce oraz łatwiejszy zdalny dostęp do danych przez sieć Internet.

1.2 Cel projektu

Celem projektu było stworzenie prototypowej implementacji opisanej wyżej wizji systemu dostępu do obrazów medycznych, przy szczególnym uwzględnieniu klienckiej aplikacji WWW. Innowacyjność projektu polegać miała na zastosowaniu jako podstawy interfejsu kontrolki Google Maps[1], stworzonej oryginalnie w celu udostępnienia bogatych rastrowych danych kartograficznych poprzez Internet przy zachowaniu tzw. rich user experience. Z technicznego punktu widzenia kontrolka ta realizuje dokładnie takie same funkcje, jakie potrzebne są do efektywnego przeglądania dużych obrazów medycznych. Kontrolka wyświetla w danej chwili obraz rastrowy na jednym z wielu poziomów przybliżenia, a każdy kolejny poziom przedstawia obraz powiększony dwukrotnie względem poprzedniego. Na stację roboczą pobierane są jedynie fragmenty obrazu potrzebne do wyrysowywania części widocznej w danej chwili na ekranie. W momencie przesunięcia obrazu przez użytkownika automatycznie pobierane są kolejne potrzebne fragmenty. Obsługa kontrolki odbywa się w sposób standardowy dla tego typu operacji – przesuwanie obrazu następuje poprzez mechanizm drag-and-drop, a przybliżanie i oddalania przy pomocy rolki urządzenia wskazującego. W przypadku braku tego ostatniego elementu możliwe jest także skorzystanie z przycisków „plus” i „minus”.

W ramach realizacji projektu zostało zaplanowane zbudowanie aplikacji umożliwiającej:

a) przeglądanie listy obrazów zorganizowanej w strukturę drzewa,
b) przeglądanie wybranego obrazu przy pomocy kontrolki Google Maps,
c) zarządzanie adnotacjami do przeglądanego w danej chwili obrazu.

[1] Google Maps – http://maps.google.com
Google Maps API – http://code.google.com/apis/maps/

Zarządzanie adnotacjami oznacza w tym przypadku zaznaczanie na obrazie obszarów będących przedmiotem adnotacji i dodawanie do nich opisów tekstowych oraz późniejszą edycję i usuwanie tych opisów wraz z odpowiadającymi im obszarami.

1.3 Harmonogram prac

Oryginalny harmonogram prac przewidywał następujące punkty kontrolne:

- 26.03.2008 – opracowanie specyfikacji protokołu Google Maps ,
- 09.04.2008 – wykonanie próbnego klienta Google Maps,
- 16.04.2008 – opracowanie interfejsu użytkownika aplikacji webowej,
- 23.04.2008 – opracowanie architektury aplikacji serwerowej,
- 07.05.2008 – prototyp aplikacji serwerowej i webowej,
- 21.05.2008 – oddanie kompletnego projektu wraz ze sprawozdaniem.

Z powodu udziału autorów sprawozdania w konkursie ImagineCup terminy te za uzgodnieniem z prowadzącym przedmiot uległy nieznacznym przesunięciom. Dokumenty wymienione w harmonogramie zostały zawarte w niniejszym sprawozdaniu lub stanowią załącznik do niego. Ostateczny termin oddania projektu został ustalony na 4 czerwca 2008.

2 Przyjęte rozwiązania

2.1 Organizacja obrazów

Ze względu na scentralizowany charakter budowanego systemu za słuszne zostało uznane opracowanie ustandaryzowanej struktury katalogów i plików, w której przechowywane będą na serwerze obrazy. Proponowana struktura zawiera trzy poziomy takiej struktury – drzewa:

1. Katalogi pacjentów - <nazwisko>.<imię>.<pesel>
2. Katalogi badań - <nazwa>
3. Pliki obrazów - <rrrr>-<mm>-<dd>.ptiff

Utrzymanie powyższej struktury nie należy do funkcjonalności opracowanego systemu i pozostaje w gestii jego administratora lub zewnętrznych narzędzi automatyzujących import obrazów z urządzeń medycznych. Nazwa badania powinna być ustandaryzowana i jednoznacznie odpowiadać źródłu obrazów, które są składowane w danym katalogu.

2.2 Kontrolka mapowa

W trakcie opracowywania prototypu systemu została podjęta decyzja o rezygnacji z kontrolki Google Maps na rzecz konkurencyjnego projektu Virtual Earth[2] firmy Microsoft. Decyzja ta została uwarunkowana istnieniem wsparcia dla technologii JScript IntelliSense[3] w środowisku Visual Studio dla kontrolki Vir-

[2] Virtual Earth – http://maps.live.com/
Virtual Earth API – http://msdn.microsoft.com/en-us/library/aa905677.aspx

[3] JScript IntelliSense – http://blogs.msdn.com/webdevtools/archive/2007/03/02/jscript-intellisense-in-orcas.aspx
Virtual Earth JScript IntelliSense Helper – http://www.codeplex.com/VEJS

tual Earth – podpowiadania dostępnych klas oraz pól i metod, które te klasy zawierają. Pod względem zasady działania i dostępnych funkcjonalności rozwiązania Google i Microsoft są w pełni analogiczne.

Sugerowana przez prowadzącego przedmiot biblioteka OpenLayers[4] nie została wybrana ze względu na mniejszą ilość dokumentacji i dostępnych przykładów wykorzystującego ją kodu. Ze wstępnego rozeznania wynika, że biblioteka ta może mieć nawet większe możliwości niż Google Maps i Virtual Earth, niestety cechuje ją bardziej stroma krzywa nauki. Wynika to po części z faktu, że firmy Google i Microsoft nastawione są na maksymalne spopularyzowanie swoich rozwiązań wśród szerokich rzeszy programistów aplikacji webowych (z różnych dziedzin), natomiast OpenLayers jest bardziej dedykowane specjalistom od rozwiązań GIS[5].

3 Architektura systemu

3.1 Źródło obrazów

Źródłem obrazów dla systemy mogą być dowolne urządzenia dostarczające gigapikselowych zdjęć. Dopuszczalne są wszystkie graficzne formaty plików, które umożliwiają konwersję do formatu TIFF, JPEG lub JPEG2000. Wymienione trzy formaty są bezpośrednio wspierane przez system podczas importu, konwersja z innych formatów pozostaje w gestii administratora systemu. Importu obrazów dokonuje się przy pomocy skryptów wykorzystujących program VIPS[6] – narzędzie do zautomatyzowanej obróbki plików graficznych. Dekompresja plików w formacie JPEG2000 realizowana przy pomocy programu kdu_expand – części pakietu Kakadu[7], będącego jedną z wzorcowych implementacji tego standardu.

Proces importu obrazu polega na przekształceniu pliku graficznego do formatu TIFF o tzw. piramidalnej strukturze[8]. Organizacja takiego pliku przypomina zbiór przekrojów poprzecznych piramidy – zawiera on oryginalny obraz, obraz 2x mniejszy, obraz 4x mniejszy itd. aż do obrazu o minimalnej rozdzielczości. Każdy z obrazów składa się z odpowiedniej liczby fragmentów o stałych rozmiarach – tzw. *tiles*. Zarówno Google Maps jak i Virtual Earth wykorzystują fragmenty o rozmiarze 256 x 256 pikseli, dlatego został on zastosowany również w budowanym systemie. Import obrazu do systemu powoduje prawie dwukrotne zwiększenie jego rozmiaru, jest to jednak niezbędne dla efektywnego udostępniania poszczególnych poziomów zbliżenia klientom systemu. Dzięki opisanej wyżej organizacji każdy fragment obrazu może być wysłany do klienta natychmiast po wczytaniu z dysku, bez potrzeby wykonywania żadnych dodatkowych obliczeń poza konwersją do jednego z „internetowych" formatów – JPEG, GIF lub PNG.

Warto wspomnieć, że pliki o piramidalnej organizacji nie wykorzystują żadnych dedykowanych struktur formatu TIFF. Kolejne poziomy przybliżenia zapisywane w pliku jako kolejne obrazy-strony (ang. directories). Z tej samej możliwości korzystają m.in. programy typu OCR podczas skanowania wielostronicowych dokumentów lub programy do obsługi faksmodemów podczas otrzymywania takich dokumentów.

Format TIFF nie jest jedynym formatem graficznym wspierającym scenariusz dostępu wykorzystywany w systemie, jest jednak spośród nich wszystkich najpopularniejszy i najlepiej wspierany przez biblioteki

[4] OpenLayers – http://www.openlayers.org
[5] Geographic Information Systems
[6] VIPS – http://www.vips.ecs.soton.ac.uk
[7] Kakadu – http://www.kakadusoftware.com
[8] Pyramidal TIFF – http://iipimage.sourceforge.net/images.shtml

programistyczne i programy użytkowe. W szczególności nie powiodły się próby wykorzystania formatu JPEG2000. Nie udało się znaleźć biblioteki, która potrafiłaby czytać wszystkie z testowych obrazów zebranych przez autorów sprawozdania i udostępniałaby mechanizm czytania z nich pojedynczych *tiles* – zarówno wśród rozwiązań open source[9] jak i komercyjnych, do których producent udostępnił wersję trial[10]. Podobnie wsparcie programów graficznych dla tego formatu pozostawia wiele do życzenia. W odczuciu autorów sprawozdania format ten nie przyjął się szerzej niż w pewnej wąskiej grupie specjalistycznych zastosowań, takich jak obrazowa diagnostyka medyczna. Korzystają na tym głównie producenci oprogramowania komercyjnego, dostarczający wsparcia dla formatu JPEG2000 często za dodatkową opłatą[11].

Warto zaznaczyć, że TIFF jest tzw. formatem kontenera – definiuje opis cech obrazu, takich jak rozdzielczość i skala barw. Dane wewnątrz pliku mogą być natomiast skompresowane różnymi algorytmami – w tym JPEG i JPEG2000. Decyzja o wykorzystaniu formatu TIFF nie wyklucza zatem możliwości przyspieszenia w przyszłości procesu importu obrazów poprzez natywną obsługę formatu JPEG2000 lub innego. Problem ten sprowadza się do napisania dodatkowych dekoderów lub ich kupna. Do obecnej wersji systemu wybrana została kompresja Deflate – bezstratny algorytm zapewniający najmniejszy rozmiar pliku spośród wszystkich algorytmów obsługiwanych przez wykorzystaną w projekcie bibliotekę LibTIFF[12] i mających zastosowanie dla obrazów kolorowych: Deflate, PackBits, LZW. Wymienione algorytmy są obsługiwane przez większość narzędzi obsługujących format TIFF.

3.2 Serwer plików

Architektura systemu przewiduje wydzielony serwer lub serwery plików, których zadaniem jest przechowywanie obrazów pozyskanych z urządzeń. W rzeczywistej realizacji byłaby to maszyna o dużej przestrzeni dyskowej zorganizowanej w macierz. Sugerowana jest organizacja RAID 1+0[13] lub inna zapewniająca dużą wydajność i redundantność. Wymaganie dużej wydajności podsystemu wejścia-wyjścia związane jest z koniecznością przetwarzania wielkich ilości danych. Dla rozważanych rodzajów obrazów, nawet przy wykorzystaniu kompresji, zupełnie typowe są rozmiary plików liczone w gigabajtach. Odpowiedni serwer plików powinien również zapewniać ciągłość pracy nawet w przypadku wystąpienia awarii części dysków. W przypadku danych medycznych ewentualna konieczność odtwarzania wielu gigabajtów z kopii zapasowej w sytuacji jednoczesnej awarii serwera i zagrożenia życia pacjenta jest niemożliwa to zaakceptowania.

Dedykowany serwer plików jest opcjonalnym elementem architektury i w wersji systemu oddawanej w ramach przedmiotu został pominięty.

---

[9] OpenJpeg - http://www.openjpeg.org
JasPer - http://www.ece.uvic.ca/~mdadams/jasper

[10] J2K-Codec – http://j2k-codec.com/
Kakadu – nie udostępnia żadnej wersji demonstracyjnej

[11] Atalasoft DotImage – http://www.atalasoft.com/Products/DotImage/Default.aspx

[12] LibTIFF – http://www.libtiff.org

[13] Redundant Array of Independent Disks – http://en.wikipedia.org/wiki/RAID

3.3 Serwer WWW

Serwer WWW jest jednym z najważniejszych elementów systemu. Działa na nim aplikacja wykonana w technologii ASP.NET[14], której zadaniem jest:

- generowanie dynamicznych stron WWW mechanizmu nawigacji po strukturze katalogów,
- dostarczanie statycznych stron oraz strony z aplikacją JavaScript do przeglądania obrazów,
- obsługa interfejsu Web Service do wczytywania i zapisywania adnotacji do obrazów,
- dostarczanie fragmentów obrazów – *tiles* – do kontrolki Virtual Earth.

Wybrane techniczne aspekty realizacji wymienionych wyżej zadań zostaną opisane w kolejnych punktach sprawozdania.

Technologia ASP.NET jest technologią tworzenia aplikacji internetowych, w której obowiązuje model programowania sterowanego zdarzeniami (ang. *event driven programming*) – w przeciwieństwie do podejścia żądanie-odpowiedź, stosowanego w tradycyjnych rozwiązaniach typu CGI oraz w technologiach czerpiących z jego tradycji, np. PHP. W ASP.NET programowanie stron internetowych odbywa się w sposób analogiczny do wizualnego programowania aplikacji desktopowych – poprzez rozmieszczenie gotowych komponentów i dopisanie kodu reagującego na generowane przez nie zdarzenia. Nowa jakość, którą wprowadza ASP.NET polega przede wszystkim na automatycznej obsłudze stanu aplikacji pomiędzy żądaniami generowanymi przez przeglądarkę użytkownika. Automatycznie pamiętane są m.in. zawartości pól formularzy, a dostęp do nich odbywa się w sposób w pełni obiektowy – z poziomu języka C# lub Visual Basic. Technologia ASP.NET sprawdza się przede wszystkim przy większych projektach, zwłaszcza w aplikacjach mocno opartych na bazie danych (ang. *data driven application*).

Wybór technologii ASP.NET do realizacji projektu podyktowany był głównie chęcią głębszego jej poznania przez autorów sprawozdania, jako że w tym konkretnym zastosowaniu nie oferuje ona wiele ponad inne rozwiązania webowe. Prosta nawigacja po strukturze drzewa oraz odesłanie do przeglądarki kodu JavaScript do przeglądania i adnotacji obrazów byłoby równie proste do zrealizowania zarówno w PHP, JSP jak i dowolnej innej technologii tego typu.

Zanim zostaną opisane dalsze aspekty techniczne związane z budowaniem systemu w technologii ASP.NET należy zaznaczyć, że pod pojęciem „komponent/kontrolka (ASP.NET) “ będzie zawsze rozumiany kod wykonywujący się po stronie serwera w oparciu o parametry z żądań GET i POST protokołu HTTP (ang. *server-side control*). Dla porównania kontrolka Virtual Earth wykonuje się zawsze po stronie klienta (ang. *client-side control*) – jest to jedyna kontrolka tego typu, o której będzie mowa w dalszej części sprawozdania.

3.3.1 Nawigacja po strukturze katalogów

Strony dostarczające funkcjonalności nawigacji po strukturze katalogów korzystają z trzech interesujących komponentów dostarczanych wraz z ASP.NET:

a) TreeView – rozwijane drzewo podobne do drzewa katalogów programu Windows Explorer,
b) SiteMapPath – popularna na stronach WWW aktualna ścieżka w strukturze serwisu,

[14] ASP.NET – http://en.wikipedia.org/wiki/ASP.NET, http://www.asp.net

c) SiteMapDataSource – źródło danych o strukturze serwisu dla powyższych kontrolek.

Wyjaśnienie wykorzystania wymienionych komponentów należy rozpocząć od ostatniego punktu.

W technologii ASP.NET istnieje liczna grupa kontrolek wspierających mechanizm tzw. `DataSource`. Idea tego rozwiązania polega na tym, że sposób prezentacji danych – np. tabelka (`GridView`), lista (`DataList`), formularz (`DetailsView`), drzewo (`TreeView`) – jest niezależny od źródła danych, które mają zostać wyświetlone – np. baza danych (`SqlDataSource`), plik XML (`XmlDataSource`), własne obiekty programisty (`ObjectDataSource`). Projektując stronę wyświetlającą dane z bazy danych w sposób tabelaryczny, korzysta się zatem z kontrolki `GridView` podłączonej do kontrolki `SqlDataSource`, ale równie dobrze na innej stronie ten sam komponent `GridView` może korzystać np. z `XmlDataSource`. Jest to rozwiązanie bardzo korzystne z punktu widzenia inżynierii oprogramowania.

`SiteMapDataSource` jest dość szczególnym komponentem typu `DataSource` – dostarcza on danych hierarchicznych z pliku `Web.sitemap`, opisującego w formacie XML strukturę serwisu. Plik `Web.sitemap`, oprócz informacji o statycznej strukturze serwisu, umożliwia wprowadzenie w dowolnym punkcie hierarchii klasy dostarczającej informacji dynamicznych. Cecha ta może być wykorzystana w sytuacji, kiedy mamy do czynienia ze strukturą serwisu zależną od innego źródła danych, np. bazy danych (drzewo kategorii produktów itp.) lub systemu plików – jak w przypadku opracowanego systemu. Aby skorzystać z opisanych wyżej, gotowych kontrolek do wyświetlania struktury katalogów i listy obrazów, została utworzona klasa `FileSystemSiteMapNodeProvider` i wpisana w odpowiednim miejscu pliku `Web.sitemap`. Dodatkowo odpowiednia implementacja interfejsu `IFileSystemSiteMapFilter` gwarantuje wyświetlanie jedynie plików *.ptiff.

3.3.2 Dostarczanie aplikacji klienckiej

Interesującym zagadnieniem jest integracja kontrolki Virtual Earth ze stronami wykonanymi w technologii ASP.NET. Osadzenie takiej kontrolki polega na wstawieniu do kodu HTML strony znacznika `div` o ustalonym atrybucie `id` i wywołaniu odpowiedniej funkcji JavaScript dostarczanej wraz z kontrolką. Funkcja ta przekształci blok `div` o wskazanej wartości `id` w kontrolkę Virtual Earth. Wsparcie dla takiego scenariusza przewiduje rozszerzenie technologii ASP.NET o nazwie ASP.NET AJAX[15], dostarczające m.in. klasy `ExtenderControl`. Komponent dziedziczący z tej klasy może zostać podłączony do dowolnej innej kontrolki ASP.NET i dodać pewne nowe zachowania po stronie klienta[16]. Komponent taki rozwiązuje w elegancki sposób kwestię dołączenia do aktualnej strony zewnętrznego kodu JavaScript (np. Virtual Earth API) i jego zainicjowania w odpowiednim momencie w przeglądarce. Jak się okazuje, w przypadku bardziej zaawansowanych bibliotek – takich jak Microsoft AJAX – zwykłe wykorzystanie zdarzenia `window.onload` okazuje się niewystarczające.

W opracowanym systemie kontrolka Virtual Earth jest zagnieżdżana i inicjowana przy pomocy klasy `VirtualEarthExtenderControl`, działającej według przedstawionej wyżej zasady. Umożliwia to m.in. ustawienie z poziomu kodu server-side początkowej pozycji i poziomu przybliżenia kontrolki, bez uciekania się do generowania kodu JavaScript przy pomocy instrukcji typu printf.

[15] ASP.NET AJAX – http://www.asp.net/ajax
[16] http://aspalliance.com/1463_Extend_ASPNET_AJAX_ClientSide_Function__The_ServerSide_Way

3.3.3 Web Service do zarządzania adnotacjami

Ponieważ obsługa adnotacji odbywa się całkowicie po stronie klienta, konieczne stało się zbudowanie odpowiedniego interfejsu umożliwiającego ich zdalny odczyt, zapis oraz usuwanie. Problem ten został rozwiązany poprzez implementację usługi Web Service. Technologia ASP.NET posiada pełne wsparcie dla standardowych usług SOAP[17] oraz automatycznego generowania ich opisu w formacie WSDL[18]. Udostępnienie metody dla zdalnych wywołań wymaga jedynie dodania do niej atrybutu[19] `[WebMetod]` i umieszczenia jej w statycznej klasie z atrybutem `[WebService]`. Format SOAP, w którym standardowo przekazywane są dane z i do takiej metody wprowadza jednak stosunkowo duży narzut komunikacyjny i wymaga dość skomplikowanego przetwarzania, dlatego nie jest stosowany w komunikacji kodu JavaScript z usługami. W takim scenariuszu wykorzystuje się częściej format JSON, umożliwiający bezpośrednią inicjalizacje obiektów JavaScript danymi z usługi. Dodanie obsługi formatu JSON do zdalnej metody wymaga przy wykorzystaniu technologii ASP.NET AJAX jedynie dodania atrybutu `[ScriptService]` do klasy zawierającej metodę.

Opisane wyżej rozwiązanie zostało wykorzystane w budowanych systemie. Za obsługę adnotacji odpowiada klasa `FileWebService`.

3.3.4 Obsługa fragmentów obrazów

Ostatnie z wymienionych zadań aplikacji ASP.NET realizowane jest w oparciu o bibliotekę LibTIFF, która umożliwia odczytywanie wybranych fragmentów obrazu bez wczytywania nadmiarowych danych – wczytywany jest tylko żądany tile, a dostęp do niego realizowany jest w czasie stałym.

Ponieważ biblioteka LibTIFF udostępnia interfejs w języku C, a pozostała część aplikacji wykonana jest w technologii .NET, konieczne było opracowanie warstwy pośredniczącej w korzystaniu z biblioteki. Jej rolę pełni kod napisany w języku C++/CLI[20]. Technologia ta umożliwia pisanie w języku C++ klas widocznych w każdym języku platformy .NET. Implementacje takich klas mogą bezpośrednio wywoływać funkcje z plików nagłówkowych języków C i C++. Jest to najefektywniejsze rozwiązanie tego typu problemu, zarówno pod względem wydajności pracy programisty jak i wydajności kodu będącego jej rezultatem. Druga możliwość na platformie .NET – mechanizm P/Invoke[21] – wymaga tłumaczenia na język C# lub Visual Basic sygnatur wywoływanych funkcji i struktur danych, które te funkcje wykorzystują. Jest to dobre rozwiązanie, kiedy ilość potrzebnych funkcji jest stosunkowo mała, a wśród ich parametrów przeważają typy proste. Na dłuższą metę P/Invoke powoduje jednak m.in. następujące problemy:

- pliki nagłówkowe zawierają często skomplikowane makrodefinicje i obszary kompilowane warunkowo, a w celu przetłumaczenia sygnatur funkcji konieczne jest ich pełne zrozumienie przez programistę,
- w przypadku błędnego przetłumaczenia sygnatury funkcji błąd w najlepszym wypadku zostanie wykryty dopiero w czasie wykonywania kodu – w postaci wyjątku naruszenia ochrony pamięci,

---

[17] Simple Object Access Protocol – http://en.wikipedia.org/wiki/SOAP
[18] Web Services Description Language – http://en.wikipedia.org/wiki/Web_Services_Description_Language
[19] C# attributes – http://www.codeproject.com/KB/cs/attributes.aspx
[20] C++/CLI – http://www.codeproject.com/KB/mcpp/cppcliintro01.aspx
[21] Platform Invoke – http://www.codeproject.com/KB/dotnet/PInvoke.aspx

- wyznaczenie odpowiedniego typu .NET dla typu określonego w pliku nagłówkowym może być w ogólności trudne.

Jeśli chodzi o kwestię wydajności, klasy pisane w języku C++/CLI umożliwiają pełną kontrolę nad konwersją danych pomiędzy maszyną wirtualną a kodem natywnym, podczas gdy mechanizm P/Invoke wykonuje wszystkie konwersje w sposób automatyczny – bez względu na to, czy dany parametr zostanie po konwersji wykorzystany.

Dla porównania, mechanizm JNI[22] służący do wywoływania natywnego kodu z poziomu kodu w języku Java jest rozwiązaniem porównywalnym do C++/CLI, z następującymi różnicami przemawiającymi na jego niekorzyść:

- funkcje stają się metodami klas w środowisku Java na podstawie ich odpowiedniego nazywania (`Java_ClassName_MethodName`) – nie jest wykorzystywane wsparcie dla klas języka C++,
- sygnatury metod trzeba powtarzać z kodzie w języku Java ze słowem kluczowym `native`,
- nie jest możliwe odwoływania się do klas środowiska Java z poziomu kodu w języku C/C++
- kod pośredniczący jest po skompilowaniu zwykłą biblioteką *.dll – razem ze wszystkimi negatywnymi implikacjami takiego rozwiązania.

Zaletą mechanizmu JNI jest natomiast możliwość implementacji metod w języku C.

We wcześniejszych punktach sprawozdania zostało dokonane pewne uproszczenie w kwestii wczytywania *tiles*. Informacja o tym, że na żądanie 1 *tile* od klienta zostaje wczytany tylko 1 *tile* z dysku, jest prawdziwa tylko w pewnych okolicznościach. Należy pamiętać, że kontrolka Virtual Earth zakłada rozmiary kolejnych poziomów powiększenia będące potęgami liczby 2. Jeśli rozmiar danego obrazu nie pasuje do tego schematu, nie wypełni całego widoku kontrolki. W tej sytuacji zdecydowano się dokonywać „w locie" wyśrodkowania wczytywanych obrazów. Wprowadzenie takiego mechanizmu zwiększa maksymalną liczbę wczytywanych tiles do 4 – dla najbardziej „niekorzystnego" ułożenia siatki *tiles* kontrolki i pliku graficznego.

3.4 Aplikacja kliencka

Klient może wejść w interakcję z systemem przy pomocy większości współczesnych przeglądarek internetowych. Warunkiem koniecznym jest wsparcie dla danej przeglądarki ze strony kontrolki Virtual Earth i technologii ASP.NET AJAX. Korzystając ze stron generowanych przez serwer WWW użytkownik może nawigować po drzewie katalogów serwera plików i wybrać obraz, który chciałbym obejrzeć. Strona dotycząca danego obrazu zawiera Virtual Earth, która pozwala przeglądać obraz i jego adnotacje. Adnotacje dodawane są do obrazu graficznie przy pomocy kontrolki, natomiast za ich ładowanie i wczytywanie odpowiada kod JavaScript komunikujący się z Web Service na serwerze. Adnotacje prezentowane są dodatkowo w formie tabelarycznej.

[22] Java Native Interface – http://en.wikipedia.org/wiki/Java_Native_Interface

4 Podsumowanie

Podsumowując realizację projektu należy podkreślić, że postawione na początku semestru cele zostały w pełni zrealizowane. Zbudowana aplikacja dostarcza wszystkich zaplanowanych funkcjonalności, a opracowana koncepcja systemu jako całości wydaje się być sensowna i nadawać do wdrożenia do rzeczywistego użycia. Nie oznacza to jednak, ze stworzona aplikacja jest gotowa do udostępnienia lekarzom. Z pewnością wymaga jeszcze wielu usprawnień, zwłaszcza w obszarze interfejsu użytkownika i dokumentacji.

Najważniejsze dla realizowanego projektu były jednak wymagania pozafunkcjonalne – efektywna obsługa bardzo dużych bitmap i udostępnienie interfejsu WWW. Wymagania te zostały spełnione. Ponadto architektura systemu została stosownie przemyślania, a zestaw wykorzystanych rozwiązań i technologii odpowiednio dobrany, co daje projektowi szansę na dalszy, bezproblemowy rozwój i ewentualne wdrożenie.

5 Załączniki

1. Opis protokołu Google Maps i Virtual Earth
2. Diagram architektury systemu