

Programowanie w Internecie 2009 - Projekt



Unabashedly striving for credits since 2006

Raport

ws. Kryzysu Streamingowego

Przyjęty dnia 9 czerwca 2009 r. uchwałą Komisji Śledczej nr 814.22e w składzie:

- ♦ **109350zt** – Chief Executive Officer, CASC Development Team
- ♦ **109252nm** – Chief Technical Officer, CASC Development Team
- ♦ **109254op** – Public Relations Enforcer, CASC Development Team

Autor nie bierze odpowiedzialności za wszelkie, zarówno odwracalne jak i nieodwracalne, uszkodzenia układu nerwowego oraz traumatyczne przeżycia będące wynikiem lektury niniejszego raportu.

Wstęp

Na początku należy zaznaczyć, że realizacja streamingu audio/video w Microsoft Silverlight, najlepiej z możliwością dynamicznego przełączania nadawcy, nie jest prosta w żadnym znanym sensie tego słowa. Najłatwiejsze rozwiązania cechuje intuicyjność na poziomie niemożliwej bryły czterowymiarowej, tzn. nikła. Utrudnia to znacznie stosowanie znanej i lubianej filozofii programowania [Worse is Better](#), opartej o pracę Mikołaja Kopernika „Monetae cudendae ratio” z 1526 r., która stawia na pierwszym miejscu prostotę implementacji.

Jako że analizowane rozwiązania problemu cechuje duża pokrętność, w celu choćby minimalnego ułatwienia zrozumienia raportu został on podzielony na następujące części:

- ♦ **Technologia streamingu** – format i sposób przesyłania danych między użytkownikami.
- ♦ **Dostęp do urządzeń** – sposób dostępu do kamery i mikrofonu użytkownika.
- ♦ **Architektura aplikacji** – podział aplikacji na moduły o różnej funkcjonalności.

Należy dodać, że zawarte w dokumencie informacje mogą nie do końca odwzorowywać rzeczywistość, gdyż prostym studentom czasami trudno załapać, czy „WMV w ASF po MMS działa w 0.512 Beta 5 w wersji PULL na HTTP pod NT 6.1” itd.



MacGyver, Silverlight Team Consultant

Technologia streamingu

1. Windows Media Services

WMS umożliwia streamowanie w formacie ASF/WMV+WMA (czyt. raczej zrozumiałym dla Silverlighta), obsługuje przesyłanie danych w trybie „push” (czyt. można nadawać z za NAT-u) i istnieje do niego *interopka* (czyt. można konfigurować serwer z poziomu C# przez wykorzystanie mnóstwa *pokrętnych* interfejsów).

Z drugiej strony, określne empirycznie opóźnienia przesyłu są raczej zniechęcające (może się zdarzyć, że dźwięk będzie opóźniony o jakieś 5 - 10 sekund w stosunku do zmieniających się slajdów).

Wykorzystanie WMS-a ogranicza wybór serwera do Windows Server 2008. Trudno określić, na ile ta technologia ma jakąkolwiek przyszłość. Biorąc pod uwagę, że w Windows Server 2003 występowała wersja 9.0, w Server 2008 dostępna jest wersja 9.5, a w Server 2008 R2 nie ma jeszcze **żadnego** WMS-a, to rozwój nie wydaje się zbyt dynamiczny. Podobno WMS Team połączył się jakiś czas temu z IIS Team, więc może IIS Media Services zostanie oficjalnym następcą WMS-a (choć pewne dane wskazują, że jednak nie).

2. VideoLAN Client (VLC)

Jeśli chodzi o serwer VLC, to najbardziej interesująco przedstawia się możliwość wykorzystania biblioteki libvlc.dll. Co prawda biblioteka jest dość mocno w stylu C, ale są dostępne pewne bardzo nieoficjalne wrappery do C#. Tzn. da się to wszystko wykorzystać z poziomu .NET, ale obsługa (tzn. API) jest raczej *pokrętna* (znowu to słowo!).

Podstawowy problem z VLC polega na tym, że *przypuszczalnie* nie obsługuje przesyłania danych w trybie „push” (klient musi wystawić strumień na jakimś porcie i podać adres, więc raczej nie prześle nic z za NAT-u). Próba potwierdzenia tej hipotezy przez zadanie [pytania na forum](#) zakończyła się fiaskiem (ogólnie wszystkie próby dowiedzenia się czegoś o streamingu przez zadawanie pytań na forach zakończyły się fiaskiem – może to temat tabu?).

3. IIS Media Services 3.0

IIS MS 3.0, jeszcze ciepły plugin do IIS-a, umożliwia wykorzystanie wspaniałej technologii Live Smooth Streaming, stworzonej specjalnie dla Silverlighta i oferującej automatyczne dopasowanie jakości przekazu do przepustowości łącza. Jest tylko jeden problem – istnieje *jeden* enkoder tego formatu. Sprzętowy. W sumie można napisać swój własny, czy to jakiś problem?

4. Red5

Red5 – ołpensorsowa alternatywa dla Flash Media Server. Obsługuje strumienie w formacie FLV, które *prawdopodobnie* da się odtworzyć w Silverlightcie 3. Jest napisany w Javie i *zdaje się* można na nim uruchamiać rodzaj aplikacji (również w Javie), które obsługują strumienie i komunikują się ze światem na podobnej zasadzie jak web service'y (wiedza z [tutoriala](#)). Problem w tym, że to jest chyba mocno nastawione na współpracę z Flashem - [Ciocia Wikipedia twierdzi](#), że obsługiwany jest protokół komunikacji Action Message Format - „[AMF is a binary format based loosely on the Simple Object Access Protocol \(SOAP\)](#)”. Prawdopodobnie „based loosely” oznacza „nic poza produktami Adobe się z tym nie porozumie”. Właściwie to trudno stwierdzić, czy można administrować Red5 z poziomu C#, ale chyba to nie jest priorytetem twórców.

5. Darwin Streaming Server

Otwarta wersja serwera Apple, nadaje w jakimś RTSP/RTP, które, znowu, *prawdopodobnie* może zostać odtworzone przez Silverlighta 3. Tu to już zupełnie nie mam pojęcia, czy można zgrać ten serwer w jakiś sposób z .NET, ani czy on działa stabilnie na Windowsie (są pewne wątpliwości na ten temat).

6. LIVE555 Streaming Media

To są jakieś biblioteki C++ do tworzenia aplikacji wykorzystujących streaming. Można na ich podstawie zrobić jakiś serwer do obsługi RTSP, ale to chyba trochę nie o to nam chodzi...

Jest też gotowy serwer z tej samej „rodziny”, ale on chyba nie obsługuje strumieni „live”.

7. Sokiety

Sockets, czyli coś, co dla odmiany jest oficjalnie wspierane w Silverlightcie, również w wersji 2, i są do tego nawet jakieś gotowe [frejmłorki](#). Ta opcja ma pewne kluczowe zalety w stosunku do wszystkich powyższych:

- prawdopodobnie pozwala zmniejszyć opóźnienia do akceptowalnego poziomu;
- łatwo można zaimplementować możliwość naprzemiennego nadawania przez różne osoby (np. wykładowca udziela chwilowo głosu studentowi);
- wszystko dzieje się na tak niskim poziomie, że przynajmniej wiadomo, *dlaczego* nie działa.

Oczywiście użycie socketów oznacza, że trzeba zaimplementować wszystko ręcznie, tzn. zestawianie połączeń, enkodowanie i dekodowanie strumienia, buforowanie.

Tu pojawiają się pewne wątpliwości odnośnie kodowania danych. Silverlight 3 obsługuje format H.264 i AAC, więc jeśli udałoby się zakodować *w locie* strumień z kamery/mikrofonu do tych formatów, to *prawdopodobnie* nie byłoby większego problemu z odtworzeniem tego w Silverlightcie (praktycznie z pojedynczych bajtów). Pytanie tylko, czy takie enkodowanie da się zaimplementować przy użyciu dostępnych bibliotek, bo własnego formatu kompresji „na szybko” raczej nie opracujemy (no, może nie licząc `System.IO.Compression.DeflateStream`).

Dostęp do urządzeń

1. DirectShow

Przez DirectShow można bardzo ładnie wyciągnąć dane z kamery tudzież mikrofonu, jest nawet [wrapper do C#](#). Sama technologia jest już wypierana przez Media Foundation, ale za to w Internecie można znaleźć sporo informacji i przykładów jej użycia.

Problem pojawia się w momencie, kiedy chcemy strumień danych zakodować. Domyślnie DirectShow udostępnia filtr enkodujący strumień do formatu ASF/WMV+WMA (WM ASF Writer). Z tym, że jedyne, co ten filtr potrafi zrobić z zakodowanymi danymi, to zapisać je do pliku. Istnieją dwa rozwiązania tego problemu (tu w kontekście wysłania strumienia bezpośrednio na serwer):

- **Profesjonalne:** „You will need to write your own sink filter that muxes the audio/video stream into an ASF transport stream and sends it using the MMS-over-HTTP transport protocol and the MMS push signalling protocol. Not an easy task.” (Na tyle *not easy*, że [niektórzy oferują za to pieniądze](#).)
- **Podejście pragmatyczne (znane jako „Dirty Hack”):** dobieranie się do zawartości obiektu WMASFWriter przy użyciu pokrętnych interfejsów Windows Media Format i wymienianie jednych obiektów na inne w „magiczny sposób”. (Mam czasami wrażenie, że to jest możliwe tylko dlatego, że twórcy DirectShow nie wpadli na to, że ktoś mógłby czegoś takiego spróbować i nie wprowadzili żadnych zabezpieczeń.)

Bardziej sensowne wydaje się w tym wypadku pobieranie danych „po ramce” (to można zrobić w *normalny* sposób) i rozsyłanie tego „na własną rękę”.

2. Windows Media Encoder

Windows Media Encoder pozwala w prosty sposób wysłać strumień z kamery/ mikrofonu na serwer z Windows Media Services. Jest też do niego SDK, które prawdopodobnie daje możliwość uruchomienia całości z poziomu C#. Dwa problemy:

- użycie WME implikuje użycie Windows Media Services;
- każdy użytkownik chcący nadawać będzie musiał instalować WME.

3. VideoLAN Client

VLC pozwala przechwycić obraz z kamery/mikrofonu i zakodować do jednego ze sporej liczby obsługiwanych formatów. Można to potem zapisać do pliku tudzież wystawić strumień na porcie. To trochę komplikuje przesyłanie tego dalej, szczególnie z za NAT-u. Nie wiem, czy istnieje możliwość pobrania pojedynczych „paczek” (tzn. byte[]) zakodowanego strumienia.

Do tego istnieje biblioteka libvlc.dll, oczywiście w C i ma raczej nieprzystępne API.

4. Microsoft Media Foundation

Czyli następca DirectShow wprowadzony w Windows Vista. Wersja dostępna w Windows 7 posiada zestaw nowych, szalonych wręcz możliwości, np. obsługę kamer. Oczywiście wszystko w C, bo przecież C# nikt nie używa.

Media Foundation ma różne ciekawe możliwości, prawdopodobnie łatwiej pobrać dane z kamery/mikrofonu, zakodować je (do H.264/AAC) i gdzieś przesłać, tym razem bez „metod pragmatycznych”. Tylko technologia jest na tyle nowa, że trochę brakuje porządnej dokumentacji/tutoriali/itp.

Co prawda używając MF ograniczamy grupę użytkowników do posiadaczy Windows 7 / 2008 R2. Ale czy my się boimy nowych technologii? W końcu od pewnego czasu cały proces rozwoju projektu i tak jest oparty o Silverlight 3 **Beta 1**, .NET Framework 4.0 **Beta 1**, Windows 7 **RC**, Visual Studio 2010 **Beta 1**, IIS 7.5 **Beta** (nawet plugin do SVN-a mamy w wersji rozwojowej). A przecież i tak już wszyscy używają Windows 7.

5. Microsoft Speech API

Pomysł z gatunku zupełnie rozpaczliwych. Zamiast implementować jakiś streaming, może lepiej zastosować rozpoznawanie mowy i przesyłać sam tekst. To oczywiście nie ma prawa działać, ale sam pomysł jest fajny.

Architektura aplikacji

Tu może należy zaznaczyć, z czego wziął się problem. Otóż Rada Nadzorcza CASC Development Team w swej naiwności przyjęła, że „to będzie działać w Silverlightcie”. Otóż, jak się okazuje, Silverlight jest na dość wczesnym etapie rozwoju. Dopiero w wersji trzeciej zaimplementowano kluczowe z punktu widzenia biznesu elektronicznego możliwości, tzn. natywną obsługę shaderów w HLSL. Bardziej wydumane funkcje, takie jak obsługa kamery (dostępne we Flashu od czasów prehistorycznych), mają pojawić się w bliżej nieokreślonej przyszłości. Co gorsza, jedyną metodą odwołania się do własnych bibliotek C/C++ z poziomu Silverlighta jest przekupienie Scotta Guthriego (a on nie wygląda na kogoś, kto da się przekupić).

Dlatego też w celu implementacji streamingu konieczne jest przeprojektowanie aplikacji w dość kategoriowy sposób.

1. Aplikacja WPF dla „wykładowcy” + Silverlight dla „studentów”

Tzw. wersja prosta i bezpieczna. „Wykładowca” ściąga aplikację (exe), i z jej poziomu obsługuje „wykład”. Z WPF można się odwoływać do jakichkolwiek bibliotek, więc można tu zaimplementować każdy rodzaj streamingu.

„Studenci” oglądają transmisję z poziomu Silverlighta, a ich jedyną możliwością włączenia się w „wykład” jest skorzystanie z tekstowego *czatu*.

2. Silverlight do odbioru + opcjonalna aplikacja WPF do nadawania

Aplikacja umożliwiająca streaming jest opcjonalna dla każdego uczestnika „wykładu”. Moduł Silverlight jest powiadamiany (to dość prosto można zaimplementować), którzy użytkownicy mają możliwość nadawania i udostępnia im odpowiednie opcje.

Takie rozwiązanie ma tę zaletę, że „wykładowca” ma możliwość czasowego przekazania głosu „studentowi” (np. wykład może być prowadzony przez dwie osoby).

3. Aplikacja jako XBAP (XAML Browser Application – WPF w przeglądarce)

W tym przypadku można przynajmniej wywoływać tzw. *funkcje niebezpieczne* (tzn. napisane w C++), jeśli się przekona użytkownika, żeby udzielił aplikacji uprawnień (czy coś).

Minusem jest to, że cały ten XBAP jest dość dziwny i pewnie mało popularny. Poza tym jest obsługiwany tylko przez Internet Explorera i Firefoksa, a my ostatnio unikamy używania takich *dziwaczných* przeglądarek.

4. Cała aplikacja w przeglądarce

Czyli opcja posiadająca tę oczywistą zaletę, że mamy wszystko „w jednym miejscu”, użytkownicy nic nie muszą ściągać itp.

Rozwiązanie opiera się na *wyjatkowo pokrętnej* technice, której opracowanie jest wynikiem długotrwałych poszukiwań i niestandardowych procesów myślowych. Dostęp do funkcji poza-Silverlightowych odbywa się w następujący sposób:

- Silverlight wywołuje funkcję z JavaScriptu
- JavaScript wywołuje funkcję z ukrytego na stronie apletu Javy
- aplet Javy albo wywołuje swoją własną funkcję (bo Java ma jakieś *pokrętne* sposoby dostępu do mikrofonu), albo wywołuje funkcję z ukrytej w hakerski sposób na stronie biblioteki DLL
- Java wysyła otrzymane mniej lub bardziej bezpośrednio z kamery, mikrofonu czy czegokolwiek innego dane i wysyła je do JavaScriptu
- JavaScript wysyła dane do Silverlighta.

Proste.

Wszystko to ma szansę działać dzięki temu, że Java ma możliwość wywoływania *funkcji niebezpiecznych*, jeśli użytkownik się na to zgodzi. A jak wiemy, użytkownik kliknie na absolutnie każdy przycisk OK/Yes/I Agree jaki zobaczy, nie zastanawiając się ani chwili, co właściwie robi.

Zresztą, [tak zrobiłby MacGyver!](#)

5. Cała aplikacja w WPF

Rozwiązanie proste i radykalne. Całość staje się komunikatorem w stylu Skype'a. To daje oczywiście olbrzymie pole manewru, ale „to już nie jest RIA”.

Podsumowanie

Jak widać z powyższej analizy, należy raz na zawsze zapomnieć o kopernikańskiej zasadzie „Worse is Better”. Być może należy ją zastąpić nową zasadą, „Worser is Betterer” (czyli zamiast „gorsze jest lepsze” → „gorsiejsze jest lepsiejsze”). Zamiast szukać na siłę prostego rozwiązania, należy przeprosić się z rozwiązaniami trudnymi i opracować jakiś własny, optymalny system. Takie podejście jest zgodne z tzw. [disaster therapy](#), popularną w nowoczesnej psychologii alternatywną techniką opracowaną przez zasłużonego na tym polu C. Avellone'a (*patent pending*).

Podsumowując (a co ja mam robić innego w podsumowaniu?!):

- ◆ **Technologia streamingu:**

Tu dość „bezpiecznym” wyjściem wydają się sockety. Technologia znana, dobrze zaimplementowana i dająca największe możliwości. Fakt, że implementacja będzie raczej skomplikowana, ale przynajmniej może uda się osiągnąć efekt zgodny z zamierzonym. Do tego unika się problemów ze *streaming serverami*, które czasami mogą nie być chętne do współpracy.

Konieczne jest tylko znalezienie jakiegoś sposobu na kompresję „w locie”.

- ◆ **Dostęp do urządzeń:**

Jeśli streaming byłby realizowany na socketach, to dane z kamery i mikrofonu można pobrać bez większych problemów z DirectShow / Media Foundation. DirectShow jest dostępny dla większej ilości użytkowników, w Internecie można znaleźć gotowy wrapper do C# oraz sporą ilość przykładowego kodu. Media Foundation jest rozwiązaniem bardziej „przyszłościowym” (cokolwiek to znaczy) i ma wbudowane wsparcie dla enkodowania H.264/AAC.

- ◆ **Architektura aplikacji:**

Tu wszystko zależy od tego, co właściwie chcemy osiągnąć. Wszystkie opcje są zasadniczo „do zrobienia”, kwestią jest tylko określenie priorytetów.

„If I had some duct tape, I could fix that.”

MacGyver, Silverlight Team Consultant