

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

ASP.NET 2.0 AJAX. Zaawansowane programowanie

Autor: Matt Gibbs, Dan Wahlin

Tłumaczenie: Krzysztof Bąbol

ISBN: 978-83-246-1300-7

Tytuł oryginału: [Professional ASP.NET 2.0 AJAX](#)

Format: B5, stron: 328



Śmiało wkraczaj w nowoczesne technologie, aby tworzyć interaktywne aplikacje internetowe!

- Jak zarządzać odwołaniami do strony?
- Jak osadzać skrypty w kontrolkach serwerowych?
- Jak używać narzędzia Fiddler do kontroli komunikatów?

ASP.NET AJAX to nazwa stworzonego przez Microsoft rozwiązania AJAX. Odnosi się do zestawu technologii klienckich i serwerowych, które koncentrują się na ulepszaniu programowania WWW pod Visual Studio. Jest to jedna z najpopularniejszych, a równocześnie najbardziej docenianych – zarówno przez użytkowników, jak i programistów - technologii do tworzenia interaktywnych witryn internetowych. ASP.NET AJAX dysponuje między innymi serwerowymi usługami aplikacyjnymi, takimi jak uwierzytelnianie i przechowywanie profilu użytkownika, oraz zawiera zestaw kontroltek ułatwiających asynchroniczną aktualizację stron i korzystanie z zasobów serwera.

Książka „ASP.NET 2.0 AJAX. Zaawansowane programowanie” szczegółowo ukazuje, na czym polega tworzenie aplikacji WWW nowej generacji. Dzięki temu podręcznikowi będziesz wiedział, jak korzystać z różnych bibliotek, jak używać zaawansowanych kontroltek z pakietu AJAX Toolkit, a także jak budować własne kontrolki. Nauczysz się asynchronicznie uaktualniać fragmenty stron i zarządzać skryptami używanymi w przeglądarce, dowiesz się, jak testować takie aplikacje i usuwać z nich błędy. Z tą książką zdobędziesz wiedzę potrzebną do kreowania nowoczesnych, interaktywnych aplikacji.

- Częściowe aktualizacje stron
- Pobieranie kodu JavaScript
- Biblioteka kliencka ASP.NET AJAX
- Używanie prototypów
- Elementy DOM
- Użycie kontrolki ScriptManager
- Dodawanie odwołań do skryptów
- Zaawansowane kontrolki z pakietu AJAX Toolkit
- Efekty interfejsu użytkownika
- Testowanie, debugowanie i wdrażanie aplikacji ASP.NET AJAX

**Korzystaj z nowoczesnych technologii podczas
tworzenia interaktywnych aplikacji internetowych!**

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl



Spis treści

O autorach	13
Wprowadzenie	15
Część I Wprowadzenie do RIA	29
Rozdział 1. Co charakteryzuje bogate aplikacje internetowe?	31
Bogate aplikacje internetowe są „plastyczne”	32
Przykład plastycznej (dynamicznej) strony WWW	32
Jak arkusze stylów tworzą lepszą stronę?	34
Bogate aplikacje internetowe: Web jest przełomem (w końcu)	36
Bogate aplikacje internetowe są niezauważalne	38
Niezauważalność Google	38
Skupianie funkcjonalności RIA	39
Bogate aplikacje internetowe burzą mury otaczające ogród	40
Bogate aplikacje internetowe tworzą nowe (nieogrodzone?) ogrody	42
Ograniczenia w nieogrodzonym ogrodzie	43
Poznajmy manifest opozycji	44
Bogate aplikacje internetowe są zawsze aktualne	47
Bogate aplikacje internetowe są zabójcami systemów operacyjnych	49
Podział na warstwy w Web 2.0	49
Kaskadowe arkusze stylów	50
Pozostawiając biuro za sobą	50
Czy Java jest zwycięzcą czy przegranym?	50
Bogate aplikacje internetowe są ukierunkowane na przeglądarkę	51
Bogate aplikacje internetowe są ukierunkowane na sieć	52
Bogate aplikacje internetowe zmieniają sposób myślenia	54
Obserwacje zmienionego myślenia programisty	54
Pewne obserwacje z naszych zmienionych umysłów	55
Bogate aplikacje internetowe są zgodne z modelem „oprogramowanie jako usługa”	55
Bogate aplikacje internetowe są ukierunkowane na użytkownika	55
Bogate aplikacje internetowe są idealne do współpracy	57
Bogate aplikacje internetowe: krótki i zwinny rozwój	58
Bogate aplikacje internetowe są skazane na ewolucję	59

Jak zmienić sposób tworzenia aplikacji?	59
Jak mogę zarabiać na RIA?	61
Bogate aplikacje internetowe są uwodicielskie	62
Rozdział 2. Podstawy RIA	69
XML: lingua franca internetu	70
Podstawy XML	71
Używanie atrybutów	72
Drzewo DOM	73
XHTML — język dokumentów	76
Struktura dokumentu	77
Struktura tekstu	79
Opisywanie typu tekstu	80
Obiekty śródtekstowe	81
Ustawienia dokumentu	82
CSS — język stylu	82
Dołączanie CSS do strony WWW	85
Zasady malowania	86
Kaskada w CSS	88
Pełne wykorzystanie zalet CSS w RIA	89
JavaScript — język działań	90
Dołączanie JavaScriptu do strony WWW	90
Wiązanie JavaScriptu z działaniami użytkownika	92
Dodawanie i usuwanie z drzewa DOM	95
Sprawdzanie i ustawianie zawartości elementu	95
Przykład podsumowujący — Restauracja Angeli	97
Podsumowanie	103
Rozdział 3. Programowalna sieć: ekosystem mashupów	105
Przygotowania do miksovania	107
Obraz ogólny	107
Obiekt XMLHttpRequest	108
Pierwsza prosta aplikacja hybrydowa	110
Praca z danymi wyjściowymi w formacie XML	110
Praca z danymi wyjściowymi w formacie JSON	122
Czym jest proxy i dlaczego jest potrzebne?	129
Praca z proxy	131
Praca bez proxy	133
XML czy JSON?	133
Mashup Yahoo i Google Traffic	134
Uzyskujemy współrzędne	134
Pobieranie raportu o natężeniu ruchu	138
Dynamiczne wstawianie znacznika SCRIPT do sekcji HEAD	141
Składamy końcową hybrydę	144
Podsumowanie	155
Rozdział 4. Zaczynamy. Tworzenie pierwszej RIA	157
Dwa scenariusze: gdyby restauracje działały tak, jak aplikacje internetowe	157
Niezwykła restauracja	158
Inne doświadczenie	159
Jaki to ma związek z aplikacjami internetowymi?	160

AJAX kontra podejście tradycyjne	161
Poznajmy tradycyjną aplikację internetową	161
Ograniczenia tradycyjnych aplikacji internetowych	163
Poznajmy aplikację internetową wykorzystującą AJAX	163
Zalety technologii AJAX w porównaniu z podejściem tradycyjnym	165
Składniki AJAX-a	166
Przygotowanie środowiska programowania	166
Nasza pierwsza RIA: Witaj świecie	167
Tworzenie HelloWorldv1.html	167
Dodawanie procedury obsługi zdarzenia w JavaScriptcie	168
Definiowanie ciała JavaScriptu	168
Tworzenie asynchronicznego żądania	169
Definiowanie funkcji obsługującej zdarzenie	169
Definiowanie funkcji zwrotnej	171
Przetwarzanie odpowiedzi serwera	172
Kompletny plik HelloWorldv3.html	173
Tworzenie katalogu helloFiles	174
Tworzenie strony serwera	176
Programowanie strony serwera w PHP	176
Programowanie serwletu Java	179
Wzbogacanie aplikacji klienta	184
Podsumowanie	188

Część II Zgłębiamy RIA 189

Rozdział 5. Debugowanie strony klienta 191

Narzędzia do debugowania	192
Firefox	192
Inspektor DOM Mozilli	193
Konsola JavaScript (konsola błędów)	194
Venkman	196
FireBug	197
Walidacja znaczników	198
Ale poczekaj, to nie wszystko!	199
Rejestrowanie	199
Funkcja alert	200
Bardzo prosty rejestrator	201
Zadanie domowe	204
Podsumowanie	204

Rozdział 6. Wzorzec MVC — Model – Widok – Kontroler 205

Model – Widok – Kontroler	205
Pochodzenie: Rzecz – Model – Widok – Edytor	206
Szczegóły wzorca Model – Widok – Kontroler	207
Używanie MVC w Javie	212
Architektura Model 2	213
Projekt historii zdrowia dla Javy	215
Frameworki MVC dla Javy	217

Frameworki aplikacji webowych	219
Ruby on Rails: framework MVC oparty na Ruby	221
TurboGears: alternatywa dla Rails oparta na Pythonie	224
Krótkie zestawienie implementacji MVC	228
Podsumowanie	228
Rozdział 7. Przegląd bibliotek JavaScript	229
Podstawy bibliotek JavaScript	230
Po co używać bibliotek JavaScript?	230
Różne biblioteki	232
Właściwości bibliotek	233
Często oferowane możliwości bibliotek	234
Szczegółowe omówienie bibliotek JavaScript	236
Dojo	236
MochiKit	240
Script.aculo.us	244
Biblioteka Yahoo! UI	248
Krótkie zestawienie bibliotek JavaScript	251
Podsumowanie	251
Rozdział 8. Kompilowanie do JavaScriptu	253
Poznajmy pierwszych kandydatów	254
Google Web Toolkit (GWT)	255
Zalety GWT	255
Wady GWT	256
Programowanie z użyciem GWT	257
Narzędzia programistyczne	259
Wrażenia z pracy z GWT	262
Automatyczna serializacja	264
Ruby JavaScript (RJS)	265
Zalety RJS	265
Wady RJS	266
Programowanie z użyciem RJS	266
Wrażenia z pracy z RJS	269
Podsumowanie	270
Rozdział 9. Wprowadzenie do ideaStax	271
Założenia dotyczące sposobu działania ideaStax	272
Tworzenie nowego Stosu, Karty lub Koncepcji	273
Tworzenie Kart	274
Zaczynamy: niezbędne elementy	279
ideaStax	279
MySQL	281
Konfiguracja Ruby on Rails	281
ideaStax — szczegóły implementacji	283
Dekonstrukcja Koncepcji, Karty i Stosu	284
Sprawdzanie Koncepcji, Kart i Stosów	285
Implementacja widoków	287

Implementacja kontrolerów	289
Tworzenie testów dla HTTP POST lub GET	290
Wykorzystanie konstruktów Ruby JavaScript jako częściowego wizualizatora	290
Sterowanie przepływem Rails	292
Podsumowanie	295

Część III Programowanie RIA. Szczegóły 297

Rozdział 10. Walidacja formularza 299

Dołączanie walidacji formularza do systemu	299
Walidacja całego formularza po stronie serwera	300
Walidacja po stronie klienta z użyciem JavaScriptu	301
Asynchroniczna walidacja jednego elementu formularza po stronie serwera	301
Pobieranie informacji od użytkownika	302
Niesprawdzany formularz	303
Walidacja w praktyce: walidacja po stronie klienta z użyciem Dojo	307
Instalacja Dojo Toolkit	309
Zaczynamy pracę z Dojo	309
Wykorzystanie funkcji walidacji po stronie klienta	310
Wykorzystanie widgetów walidacji Dojo	311
Walidacja w praktyce: walidacja całego formularza po stronie serwera	325
Implementacja strony serwera	325
Definiowanie modelu	325
Definiowanie kontrolera	328
Definiowanie klasy pomocniczej	330
Kompilacja i uruchamianie serwletu	334
Implementacja strony klienta przez wzbogacenie widoku	335
Rozbudowa klasy pomocniczej o walidację pozostałych pól formularza	337
Walidacja w praktyce: walidacja nazwy użytkownika po stronie serwera	342
Rozbudowa widoku	343
Rozbudowa kontrolera	345
Podsumowanie	347

Rozdział 11. Użyteczność formularza 349

Podstawowe informacje	350
Jak utworzyć użyteczny formularz?	351
Ograniczenia synchronicznego JavaScriptu w formularzach	351
Zalety technologii AJAX i efektów JavaScript w formularzach	352
Użyteczność formularzy w praktyce: umieszczanie w polu formularza danych z serwera	353
Implementacja strony serwera	354
Implementacja strony klienta poprzez rozbudowę widoku	360
Użyteczność formularzy w praktyce: wykorzystanie pola kombinowanego zasilanego z serwera	365
Przykład pola kombinowanego z autouzupełnianiem: marka samochodu	365
Przykład pola kombinowanego zasilanego z serwera: marka samochodu	365
Implementacja strony serwera	368
Implementacja strony klienta poprzez rozbudowę widoku	371
Podsumowanie	379

Rozdział 12. Technika przeciągnij i upuść	383
Poznaj Prototype i script.aculo.us	384
Poznaj możliwości Prototype i script.aculo.us	384
Skąd uzyskać Prototype i script.aculo.us?	385
Elementy DOM w świecie fizycznym	385
Klasa Draggable	386
Droppable	388
Listy sortowalne	390
Zastosowanie techniki przeciągnij i upuść: edytor ideaStax	391
Podstawowy projekt	391
Planowanie przeciągania i upuszczania	393
Plan implementacji	394
Edytor ideaStax pracujący wyłącznie po stronie klienta	395
Dynamiczne tworzenie slajdów z użyciem klasy Builder z biblioteki script.aculo.us	395
Zapełnianie Biblioteki przeciągalnymi slajdami	398
Tworzenie panelu edytora typu Droppable	400
Implementacja sortowalnej Osi czasu	403
Podsumowanie implementacji pracującej wyłącznie po stronie klienta	406
Edytor ideaStax wspierany przez serwer	407
Rozpoczynamy pracę z Rails	408
Tworzenie modelu	410
Tworzenie widoku	415
Tworzenie kontrolera	421
Kompletny edytor ideaStax	425
Podsumowanie	426
Rozdział 13. Interakcja z użytkownikiem, efekty i animacje	429
Wprowadzenie do efektów	430
Klasa Effect z biblioteki script.aculo.us	431
Używanie klasy Effect	432
Przykład: powiadomienia oparte na efektach	432
Wiązanie efektów z innymi akcjami script.aculo.us	435
Łączenie efektów	439
Okna modalne i podgląd w ideaStax	440
Pozycjonowanie okna podglądu	441
Pobieranie okna podglądu	444
Wyświetlanie i zamykanie okna podglądu	446
Przekształcanie okna podglądu w okno modalne	447
Ukończone okno modalne podglądu	449
Podsumowanie	451
Rozdział 14. Znacznikowanie i klasyfikowanie (I): tworzenie infrastruktury	453
Problem znacznikowania	454
Aplikacja umożliwiająca znacznikowanie	455
Zaczynamy: elementy podstawowe	459
Python	460
MySQL	460
MySQLdb	461
Api zdjęciowe Flickr	461
Serwer Apache	462
MochiKit	463

Eksploracja danych	465
Pobieranie metadanych zdjęć	465
Uzyskiwanie dostępu do metadanych zdjęć	466
Ustawianie modelu	468
Tworzenie tabel na dane	468
Zapełnianie bazy danych zdjęć	473
Ustawianie interfejsu Flickr Photo	476
Zdobywanie znaczników	480
Implementacja chmury znaczników przy użyciu stylów	480
Dane JSON ze skryptu Tagger	482
Przeszukiwanie bazy danych	483
CGI po stronie serwera do pobierania oznakowanych zdjęć	485
Dynamicznie generowana strona RIA	487
Podsumowanie	487

Rozdział 15. Znacznikowanie i klasyfikowanie (II): wykorzystanie społeczności 489

Stax do tej pory	489
Centrum projektowe Stax	490
Wydobywanie koncepcji i kart	492
Zaczynamy: podstawowe elementy	493
Implementacja kontrolerów i widoków	502
Podsumowanie	513

Część IV Wybrane zaawansowane zagadnienia 517

Rozdział 16. Udostępnianie API 519

Wybór sposobu prezentacji usługi	520
Rozwiązania własne	520
Rozwiązanie z wykorzystaniem usługi sieciowej	520
Formalnie czy nieformalnie	521
Udostępnianie API poprzez HTTP	522
Monitorowanie dostępu	522
Definiowanie interfejsu	525
Przykładowe API dla Stax	525
Pisanie dokumentacji	526
Przykładowa dokumentacja	526
Konstruowanie API	528
Podsumowanie	532

Rozdział 17. WIDGETY RIA 533

Badanie gruntu	534
Zaczynamy: podstawowe elementy	534
Rozpakowywanie widgetu przy użyciu widgetu Widget Converter	535
Struktura pliku i jego zawartość	537
Przegląd pliku specyfikacji widgetu	537
Definiowanie pierwszego widgetu — ćwiczenie „Witaj świecie”	538
Podstawowy widget	538
Pakowanie widgetu za pomocą widgetu Widget Converter	541
Rozszerzanie funkcjonalności widgetu	541

Bardziej użyteczny przykład: użycie API Stax	548
Czynności przygotowawcze	549
Tworzenie podstawowego widgetu	550
Dodawanie obsługi funkcji przeciągnij i upuść	552
Wysyłanie koncepcji do serwera Stax	554
Dodawanie preferencji użytkownika	557
Podsumowanie	562

Rozdział 18. Światy wirtualne 563

Spojrzenie wstecz na wirtualny świat: Magic Cap	564
Skok w przód: świat Zamieci	565
Świat Second Life	566
Second Life: od strony programisty	568
Zaczynamy: podstawowe elementy Second Life	576
Łączność pomiędzy Second Life a prawdziwym życiem	577
Demonstracja Second Life w Web 2.0	581
Podsumowanie	584

Dodatki 591

Dodatek A Funkcje i znaczniki walidacyjne biblioteki Dojo 593

Skorowidz 601

1

Co charakteryzuje bogate aplikacje internetowe?

„Powiem ci tylko jedno, młody człowieku — tworzywa plastyczne!”¹

— *Starszy mężczyzna podpowiadający dobrą inwestycję Benjaminowi Braddockowi, granemu przez Dustina Hoffmana w filmie Absolwent (1964)*

Niniejsza książka została napisana przez programistów dla programistów. Jako programiści skłaniamy się ku bezpośredniemu przejściu do szczegółów technicznych. Na przykład, gdy poznajemy nowy język, chcemy poznać takie jego cechy, jak typizacja, deklarowanie, przypisywanie wartości zmiennym, zawieranie i struktury sterujące: iteracje, testowanie i rozgałęzianie, modularyzację i tak dalej. Ten rozdział będzie nieco inny, ale nie mniej wartościowy.

Ten rozdział wyjaśnia, dlaczego warto uczyć się projektowania aplikacji zgodnie z RIA (Rich Internet Applications). W tym sensie będzie on bardziej „filozoficzny” niż pozostałe. Zawiera jedynie kilka przykładów kodu, przede wszystkim wyjaśniając nadchodzącą rewolucję w oprogramowaniu i starając się pomóc w jej zrozumieniu. Reszta książki omawia kod i programowanie, stąd dowiemy się, co się zmieni i jak można to wykorzystać. Dyskusja jest bardziej ukierunkowana na niezależnych twórców oprogramowania przez to, że objaśnia, jak zmienia się krajobraz konkurencyjności. Jednak programiści pracujący w przedsiębiorstwach również skorzystają, gdyż dowiedzą się, jak zwiększyć konkurencyjność zewnętrzną firmy poprzez tworzenie bardziej skalowalnych i skupionych na użytkowniku funkcjonalności, a także dowiedzą się, jak zwiększyć wydajność wewnętrzną poprzez tworzenie lepszych narzędzi umożliwiających pracę grupową.

W tym duchu spróbujemy odpowiedzieć na pytanie, co sprawia, że warto zainteresować się RIA.

¹ Oficjalne tłumaczenie tego cytatu brzmi również „Pozwól młody człowieku, że powiem ci tylko jedno słowo — plastik!” — *przyp. tłum.*

Bogate aplikacje internetowe są „plastyczne”

Prawidłowa wersja cytatu z filmu *Absolwent* z pierwszej strony niniejszego rozdziału odnosi się do najpowszechniejszego rozumienia słowa „plastik” — powszechnie stosowanego, sztucznego materiału, często używanego jako substytut prawdziwego (zwykle droższego) materiału. W filmie tworzywa sztuczne zostały polecone bohaterowi jako dobra inwestycja na przyszłość. Gdy my, jako programiści, patrzymy wstecz na początkowe dni obecnego okresu związanego z nowym wcieleniem internetu, również możemy mówić o plastyczności — w sensie „przekształcania, wzrostu, zmiany, dynamizmu”.

A więc pierwszą cechą bogatych aplikacji internetowych jest *plastyczność*. Nowa generacja aplikacji internetowych jest rozszerzalna, jej interfejs można łatwo przekształcać, czasem jedynie delikatnie modyfikując kaskadowy arkusz stylów (CSS). Wygląd aplikacji internetowych i stron WWW poprzedniej generacji zawsze można było oprzeć na „skórkach”. Poniżej znajduje się prosty przykład demonstrujący, jak bardzo plastyczna (dynamiczna) może stać się zwykła strona HTML, dzięki kaskadowemu arkuszowi stylów (CSS). Jeżeli znasz się na plastyczności stron WWW, możesz pominąć ten podrozdział, w przeciwnym razie czytaj dalej.

Przykład plastycznej (dynamicznej) strony WWW

Na początek strona WWW bez formatowania — nudna i przeciętna. Rysunek 1.1 przedstawia stronę WWW, dla której nie zastosowano stylów. Znajdziemy na niej informacje, ale strona jest statyczna i bez życia.

Poniższy kod tworzy banalną stronę, przedstawioną na rysunku 1.1. Domyślnie umieszczonych jest kilka elementów DOM (ang. *document object model* — obiektowy model dokumentu) — kilka akapitów oznaczanych parami znaczników `<p>` i `</p>` oraz obraz JPEG zadeklarowany w znaczniku `<img />`.

```
<html>
<head>
  <title>Bez CSS</title>
</head>
<body>
  <p> S T . M A R T I N S , A R C H I P E L A G M A Ł Y C H A N T Y L I </p>
  
<p>
  Najmniejsza na świecie wyspa podzielona pomiędzy dwa różne narody.Od prawie 350 lat
  ↳wyspa St. Martin/St. Maarten jest współdzielona przez Francję i Holandię w duchu
  ↳sąsiedzkiej współpracy i obopólnej korzyści.
</p>

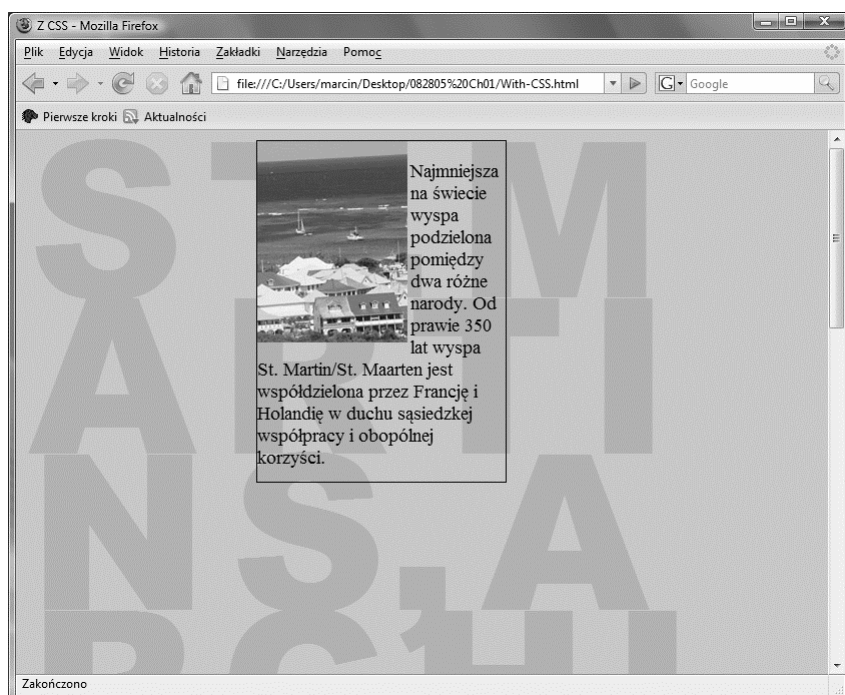
</body>
</html>
```

Dla kontrastu — zastosowanie arkusza stylów dla identycznego DOM gruntownie zmienia stronę, nadając jej styl i szyk, co widać na rysunku 1.2.

Rysunek 1.1



Rysunek 1.2



Jak można zauważyć na poniższym listingu, żaden z elementów DOM nie został zmieniony, ale do kodu strony dodano kilka stylów. Zawartość informacyjna strony jest taka sama, jak na rysunku 1.1, ale teraz się ona wyróżnia:

```

<html>
  <head>
    <title>Z CSS</title>
    <style>
      #sxm {
        position: absolute;
        left: 10px;
        top: 1px;
        font-family: 'Arial Black';
        font-size: 180px;
        line-height: 130px;
        color: rgb(173, 181, 184);
        word-spacing: -50px;
        margin: 0;
        padding: 2px;
        z-index: 1;
      }
      #stm {
        position: absolute;
        left: 240px;
        border: 1px solid #000;
        top: 10px;
        width: 248px;
        height: 340px;
        z-index: 2;
      }
    </style>
  </head>
  <body style="background-color: rgb(194, 203, 207); margin: 0; padding: 0;">
    <div id="sxm">
      S T . M A R T I N S , A R C H I P E L A G M A Ł Y C H A N T Y L I
    </div>
    <div id="stm">
      
      <p>
        Najmniejsza na świecie wyspa podzielona pomiędzy dwa różne narody.
        Od prawie 350 lat wyspa St. Martin/St. Maarten jest współdzielona przez Francję
        i Holandię w duchu sąsiedzkiej współpracy i obopólnej korzyści.
      </p>
    </div>
  </body>
</html>

```

Jak arkusze stylów tworzą lepszą stronę?

Wszystkie tajemnice poprawionej strony znajdują się między parą znaczników `<style>...</style>`, gdzie zostały utworzone dwa style o nazwach `#sxm` i `#stm`. Ponadto elementy informacyjne strony zostały umieszczone we własnych sekcjach oddzielonych parami znaczników `<div>...</div>`. W tych dwóch przykładach, pokazujących tradycyjne wykorzystanie

stron WWW, określone sekcje są wykorzystane do (między innymi) wskazania przeglądarce, gdzie zastosować style. Jest to prosta, lecz skuteczna demonstracja plastyczności, którą można wykorzystać, tworząc nowy styl aplikacji, co wykorzystamy w kolejnych rozdziałach.

Sekcje stają się jeszcze istotniejsze w projektowaniu RIA, gdzie oddzielają segmenty strony, które nowoczesne przeglądarki mogą dynamicznie aktualizować bez konieczności przeładowywania całej strony. W tej książce, a zwłaszcza w rozdziale 10., opisującym aplikację *ideaStax*, spotkamy się z wieloma przykładami, gdy dynamiczne odświeżanie fragmentu strony tworzy istotną więź między użytkownikiem i aplikacją, pozwalając uniknąć całkowitego przeładowania strony, co sprawiało, że praca z aplikacjami internetowymi poprzedniej generacji przypominała bardziej wypełnianie formularza niż pracę z interaktywną aplikacją biurkową.

Ten fragment przedstawiał banalny przykład plastyczności stron pierwszej generacji. Aby zasmakować tego, co projektanci potrafią zrobić ze stroną, odwiedź stronę CSSZenGarden (www.csszengarden.com). Znajdują się tam świetne przykłady plastyczności, uzyskiwanej poprzez zastosowanie różnych CSS dla tej samej zawartości.

Jeden z prostszych sposobów uruchomienia przykładów polega na utworzeniu katalogu, wpisaniu kodów przykładów do osobnych plików HTML i (zakładając, że w systemie zainstalowany jest Python, co w przypadku Mac OS/X i Linuksa jest standardem) uruchomieniu serwera WWW w tym samym katalogu za pomocą poniższej instrukcji:

```
python -m SimpleHTTPServer
```

Zostanie uruchomiony serwer WWW zdolny do obsługi wszystkich stron znajdujących się w tym katalogu i jego podkatalogach. Jeżeli w systemie nie jest zainstalowany Python w wersji 2.4 lub 2.5, można go pobrać z różnych źródeł, na przykład tutaj: www.activestate.com. Jako adresu URL strony należy użyć:

```
http://localhost:8000/<nazwa pliku HTML>
```

Na przykład, jeżeli nazwa pliku ze stroną bez arkusza stylów to `noStyle.html`, należy podać:

```
http://localhost:8000/noStyle.html
```

Zademonstrowaliśmy plastyczność tradycyjnych stron WWW, aby podkreślić fakt, iż pewien stopień dynamizmu był zawsze możliwy. Inną kwestią jest interaktywność. Przed pojawieniem się RIA, zapewnienie interaktywności nie było proste, ale aplikacje Web 2.0 czerpią podstawowe elementy z tradycyjnej specyfikacji HTML dostarczanej przez W3C, jednak nieco zmieniają ich przeznaczenie, aby miały zastosowanie w konstruowaniu bardziej interaktywnych aplikacji.

Bardziej fascynujący styl plastyczności można osiągnąć, wykorzystując API różnych usługowych aplikacji do „zmiksowania” własnej aplikacji. Mashupy przenoszą koncepcję plastyczności na znacznie wyższy poziom, o czym przekonamy się w kolejnym rozdziale.

RIA również różni się od tradycyjnych aplikacji internetowych na wiele istotnych sposobów i zanim przejdziemy do omawiania kodu, zajmiemy się trochę tą kwestią.

Bogate aplikacje internetowe: Web jest przełomem (w końcu)

Często wydaje się, że przełomowe technologie, takie jak RIA, pojawiają się nagle, bez poprzedzającej ich ewolucji. Można argumentować, że tak naprawdę to tylko kwestia zauważenia przez popularną prasę i pojawienie się w krótkim odstępie czasu kilku opisowych sloganów. Technologie leżące u podstaw RIA powstały i dojrzewały jeszcze przed rozdmuchaniem oryginalnej bańki internetowej.

Rozważmy czynniki, dzięki którym aplikacje Web 2.0 mogą w ogóle działać. Aby RIA osiągały podobny czas reakcji, jak ich przodkowie z rodzaju RCP, szybki dostęp do internetu musi być niemal nieograniczony. Bez łącza szerokopasmowego nie można zaoferować użytkownikowi niezakłóconej pracy lub przekonująco oddzielonych od siebie widoku w przeglądarce od obsługiwanych przez serwer modelu i logiki kontrolera. A w latach 90., gdy internet dopiero raczkował, takich łączy nie było. Wystarczy spróbować użyć Google Map, korzystając z łącza modemowego o prędkości 56 Kb/s, a problem stanie się jasny. Na szczęście łącza szerokopasmowe są teraz powszechnie dostępne, a wraz z nimi nasi docelowi klienci. Szerokopasmowy dostęp do internetu jest dostępny niemal w każdym zakątku Stanów Zjednoczonych, a w kwietniu przynajmniej 73 miliony mieszkańców dysponowało takim łączem.

Poza tym, jak zostało to wspomniane wcześniej, zmieniły się podstawy projektowania stron WWW. Początkowo projekt polegał na utworzeniu ładnej strony lub strony zastępującej czasopismo. Teraz projektuje się rzeczy mające przypominać prawdziwe aplikacje. Projektowanie stron WWW stało się projektowaniem aplikacji, a przeznaczeniem CSS nie jest już wzbogacanie układu treści witryny, ale wzbogacanie doświadczeń użytkownika.

Rozwój technologii zajmuje więcej czasu niż nam się to wydaje — zmiany są stopniowe, a zauważalne różnice pojawiają się po latach. Powody, dla których tak właśnie jest, wykraczają poza tematykę tej książki, ale stopniowa ewolucja od przyswojenia marnego spinacza do powstania technik projektowania i implementacji RIA może trwać dziesięciolecia. Tak naprawdę technologie obsługujące RIA były dostępne w różnych formach od dłuższego czasu. Pod pewnymi względami rozwój RIA jest unowocześnianiem modelu klient — serwer. Zasadniczą różnicę stanowi liczba możliwych klientów, po uwzględnieniu rozpowszechnienia urządzeń telefonii bezprzewodowej, przenośności laptopów, rozprzestrzenienia WiFi oraz nowej generacji spakobierców PDA.

Z dalszej perspektywy możemy postrzegać kolejny czynnik umożliwiający tworzenie RIA, AJAX w przeglądarce, jako prostą sztuczkę, dzięki której przysyłanie danych do serwera podczas wprowadzania przez użytkownika danych mających wpływ na inne pola lub dane na stronie implementuje „częściowe odświeżanie strony”. Jak wspomniano wcześniej, współczesne przeglądarki mogą odświeżać określone segmenty strony (wydzielane parami znaczników `<div>...</div>`), nie odświeżając przy tym całej strony. Częściowe przeładowanie znacząco poprawia komfort pracy użytkownika, ponieważ reakcje aplikacji wydają się być bardziej nieprzerywane.

Żadna z tych technologii, czy to samotnie, czy też w grupie, jak w RIA, nie wydaje się być podstawą rewolucji w projektowaniu i dostarczaniu aplikacji, ale rozważmy kilka przykładów, gdy małe zmiany po cichu powodowały dramatyczną zmianę. E-maile i komunikatory internetowe są pod pewnymi względami bardzo do siebie podobne. Obydwie technologie są jakkolwiek na to patrzeć strategiami komunikowania się typu zapisz i prześlij. Obydwie do działania potrzebują sieci. W niektórych przykładach e-mail i komunikat wysłane w tym samym czasie do tej samej osoby mogą przybyć równocześnie. Ale jednak użytkownicy postrzegają je jako zupełnie różne aplikacje. W rzeczy samej, chociaż mają wspólne cechy projektowe, ewoluowały w zupełnie różne aplikacje, inaczej na siebie zarabiające i mające inne grupy projektantów (jeżeli wciąż można mówić o społeczności projektantów e-mail). Poznanie różnic dzielących e-maile i komunikatory internetowe może być pouczające, jeżeli uznamy nową generację aplikacji jako RIA i zastanowimy się, czym różnią się od poprzedników. Rozważmy tabelę 1.1, z której wynika, że dwie różne platformy oprogramowania mogą mieć podobne założenia projektowe, ale mimo to mogą się znacznie różnić dzięki społecznościom, które się wokół nich formują.

Tabela 1.1. Różnice między komunikatorami internetowymi a e-mailem

Charakterystyka	E-mail	Komunikator internetowy
Wzorzec użycia	Środek formalnej komunikacji korporacyjnej.	Wygodny sposób komunikacji, gdy tylko chcemy coś komuś powiedzieć, zwykle w czasie rzeczywistym i zwykle zwięzłego.
Co zastępuje	List biznesowy i formalne kanały komunikacji biznesowej.	Telefon, a nawet rozmowę w cztery oczy.
Łatwość użycia	Wymaga specjalnej konfiguracji i administrowania systemem.	Jest zawsze „włączony”, bez wysiłku ze strony użytkownika. Mówimy o „sprawdzeniu e-maili”, ale raczej nie mówimy o „sprawdzeniu wiadomości z komunikatora”.
Natychmiastowość	Wolniejszy i bardziej ociężały; przed wysłaniem treść musi być przemyślana i wyrażona zwykle językiem formalnym.	Natychmiastowy, rozmowny i społeczny. Mówimy, co chcemy powiedzieć i kiedy chcemy powiedzieć. Choć obydwie platformy wykorzystują protokół zachowań i prześlij, zmiana tempa wymiany informacji w komunikatorach internetowych spowodowała zmianę w postrzeganiu technologii przez użytkowników.
Ciągłość aplikacji	Przerywana, z formalnym odebraniem i wysłaniem oraz formalną prezentacją i argumentacją.	Ciągła, wymiana myśli po prostu „się dzieje”.
Administracyjne obciążenie użytkownika	Często czujemy potrzebę katalogowania i kategoryzowania e-maili, ot tak, „na przyszłość”.	Rzadko powracamy do zapisów z komunikatorów internetowych, choć można użyć narzędzi wyszukiwujących do przeglądania zalogowanych rozmów. To rozróżnienie może się już kończyć za sprawą narzędzi e-mailowych przenoszących pracę do internetu, takich jak Gmail lub RIA, gdzie wyszukiwanie zastąpiło nawigowanie po folderach.

Można przekonywać, że różnice między tymi dwiema aplikacjami mającymi podobne zalety technologiczne są różnicami między tradycyjnymi aplikacjami RCP lub Web 1.0 a RIA. RIA oferują niezauważalność, natychmiastowość, ciągłość doświadczeń użytkownika i brak konieczności formalnej konfiguracji. Są to przełomowe cechy bogatych aplikacji internetowych, które zostaną po kolei omówione w dalszej części rozdziału.

Bogate aplikacje internetowe są niezauważalne

Jesteśmy programistami. Zgadza się, ale czasem jesteśmy też użytkownikami końcowymi i jako użytkownicy aplikacji biurowych przyzwyczailiśmy się do zdobywania (czasem nawet kupowania), instalowania i pielęgnacji aplikacji. Dbamy o ich dobrostan, wkładając wiele niewynagradzanej pracy, by odpowiednio „przystrzyc ją” (ramka na stronie 40), aby doprowadzić aplikacje do stanu, w którym będą mogły „zaspokajać nasze potrzeby”. Rysunek 1.3 przedstawia Google — doskonały przykład niezauważalnej aplikacji. Jest czymś oczywistym, ale zawsze jest w zasięgu użytkownika.

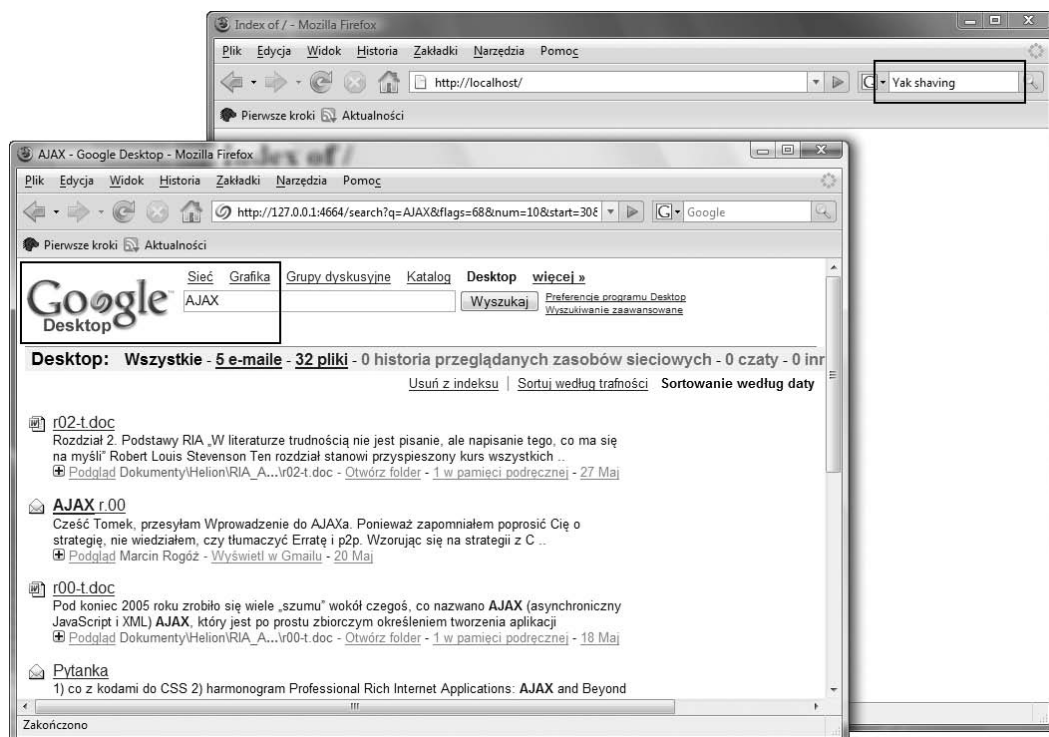
Rysunek 1.3



Niezauważalność Google

Google jest doskonałym przykładem niezauważalnej aplikacji, bez której trudno się obejść. Twórcy Google celowo zwiększyli jego wartość i niezauważalność, osadzając proste okienko w pasku narzędzi w nowoczesnych przeglądarkach i obszarze powiadomień systemu Windows dla wyszukiwania lokalnego (Google Desktop), co obrazuje rysunek 1.4. Bardziej niż jakkolwiek aplikacja Google stało się elementem przeglądania w takim stopniu, że wielu użytkowników przywołuje go niemal nieświadomie. Nieświadome użycie jest cechą charakterystyczną usług lub produktów najwyższej cenionych przez użytkowników.

Tradycyjne aplikacje przypominają związek wymagający wiele nakładów pod tym względem, że dostarczają one wartość, ale niekoniecznie w takim stopniu, by wyrównać inwestycje poczynione przez użytkownika. Im więcej eksperymentujemy z nowym paradygmatem, tym więcej rodzajów aplikacji, które dotąd pisaliśmy, wydaje się nam pokracznych i prymitywnych.



Rysunek 1.4

Im więcej korzystamy z RIA jako użytkownik, tym rzadziej zauważamy fakt, że one po prostu są „tam”, że w jakiś sposób znikają w tle. Jako programista możesz pamiętać objawienie, gdy nabyłeś biegłość z wieloma aplikacjami w wierszu poleceń Linuksa (lub jeżeli jesteś wystarczająco dojrzały — Uniksa). Choć rzadko postrzegane w taki sposób, `find`, `grep`, `ps`, `ls` i `grep` są aplikacjami, a „zaklęcia” wpisywane w konsoli, takie jak:

```
$ ls -lat | grep html | wc -l
```

aby policzyć liczbę plików HTML w katalogu, były wczesnym przykładem łączenia aplikacji w Uniksie w celu utworzenia złożonej aplikacji, czyli czegoś w rodzaju współczesnego mashupa. Po chwili jednak prawdopodobnie o nich zapominałeś — były używane do wykonania pracy i nawet nie były postrzegane jako oprogramowanie.

Skupianie funkcjonalności RIA

Poruszając się po świecie aplikacji Web 2.0 można zauważyć, że poszczególne aplikacje (jak dotąd) nie cierpią na rozděcie funkcjonalności, takie jak w wypadku aplikacji natywnych. Wydają się skupiać na robieniu jednej rzeczy, ale robieniu jej dobrze.

Nie można jednak powiedzieć, że nie da się napisać naprawdę złej RIA, albo że wskutek nacisków marketingowych nie powstaną aplikacje internetowe przeładowane niepotrzebną funkcjonalnością. Może się tak zdarzyć. Chodzi o to, że to, czym się teraz zajmujemy i co jest już gotowe, jest warte docenienia za swą prostotę. Aby poznać przemysłowy punkt widzenia na temat zwalczania nadmiarowości, przeczytaj „Wywiad z 37Signals”.

Jednak dwa czynniki mogą powstrzymać nadmiarowość i tworzenie złych projektów:

- Przeładowane oprogramowanie nie będzie się tak szybko uruchamiało ani reagowało.
- Dla użytkowników koszty migracji będą niższe.

Rada często powtarzana w niniejszej książce i miejmy nadzieję wystarczająco podkreślona w kodzie brzmi, aby projektować z przeznaczeniem dla jednej funkcji i ze zrozumieniem tego, że użytkownicy nie chcą „wgryzać” się w oprogramowanie internetowe. Najważniejszą zasadą do zapamiętania jest ta, że w internecie doświadczenie użytkownika jest najważniejsze.

Używamy tego, co oferujemy

Jedną z decyzji, którą autorzy niniejszej książki podjęli na samym początku jej pisania, było wykorzystanie edytora tekstu opartego na AJAX, a nie narzędzia biurkowego. Nie tylko dlatego, aby używać tego czym się zajmują, jak to mawiano w erze dotcomów, ale dlatego, że często musieli nagle zacząć współpracę, niezależnie od swej lokalizacji, niezależnie od aktualnie używanego systemu operacyjnego. Dzięki temu nie musieli się martwić o to, czy zainstalowane jest „odpowiednie” oprogramowanie, czy też gdzie dokładnie znajdowała się treść, nad którą pracowali. Dołożenie wysiłku, by tworzone oprogramowanie było czymś oczywistym dla dużej liczby osób, jedynie zwiększa naszą reputację i wartość. (Więcej informacji na ten temat znajdziesz w podrozdziale „Jak mogę zarabiać na RIA?”).

Inny powód, dla którego następuje przejście do innej generacji aplikacji, z punktu widzenia użytkownika jest oczywisty, ale ma niewiele wspólnego bezpośrednio z RIA. Przeglądarki obsługujące karty przypominają pulpit, o jakim zawsze marzyliśmy, a jakiego nigdy nie dostaliśmy.

Użytkownicy przyzwyczaili się do chaotycznego środowiska Windows, jakby było to podstawowe prawo natury. Towarzyszyli im programiści, biorąc „współudział w przestępstwie”, ale zastanówmy się chwilę. Okienkowy świat tradycyjnego pulpitu jest chaotyczny, a przełączanie z obszaru jednej aplikacji do innej jest kłopotliwe, bo wiąże się z nieustannym szukaniem, przeciąganiem okien i manipulowaniem systemem plików, aby umieścić zawartość w odpowiednim miejscu dla dobra aplikacji.

Bogate aplikacje internetowe burzą mury otaczające ogród

Aktualnie używane platformy skłaniają się do kontrolowania i umieszczania własności intelektualnej w „ogrodzie otoczonym murem”. Gdy tworzymy dokument za pomocą aplikacji RCP, choć to my jesteśmy autorami, nie posiadamy zawartości w oderwaniu od narzędzia,

którego użyliśmy do jej utworzenia. Nie możemy napisać lub odczytać dokumentów Microsoft Worda bez czegoś, co działa podobnie do Microsoft Worda. W rzeczywistości producent Worda wolałby, abyśmy nie używali niczego oprócz Worda do pracy z czymś, co powinno być „naszymi” danymi. Posuwając to rozumowanie krok dalej, oni *nalegają*, abyśmy używali systemu operacyjnego, który z chęcią nam sprzedadzą, aby uruchomić aplikację, którą równie chętnie nam sprzedadzą, a to wszystko po to, abyśmy mieli dostęp do informacji, które tak naprawdę powinny być naszymi informacjami.

W świecie idealnym (z punktu widzenia *producenta tradycyjnych aplikacji biurkowych*):

- Zawartość nie może istnieć bez aplikacji pośredniczącej. Producenci sprzedają aplikacje i przechowują model zawartości w formacie uniemożliwiającym komukolwiek łatwe przerwanie ich dominacji nad użytkownikiem. To producent decyduje, czy, kiedy i jak poprawiane są błędy. Producenci muszą tworzyć ogromne systemy masowej dystrybucji, zależące od wysiłków użytkowników, którzy będą mieli tak aktualne oprogramowanie, jak będzie tego chciał producent.
- Aplikacje nie mogą istnieć niezależnie od określonego systemu operacyjnego. Producent opublikuje tylko tyle stosu wywołań systemowych, ile uzna za stosowne, by zachęcić niezależnych producentów oprogramowania do tworzenia aplikacji, które będą obsługiwane przez system operacyjny. Zwykle „pobłogosławia” kilka języków, a resztę pomina.
- System operacyjny i aplikacja są ciasno związane, dzięki czemu producent wedle własnego uznania może stwierdzić, że są przestarzałe. Producenci będą tak robić pod hasłem zwiększania wydajności, ale oznacza to również, że mogą oni (i to robią) dowolnie łamać paradygmat Model-Widok-Kontroler czy dodawać inne, nieudokumentowane fragmenty do głównego kodu. Oznacza to także, że wewnętrzny mechanizm systemu operacyjnego lub aplikacji staje się tajemny, kruchy i niejasny dla niezależnych producentów oprogramowania i twórców spoza przedsiębiorstwa. Więc zewnętrzni programiści, chcąc dostarczyć konkurencyjne aplikacje, muszą nieźle się napocić, by osiągnąć rezultaty porównywalne z aplikacjami producenta systemu. Dominujący producenci chronią więc swą władzę nad „dojnymi krowami” — aplikacjami, dzięki którym płynie ich strumień dochodów.
- Każde narzędzie tworzy „wyspę automatyzacji” z własnym modelem danych, własnym kontrolerem i widokiem zależnym od systemu operacyjnego i narzędzia jako platformy. Ponieważ producenci są zafascynowani aktualnym stanem świata, będą mieli problemy z tworzeniem (na przykład) aplikacji opartych na sieci, umożliwiających łatwą współpracę, niezależnych od systemu operacyjnego, czy rozszerzalnych — czy to przez użytkownika końcowego, czy też przez innych twórców oprogramowania.
- Podczas kampanii marketingowych, konferencji programistycznych i innych interakcji, producenci starają się przekonać zarówno niezależnych producentów oprogramowania, jak i użytkowników końcowych, że pewnego rodzaju pogodzenie się z tymi kłopotami skutkuje bezproblemową pracą (w przypadku użytkowników) i ochroną oraz gotowym rynkiem zbytu (dla niezależnych producentów). Dzięki temu tworzą barierę wejścia, przez co ceny oprogramowania są sztucznie zawyżone i trudne do powielenia.

Byłoby zaskakujące, gdyby którykolwiek element powyższej listy stanowił zaskoczenie, czy to dla twórcy programowania, czy dla użytkownika końcowego. Jest to podstawowa doktryna w przemyśle oprogramowania, obowiązująca przez ostatnie 30 lat. Pisząc niniejszą książkę, przedstawiając nowe techniki, prosimy Cię, byś zadał sobie pytanie, czy świat musi być w ten sposób urządzony w imię jakiś wyższych, nienaruszalnych celów, czy też jest tak tylko dlatego, że stało się to zwyczajem, ograniczającym naszą innowacyjność i ducha przygody.

W tradycyjnym świecie oprogramowania — tym, który powinien pozostać za nami — narzędzia i aplikacje stały się pułapkami zarówno dla twórcy, jak i dla użytkowników, ponieważ wiążą zawartość i własność intelektualną. Jednak słowo ostrzeżenia: Sam fakt nauczenia się tworzenia rozszerzalnych, możliwych do łączenia aplikacji tworzonych w paradygmacie oprogramowanie-jako-usługa nie sprawi, że staniesz się odporny na zagrożenia z innych stron, tak samo nieprzyjaznych dla oprogramowania o otwartym kodzie, jak tradycyjni producenci systemów operacyjnych i aplikacji. Więcej informacji na ten temat znajdziesz w podrozdziałach „Bogate aplikacje internetowe są ukierunkowane na sieć” i „Bogate aplikacje internetowe są ukierunkowane na przeglądarkę”.

Jak na ironię, choć klienci przywykli do otoczonych murem ogrodów, programistom one niezbyt przeszkadzają. Rozważny szereg sposobów, w jakie możemy tworzyć i manipulować kodem Java/Python/Ruby. Wystarczy VI, Eclipse, Netbeans, EMACS lub różnego rodzaju notatniki (których liczba jest nie do ogarnięcia). Gdy potrzebujemy przejrzeć kod, wystarczy otworzyć okno terminala, wpisać `cat mojkod.py` i to wystarczy. Jednym z powodów, dla których możemy działać w takiej nieskrępowanej atmosferze, jest fakt, że na całym świecie ASCII to ASCII i żaden producent nie może przywłaszczyć sobie tego formatu. Dopóki możemy tworzyć dokumenty ASCII, dopóty możemy programować.

Bogate aplikacje internetowe tworzą nowe (nieogrodzone?) ogrody

Celem tej książki jest zmotywowanie programistów do pisania nowej generacji aplikacji, tworzących ogród innego rodzaju. W tym ogrodzie bogactwo aplikacji jest uwolnione, aby interakcja twórców i treści przebiegała z mniejszymi zakłóceniami i dzięki mniejszemu nakładowi w celu zdobycia i zarządzanie narzędziami, niż miało to miejsce w przypadku poprzednich generacji aplikacji z ogrodem „otoczonym murem”. Jeżeli pierwsza odsłona internetu była związana z dostarczaniem i prezentowaniem połączonych informacji, a druga była związana z przysyłaniem informacji osobistych w celu zakończenia transakcji, wówczas trzecia z pewnością jest związana z tworzeniem, używaniem zawartości i manipulowaniem nią w *społecznym lub współpracującym środowisku*. W szczególności powstały aplikacje zasilane przez nas — użytkowników.

Aplikacje generacji biurowej umożliwiały użytkownikom pisanie, rysowanie, drukowanie i przysyłanie swoich wytworów do względnie niewielkiego koła konsumentów. Ale obecnie zawartość jest potencjalnie otwarta dla nieograniczonej liczby konsumentów, recenzentów, opiniodawców, bloggerów, którzy przenieśli poziom tworzenia zawartości, konsumpcji

i potencjał debaty i dyskusji na zupełnie nowe poziomy. Z oceniania zawartości przez „mądrość zbiorową” robi się w tej chwili biznes. Zastanówmy się na przykład, jak trudno byłoby utworzyć del.icio.us, Slashdot.org czy dig.com w generacji biurkowej.

Ograniczenia w nieogrodzonym ogrodzie

Spółeczność programistów pracujących z tą nową architekturą (do której należysz lub będziesz należał wkrótce) ustaliła kilka istotnych pomysłów, jak tworzyć wartość z użyciem RIA. Niezależnie od tego, czy tworzymy aplikację do tworzenia, przechowywania i dzielenia się treścią (zbyt wiele do wymienienia, ale na przykład Flickr), komentowania treści (na przykład Slashdot), współpracy nad zawartością (weźmy Writely i aplikację tworzoną w niniejszej książce — *ideaStax*), czy też remiksującą zawartość (ponownie, zbyt wiele, by wymienić, ale weźmy mashupy Google Map, czy też widżety Apple lub Yahoo!), łańcuch rozumowania architekt/programista wymusza kilka interesujących ograniczeń projektowych dla aplikacji nowej generacji:

- Współdzielona zawartość i dynamiczne publikowanie treści są podstawowymi elementami nowej epoki internetu.
- Istnieje mniejszy opór związany z tworzeniem i publikowaniem treści niż w przeszłości. Do pewnego stopnia, blogi zastępują czasopisma i książki. A w najgorszym razie rozszerzają możliwości publikacji.
- Im coś łatwiej współdzielić, tym lepiej. Może być tak dobrze, że możliwe będzie zarobienie na tym.
- Szybkie łączy się powszechne wśród użytkowników korzystających z aplikacji nowej generacji.
- Po uwzględnieniu możliwości tworzenia dużych społeczności wokół określonych kategorii zawartości, aplikacja musi dotrzeć do dużej liczby użytkowników końcowych.
- W swej dynamice aplikacja musi przypominać sieć radiową czy telewizyjną — oczywiście z tą różnicą, że sieci tego rodzaju mogą być jedynie „konsumowane”. Nie umożliwiają one bogactwa interakcji, do których zachęcają sieci nowego rodzaju.
- Użytkownik końcowy będzie często korzystał z przeglądarki do uzyskania dostępu do aplikacji.
- W obszarze przeglądarek nie ma monopolu. Są one obsługiwane praktycznie wszędzie (komputery osobiste, telefony komórkowe i inne urządzenia osobiste); z tego też względu nowa aplikacja nie musi korzystać z API określonego przez producenta. W rzeczywistości z cyklu deweloperskiego usunięte zostaje pisanie dla specyficznego API po stronie klienta. Poza tym może jeszcze bardziej zoptymalizować wartość nowej aplikacji zarówno dla programisty, jak i dla potencjalnego użytkownika końcowego.

- Istnieje znormalizowany, powszechny API przesyłania danych (HTTP) oraz znormalizowany powszechny standard wyświetlania po stronie klienta (HTML).
- Twórca aplikacji wciąż może przechowywać prawdziwą wartość po stronie serwera w modelu danych oraz logice pracy z danymi (które to nie muszą być ujawniane użytkownikom aplikacji).

Żadne formalne grono nie zasiadło gdzieś w Dolinie Krzemowej i zdecydowało, że te zasady będą podstawami RIA. Żadna tego typu lista nie została nigdzie formalnie opublikowana, ale rozumowanie wydaje się być rozsądne i mocne, jeżeli weźmiemy pod uwagę następujące sprawy:

- Ta książka jest tworzona w czasie, gdy jest sens, aby programiści ją pisali i czytali.
- Charakterystyki większości RIA są poznawane „na żywo”.

Niektóre z tych spostrzeżeń po prostu wyewoluowały z powodu dostępności technologii, które je umożliwiają. Odkąd pękła pierwsza bańka internetowa, twórcy oprogramowania powrócili do desek kreślarskich z dobrym zrozumieniem możliwości architektonicznych. Innowatorzy zapoczątkowali aktualny boom internetowy, tworząc wartość wokół pomysłów i aplikacji, które wcześniej nie były możliwe. Istnienie tej nowej architektury dla aplikacji tworzy nowy „obszar powierzchni” dla treści, który trudno było uzyskać w erze biurkowej.

Technicznie rzecz ujmując, niezmaczone doświadczenie przy korzystaniu z narzędzi bazujących na internecie jest jednym z filarów nowej generacji aplikacji. Rozpowszechnienie łączy szerokopasmowych jest jednym z powodów, dla których możemy pisać aplikacje rozproszone, nie martwiąc się (zanadto) o czas reakcji, podobnie jak przy tworzeniu aplikacji poprzedniej generacji. Dostępność nieprzeciążonych przeglądarek stanowi gotowy pojemnik na aplikacje.

Jednakże RIA to coś więcej niż proste przeniesienie możliwości tradycyjnych aplikacji na nową platformę. Jeżeli chcesz utworzyć nową „niezbędną” aplikację, musisz pamiętać o własnie wyłożonej wizji: Udostępnij wierzchnią warstwę aplikacji opartej na treści jak największej liczbie „nas”. To, bardziej niż cokolwiek innego, jest nowym głównym założeniem projektowym.

W okresie pisania niniejszej książki około miliard osób tworzyło w zawartości online lub korzystało z niej. To są nowi użytkownicy końcowi, a jeżeli im to ułatwimy, zapewnią nieopisane bogactwo — dzieląc się wiedzą, tworząc sieci społeczne — osobistymi perspektywami i chęcią współpracy. Wszystko to może być wykorzystane dla dobra aplikacji. Nagle stało się możliwe (być może nawet po raz pierwszy w zapisanej historii) doświadczenie masowej współpracy, zwanej ekonomią daru. Jest to możliwe, gdy piszemy aplikacje umożliwiające wniesienie małego wkładu.

Poznajmy manifest opozycji

Nowy rodzaj ogrodu, który będziecie tworzyć Ty jako programista oraz Twoi użytkownicy końcowi można wyrazić w czymś w rodzaju „manifestu opozycji”, który jest użyty jako podstawa projektu przykładów kodu oraz aplikacji *ideaStax* przedstawianych w niniejszej

książce. Ten manifest opozycji brzmi całkowicie odmiennie od opisanego wcześniej manifestu ery aplikacji biurkowych. Prezentujemy go jako zestaw wzorców projektowania na poziomie aplikacji, które omówimy w kolejnych podrozdziałach.

Wzorzec 1: ujawnij (przynajmniej część) zdalnego API

Programiści RIA powinni tworzyć zawartość w taki sposób, aby inne aplikacje pośredniczące, o których mogliby marzyć inni programiści lub użytkownicy, mogły z nią współpracować, wyświetlać ją, czy też zmieniać jej przeznaczenie.

Więcej informacji na ten temat można znaleźć w rozdziale 3., gdy będziemy tworzyli mashupy. O mashupach i remiksach mówimy, gdy nowe aplikacje (początkowo niezamierzone) są tworzone z danych i usług mających pierwotnie inne przeznaczenie.

Ciekawe mashupy GoogleMaps można znaleźć pod adresami www.housingmaps.com (domy do wynajęcia) oraz www.mywikimap.com (wyszukiwarka tanich stacji benzynowych).

Wzorzec 2: używaj popularnego formatu danych

Inny (choć podobny) wzorzec sugeruje, że programiści RIA powinni tworzyć modele danych leżące u podstaw aplikacji opartych na usługach w taki sposób, aby mogli oni lub inni programiści je wykorzystać. Podczas gdy serwer może przechowywać model danych w dowolnej wygodnej formie (nie ma znaczenia, czy będzie to relacyjna baza danych SQLite, zamontowany sieciowy system plików, czy dane w formacie XML), dane wychodzące do użytkownika muszą być wyrażone jako coś, co obsłuży dowolna liczba czytelników/parserów.

Nazywanie tego „wzorcem projektowania” i sugerowanie, byś to wspierał, jest drobnym oszustwem. Z rozdziałów 10. i 11. wynika, że dane wychodzące do przeglądarki muszą być albo w formacie XML albo JavaScript Object Notation (JSON). Podobnie, jak prędkość świata — to nie jest po prostu dobry pomysł — to jest prawo należycie egzekwowane przez wszystkie przeglądarki.

Wzorzec 3: przeglądarka tworzy doświadczenie użytkownika

Jedynie prawdziwe ograniczenie możliwości aplikacji stanowią ograniczenia popularnych przeglądarek. Jako programiści możemy sobie poradzić z tym ograniczeniem, ponieważ, jako wymiana, i tak jest to lepsze niż wysoki stos ograniczeń wejścia i barier, doprowadzających nas do szału w poprzedniej epoce oprogramowania.

Jeżeli stopień złożoności wyświetlanego obrazu przekracza możliwości współczesnych przeglądarek, można użyć dodatków, takich jak Flash. Dodatkowo, w sytuacjach skrajnych można uruchamiać zewnętrzne aplikacje, choć może to powodować rażącą przerwę w pracy. Jednym z dość udanych tego typu wzorców na poziomie aplikacji jest Java Web Start, gdzie użytkownik przechodzi do adresu URL, który zawiera JavaScript uruchamiający aplikację.

Wzorzec 4: aplikacje są zawsze usługami

Aplikacje są częścią architektury zorientowanej na usługi (ang. *service-oriented architecture* — SOA), w której aplikacje są usługami tworzonymi od podstaw lub jako pochodne połączenia zewnętrznych warstw innych aplikacji. Choć podlega ograniczeniom (takim jak dostępność usług, od których może zależeć aplikacja), SOA stanowi odpowiednik ewolucyjnej adaptacji w dziedzinie oprogramowania. Co ważne, do utworzenia tego wzorca nie był potrzebny żaden stos protokołu tworzony przez komitet producentów. Co więcej, w przeciwieństwie do stosu UDDI/WSDL, Web 2.0 SOA jest prosty i niemal intuicyjny.

Wzorzec 5: zezwalaj na agnostycyzm użytkowników

RIA nie zmuszają użytkowników do przejmowania się systemami operacyjnymi czy przeglądarkami. Jeżeli tylko przeglądarka obsługuje skrypty, aplikacje „po prostu działają”. Programiści mogliby wykorzystać „tajemną wymianę potwierżeń”, czy inne implementacje na zapleczu, powodujące związanie z określonym producentem lub systemem operacyjnym (prawdopodobnie możesz wymienić jedną przeglądarkę i jednego producenta systemów operacyjnych, który tak właśnie czyni). Jednak wadą takiego diabelskiego paktu jest fakt, że ostatecznie programista będzie miał znacznie więcej pracy, gdy po zdobyciu pewnej liczby użytkowników na jednej platformie, okaże się, że w jego interesie leży rozszerzenie aplikacji na wszystkie pozostałe platformy. Lepiej nie wspierać producentów posiadających własny porządek implementacji zaplecza i projektować od początku, aby uniknąć śliskiego zbocza, z którego można wpaść w pułapkę kolejnego ogrodu otoczonego murem. A więc najlepiej jest unikać zależności od systemów operacyjnych i natywnych bibliotek. Jeżeli jesteśmy w stanie to zrobić, możemy pominąć mnóstwo niepotrzebnych szczegółów podczas pisania aplikacji, a w szczególności części, które zmniejszają stabilność aplikacji. Zamiast tego możemy się skupić na logice tego, co naprawdę chcemy zaoferować użytkownikowi.

Wzorzec 6: sieć jest komputerem

Pierwotnie będący hasłem kampanii reklamowej Sun Microsystems z lat 80., slogan ten jest lub nie jest trendy. Jego przykładem mogą być Java Applets wraz z rozproszonym modelem X11/Motif z lat dziewięćdziesiątych. Obydwa te modele nie sprawdziły się z powodu słabości infrastruktury. X11 się nie sprawdził, ponieważ wymagał dużej wydajności od drogich procesorów „stacji roboczych” z tamtej epoki, a także dlatego, że jego model sieciowy nie mógł wykroczyć poza LAN, ponieważ era łączy szerokopasmowych rozpoczęła się około 10 lat później. Java Applets zawiodły przez swoje rozmiary (długi czas ładowania) i niską wydajność — ponownie, do rozpowszechnienia łączy szerokopasmowych pozostało jeszcze kilka lat.

Obecnie szerokopasmowe łącza są niemal wszechobecne w cyfrowym krajobrazie. Ponadto, ponieważ aktualne aplikacje RIA nie są zanadto rozbudowane i doskonale mieszczą się w obszarze pamięci zajmowanym przez nowoczesną przeglądarkę, porażka wydaje się mniej prawdopodobna. A więc „systemem operacyjnym” dla tej nowej generacji aplikacji nie jest jeden, określony system operacyjny, ale raczej internet jako całość. Zgodnie z tym możemy programować dla osobnych komponentów paradygmatu Model-Widok-Kontroler, pomijając

(w większości) sieć pośredniczącą w aplikacji. Wciąż pozostaje kilka prawdziwych trosk związanych z siecią tkwiącą w środku aplikacji, i potrzebujemy kodu chroniącego doświadczenie użytkownika. Wiele z tych kwestii zostanie omówionych w części III.

Wzorzec 7: Web 2.0 jest zjawiskiem kulturowym

Poza najwcześniejszymi latami komputerów osobistych, gdy grupa Homebrew Computer Club złożona niemal z samych mężczyzn, która stworzyła pierwszy komputer Apple, zdefiniowała kulturę hakerską, komputer osobisty nigdy nie stanowił zjawiska kulturowego. Przeciwnie rzecz się ma z Web 2.0, które zawsze było związane z kulturą. Użytkownicy zbierają się wokół „podobnych treści”. W kulturze Web 2.0 społeczność jest wszędzie. Digg, craigslist i Slashdot istnieją jako skupiska komentarzy użytkowników. Wąsko tematyczne podcasty zastępują telewizję i radio. Blogowanie zastępuje pozostałe formaty publikowania i masowej dystrybucji. Remiksowanie i mashupy są możliwe do wykonania, ponieważ modele danych, normalnie zamknięte lub zastrzeżone są udostępniane przez API usługi, ale również one nie istniałyby bez pasji i trudu wnoszonego przez użytkowników. Niektóre przedsiębiorstwa (na przykład Ning na *ning.com*) nawet są w stanie na tej kulturze zarabiać, oferując platformę dla twórczych *hobbyistów*, którzy tworzą społeczne aplikacje w oparciu o istniejące API. W rozdziale 3., dotyczącym remiksów i mashupów, dowiemy się, co mogą zrobić użytkownicy, mając skromne umiejętności programistyczne i internet jako platformę.

Tworząc aplikację, należy brać pod uwagę kulturę społeczną, która może powstać wokół aplikacji. Możliwości społeczne i umożliwiające współpracę (na przykład współedycję, subiektywną klasyfikację dostępnych treści) możemy wbudować bardzo łatwo. Jeżeli fakt, iż aplikacja istnieje jedynie „w chmurach” uznamy za wadę, zaletą jest fakt, że aplikacji sieciowej dotyczy również prawo sieci: wartość aplikacji wzrasta jako funkcja logarytmiczna liczby użytkowników. Wiele dostępnych obecnie RIA posiada wszystkie te cechy lub dużą ich liczbę.

Weźmy poprzednika aplikacji opartych na dzieleniu się treścią — blog i jego bliską krewniaczkę, wiki. Liczba czytelników wielu blogów zbliża się do liczby czytelników tytułów drukowanych lub przekracza ją, a pod względem obsługi koordynacji i współpracy wielu użytkowników, wiki są znacznie bardziej elastyczne od swoich biurkowych poprzedników papernych na zasoby systemowe (takich, jak na przykład Groove).

Bogate aplikacje internetowe są zawsze aktualne

Ważny powód, dla którego zaczęto używać obniżonych barier (czynnik „strzyżenia jaka”, jak to nazwaliśmy) polega na tym, że w momencie, gdy użytkownicy zaczęli częściej korzystać z niezauważalnych i wszechobecnych aplikacji stało się coś niezmiernie gwałtownego, a jednocześnie oczywistego — uaktualnienia oprogramowania, poprawki bezpieczeństwa i masowa dystrybucja service packów po prostu zniknęły; nagle stały się artefaktem minionej epoki. Powód tej dużej zmiany, która zaszła wśród użytkowników końcowych, jest teraz prosty, w jakiś sposób „naturalny”.

Jako programiści możemy zrozumieć dogłębne, ale wciąż proste wyjaśnienie w sposób, który mogą pominąć końcowi użytkownicy. Ponieważ RIA dostarczają do przeglądarki świeży egzemplarz aplikacji przy każdym przeładowaniu, użytkownik końcowy zawsze automatycznie otrzymuje najnowszą wersję z wszystkimi poprawkami i aktualizacjami, nawet jeżeli skończyliśmy je tworzyć poprzedniej nocy. Brak bólów głowy związanych z masową dystrybucją, brak skomplikowanych łat w zabójczym uścisku końcowego użytkownika.

Strzyżenie jaka oraz style tworzenia tradycyjnych aplikacji kontra RIA

Wielkie brawa za każdą wymyśloną sytuację końcowego użytkownika, w której można obniżyć bariery korzystania. Dużą część tworzenia tradycyjnych aplikacji RCP zajmuje to, co obecnie nazywa się *strzyżeniem jaka* (ang. *yak shaving*). Według Wikipedii termin ten mógł powstać w CSAIL w MIT. W każdym razie odnosi się on do każdej, wydawałoby się bezcelowej aktywności, która tak naprawdę jest niezbędna do rozwiązania problemu rozwiązującego problem, który po kilku rekurencjach rozwiązuje prawdziwy problem, którym się zajmujemy.

List znajdujący się pod adresem <http://projects.csail.mit.edu/gsl/gsb-archive/gsb2000-02-11.html> zawiera jeszcze jedną definicję i przykład:

...ze strzyżeniem jaka mamy do czynienia wtedy, gdy wykonujemy jakieś głupie, nużące zadanie, niemające oczywistego związku z tym, nad czym powinniśmy pracować, ale łańcuch kilku nastu związków przyczynowo-skutkowych łączy to, co robimy z pierwotnym metazadaniem.

Oto przykład:

„Pisałem pracę naukową i okazało się, że potrzebuję cytatu. Widziałem ostatnio post na grupie Comp.arch, który cytował ten artykuł, więc uruchomiłem gnus. Gdy szukałem postu, natrafiłem na inny post, którego kodowanie MIME popsło moją starożytną wersję gnus, więc przebrałem i pobrałem najnowszą wersję gnus.

Niestety nowa wersja gnus nie działała z emacs 18, więc pobrałem i skompilowałem emacs 20. Oczywiście musiałem potem zainstalować zaktualizowane wersje kilku innych pakietów, aby inni użytkownicy nie zrobili mi krzywdy. Gdy w końcu spróbowałem uruchomić nowe gnus, nie działały ze starą konfiguracją. Dlatego też zgłębiłem właśnie strony informacyjne gnus i plik .emacs — to wszystko jest częścią pisania mojej pracy naukowej”.

I to moi drodzy jest właśnie strzyżenie jaka.

Prawdopodobnie przekonasz się, że tworzenie RIA nie wymaga tak częstego strzyżenia albo że przynajmniej jest ono zastąpione czymś bardziej satysfakcjonującym — na przykład strzyżeniem wełny, aby uatrakcyjnić aplikację. Albo zwykłym strzyżeniem owiec, ponieważ:

1. Technologie przestają być tak zależne od tajemnych wywołań systemowych wymieniających potwierdzenia.
2. Cele większości RIA są jaśniej określone i mniej przeładowane funkcjonalnością, a tym samym szybsze do osiągnięcia i dostarczenia.
3. Cykl kodowania, testowania, obserwowania i refaktoryzacji wydaje się (przynajmniej nam) krótszy i zmierzający szybciej w kierunku ukończonego produktu.

Ponadto jako programista, poprawiając aplikację po stronie serwera, możesz do późna poprawiać model danych czy przekształcać bazę danych w inną. Nie ma to żadnego wpływu na użytkownika końcowego. Natywne oprogramowanie biurowe już nigdy nie będzie przedstawiało się tak atrakcyjnie, a oczekiwania użytkowników zmieniały się już na zawsze.

W końcu oczekiwania programistów również się zmieniły. Ponieważ nie musimy koncentrować się na kwestiach pobocznych, takich jak dystrybuowanie wielu wersji i zarządzanie nimi, możemy się skupić na przydatnej refaktoryzacji czy tworzeniu następcy projektu.

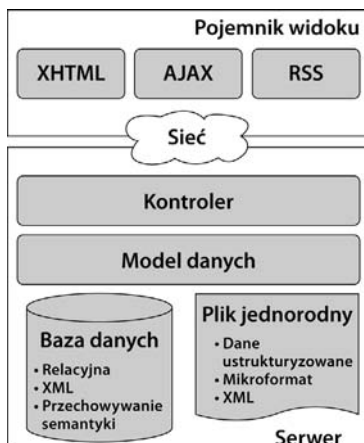
Bogate aplikacje internetowe są zabójcami systemów operacyjnych

Od swych początków oprogramowanie było związane z określonymi systemami operacyjnymi. Pierwotnie oprogramowanie było oferowane niemal ze stratą dla producenta, aby zapewnić lojalność względem sprzętu. W czasach komputerów osobistych, użytkownicy wciąż nie mogli kupić dowolnego komputera, by uruchomić określoną aplikację. Nawet gdy producenci opanowali sztukę sprzedawania oprogramowania jako osobnego produktu, użytkownicy byli zmuszeni do zakupienia odpowiedniego systemu operacyjnego i właściwej architektury PC. Gdyby w taki sposób działały samochody, motoryzacyjna aplikacja — powiedzmy przemieszczenie się z Krakowa do Gdańska wymagałoby zakupu samochodu określonej marki, być może z określonymi oponami, a samochód mógłby poruszać się jedynie po drogach asfaltowych. Byłby to jedyny sposób osiągnięcia celu aplikacji, czyli odbycia podróży.

Podział na warstwy w Web 2.0

Na szczęście, tak ciasne ograniczenia nie dotyczą ani systemu transportu, ani aplikacji Web 2.0. Jak obrazuje to rysunek 1.5, rozwiązania RIA nie są tak ograniczone. Naturalny podział na warstwy sieciowych aplikacji Web 2.0 ma pozytywny wpływ na programistów — wymusza właściwy porządek programowania poprzez rozdzielenie problemów; a także pozytywny wpływ na wzrost i ewolucję samej sieci — znacznie łatwiej jest wstawiać nowe (i jeszcze nieoczekiwane) warstwy i ostatecznie zwiększa skalowalność, co zostało podkreślone w podrozdziale „Wywiady z liderami przemysłu: Jasonem Friedem i Davidem Heinemeierem Hanssonem z 37Signals” na końcu niniejszego rozdziału.

Rysunek 1.5



A więc ponieważ RIA wykorzystują internet jako własną platformę, w efekcie nie ma znaczenia, czy użytkownik korzysta z Windowsa, Linuksa, Mac OS/X czy czegokolwiek innego. Jest to niesamowita wiadomość dla programistów. Założenie projektowe Web 2.0 wymaga od programisty dostarczenia aplikacji przez URL, poprzez XHTML (opartą na standardach wersję „starego dobrego HTML”) do przeglądarki. Do dynamicznego wyświetlania i umożliwienia interaktywności bez konieczności pełnego odświeżania używany jest obiektowy model dokumentu (DOM). Programista osadza JavaScript wewnątrz XHTML; to strona doświadczenia użytkownika.

Kaskadowe arkusze stylów

Wygląd i sposób działania po stronie użytkownika są określane za pomocą kaskadowych arkuszy stylów (przeważnie specyfikacji CSS w XHTML). Logika JavaScript w przeglądarce wykorzystuje wygodną funkcjonalność każdej nowoczesnej przeglądarki jako sposób zdalnego wywoływania metod (nosi to nazwę XMLHttpRequest i zostało szczegółowo omówione w rozdziale 3.). Logika aplikacji i model danych są często napisane w popularnej platformie (omawiamy platformy w językach Python, Ruby i Java), ale platforma pozwala nam jedynie zwiększyć produktywność i oszczędzić pisanie kodu. Pisanie pod platformę nie jest konieczne — na przykład wystarczy PHP.

Pozostawiając biuro za sobą

W świecie dekstopowym, radzenie sobie z określonymi irytującymi sprawami tak zakorzeniło się w naszym życiu, że z trudem je zauważamy lub zakładamy, że są one podstawowym prawem wszechświata. Wiele z tych zdawałoby się niezmiennych praw stanowi nieodłączną część nieprzyjemności pracy z aplikacjami w natywnym systemie operacyjnym: kompulsywne klikanie przycisku *Zapisz* co parę minut, aby zabezpieczyć się na wypadek niespodziewanej awarii aplikacji; obsesyjne porządkowanie hierarchii folderów; nerwowe uruchamianie programu antywirusowego kilka razy dziennie. RIA ucieleśnia skok kulturowy oraz technologiczny dla użytkowników.

W przypadku, gdyby użytkownicy musieli stopniowo rozstawać się z erą oprogramowania biurkowego, została podjęta przynajmniej jedna inicjatywa, by przestawić cały desktop na AJAX: AJAXOS (<http://www.ajax13.com/en/ajaxos/>). Wyróżnikiem AJAXOS-a jest fakt, iż zakłada on istnienie „chmury sieciowej” nawet dla plików znajdujących się na platformie komputerowej, na której AJAXOS jest uruchomiony. Można pomyśleć, że jest to celowe zacieranie granicy między „lokalnym” a „sieciowym”, czynione w wierze, że te opcje powinny pozostać w gestii użytkownika. Wszystko, począwszy od systemu plików, a na odtwarzaczu MP3 skończywszy, działa w AJAXOS w ten sposób.

Czy Java jest zwycięzcą czy przegranym?

Jedno z pytań, jakie zadawano autorom niniejszej książki, brzmi, czy w RIA, które są zmianą zarówno kulturową, jak i technologiczną, Java jest zwycięzcą czy przegranym. Nie znamy odpowiedzi, ale możemy zaryzykować pewne przewidywania. Większość programistów

zgodzi się z opinią, że dostarczanie do przeglądarki aplikacji w Javie jest ślepym zaułkiem i to od dawna. W ramach porównania warto uruchomić edytor tekstu RIA, taki jak Writley (docs.google.com), oraz apletową wersję Thinkfree (thinkfree.com). Choć w długim okresie może oferować podobną funkcjonalność i czas reakcji, i tak wygoda związana z brakiem konieczności radzenia sobie z kwestiami desktopowymi powoduje zwycięstwo RIA. Na ironię losu zakrawa fakt, że Java, która na tyle sposobów stara się być własną platformą, jest niezależna od systemu operacyjnego i rozpoczęła tak naprawdę przenoszenie aplikacji z komputera lokalnego do sieci, przestaje być znaczącą technologią przyszłości.

Zupełnie inną kwestią jest strona serwera. Java jest tu tak zniewalająca, jak w kontekście aplikacji biurkowych. Ponadto Sun Microsystems, firma, która opracowała Javę, może być jednym z wielkich zwycięzców zmiany kulturowej związanej z przejściem na RIA po prostu dlatego, że firma ta jest dostawcą sprzętu serwerowego. Dodatkowe spojrzenie na stosowanie Javy lub języków zwinnych (agile) po stronie serwera można znaleźć w podrozdziale „Wywiady z liderami przemysłu: Jasonem Friedem i Davidem Heinemeierem Hanssonem z 37Signals”.

Na koniec należy zrozumieć, że AJAX niesie ze sobą własne wyzwania: zarządzanie asynchronicznymi operacjami i programowaniem zdarzeniowym, zwłaszcza jeżeli istnieje potrzeba wielu wątków sterowania; opóźnienie sieciowe dla tych żądań asynchronicznych; koordynacja wielu zadań w interfejsie WWW — by wymienić tylko kilka. Więcej informacji na temat tych wyzwań można znaleźć w rozdziale 15. W sumie jednak RIA stanowią przyszłość systemów operacyjnych i „pakietów biurowych”.

Bogate aplikacje internetowe są ukierunkowane na przeglądarkę

Zastosowanie przeglądarek ewoluowało od łączenia informacji, przez skromne aplikacje (w Web 1.0), aż do pełnego hostowania aplikacji Web 2.0. I to zmusza nas do myślenia wykraczającego poza desktop. Podczas gdy poprzednio musieliśmy testować aplikacje jedynie w monokulturze oprogramowania (przeważnie), teraz musimy testować z użyciem wszystkich liczących się przeglądarek — przynajmniej z aktualną i nadchodzącą wersją każdej z nich.

Przeglądarki „zawiadują grą” w ewolucji aplikacji Web 2.0 i możemy znacznie zwiększyć ich oryginalne możliwości. Jak pokazują liczne przykłady kodów z kolejnych rozdziałów, prosta możliwość zdalnego wywoływania procedur za pomocą wbudowanego XMLHttpRequest z JavaScript już sporo zmieniła „w grze”. Zauważmy jak niewiele wymaga od dowolnej współczesnej przeglądarki „podniesienie stawki” poprzez wcielenie odpowiedników pakietów, takich jak Microsoft Office, OpenOffice i Mac iWork. Istnieją już edytory zawartości formatowanej — takie jak FCKEditor (www.fckeditor.com) i Tiny MCE (tinymce.moxiecode.com) — napisane całkowicie w JavaScriptcie i możliwe do bezpośredniego osadzenia w przeglądarce. Z kolei *ideaStax*, aplikacja, którą będziemy tworzyć w niniejszej książce, jest podobna do programów służących do tworzenia prezentacji.

Gdy treść opuści komputer lokalny i będzie istniała w „internetowej chmurze”, dostępna dla użytkownika końcowego poprzez łącze bezprzewodowe czy szerokopasmowe, gra zmieni się na dobre. Telewizory, telefony, urządzenia kieszonkowe i inne jeszcze niewymyślone będą na równej stopie z tradycyjnymi PC i laptopami. Gdy pisaliśmy tę książkę, przeglądarkę można było uruchomić na większości urządzeń dostępnych na rynku. Jeden z systemów operacyjnych, Damn Small Linux (www.damnsmalllinux.com) może być uruchomiony z pen-drive’a podłączanego do portu USB i można w nim uruchomić przeglądarkę dysponującą pełnią możliwości.

Oczywiście wciąż będziemy testować każdą nową aplikację pod względem kompatybilności z przeglądarkami. Prawda jest jednak taka, że kwestie związane z przeglądarkami będą malały, ponieważ narzędzia programistyczne AJAX-a hermetyzują przeglądarki na wysokim poziomie abstrakcji. Co więcej, nawet producenci telefonów komórkowych (w szczególności Nokia) zaczynają wspierać przeglądarki zgodne z AJAX-em. Zwiększona niezależność od platformy pozwala na dostarczenie aplikacji znacznie większej liczbie osób, niż można było przyciągnąć w czasach czegoś, co nazywane było „monokulturą” oprogramowania/sprzętu.

Bogate aplikacje internetowe są ukierunkowane na sieć

Istnieje jeszcze jeden powód, dla którego, jak zostało to wspomniane w poprzednim podrozdziale, RIA zmniejszają znaczenie systemów operacyjnych. Wiele użytkowników końcowych działa w kilku lokalizacjach i nie przestaje pracować tylko dlatego, że wróciło z pracy do domu. Tak samo nie odkłada na bok spraw osobistych po przyjeździe do pracy. W wyniku tego użytkownicy spędzają sporo czasu na próbach zsynchronizowania komputera domowego i znajdującego się w pracy — dotyczy to zarówno samej treści, jak i oprogramowania z tą treścią pracującego. Wartość końcowa tworzona w przestrzeni RIA — czy mówimy tu o kolejnym świetnym edytorze tekstu, czy programie do zarządzania informacjami osobistymi — umożliwia użytkownikowi uzyskanie dostępu do oprogramowania i całej treści, które są zawsze dostępne przez URL. Dzięki ukierunkowaniu na sieć dzielenie się jest również proste. Zastanówmy się nad wszystkimi sztuczkami użytkowników, stosowanymi do przezwyciężenia odległości i faktu pracy na kilku komputerach: kopiowanie plików na pen-drive’y, wysyłanie treści na konta e-mail, korzystanie z niezgrabnych i ospałych aplikacji synchronizujących (na przykład Groove). Wszystkie te czynności wymagają planowania i przewidywania, jaka treść powinna być skopiowana, by była dostępna w wielu miejscach.

Zwolennicy aplikacji z platformy RCP podają argument, że wadą RIA jest fakt, iż w przypadku, gdy u użytkownika wystąpi awaria sieci, jego aplikacja również jest popsuta. Jest to oczywiście prawdą, tak samo, jak prawdą jest, że komputer zainfekowany wirusem traci swą przydatność. Jednakże ostatecznie sieć jest znacznie bardziej odporna i lepiej pielęgnowana niż komputer przeciętnego użytkownika. Weźmy również pod uwagę wszystkie aplikacje ukierunkowane na sieć, których używamy i na których polegamy bez zastanowienia — satelity czy radio, telefonię komórkową czy telewizję. Końcowi użytkownicy aplikacji internetowych są już przyzwyczajeni do aplikacji i urządzeń sieciowych. Ponadto, kiedy weźmiemy

pod uwagę liczbę miast amerykańskich, które utworzyły już lub planują utworzenie miejskich sieci WiFi, widać, że rośnie świadomość, że powszechność dostępu do sieci jest koniecznością dla współczesnego społeczeństwa.

Kwestię kolejnego oczywistego zmartwienia (przynajmniej użytkowników końcowych) związanego z ukierunkowaniem na sieć wyraża często słyszany komentarz: „Martwię się, że wszystkie moje dokumenty nie znajdują się już na moim komputerze. Są gdzieś tam w internecie. Są w sieci i wydaje mi się to naprawdę niebezpieczne”.

Czy to na pewno prawda? W wielu sytuacjach trzymanie własnych rzeczy pod ręką nie jest dobrym pomysłem: użytkownicy raczej nie trzymają oszczędności swojego życia pod materacem ani nie chomikują zdjęć RTG klatki piersiowej gdzieś w szafie na korytarzu. Mimo to komputer osobisty często daje użytkownikom fałszywe poczucie bezpieczeństwa, mimo że każdego roku kradziona jest spora liczba laptopów.

Gdy użytkownicy podnoszą ten argument, warto wspomnieć, że namacalność może być trudno dostępna. Użytkownik może mieć treść zabezpieczoną na laptopie, który (niestety) ma kłopoty z zasilaniem, popsuty twardy dysk, popsutą płytę główną czy też jest zainfekowany wirusem. Treść należąca do użytkownika jest więc namacalna, a nawet znajduje się pod ręką, ale nie jest dostępna bez żmudnego „strzyżenia jaka”, wymaganego do wskrzeszenia komputera.

Wystarczy jedynie zapytać użytkowników, jak wielu z nich straciło ważne dane na twardych dyskach, czy też dało pełne prawa dostępu indeksatorom, takim jak Google Desktop, czy uruchamiało programy wspomagające tworzenie rozliczenia podatkowego, czy też pozwoliło ukrytemu spybotowi, takiemu jak rootkit BMG/Sony znaleźć drogę do ich komputera. Dla większości użytkowników lepiej będzie, jeżeli pozwolą na przechowywanie danych dobrze zabezpieczonym centrom danych. Jednak droga do przekonania użytkowników jest długa i musi upłynąć jeszcze sporo czasu. Najlepszą radą może być tutaj jedynie edukacja.

Ostatnia uwaga na temat ukierunkowania na sieć skierowana jest do programistów. Nie należy zakładać, że proste przeniesienie aplikacji na architekturę RIA spowoduje, że nasza praca będzie w pełni zabezpieczona i będziemy tworzyć ogrody rozkoszy, zamiast kolejnych ogrodów otoczonych murami. W szczególności w Stanach Zjednoczonych użytkownicy mogą zastępować jeden rodzaj ogrodu otoczonego murem innym.

W okresie pisania niniejszej książki wiele uwagi poświęca się sprawie dwuwarstwowego (ang. *two-tiered*) dostępu do internetu, gdy dostawcy usług mogliby naliczać dodatkową opłatę za usługi lepszej jakości i większą przepustowość. Oczywiście ze swej natury internet nie jest odporny na tworzenie ogrodzonych ogrodów. Jeżeli dostawca usług internetowych będzie w stanie wymóc na użytkownikach (czy to poprzez regulacje rządowe, korzyści ekonomiczne, czy też masową hipnozę) wyrażenie zgody na umożliwienie mu pośredniczenia w przeglądaniu, będzie mógł kształtować treści odbierane przez użytkowników, promować cele polityczne czy sprzedawać użytkownikom interesujące (dla dostawcy) rzeczy.

Jako przykład możemy podać Chiny, gdzie rząd skrepiował wyszukiwanie i dyskusje polityczne poprzez regulacje ograniczające działania dostawców internetu. Natomiast Digital Millenium Copyright Act w Stanach Zjednoczonych stanowi przykład obmurowanego ogrodu

służącego celom politycznym lub komercyjnym. A więc najlepsze realizacje globalnej społeczności programistów mogą zostać niedokończone na skutek okoliczności poza ich kontrolą. Najlepszą radą może być tutaj wpływanie na zmiany zachodzące w kraju.

Rządy nie są jedynymi podmiotami próbującymi utworzyć obmurowany ogród wokół doświadczania internetu przez użytkowników. America Online jest najwcześniejszym przykładem dostawcy usług internetowych, umieszczającym użytkowników w takim ogrodzie. Pozostaje też jednym z nielicznych, którzy wciąż tak robią.

Bogate aplikacje internetowe zmieniają sposób myślenia

Możemy rozpatrywać to stwierdzenie na dwa sposoby. Najpierw przyjrzyjmy się, jak zmienia się nasz sposób myślenia jako twórców oprogramowania i architektów systemów, a następnie jako użytkowników końcowych.

Obserwacje zmienionego myślenia programisty

Większość programistów, w szczególności programujących już stosunkowo długo, doświadczyła już sytuacji, w której opanowanie nowego języka programowania powodowało gwałtowną zmianę postrzegania świata. Gdy programiści odkryli język C, przekonali się, że mogą manipulować wewnętrznym stanem systemu komputera w sposób znacznie mniej prymitywny niż poprzednio.

Gdy nastał Smalltalk, rozumni programiści wykryli, że mogą myśleć w sposób zorientowany na komunikat — aplikacje były „rozmowami”, gdzie komunikaty były tworzone i zużywane. Gdy Visual Basic zaczął zaspokajać potrzeby ery biurkowej, wniósł dokument, jako wizualny pojemnik dla aplikacji, a wielu programistów uznało, że myślenie za pomocą metafory dokumentu jest przydatne. Gdy rozpowszechnił się C++, a w szczególności Java, cała generacja architektów oprogramowania oraz programistów zaczęła myśleć w sposób zorientowany obiektowo (taki, w którym dokumenty były jednym z modeli architektury aplikacji).

Ostatnio niewielka, choć znacząca liczba programistów zaczęła pisać poważne aplikacje korzystając z dynamicznych języków, takich jak Python i Ruby, co wprowadziło wielu niezależnych i kilku korporacyjnych programistów w świat zwinnego programowania. Na każdym etapie odysei twórcy oprogramowania zyskiwali nowe perspektywy, które pomagały im projektować i pisać kod inaczej i również pisać inny kod.

Teraz, nagle, napędzane dynamicznymi zachowaniami w przeglądarce i językami zwinnymi po stronie serwera, koło się obróciło ponownie, dając programistom kolejną perspektywę. Krajobraz, który, jak się wydaje, widzą programiści, którzy zdążyli się już „przestać” na nowy sposób myślenia, stanowi robiącą wrażenie listę atrybutów projektowania

współczesnych aplikacji. W tej części przedstawimy kilka obserwacji sposobu, w jaki nasza perspektywa została zmieniona. Ile z nich stanowi przykład Twoich doświadczeń programistycznych z Web 2.0?

Pewne obserwacje z naszych zmienionych umysłów

Pierwsze spostrzeżenie, jakim chcielibyśmy się podzielić, jest takie, że bieżąca zmiana perspektywy, jakiej możesz teraz doświadczać, nie jest spowodowana wprowadzeniem nowego języka, ale całej platformy. Python wprowadził świeżą filozofię *tworzenia ogólnych aplikacji*. Ci, którzy wiedzą o „Zen Pythona”, znają to założenie, a ci, którzy nie, niech spróbują uruchomić interpreter Pythona i wpisać `import this` w wierszu poleceń, aby przywołać interesującą „niespodziankę” przedstawioną w lewej części rysunku 1.6. W opozycji do „zen” konkretnych języków, w prawej części tego rysunku sugerujemy, że RIA wprowadzają nową filozofię tego, czym aplikacje są. Te spostrzeżenia są omówione w kolejnych podrozdziałach.

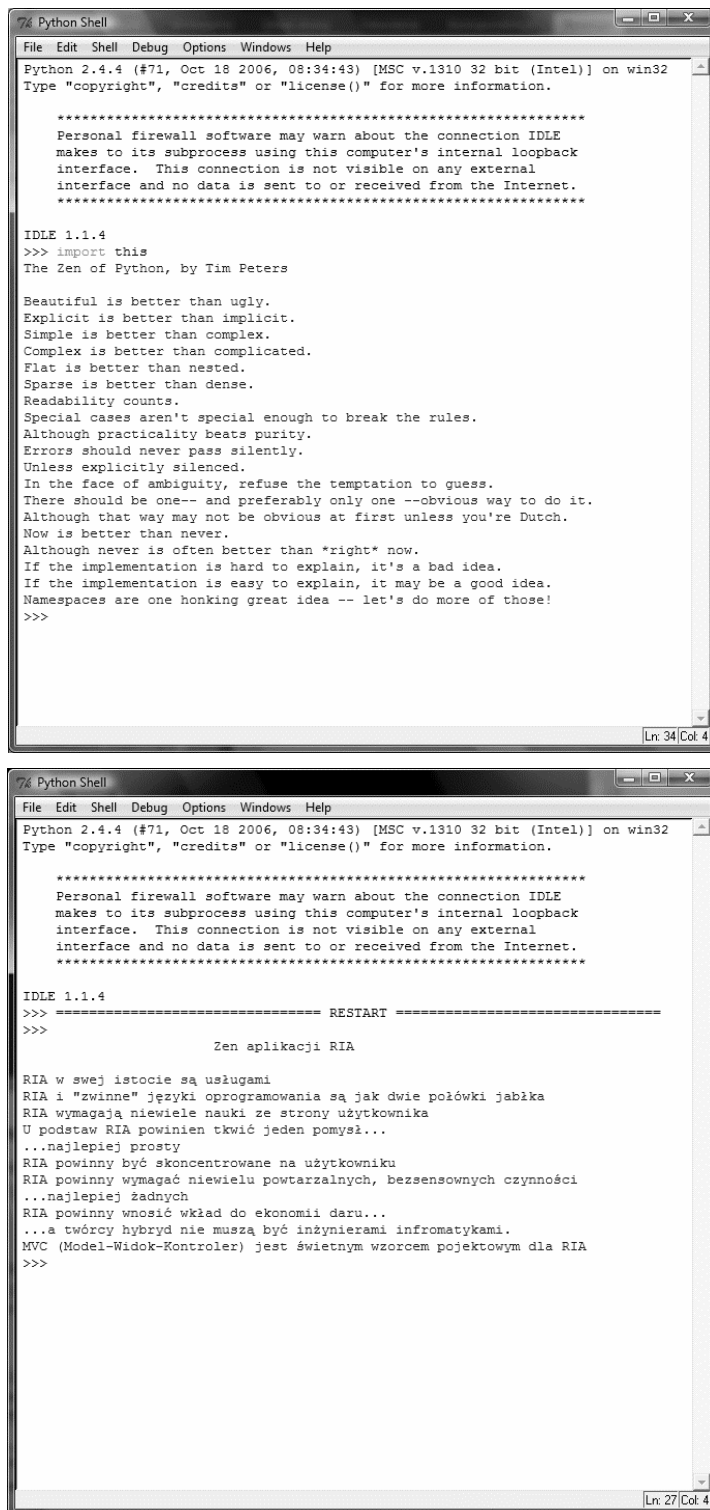
Bogate aplikacje internetowe są zgodne z modelem „oprogramowanie jako usługa”

Zgodnie z naszym nieoficjalnym „Zen RIA” aplikacje są po pierwsze i przede wszystkim usługami. Oznacza to, że są przywoływane przez nawigację URL, a nie przez instalację, przygotowania i konfigurację na określonej platformie. Jeżeli RIA nie jest „po prostu dostępna”, oznacza to, że zawiedliśmy jako programiści, a potencjalni użytkownicy odrzucają naszą aplikację. W tym przypadku model „oprogramowanie jako usługa” implikuje otwarte API, co z kolei wiąże się z możliwością remiksowania i tworzenia aplikacji hybrydowych. Mashupy i remiksy są możliwe, ponieważ wielu twórców RIA ujawniło API po to, aby umożliwić tworzenie remiksów. Prawdę mówiąc, raczej nie mogli temu zapobiec, ponieważ wywołania metod serwera są ujawnione w JavaScriptcie. Wielu dostawców prosi programistów, by „wnioskowali” oni o klucz API, który będzie przesyłany wraz z wywołaniem do serwera, by identyfikować aplikację wywołującą. Można to potraktować jako sposób, dzięki któremu twórca pierwotnej aplikacji zbiera informacje na temat wykorzystania API przez poszczególne remiksy i mashupy.

Bogate aplikacje internetowe są ukierunkowane na użytkownika

RIA są ukierunkowane na użytkownika na dwa sposoby, co należy wziąć pod uwagę podczas projektowania aplikacji. Po pierwsze, są one ukierunkowane na użytkownika, ponieważ działają na poziomie wyższym od systemu operacyjnego. Jest to ważne, ponieważ tradycyjnie

Rysunek 1.6



wiele produktów technologicznych rozpoczyna od organizacji wewnętrznej i owija doświadczenie użytkownika wokół technologii. Na przykład większość systemów operacyjnych posiada sporo rusztowań obracających się wokół manipulowania systemem plików (istnieje tylko garstka, gdzie jest inaczej, na przykład Magic Cap firmy General Magic). Większość technologów rozpoczyna od interesującej architektury i od tego punktu tworzy coś, z czym ewentualnie styka się użytkownik.

Dobrym przykładem tej złej praktyki może być odtwarzacz MP3 obsługiwany z wiersza poleceń. Co charakterystyczne, nikt nie przemyślał interfejsu lub ponieważ został on utworzony na bazie technologii, jest tak głęboko zakorzeniony w wewnętrznej pracy systemu, że jedyną osobą, która naprawdę wie, jak z niego korzystać, jest jego twórca.

Dla odmiany jedynym punktem styku użytkownika z RIA jest doświadczenie użytkownika, więc musi być ona zaprojektowana z uwzględnieniem tego faktu. Zauważmy na przykład, że choć Writely tworzy dokumenty, nie ma wyraźniej potrzeby zapisywania dokumentu. W idealnej sytuacji użytkownik powinien mieć po prostu możliwość przechodzenia z jednego kontekstu funkcjonalności do kolejnego w sposób zupełnie niemodalny.

Opinię na temat rozpoczynania od doświadczenia użytkownika i przechodzenia w głąb, w kierunku wewnętrznych mechanizmów, można znaleźć w podrozdziale „Wywiady z liderami przemysłu: Jasonem Friedem i Davidem Heinemeierem Hanssonem z 37Signals”, w dalszej części niniejszego rozdziału. Niemodalne działanie stanowi pewne wyjaśnienie sukcesu Google w przekształceniu się w narzędzie niezbędne dla użytkowników. Internet jest znacznie większy niż czyjkolwiek system plików. Z tego też względu wyszukiwanie i subiektywne ocenianie stało się ewolucyjnym zamiennikiem nawigowania po folderach oraz zarządzania folderami.

Podczas gdy witryny Web 1.0 starały się wyglądać i działać jak czasopisma, wszystkie witryny Web 2.0 wydają się odmiennie (lecz podobnie) wyglądać i działać — na porządku dziennym są duże, widoczne przyciski i stałe składniki. Jest tak dlatego, że zdecydowanie *nie* są to książki przenoszone na ekran. Programiści nie są wydawcami w HTML, umieszczającymi polityczne tyrady lub kolejną „świetną amerykańską powieść”, by ludzie sobie poczytali. Zamieszczają oni bowiem aplikacje w pełnym tego słowa znaczeniu.

Z drugiej strony wydaje się, że użytkownicy końcowi przyzwyczaili się do samodzielnego komponowania i integrowania treści. Doskonale wiedzą, że przeglądarki oferują podział tematyczny trosk/zainteresowań, którymi można się zająć lub które można odłożyć dzięki kartom. Zaawansowany użytkownik Web 2.0 nie przechwala się doskonałą znajomością Excela. Zamiast tego ma on uruchomione trzy lub cztery egzemplarze przeglądarki, w każdej po kilka kart i przechwala się umiejętnością zarządzania wynikami z RSS.

Bogate aplikacje internetowe są idealne do współpracy

RIA są również ukierunkowane na użytkownika pod tym względem, że społeczności mogą brać udział w rozwijaniu wytworów innych osób. Niezależnie od tego, czy chodzi o komentowanie bloga, ocenianie wiadomości, dodawanie metadanych do zdjęcia, czy tworzenie

lokalnych miejsc wymiany informacji, ludzie tworzą społeczności wokół aplikacji internetowych, a wiele ważnych aplikacji Web 2.0 jest w jakiś sposób ukierunkowanych na słuzenie społecznościom.

Choć firma, w której pracują autorzy, BBN Technologies, jest dużym graczem w dziedzinie prac nad formalnymi aspektami semantyki w internecie, podczas naszej pracy nad OWL, językiem opisu znaczeń w internecie, byliśmy pod niesamowitym wrażeniem „folksonomii”, prostszej kategoryzacji treści, zdającej się pracować w sposób niemal doskonały, mimo swej prostoty i subiektywnej natury. Oto definicja z Wikipedii:

Folksonomia to neologizm oznaczający praktykę kategoryzacji treści z wykorzystaniem dowolnie dobranych słów kluczowych. W znaczeniu potocznym termin ten odnosi się do grupy ludzi współpracujących spontanicznie w celu uporządkowania informacji w kategoriach.

W przeciwieństwie do formalnych metod klasyfikacji z fenomenem tym spotkać się można raczej w niehierarchizowanych wspólnotach, takich jak publicznie dostępne strony WWW.

Ponieważ klasyfikatorami informacji są zwykle osoby ją wykorzystujące, zwolennicy folksonomii uważają, że daje ona rezultaty lepiej oddające model informacji we wspólnocie.

Folksonomia jest obecnie rozumiana w pewnym stopniu jako tagowanie, oznaczanie.

Folksonomia sprawdza się najlepiej, kiedy duża liczba osób opisuje tą samą porcję informacji. Przykładowo w serwisie Del.icio.us wiele osób dodało do ulubionych Wikipedię, każdy z innym zestawem słów kluczowych (tagów) na jej opisanie. Pośród wielu używanych słów kluczowych najpopularniejsze to reference, wiki oraz encyclopedia.

Współpraca poprzez internet rodzi nowy rodzaj współuczestniczenia, nowy rodzaj kontaktu z informacją, gdzie potencjalnie obejmująca swym zasięgiem cały świat wspólnota użytkowników ma możliwość modyfikowania własnego doświadczenia online oraz wnoszenia wkładu do kształtowania doświadczenia innych. W aplikacji rozwijanej w niniejszej książce zapewniamy platformę, dzięki której może ze sobą współpracować wiele osób w tworzeniu treści z blokadą treści. Co ciekawe, jest to coś, co w ogóle nie jest dostępne w nawet najbardziej znanych aplikacjach biurkowych (na przykład współpraca w czasie rzeczywistym nad edycją tekstu nigdy nie była możliwa poza jednym lub dwoma rozwiązaniami, na przykład w SubEthaEdit na Macintoshach; dostępne dla wielu użytkowników tworzenie kodu w czasie rzeczywistym również nie wykroczyło poza IDE Javy, Netbeans).

Bogate aplikacje internetowe: krótki i zwinny rozwój

Gdy piszemy tę książkę, tworzenie się nowej generacji aplikacji internetowych wciąż nie jest zakończonym procesem, więc trudno opowiedzieć się za niektórymi opiniami wyrażonymi w tym podrozdziale. Jednakże jako porównania możemy użyć badań przeprowadzanych

nad językami zwinnymi (na przykład projektami tworzonymi w Ruby i Pythonie), przede wszystkim dlatego, że najważniejsze platformy rozwoju Web 2.0 implementują aplikacje w tych językach.

Badania prowadzone nad językami zwinnymi (na przykład Lutz Prechelt, „And empirical comparison of seven programming languages”, *IEEE Computer*, 33(10):23-29, October 2000) wykazują, że w językach zwinnych programiści piszą mniej kodu i szybciej uzyskują rezultaty. Wydaje się, że możliwości pojedynczego programisty czy małych zespołów zostały wzmocnione. W przebadanych wielu przypadkach okazało się, że zespoły są mniejsze i bardziej zwinne z różnych powodów. Wydaje się, że dzięki wykorzystaniu platform omawianych w niniejszej książce, zespoły mogą być mniejsze i szybciej osiągać cele: Writely, nabyte przez Google w 2006 roku, było tworzone przez trzech inżynierów (trzech współzałożycieli). 37Signals, wynalazcy i użytkownicy platformy Ruby on Rails, w 2006 liczyli cztery czy pięć osób.

Bogate aplikacje internetowe są skazane na ewolucję

W okresie pisania niniejszej książki opracowywane są nowe wersje wszystkich liczących się przeglądarek. Rewolucja Web 2.0 wydaje się zmierzać ku punktowi krytycznemu pod względem liczby aplikacji konwertowanych z desktopu, zarówno pod względem bazy użytkowników, jak i liczby platform. Braki współczesnych przeglądarek i wymagania stawiane nowym staną się jasne. Oznacza to po prostu, że nasza wiedza i umiejętności również będą musiały ewoluować. Niezależnie od tego, jakie zmiany czekają przeglądarki, z pewnością będą ewoluowały one w kierunku jeszcze bardziej „ogólnoplikacyjnych” platform niż dotychczas.

Jak zmienić sposób tworzenia aplikacji?

Można się zastanawiać, czy fakt że, jak to sugerowaliśmy, otaczający nas świat aplikacji jest w środku dramatycznych zmian, oznacza, iż musimy zmienić nasze obecne praktyki programistyczne, by dopasować się do tego nowego świata. Jakie są obecnie najlepsze praktyki przy tworzeniu aplikacji nowej generacji? Coś się dzieje: aplikacje są w wersji beta, znacznie dłużej niż wiele ostatecznych wersji tradycyjnego oprogramowania. Ekstremalne programowanie i nieustanna integracja, jeszcze niedawno tak awangardowe, wydają się teraz nieco wyświechtane. W naszym przeglądzie projektów Web 2.0, które odniosły sukces, wyraźnie zaznacza się różnica między tradycyjnym stylem rozwoju, z formalnymi cyklami wydawniczymi i koncentracją na technologii, a nowym wzorcem aplikacji nieustannie ewoluujących i koncentracją na kliencie. Poniższa lista opisuje niektóre oznaki tej zmiany:

- **Wszystkie aplikacje są wersjami beta.** Nie wiadomo kiedy ten trend się rozpoczął, ale jak w wielu innych obszarach Google mógł wskazać drogę. Z punktu widzenia końcowego użytkownika nie ma to znaczenia. Jeżeli użyteczność aplikacji jest zadowalająca, użytkownik nie dba o etykiety. Beta wprowadzają pozytywne oczekiwania użytkowników i umożliwiają programistom podejmowanie ryzyka, a czasami nawet ponoszenie porażki. Aplikacje z cyklami wydawniczymi wiążą się również z końcem wspierania poprzedniego wydania aplikacji i nie tak znowu subtelnym, ukrytym komunikatem do użytkownika, że twórca pragnie zakończyć związek ze społecznością użytkowników.
- **Wersje i błędy.** To, że „wszystko jest wersją beta” nie oznacza jednak, że możemy sobie pozwalać na partackie implementacje. Strategia z wersjami beta pozwala nam dodawać funkcjonalności do aplikacji, gdy te będą już gotowe, a nie wiązanie nowych funkcjonalności z mocno sztucznym systemem stopniowego wypuszczania wersji. Jak zawsze, ważne jest dobre oznaczanie kolejnych wersji (CVS lub SVN), budowania (Ant, Cruise Control) i śledzenie błędów (Bugzilla), ale biorąc pod uwagę to, co zostało napisane powyżej, należy również określać sobie wewnętrzne etapy, po których nastąpi publikacja. Jak rekin należy zawsze przeć do przodu. Nie jest ważne, jakie błędy zostały zgłoszone przez użytkowników w zeszłym tygodniu czy miesiącu. Jeżeli błąd nie może być odtworzony w kodzie, możesz go spokojnie pominąć. Jedyny problem, którego nie wolno nam ignorować, polega na tym, że podczas gdy *możemy* wycofać kod, nie *możemy* wycofać treści wprowadzonych przez użytkownika. Na koniec, niezależnie od tego czy zdecydujemy się samodzielnie prowadzić serwer, czy też skorzystamy z jednego z rozlicznych serwerów do wynajęcia, nie możemy pozwolić, by błędy po stronie serwera uniemożliwiły działanie witryny. Należy więc myśleć o tym aspekcie, niezwiązanym z funkcjonalnością aplikacji, równie mocno, jak o możliwościach aplikacji. Chcemy, aby użytkownicy traktowali naszą witrynę i aplikacje jak narzędzie — zawsze włączone, zawsze działające, tak jak wyłącznik światła. (Pamiętasz fragment o niezauważalności?).
- **Rozwój napędzany testowaniem** (ang. *test-driven development* — *TDD*). Rozwój napędzany testowaniem staje się szczególnie istotny z powodu rozproszonych zespołów programistów, niewielkiego zatrudnienia i ograniczonych budżetów kontroli jakości.
- **RIA jest czynnym laboratorium.** Liczba unikalnych użytkowników, jaką przyciągają witryny RIA, często szokuje początkujących twórców lub przedsiębiorców. RIA mogą rozrosnąć się w tempie wykładniczym od zera do setek tysięcy użytkowników. Nawet niewielka próbka użytkowników może być tak naprawdę dużą liczbą. Nasza aplikacja jest naszym wytworem. Możemy bez przeszkód zmieniać funkcjonalność i rozmieszczenie elementów sterujących i bezpośrednio poznawać reakcje użytkowników. Wbudowywanie liczników kliknięć w celu poznania, jak użytkownicy poruszają się po aplikacji, z których funkcji korzystają, które rozumieją, może być osiągnięte za ułamek kosztu tradycyjnych testów użyteczności.

- **Nie pisz jednej aplikacji, pisz dwie.** Jak zostanie to omówione w kolejnym podrozdziale, istnieje wiele sposobów zarabiania pieniędzy, ale zrozumienie, że społeczność gromadząca się wokół aplikacji jest jednym z najpewniejszych z nich. Oczywiście, przedsiębiorstwa próbowały kategoryzować osoby kupujące ich aplikacje, by zapewnić ponowny zakup. Wydajność ich nachalnych badań rynkowych — takich jak ankiet związanych z rejestracją produktu — jest dyskusyjna. Nawet w aplikacjach Web 1.0 śledzenie ruchu i inne sposoby „szatkowania” populacji użytkowników były mało wydajne. Dla odmiany rozważmy „połączoną kulturę” Web 2.0. Użytkownicy sami określają swoje sieci społeczne i zapraszają nowych użytkowników do dzielenia się i innych sposobów dołączenia do świata aplikacji. Flickr i YouTube są przykładami aplikacji, w których społeczność i aplikacje stały się nierozróżnialne. Rodzaj metadanych użytkowników, który będzie nam przydatny w tworzeniu wartości wokół aplikacji, będzie oczywiście różnił się między poszczególnymi aplikacjami i społecznościami użytkowników. W tym przypadku musimy się wykazać taką samą pomysłowością, jak w przypadku samej aplikacji.

Jak mogę zarabiać na RIA?

Jeżeli jesteś korporacyjnym twórcą oprogramowania, nie jest to może dla Ciebie zbyt paląca kwestia. Jednak dla innych osób odpowiedź na to pytanie wykracza poza zwykłą ciekawość. Na pierwszy rzut oka można pomyśleć, że własność intelektualna jest silnie wyeksponowana dla naszej podstępnej konkurencji, która może ją poznać i klonować. W końcu to, co widzi użytkownik (porcja z „widokiem” aplikacji), łącznie z kodem HTML, wpisanym arkuszy stylów i wplecionego JavaScriptu, jest dostępny w postaci zwykłego tekstu w każdej przeglądarce poprzez funkcję „Pokaż źródło”.

Oczywiście bardzo trudno jest naruszyć fundamentalny podział podstawowego schematu łączącego aplikację (wzorca Model – Widok – Kontroler) w RIA. Należy jednak pamiętać, że prawdziwa intelektualna zawartość aplikacji internetowej jest chroniona na serwerze. Jak dużą część interfejsu usługi udostępnimy, zależy jedynie od nas samych. Jak przedstawimy w kolejnym rozdziale, kilku dostawców o silnej marce ujawnia całkiem spore części interfejsów wywołań usługi poprzez zewnętrzne API. Pokażemy, jak można wykorzystać ujawnione API do tworzenia aplikacji będących hybrydami złożonymi z części innych aplikacji.

Rozważając kwestie utrzymania się na rynku, należy wziąć pod uwagę fakt, że modele ekonomiczne szybko się zmieniają. Po pierwsze, koszty produkcji są naprawdę niskie. Jednym z założeń niniejszej książki była prezentacja jedynie tych platform, które są dostępne i wspierają otwarty kod źródłowy. Po drugie, koszty serwera są również bardzo niskie. W Stanach Zjednoczonych koszty usług internetowych bezlitośnie spadają. Co więcej, gdy zaczniesz rozumieć, że w przypadku RIA aplikacja staje się usługą, znacząca (niefunkcjonalna) część oferty — przechowywanie, bezpieczeństwo i nieustanna dostępność — staje się wartością dodaną bez lub z niskimi kosztami.

Według Michaela Robertsona, jednego z twórców ajaxWrite (jednego z edytorów tekstów RIA), dziesiątki tysięcy użytkowników pracowało z dokumentami. Wydaje się jasne, że im więcej możliwości manipulowania treścią przeniesie się do „chmury”, tym szybciej policzone

będą dni pakietów „Office” po 400 dolarów. Co więcej wydaje się jasne, że jako rezultat ekonomii „daru” czy „dzielenia się”, która wydaje się być tworzącą się cechą kultury Web 2.0, wiele treści nowych aplikacji pochodzi od końcowych użytkowników. Wartością serwisów takich, jak Flickr czy YouTube, nie jest fakt, iż użytkownicy mogą umieścić swoje zdjęcia czy filmy w internecie, ale że mogą oni opublikować swoją treść, by podzielić się nią z potencjalnie milionami odbiorców. Twórcy serwisów, takich jak Flickr i Writely, po prostu dostarczyli platformę, umieszczania, przechowywania i oznaczania treści.

Choć wszystkie rodzaje modeli istnieją, by tworzyć wartość w zamian za poniesione nakłady, oto obserwacja, którą pragniemy się podzielić: Nie należy używać Microsoftu jako punktu odniesienia. Sfatygowane modele z ery biurowej mogą być już na prostej drodze na śmietnik historii. Co więcej, twórz różne rzeczy — rzeczy wykorzystujące nieodłączną przewagę faktu, że tworzysz aplikację sieciową, dostarczającą platformę dla wkładu i współpracy wielu użytkowników. Możemy też chcieć, by aplikacja była „niezauważalna”, umożliwiając użytkownikowi tworzenie treści, lecz tworząc wokół aplikacji otoczkę niezbędności.

Jak wielka wartość drzemie w fakcie stania się tak niezbędnym, że aż pewnym? Jak wspomniano wcześniej w niniejszym rozdziale, aplikacją, którą większość użytkowników przyjmuje za pewnik i uważa za istotną część tkanki internetu, jest wyszukiwarka Google. Kapitalizacja Google w kwietniu 2006 wynosiła 128 miliardów dolarów.

Podejście Google polega na tym, że pomagając użytkownikom w wyszukaniu interesujących ich treści, pomagają również reklamodawcom wyszukać faktycznie zainteresowanych reklamą, zakładając, że użytkownik *może* poszukiwać dóbr i usług. Najczęściej nie poszukuje, ale Google nie musi mieć w większości racji. W 2005 roku reklamodawcy wydali w Stanach Zjednoczonych 9 miliardów dolarów na reklamy radiowe i telewizyjne, gdzie niemal wszystkie komunikaty nie miały znaczenia dla niemal wszystkich odbiorców. Funkcja AdSense Google przynajmniej jest w stanie w choć odrobinę inteligentny sposób oszacować zainteresowania klienta. Być może nawet reklamodawcy zaczynają to rozumieć — w roku 2005 wydatki na reklamę internetową zwiększyły się o 28 procent.

Wniosek jest taki, że gdy utworzymy docelową społeczność wokół aplikacji, mogą zdarzyć się ważne i doniosłe rzeczy. Rozważmy na przykład potencjał aplikacji utworzonej jako witryna dotycząca siatkówki — zawierająca kalendarz rozgrywek ligowych, wyniki meczów, statystyki drużyn i tak dalej. Taka witryna, nawet mając stosunkowo niewielką społeczność, byłaby niesamowicie wartościowa dla producentów sprzętu. Inne społeczności internetowe (na przykład del.icio.us, ClipMarks) tworzą źródła danych pojawiające się tylko w internecie. Wartość tych źródeł danych jest jeszcze niedoszacowana, ale należy zdawać sobie sprawę, że ta wartość istnieje.

Bogate aplikacje internetowe są uwodzicielskie

Dla użytkowników końcowych RIA są zdecydowanie uwodzicielskie, biorąc pod uwagę możliwość tworzenia aplikacji hybrydowych z tak wielu aplikacji. Co więcej, jak zostało to wspomniane wcześniej, są one uwodzicielsko łatwe w obsłudze. Tutaj jednak mamy na myśli, że są one uwodzicielskie dla nas, programistów. Przejdźcie od koncepcji do działającej aplikacji wydaje się być znacznie łatwiejsze. Kursy online TurboGears i Ruby on Rails

pokazują programistom, jak napisać i uruchomić lepszą aplikację Web 2.0 niż *Witaj świecie*, a to może wystarczyć, by pokazać początkującemu ogólną strukturę tych dobrych platform. Prawdę mówiąc, znacznie więcej wysiłku wymagają niebanalne prace, ale początkowe „uwiedzenie” dostarcza prawdziwego „programistycznego kopa”.

Ujmujący instruktaż Ruby on Rails „Creating a weblog in 15 minutes” (www.rubyonrails.org/screencasts); *ekwiwalent Turbo Gears „The 20-Minute Wiki”* (www.turbogears.org/docs/wiki20).

Ponieważ tworzenie RIA jest najczęściej związane ze zwinnymi językami, gdzie często niewielka ilość kodu wnosi większe korzyści, osiąganie celów zachodzi szybciej. Ci z nas, którzy używają zwinnych języków, wiedzą dlaczego tak jest. Programiści języków zwinnych zawsze byli nieco podejrzliwi względem wodospadowej metody WIELKIEGO PROJEKTU. Wydaje się, że to techniczne podejście zawsze produkuje więcej UML niż kodu. Tym programistom zawsze wydawało się, że mając w pamięci dobrze określony opis aplikacji oraz uznane już podejście ekstremalnego programowania — „zaprojektuj mały aspekt, napisz trochę kodu, przetestuj trochę, refaktoryzuj i powtórz” — pozwala osiągnąć lepsze i szybsze rezultaty niż nieustanne męki z rozszyfrowywaniem obrazu wielkiego systemu.

W naszym doświadczeniu, projekt Web 2.0 wydaje się zachowywać właściwe proporcje między robieniem planów a modyfikowaniem aplikacji, ponieważ doświadczenie moderuje zarówno cele, jak i metody. Co więcej platformy, które omawiamy w niniejszej książce (Ruby on Rails, TurboGears, Django), jak najbardziej umożliwiają robienie po prostu tego — powiększanego projektowania i dostarczania. Nie znaczy to, że robienie jakichkolwiek planów jest złe. Praktyki zwykle wykorzystywane przez programistów — rozdzielanie zagadnień, dekompozycja problemu na obiekty i tak dalej — wciąż mają zastosowanie i będziemy przestrzegali tych zasad w tej książce (co, jak mamy nadzieję, nam się uda).

Nawet jeżeli wykorzystane zostaną do tworzenia aplikacji powiązanych z systemem operacyjnym, standardy RIA mogą być łatwiejsze do napisania niż API aplikacji biurkowych. Powoli Macintosh Dashboard i Yahoo Widgets, prawnuki dawnych aplikacji rezydentnych (takich, jak szacowny SideKick w DOSie), zaczynają być zauważane przez programistów, a jednym z powodów, dla których tak się dzieje, jest fakt, że po opanowaniu umiejętności i przestawieniu się na sposób myślenia RIA, trudno powrócić do dawnych (złych) dni nudnego kodu. Biurkowe RIA omówione zostały na początku rozdziału 17.

Wywiady z liderami przemysłu: Jasonem Friedem i Davidem Heinemeierem Hanssonem z 37Signals

W tym rozdziale znalazło się szereg przypuszczeń na temat przyszłego kształtu internetu oraz potencjału wykraczającego poza jego pierwsze wcielenie. Można się z nimi zgadzać lub nie. Wiele z naszych wniosków wysnuliśmy po uważnym wysłuchaniu liderów przemysłu i wizjonerów. Od czasu do czasu w tej książce zamieszczamy opinie autorzytetów. Przeprowadziliśmy wywiad z Jasonem Friedem i Davidem Heinemeierem Hanssonem z 37Signals (37Signals.com), liderami w tworzeniu wartości w internecie nowej generacji.

O byciu wczesnym innowatorem Web 2.0

Pytanie: *Zaczynaliście jako firma projektująca strony WWW. Teraz dostarczacie internet jako platformę aplikacji. Kiedy zdaliście sobie sprawę, że w internecie zachodzą zmiany — od przeglądarki informacji połączonych odnośnikami do platformy aplikacji, której rozwój obserwujemy obecnie? Czy w ogóle był ten moment „Aha!”, czy był to naturalny postęp od projektowania stron do projektowania aplikacji?*

Współzałożyciel 37Signals, Jason Fried: Myślę, że tak naprawdę nie było żadnego momentu „Aha!”. Potrzebowaliśmy stworzyć produkt dla nas samych. Początkowo Basecamp był utworzony dla nas. Zdecydowaliśmy się umieścić go w internecie, ponieważ wiedzieliśmy, że musiał być scentralizowany. Nie za bardzo wiedzieliśmy, jak pisać oprogramowanie biurkowe, a poza tym takie oprogramowanie niezbyt dobrze radzi sobie ze scentralizowanymi informacjami. Czuliśmy, że internet jest tym, co najlepiej rozumiemy. Nie było tak, że mieliśmy jakąś długoterminową wizję, w której planowaliśmy, że wszystko przeniesie się do internetu. Było to coś, co stworzyliśmy, bo pojawiła się taka potrzeba, a wiedzieliśmy, jak pisać oprogramowanie oparte na internecie. Mieliśmy szczęście, robiąc najłatwiejsze dla nas rzeczy, a zdarzyło się, że były to rzeczy związane z internetem.

Pytanie: *Czy pozostałe projekty powstały z Basecampa lub czy są to te projekty, które uważaliście za przydatne w waszej firmie, a następnie zaczęliście udostępniać je jako zewnętrzne usługi?*

Jason: Wszystko, co utworzyliśmy, tworzyliśmy najpierw dla siebie. Zawsze zaspokajaliśmy nasze potrzeby i zdaliśmy sobie sprawę, że, jakby to powiedzieć, „nie jesteśmy jedynymi płatkami śniegu”. Skoro potrzebujemy narzędzia, to samo dotyczy setek tysięcy firm podobnych do naszej. Nie ma w nas nic na tyle specjalnego, że dane narzędzie mogłoby służyć tylko naszej firmie. Więc powiedzieliśmy sobie, poświęćmy czas na tworzenie tego, a potem możemy zmienić to w produkt, ponieważ inni ludzie również tego potrzebują, więc możemy na tym zarobić.

O staraniach w utworzeniu modelu subskrypcji oprogramowania

Pytanie: *Wydaje się, że znaleźliście złoty środek, gdzie użytkownicy nie wahają się subskrybować dostępu do oprogramowania, podczas gdy jeden z największych producentów oprogramowania próbował to osiągnąć (bez powodzenia) przez ponad dekadę. Jak myślisz, dlaczego wam się powiodło?*

Jason: Wydaje mi się, że stało się tak dlatego, że proponujemy zupełnie inną wartość... czyli uczciwe ceny za dobre produkty — produkty nieprzeładowane funkcjonalnością; produkty, które dobrze robią kilka rzeczy i potem schodzą z drogi. Właśnie to jest istotą tego, co robimy dla ludzi. Jeżeli spojrzymy na wiele innych firm tworzących oprogramowanie — one są w „biznesie oprogramowaniowym”, więc wydaje im się, że muszą tworzyć coraz więcej oprogramowania, wykonującego coraz więcej i więcej rzeczy po to, by usprawiedliwić własne koszty i podtrzymać przekonanie, że są najlepszą firmą na świecie. Wieleść funkcji nie oznacza, że dostarczają oni tego, czego ludzie naprawdę potrzebują. Oni powinni się skupić się na prostszych rzeczach, które pozwalają ludziom lepiej pracować. Nie przejmujemy się tym, co robią inni i ustalamy rozsądne ceny. Na przykład cały przemysł oprogramowania wycenia wszystko według stanowisk lub użytkowników. Jeżeli z programu korzysta jedna osoba, cena wynosi X dolarów, ale jeżeli jest to dziesięć osób, mamy 10 razy X dolarów. Zamiast tego mówimy: „Niech jak największa liczba ludzi korzysta za tę samą cenę”. Nie ściągamy, czegoś, co nazywamy „podatkiem od uczonectwa”, co robią niemal wszystkie inne firmy. Gdy zaczniesz tak robić, zniechęcisz więcej osób do korzystania z produktu. Wszystkie te rzeczy zebrane razem dają nam niezły początek w modelu oprogramowania subskrybowanego.

O pamiętaniu o celu

Pytanie: Wydaje się, że jednym z wątków przewijających się w wywiadach z pracownikami 37Signals jest rada, by oprogramowanie było proste, by nie nękać użytkownika. Teraz, gdy część waszych produktów jest już od pewnego czasu w internecie i zyskaliście większą liczbę użytkowników, prawdopodobnie uzyskaliście wiele informacji zwrotnych. Czy musicie aktywnie walczyć z pragnieniem „przemycania funkcjonalności” do aplikacji?

Jason: Powiedzenie „nie” jest zdecydowanie wyzwaniem i wymaga sporo dyscypliny, ale zakodowaliśmy sobie jako pierwszą reakcję — na wszelkie, włącznie z naszymi, pomysły — mówienie „nie”. Nie oznacza to, że ludzie później nie powiedzą „tak”, ale początkowo automatycznie mówimy „nie” każdemu nowemu pomysłowi, co pozwala nam wybierać i nadal słuchać i słyszeć coraz większą liczbę prośb. Nie reagujemy na prośbę jednej osoby, ale jeżeli usłyszymy ją od dużej liczby użytkowników, natychmiast powiemy „tak”. Nie reagujemy na prośbę jednej osoby, nawet naszą. Gdy masz setki tysięcy użytkowników, nie możesz odpowiadać na pojedyncze prośby.

Utrzymanie prostoty wymaga sporej dyscypliny, a ogólnie rzecz biorąc, przemysłowi oprogramowania brakuje samodyscypliny. W przypadku większości przedsiębiorstw, w miarę dojrzewania produktów, stają się one coraz bardziej i bardziej złożone. Wydaje mi się, że nie jest to właściwa droga. Jednak cały czas się uczymy.

O roli języków zwinnych w bogatych aplikacjach internetowych

Pytanie: Jak natknęliście się na Ruby? Kilka lat temu nie był to najchętniej wybierany język przez większość osób, nawet jeżeli byli oni zapaleńcami zwinnych języków. Od Ruby odstręczał brak natywnego sposobu utworzenia interfejsu użytkownika. Java ma Swing, Python ma TK, WX, PythonCard i szereg innych UI. TCL posiada biblioteki, a Ruby nie miał nic oprócz internetu. Dlaczego więc zdecydowaliście się na Ruby?

David Heinemeier Hansson, twórca Ruby on Rails, pracownik 37Signals: Ruby jest językiem, w którym pisze się piękny kod. Piękny kod jest kodem, który sprawia, że programista jest szczęśliwy, a szczęśliwy programista jest bardziej produktywny. Unikalność Ruby polega na tym, że pozwala on wyrazić coś krótko, zwięźle i pięknie, podczas gdy PHP i Java są albo „przegadane” albo brzydkie w strukturze i sposobie, w jaki strukturyzują kod.

O roli platform w bogatych aplikacjach internetowych

Pytanie: Czy widzieliście platformę Java od Google (<http://code.google.com/webtoolkit/>)? Jak ją oceniacie? Czy widzieliście TurboGears (www.turbogears.com) albo Django (www.djangoproject.com)?

David: Tak. Produkt Google to wciąż Java, więc w czystej formie mnie nie interesuje. Przyjęte przez nich podejście jest tym, z czym zmagamy się już od około roku, czyli z pisaniem JavaScriptu w innym języku, niebędącym JavaScriptem. W Rails mamy RJS. Piszemy JavaScript w Ruby i w momencie uruchomienia jest to kompilowane do JavaScriptu. Fakt, że Google podąża tą samą drogą jest całkiem przyjemny.

Jeżeli chodzi o TurboGears i Django, są to naprawdę interesujące platformy. Nie istnieją jeszcze tak długo, jak Rails i zdecydowanie nie rozwijają się tak szybko, jak Rails. Ponadto są napisane w Pythonie, który jest świetnym językiem, ale to nie jest Ruby. W porównaniu jest im bliżej do Javy niż PHP. Wolałbym pisać w Pythonie niż w Javie czy PHP.

Django wydaje się być bardziej interesujące. TurboGears są w zasadzie „posklejane” z innych komponentów w taki sposób, by pasowały one do rozwiązania. Sukces Rails opiera się w dużej mierze na tym, iż nie jest utworzony w ten sposób. Wszystkie komponenty są rozwijane w sytuacji, gdy „znają” się nawzajem, i dzięki temu łagodzimy kłopoty ze świata Javy, gdzie wszystko jest „klejone”. Trzeba „upolować” wiele różnych platform i spróbować je razem połączyć. Myślę, że platformy Pythona podążają tą samą drogą, którą obraliśmy, czyli korzystania z produktywnego języka, w którym wszystko można zrobić samodzielnie, z większym pożytkiem dla siebie.

O językach zwinnych i skalowalności

Pytanie: Wybierając Ruby, w ogóle martwicie się o skalowalność?

David: Odpowiedź na to pytanie jest łatwa, ponieważ jest dokładnie taka sama, jak dla pozostałych języków open-source. Jest zgodna z tym, co nazywamy *architekturą niewspółdzielenia*. Oznacza to, że rozbijamy wszystko na trzy warstwy: internet, aplikację i bazę danych. Następnie pilnujemy, by żadne dane nie znajdowały się w warstwie internetowej i aplikacji. Przestrzegając tych reguł, uzyskujemy wspaniałą możliwość dodawania nieograniczonej liczby serwerów WWW i nielimitowanej liczby serwerów aplikacji, ponieważ wszystkie dane zostają umieszczone w bazie danych, w warstwie pamięci współdzielonej czy sieciowym systemie plików, więc przestaje to być naszym problemem. Jest to podejście, które przyjęliśmy. Skalowalność w przypadku danych jest trudna. Nie chcemy zajmować się tym problemem. Są ludzie, którzy rozwiązali ten problem dawno temu. Część z nich stanowią specjaliści baz danych. Więc skalowalność nie stanowi żadnej kwestii w przypadku języków zwinnych. Warstwa internetowa i aplikacji są nieskończenie skalowalne — wystarczy dodać więcej serwerów.

Sam język (Ruby) jest nieco wolniejszy niż pozostałe języki. Należy do tej samej kategorii co Perl, PHP i Python. Oczywiście można mocno usprawnić jakiś kod w Javie, czy napisać w C czy C++ coś, co będzie szybsze. To świetnie, ale nie za bardzo ma to znaczenie. Żyjemy w czasach, gdy cykle CPU z dnia na dzień tanieją, ale cykle programistyczne nie tanieją, ani nawet od bardzo dawna nie potaniały. Parametrem, który należy optymalizować, jest liczba cykli programistycznych uzyskiwanych z zespołu. Poprawną drogą jest w tym przypadku użycie produktywnego języka i platformy. Jako miara wydajności — wiemy, że przetwarzamy miliony żądań dziennie. Robimy to bardzo dobrze, a wiele innych witryn obsługuje setki tysięcy użytkowników.

O tworzeniu nowego środowiska programistycznego dla bogatych aplikacji internetowych

Pytanie: *Jakie są ogólne zasady, którymi kierujecie się, decydując, czy nowa cecha czy pomysł wejdą do Rails? Czy jest to łaskawość dyktatury, jak w przypadku Linuksa? Czy decyza grupowa?*

David: Jest to grupowa decyzja, choć zawsze unosi się nad nią cień łaski dyktatora, czyli mój. Jeżeli jest to coś, co mi się nie podoba, nie wchodzi. Bardzo proste. Przeważnie po korzystaniu z Rails przez jakiś czas ludzie dostosowują się do kultury i rozumieją cele oraz co prowadzi do poprawek zgodnych z tą wizją.

Pytanie: *W jakim kierunku będzie się rozwijało Rails?*

David: Będzie wciąż stanowić zbiór rozwiązań problemów zgłaszanych przez uczestników. Nie mamy mapy drogowej. Nie mamy też zamiaru jej tworzyć. Nie opracowujemy funkcjonalności dla innych osób. Jedyne, jakie opracowujemy, to te, które są nam potrzebne. Jest to jedyny sposób uzyskania wydajnej platformy. Jeżeli zaczynasz sobie wyobrażać, co mógłby potrzebować inny programista, zaczynasz stąpać po cienkim lodzie.

O pracy rozpoczynającej się od doświadczenia użytkownika jako specyfikacji funkcjonalnej

Pytanie: *Słyszeliśmy, że filozofią 37Signals jest rozpoczynać od doświadczenia użytkownika i, używając tego jako specyfikacji funkcjonalnej, stawiać na głowie tradycyjny sposób projektowania aplikacji. Spędzanie 80 procent czasu nad interfejsem użytkownika wydaje się przeczyć zdrowemu rozsądkowi. Czy to osobista wrażliwość, czy też wpadliście na to, ponieważ rozpoczynaliście jako firma projektująca strony WWW?*

David: Tak, to zdecydowanie pomogło. Gdy przestaliśmy starać się sprostać oczekiwaniom innych ludzi dotyczących tego, jak robić różne rzeczy, zrobiliśmy więcej rzeczy i to więcej dobrych rzeczy.

O aplikacjach hybrydowych

Pytanie: *Czymś, co zdumiewa osoby niemające wcześniej do czynienia z Web 2.0, jest łatwość tworzenia mashupów. Wydaje się, że jesteście nieco przeciwko wyjawianiu zbyt dużej części API, by ludzie mogli tworzyć mashupy.*

David: Większość naszych produktów ma ujawnione API, ale jest to raczej więcej „gadania” niż „działania”. Nie spotkaliśmy się ze zbyt dużą liczbą mashupów, które byłyby fajne i przydatne, a nie tylko fajne. W większej skali ujawnione API nie mają tak dużego znaczenia, jak chciano by, żebyśmy uwierzyli. Naszym kierunkiem w Rails jest ułatwienie tworzenia API prosto z pudełka.