

## » Idź do

- Spis treści
- Przykładowy rozdział

## » 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

## » Kontakt

Helion SA  
ul. Kościuszki 1c  
44-100 Gliwice  
tel. 032 230 98 63  
e-mail: helion@helion.pl  
© Helion 1991-2008

## Oracle Database. Tworzenie aplikacji internetowych w AJAX i PHP

Autorzy: Lee Barney, Michael McLaughlin

Tłumaczenie: Andrzej Stefański

ISBN: 978-83-246-1977-1

Tytuł oryginału: [Oracle Database Ajax & PHP Web Application Development](#)

Format: 168×237, stron: 408



### Wykorzystaj możliwości najlepszych technologii!

- Jak wykorzystać obiekt XMLHttpRequest?
- Jak manipulować modelem DOM?
- Jakie możliwości kryje w sobie język PL/SQL?

Duet PHP i MySQL cieszy się nieustannie popularnością. Jednak na rynku rozwiązań bazodanowych istnieje także wiele innych produktów. Wśród nich znajdziesz również bazę danych firmy Oracle. Przez znaczną część profesjonalistów uważana jest ona za najlepszą, najbezpieczniejszą i najwydajniejszą platformę do tworzenia zaawansowanych rozwiązań. Dzięki tej książce dowiesz się, jak wykorzystać bazę Oracle w połączeniu z PHP oraz dynamicznymi możliwościami technologii AJAX.

W pierwszej kolejności zapoznasz się z możliwościami skalowania rozwiązań korzystających z JavaScriptu i PHP. Następnie nauczysz się wykorzystywać obiekt XMLHttpRequest, a stąd już tylko krok do opanowania technologii AJAX. Ponadto dowiesz się, jak manipulować elementami w modelu DOM oraz pracować zdalnie z obiektami PHP. Autorzy książki w wyczerpujący, a równocześnie przejrzysty i przyjazny sposób wprowadzą Cię w tajniki administrowania bazami Oracle oraz posługiwania się składnią języków SQL i PL/SQL, wykorzystywanych w tej bazie. Książka ta w umiejętny sposób łączy wiedzę na temat projektowania przyjaznego i dynamicznego interfejsu użytkownika oraz zastosowania najlepszych rozwiązań w dziedzinie baz danych.

- Skalowalność JavaScriptu i PHP
- Obiektość w tych językach
- Możliwości i sposoby wykorzystania obiektu XMLHttpRequest
- Manipulowanie modelem DOM za pomocą JavaScriptu
- Zdalne wywołania w PHP i HTTP-RPC
- Obsługa przycisku „Cofnij” w AJAX-ie
- Zalecenia przy tworzeniu skalowalnych i elastycznych aplikacji
- Udostępnianie VOIP oraz IM
- Wykaz znaczników HTML
- Podstawy języka PHP
- Administrowanie bazami danych Oracle
- Składnia i wykorzystanie języka SQL oraz PL/SQL

**Projektuj skalowalne i elastyczne aplikacje!**

# Spis treści

<b>O autorach</b> .....	<b>9</b>
<b>O redaktorze technicznym</b> .....	<b>11</b>
<b>Wprowadzenie</b> .....	<b>13</b>
<b>Część I Tworzenie podstaw</b> .....	<b>17</b>
<b>Rozdział 1. Skalowalność JavaScript i PHP</b> .....	<b>19</b>
Skalowalność .....	20
Redukcja obciążenia procesora i pamięci .....	21
Skalowanie struktur kontrolnych .....	22
Skalowanie przetwarzania ciągów znaków .....	26
Skalowanie prostych operacji matematycznych .....	28
Wpływ obiektów i ich metod na skalowanie .....	33
Wpływ wielokrotnych żądań wydruku na skalowanie .....	35
Redukcja obciążenia sieci .....	36
Wykorzystanie AJAX do zmniejszenia obciążenia sieci .....	36
Wykorzystanie kompresji do zmniejszenia obciążenia .....	38
Podsumowanie .....	40
<b>Rozdział 2. Modularność PHP i JavaScript</b> .....	<b>41</b>
Modularność .....	41
Projektowanie w rzeczywistym świecie .....	42
Główny moduł i kontrolery .....	49
Kontroler aplikacji i odwzorowania .....	50
Modularność JavaScript z kontrolerami i odwzorowaniami .....	52
Podsumowanie .....	54
<b>Rozdział 3. Obiekty JavaScript i PHP</b> .....	<b>55</b>
Definiowanie i tworzenie klas .....	55
Dziedziczenie .....	59
Konstruktory .....	61
Publiczne, chronione i prywatne .....	63
Atrybuty i metody w JavaScript .....	66
Obiekty modelu, kontrolera i obiekty kontrolne w PHP .....	69
Obiekty modelu, kontrolera i obiekty kontrolne w JavaScript .....	74
Podsumowanie .....	76

<b>Część II</b>	<b>Dynamiczna prezentacja: komunikacja między interfejsem użytkownika i serwerem .....</b>	<b>77</b>
<b>Rozdział 4.</b>	<b>Obiekt XMLHttpRequest .....</b>	<b>79</b>
	Tworzenie żądania i obsługa jego rezultatów .....	79
	Tworzenie modułu Server Access Object .....	83
	Podsumowanie .....	90
<b>Rozdział 5.</b>	<b>AJAX, zaawansowany HTML i komunikacja HTTP .....</b>	<b>93</b>
	Tworzenie rozwijanych tabel HTML .....	93
	Nagłówki HTTP, błędy i komunikacja z serwerem .....	102
	Podsumowanie .....	108
<b>Rozdział 6.</b>	<b>Modyfikowanie DOM za pomocą JavaScript .....</b>	<b>109</b>
	Obiektowy model dokumentu (DOM) .....	110
	Przeciagnij i upuść .....	112
	Biblioteka .....	112
	Obiekty kontrolne w PHP .....	122
	Obiekty kontrolne JavaScript .....	125
	Zapisywanie informacji z mechanizmu przeciagnij i upuść w bazie danych .....	130
	Podsumowanie .....	133
<b>Część III</b>	<b>Zaawansowane zagadnienia dynamicznej prezentacji i komunikacji .....</b>	<b>135</b>
<b>Rozdział 7.</b>	<b>Dostęp do danych z innych aplikacji .....</b>	<b>137</b>
	Zdalne wywołania w PHP i HTTP-RPC .....	138
	Zdalne wywołania za pomocą klientów i usług XML-RPC z PEAR .....	142
	Podsumowanie .....	152
<b>Rozdział 8.</b>	<b>AJAX, tworzenie wykresów i proste przesyłanie danych .....</b>	<b>155</b>
	Korzystanie z biblioteki SimplePlot .....	155
	Generowanie danych do wykresu .....	159
	Pobieranie danych za pomocą AJAX .....	162
	Podsumowanie .....	165
<b>Rozdział 9.</b>	<b>Przycisk cofania w AJAX .....</b>	<b>167</b>
	Dodawanie śledzenia historii .....	167
	Korzystanie z historii iframe .....	175
	Sesje po stronie klienta .....	177
	Podsumowanie .....	179
<b>Część IV</b>	<b>Tworzenie bardzo elastycznych, skalowalnych aplikacji .....</b>	<b>181</b>
<b>Rozdział 10.</b>	<b>Sesje po stronie klienta i po stronie serwera .....</b>	<b>183</b>
	Śledzenie i wykorzystywanie stanu logowania po stronie klienta .....	183
	Śledzenie i wykorzystywanie stanu logowania po stronie serwera .....	192
	Definiowanie i przechowywanie preferencji użytkownika między sesjami .....	197
	Wykorzystanie preferencji zapisanych przez użytkownika .....	201
	JSON .....	204
	Podsumowanie .....	205

<b>Rozdział 11. Tworzenie modyfikowanych przez użytkowników serwisów typu mashup ....</b>	<b>207</b>
Tworzenie prostej strony typu mashup .....	208
Wbudowanie mashupa do systemu obsługi danych medycznych .....	211
Jak to działa .....	214
Podsumowanie .....	218
<b>Rozdział 12. Wielowymiarowa komunikacja: VOIP, IM oraz predefiniowane raporty .....</b>	<b>219</b>
Udostępnianie VOIP i IM .....	220
Rozproszone, ukierunkowane, raportowane .....	225
Podsumowanie .....	243
<b>Dodatki .....</b>	<b>245</b>
<b>Dodatek A Indeks znaczników HTML .....</b>	<b>247</b>
<b>Dodatek B Wprowadzenie do PHP .....</b>	<b>255</b>
Historia i tło .....	256
Czym jest PHP? .....	256
Czym jest Zend? .....	257
Tworzenie aplikacji internetowych .....	257
Co z czym łączyć i dlaczego? .....	257
Co Oracle wnosi do PHP? .....	258
Dlaczego PHP 5 jest ważny? .....	259
Jak korzystać z PHP? .....	259
Jak za pomocą PHP i OCI8 korzystać z bazy danych Oracle? .....	278
Podsumowanie .....	299
<b>Dodatek C Wprowadzenie do administracji bazą danych Oracle .....</b>	<b>301</b>
Architektura baz danych Oracle .....	302
Uruchamianie i wyłączanie bazy danych Oracle .....	307
Operacje w systemie Linux .....	307
Operacje w systemie Windows .....	311
Uruchamianie i wyłączanie procesu nasłuchującego .....	314
Korzystanie z programu SQL*Plus .....	319
Interfejs wiersza poleceń .....	321
Zmienne łączące .....	323
Podsumowanie .....	323
<b>Dodatek D Wprowadzenie do SQL .....</b>	<b>325</b>
Typy danych Oracle SQL .....	326
Język definicji danych (DDL) .....	329
Zarządzanie tabelami i ograniczeniami .....	330
Zarządzanie widokami .....	333
Zarządzanie procedurami składowanymi .....	334
Zarządzanie sekwencjami .....	335
Zarządzanie własnymi typami .....	336
Język pobierania danych (DQL) .....	338
Zapytania .....	338
Język modyfikacji danych (DML) .....	342
Polecenia INSERT .....	342
Polecenia UPDATE .....	344
Polecenia DELETE .....	345
Język sterowania danymi (DCL) .....	345
Podsumowanie .....	346

<b>Dodatek E</b>	<b>Wprowadzenie do PL/SQL</b>	<b>347</b>
	Struktura bloków PL/SQL	348
	Zmienne, przypisania i operatory	352
	Struktury sterujące	354
	Instrukcje warunkowe	355
	Pętle	358
	Procedury składowane, funkcje składowane oraz pakiety	360
	Funkcje składowane	361
	Procedury	364
	Pakiety	366
	Wyzwalacze bazodanowe	369
	Wyzwalacze DDL	370
	Wyzwalacze DML	370
	Wyzwalacze „zamiast”	372
	Wyzwalacze systemowe lub bazodanowe	372
	Kolekcje	373
	Typ danych VARRAY	374
	Typ danych NESTED TABLE	375
	Tablica asocjacyjna	376
	Interfejs kolekcji	379
	Wykorzystanie pakietu DBMS_LOB	379
	Konfiguracja i weryfikacja środowiska dla typów LOB	379
	Zapis i odczyt typu danych CLOB	381
	Podsumowanie	386
	<b>Skorowidz</b>	<b>387</b>

## Rozdział 11.

# Tworzenie modyfikowanych przez użytkowników serwisów typu mashup

Gdy wydajne komputery stały się popularniejsze, użytkownicy zaczęli postrzegać je jako narzędzia do agregacji informacji. Wpłynęło to bezpośrednio na rozwój funkcjonalności serwisów web. Takie serwisy pozwalają użytkownikom wybierać informacje do przeglądania. Dzisiejsze przeglądarki umożliwiają też wyświetlanie kanałów RSS w postaci odnośników na stronie lub zakładek. E-mail już dawno został połączony z przeglądarką. Wszystko to, łącznie z aplikacjami AJAX, prowadzi użytkownika do wniosku, że skoro przeglądarka służy do agregacji danych, to takie jest też najważniejsze zastosowanie komputera.

Ponieważ użytkownicy wymagają coraz większej elastyczności aplikacji internetowych, te muszą się dostosować. Jednym ze sposobów na to jest wykorzystanie serwisów typu mashup wyświetlających kilka części różnych stron na jednej stronie. Przykładową definicję tego, czym jest mashup, można znaleźć w angielskiej Wikipedii pod adresem [http://en.wikipedia.org/wiki/Mashup\\_\(web\\_application\\_hybrid\)](http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)).

Przeczytanie tego rozdziału pozwoli stworzyć strony umożliwiające użytkownikom połączenie danych z różnych zdalnych serwisów internetowych lub aplikacji w jedną, opartą na AJAX aplikację internetową. Będą oni mogli zdefiniować, jakie części stron wyświetlić i gdzie mają się one znaleźć na stronie.

W tym rozdziale zostaną poruszone zagadnienia takie jak:

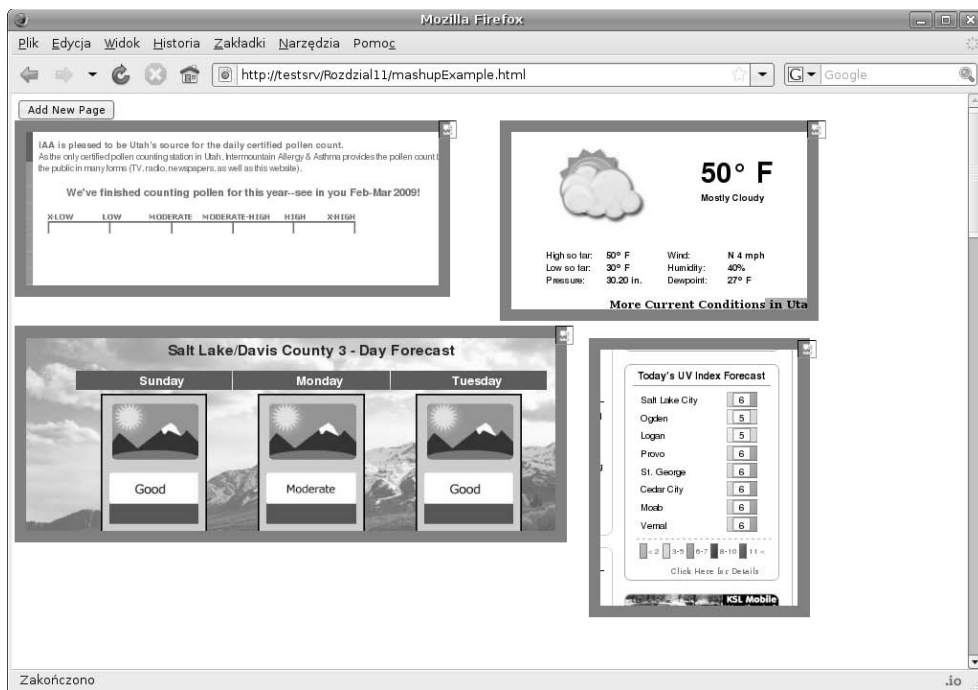
- ◆ wykorzystanie JavaScript do definiowania i wyświetlania wielu fragmentów stron internetowych na jednej stronie,
- ◆ wykorzystanie JSON do zapisywania i pobierania danych,
- ◆ API biblioteki mashup.

Dynamiczne usługi typu mashup umożliwiają użytkownikowi dostosowanie aplikacji przy małym wysiłku zarówno ze strony samego użytkownika, jak i programisty.

## Tworzenie prostej strony typu mashup

Pielęgniarze opiekujący się pacjentami w domu nie mają tak dużej kontroli nad otoczeniem swoich pacjentów jak pielęgniarze w szpitalu. Dlatego potrzebują narzędzia do kontroli temperatury i jakości powietrza w okolicy, w której mieszka pacjent. Choć istnieje wiele serwisów, które mogą dostarczyć części tych informacji dla dowolnej lokalizacji, niewiele z nich lub wręcz żaden nie dostarcza wszystkich. Aby mieć aktualne informacje i działać profilaktycznie, pielęgniarze musieliby regularnie odwiedzać kilka serwisów w ciągu dnia. Duże ograniczenia czasowe, jakie ludzie ci mają w pracy, i potencjalne zagrożenie zdrowia lub życia pacjentów w wypadku ignorowania tego typu informacji powodują, że potrzebny jest prosty sposób wybrania preferowanych źródeł informacji i ich wyświetlania. Serwis typu mashup nadaje się do tego idealnie.

Serwis typu mashup składa się z fragmentów innych stron internetowych umieszczonych na jednej stronie. W swojej najprostszej postaci mashup jest definiowany przez programistę lub inżyniera i nie pozwala użytkownikowi na dodawanie nowych stron. Plik *mashupExample.html*, który można pobrać z <ftp://ftp.helion.pl/przyklady/ordatw.zip>, pokazano na rysunku 11.1. Stanowi on przykład takiego prostego serwisu mashup i wyświetla informacje o parametrach powietrza z obszaru Salt Lake City w amerykańskim stanie Utah.



Rysunek 11.1. Prosta strona typu mashup w przeglądarce Firefox

Ramki mashupa składają się z wyświetlanych stron i dają użytkownikowi możliwość zmiany rozmiaru, wybrania części strony źródłowej do wyświetlenia, przesunięcia wyświetlanej ramki i jej usunięcia. Strona *mashupExample.html* zawiera również przycisk pozwalający użytkownikowi

kowi na dodanie nowych ramek do mashupa. Wszystkie zmiany na tej stronie mają charakter tymczasowy, ponieważ nie są przesyłane do zapisania na serwerze. Sposób zapisywania tych informacji w bazie danych Oracle został opisany w dalszej części tego rozdziału.

Ponieważ ramki mashupa odwołują się do źródłowych stron przez sieć, zawsze wyświetlają informacje z aktualnej strony zdalnego serwera. Dlatego gdy strona źródłowa się zmienia, zmienia się także zawartość ramki wyświetlanej w mashupie. Umożliwia to utrzymanie aktualności danych. Z tego powodu jedyny czynnik do wzięcia pod uwagę przy wybieraniu strony źródłowej dla ramki mashupa stanowi stabilność wyglądu strony. Ponieważ wyświetlany jest tylko wybrany obszar strony źródłowej, jeśli zmieni się układ tej strony w obszarze wybranym do wyświetlania, w ramce mashupa będzie znajdowała się inna treść.

Opisane w tabeli 11.1 API wykorzystywane do tworzenia mashupów zawiera trzy funkcje i zależy od biblioteki dostarczającej mechanizm przeciągnij i upuść, opisanej w rozdziale 6. Te trzy funkcje umożliwiają programiście zdefiniowanie ramek mashupa i pobranie opisu stanu każdej z nich. Biblioteka mashup znajdująca się w pliku *mashup.js* do pobrania z <ftp://ftp.helion.pl/przyklady/ordatw.zip> została napisana w taki sposób, by obsługiwać najnowsze wersje przeglądarek Firefox i Safari na OS X oraz Firefox i IE w Windows. Dopracowania wymaga wsparcie przeglądarki Firefox dla systemu Linux.

Plik *mashupExample.html* korzysta z dwóch funkcji API opisanych w tabeli 11.1. Składa się on z czterech ramek mashupa — rozmieszczonych w różnych miejscach i mających różne rozmiary — wyświetlających aktualną temperaturę i informacje o jakości powietrza w Salt Lake City w amerykańskim stanie Utah.

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="mashup.css" />
  <script src="util.js" type="text/javascript"></script>
  <script src="mashup.js" type="text/javascript"></script>
  <script src="JSON_Util.js" type="text/javascript"></script>

  <script>
    function init(){
      new MashFrame('www.intermountainallergy.com/pollen.html',
        ↪ 'displayDiv', 0, 30, 550, 200, 300, 60);
      new MashFrame('http://www.airquality.utah.gov/sl1c.html',
        ↪ 'displayDiv', 0, 295, 700, 250, 210, 200);
      new MashFrame('www.ks1.com/index.php?nid=88', 'displayDiv', 625, 30, 400, 230, 850, 0);
      new MashFrame('www.ks1.com/index.php?nid=88',
        ↪ 'displayDiv', 740, 310, 380, 220, 550, 40);
    }
  </script>
</head>
<body id='mainBody' onload='init()'>
  <input type = 'button' value='Add New Page' onclick='requestNewMashFrame("displayDiv")' />
  <div id='displayDiv' style='width: 1000px; height: 3000px; top: 50px;' >

</div>

</body>
</html></div>
</body>
</html>
```

Tabela 11.1. API biblioteki *mashup*

Funkcja	Opis
MashFrame (aURL, parentID, xLoc, yLoc, width, height, scrollDown, scrollRight)	<p>Ta funkcja konstruuje ramkę mashupa. Wymaga podania dwóch parametrów, ale może przyjmować sześć dodatkowych parametrów opcjonalnych.</p> <p>Wymagane parametry:</p> <ul style="list-style-type: none"> <li>♦ aURL URL źródła strony. Nie musi zawierać prefiksu <code>http://</code>. Wszystkie URL powinny korzystać z protokołu HTTP, a nie FTP czy innego.</li> <li>♦ parentID Identyfikator elementu HTML, który będzie zawierał ramkę mashupa. Jest to zazwyczaj element <code>div</code> HTML.</li> </ul> <p>Opcjonalne parametry:</p> <ul style="list-style-type: none"> <li>♦ xLoc Poziome przesunięcie lewej krawędzi ramki mashupa od brzegu zajmowanego elementu HTML. Domyślna wartość to 0 pikseli.</li> <li>♦ yLoc Pionowe przesunięcie górnej krawędzi ramki mashupa w stosunku do górnej krawędzi zajmowanego elementu HTML. Domyślna wartość to 0 pikseli.</li> <li>♦ width Szerokość wyświetlanej ramki mashupa w pikselach. Domyślna wartość to 300 pikseli.</li> <li>♦ height Wysokość wyświetlanej ramki mashupa w pikselach. Domyślna wartość to 294 piksele.</li> <li>♦ scrollDown Pionowe przesunięcie w dół widocznego obszaru strony źródłowej w pikselach. Domyślna wartość to 0 pikseli.</li> <li>♦ scrollRight Poziome przesunięcie w prawo widocznego obszaru strony źródłowej w pikselach. Domyślna wartość to 0 pikseli.</li> </ul>
requestNewMashFrame(parentID)	<p>Ta funkcja opakowuje konstruktor <code>MashFrame</code>, korzystając z wartości domyślnych wszystkich opcjonalnych parametrów. Po uruchomieniu funkcja ta pyta użytkownika o URL strony źródłowej. Wymaga podania jednego parametru.</p> <p>parentID Identyfikator elementu HTML, który będzie zawierał ramkę mashupa. Jest to zazwyczaj element <code>div</code> HTML.</p>
getMashupDescriptor()	<p>Ta funkcja zwraca tablicę opisów ramek mashupów. Każdy deskryptor zawiera wszystkie aktualne wartości ramki mashupa. Zwrócona tablica jest wykorzystywana do zapisywania aktualnego stanu wszystkich ramek mashupa na serwerze.</p>

Należy zauważyć, że element `displayDiv` ma ustawioną dużą szerokość i wysokość. Jest to niezbędne, ponieważ gdy ramka mashupa jest przenoszona lub zmieniany jest jej rozmiar, czasem kursor może opuścić szary pasek wykorzystywany do przeciągania lub zmiany rozmiarów ramki mashupa. Jest to typowe zjawisko w przeglądarkach z powodu ilości czasu potrzebnego do interpretowania i wykonywania kodu JavaScript, a następnie rysowania zmian. Aby to pokonać, biblioteka *mashup*, za pomocą biblioteki przeciągnij i upuść *util.js*, dodaje funkcję obsługującą zdarzenia myszy do elementu nadrzędnego, co powoduje, że ramka mashupa „przechwytuje” mysz.

Czasem ten sam problem występuje też w obszarze wyświetlania ramki mashupa. We wszystkich przeglądarkach oprócz IE ramka mashupa ponownie łączy się z myszą. Przy przemieszczaniu lub zmianie rozmiaru ramki mashupa w IE należy zachować ostrożność i przesuwać mysz dość wolno, by nie weszła w obszar wyświetlania ramki. Powodem tego jest fakt, że w IE wiele aktywnych komponentów, takich jak odnośniki, zawsze ma najwyższy priorytet w kolejności wyświetlania, niezależny od priorytetu deklarowanego dla nich przez elementy nadrzędne, i dlatego kod przechwytyjący nie może być zaimplementowany.

Ponieważ biblioteka *mashup.js* korzysta z pustego, przykrywającego obszar wyświetlania znacznika `div`, nazywanego też `glass pane`, może wykrywać opuszczone ruchy myszy i zatrzymać ramkę mashupa. W IE obiekt `glass pane` został usunięty, ponieważ aktywne komponenty wyświetlanej strony i tak byłyby powyżej niego. Z powodu takiej implementacji priorytetów kolejności wyświetlania w IE pojedyncze aktywne komponenty przechwytyją zdarzenia związane z ruchem myszy i ramka mashupa nie jest przywiązywana do kursora myszy. Dla zachowania spójności warstwa `glass pane` została tutaj usunięta. W bibliotece mashup pojawiają się też inne ograniczenia.

Jeśli wyświetlana strona zawiera Flash lub inne wbudowane interpretery, może się zdarzyć, że film lub inna zawartość tego typu będzie widoczna nawet poza ramką mashupa. Problem ten występuje częściej w Windows, zarówno w przeglądarce Firefox, jak i IE, niż w OS X i nie został przetestowany w systemie Linux. Wydaje się bardziej zależny od wyświetlanej strony niż od przeglądarki, dlatego należy ostrożnie wybierać wyświetlane w mashupie strony.

Innym ograniczeniem są menu w JavaScript. Niektóre serwisy mają menu, które automatycznie pojawiają się po załadowaniu strony. Takie menu również mogą czasem się pojawiać nawet wtedy, gdy znajdują się poza obszarem wyświetlania ramki mashupa.

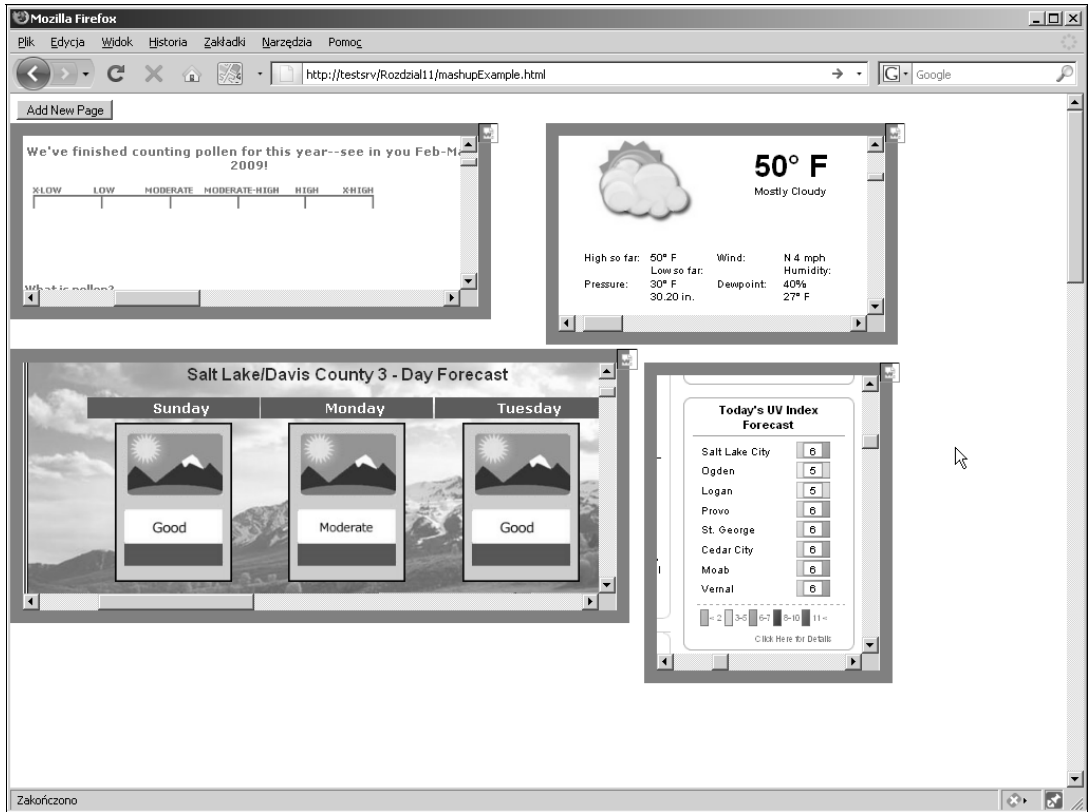
Istnieje również różnica w funkcjonalności pomiędzy tym, jak biblioteka mashup działa w systemie Windows, i w innych systemach operacyjnych. Rysunek 11.1 pokazuje prosty przykład w przeglądarce Firefox pracującej pod kontrolą systemu Linux. Należy zauważyć, że okna mashupów nie mają tam pasków przewijania. Przeciąganie zawartości w ramce mashupa zmienia wyświetlany tam obszar strony źródłowej. Przy takim podejściu do wyświetlania strony źródłowej w Firefoksie i IE pod systemem Windows obie przeglądarki źle odrysowywały wyświetlany obszar i poza oknem wyświetlania pojawiały się widoczne pozostałości. Problem ten znikł przy użyciu pasków przewijania pokazanych na rysunku 11.2.

Te widoczne pozostałości nie pojawiają się w przeglądarkach Firefox i Safari działających w systemach OS X i Linux, dlatego wygląda na to, że problem dotyczy biblioteki MFC Windows wykorzystywanej do wyświetlania grafiki w obu tych przeglądarkach pod Windows.

Jest nadzieja, że z czasem biblioteka mashup zostanie usprawniona oraz rozbudowana, a problemy te zostaną pokonane.

## Wbudowanie mashupa do systemu obsługi danych medycznych

Jak już wspomniano na początku poprzedniego podrozdziału, wykorzystanie mashupa bardzo pomogłoby pielęgniarzom opiekującym się ludźmi w ich własnych domach. W tej sytuacji ważne jest, aby mashup był prosty w użyciu oraz pamiętał ustawienia. Jeśli takie same komponenty interfejsu użytkownika są stosowane w innych częściach systemu obsługi danych medycznych, użytkownik może łatwo wykorzystać tę funkcjonalność. Wykorzystując takie samo podejście projektowe jak w innych rozdziałach tej książki, programista może łatwo dołączyć potrzebną funkcjonalność. Rysunek 11.3 pokazuje ramki mashupa umieszczone na głównej stronie.



**Rysunek 11.2.** Prosta strona mashup wyświetlona w przeglądarce Firefox pod Windows

W rozdziale 10. zostało omówione, jak obiekty sesji po stronie klienta są zapisywane i pobierane z bazy danych Oracle za pomocą AJAX. Ponieważ ustawienia mashupa wprowadzone przez użytkownika muszą zostać zapisane, można je dodać do obiektu sesji w taki sam sposób jak informację o wybranej stronie startowej w poprzednim rozdziale. Proces logowania z rozdziału 10. w tej sytuacji powinien pobrać te ustawienia i umieścić je w obiekcie sesji, tak aby były dostępne, gdy użytkownik zażąda wyświetlenia mashupa.

Aby wykorzystać kod służący do zapisywania obiektu sesji, musi być stworzone BCO według wzoru opisanego w rozdziale 3. Obiekt `saveMashupBCO` znajduje się w pliku `CO.js`, który można pobrać z <ftp://ftp.helion.pl/przyklady/ordatw.zip>.

```
function saveMashupBCO(){
    // utwórz atrybut lub zastąp nowym
    session.addAttribute('mashupDesc', getMashupDescriptor());
    // umieść ciąg opisujący sesję na serwerze bez korzystania z VCO
    theSAO.makeCall('POST', null, 'Text', true, '',
    'cmd=store&sessDef='+session.toJSONString());
}
```

Obiekt `saveMashupBCO` wstawia opisy wszystkich ramek zdefiniowanego przez użytkownika mashupa, dopisując rezultat działania funkcji `getMashupDescriptor` API jako atrybut obiektu `Session`. Następnie, wykorzystując funkcję, która służy do zapisywania sesji klienta i którą



Rysunek 11.3. System obsługi danych medycznych po wybraniu opcji Pokaż Mashup

stworzono w rozdziale 10., generuje widoczne tutaj żądanie HTTP POST. Spowoduje to, że opisy zostaną zapisane w polach bazy danych zawierających sesję w postaci ciągu znaków JSON. Więcej o JSON można dowiedzieć się z ostatniego podrozdziału rozdziału 10.

Jak widzieliśmy w poprzednich rozdziałach, w sytuacji, gdy użytkownik chce wyświetlić mashup, musi zostać wywołany BCO i odpowiadający mu VCO. Obiekt `mashupBCO` pokazany w poniższym kodzie i w pliku `CO.js` różni się od większości omawianych dotychczas obiektów BCO w JavaScript tym, że nie pobiera danych z serwera. Gdy użytkownik jest zalogowany, utworzony obiekt `Session` zawiera wszystkie zapisane informacje na temat mashupa.

```
function mashupBCO(){
    //nie jest potrzebne połączenie z serwerem, ponieważ
    //informacje o sesji zostały pobrane przy logowaniu
    var mashupDescriptorArray = session.getAttribute('mashupDescre');
    var aVCO = new mashupVCO();
    aVCO.notify(mashupDescriptorArray);
}
```

Z tego powodu BCO musi tylko pobrać te informacje z obiektu sesji i wywołać bezpośrednio odpowiedni VCO, przekazując dane jako parametr.

Oba te obiekty są bardzo proste. Obiekt `mashupVCO` znajdujący się w `CO.js` jest również bardzo prosty dzięki wykorzystaniu biblioteki `mashup`.

```
function mashupVCO(){
    this.notify = function(data){
        var displayString = "<div>";
```

```

displayString += "<input type='button' value='Dodaj kolejną stronę'
↳onclick='requestNewMashFrame(\"mashupContainer\")' />";
displayString += "<input type='button' value='Zapisz zmiany w mashupie'
↳onclick='saveMashupBCO()' />";
displayString += "<div id='mashupContainer' style='height: 4000px; width:
↳2000px;'></div></div>";
document.getElementById('content').innerHTML = displayString;

MashFrame.mashCount = 0;
MashFrame.mashArray = new Array();
if(data != null){
    var numFrames = data.length;
    for(var i = 0; i < numFrames; i++){
        var aDescription = data[i];
        new MashFrame(aDescription.URL, 'mashupContainer', aDescription.left,
↳aDescription.top,
            aDescription.width, aDescription.height,
↳aDescription.scrollDown, aDescription.scrollRight);
    }
}
}
}
}

```

Obiekt `mashupVCO` powoduje wyświetlenie dwóch obiektów `div`. Pierwszy zawiera przyciski umożliwiające dodawanie nowych ramek mashupa oraz zapisanie opisów mashupów na serwerze. Drugi `div` zawiera same ramki mashupa. Dodatkowo ten `VCO` tworzy obiekt `MashupFrame` dla każdego zapisanego opisu i umieszcza go w odpowiednim miejscu.

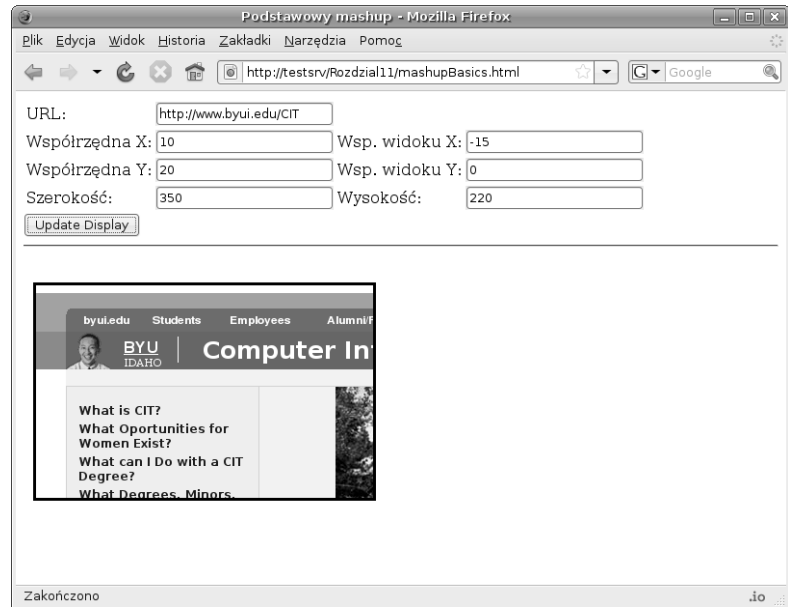
## Jak to działa

Biblioteka `mashup` znajdująca się w pliku `mashup.js` do pobrania z <ftp://ftp.helion.pl/przyklady/orodatw.zip> wymaga zrozumienia najważniejszej koncepcji, modyfikowania elementów za pomocą JavaScript, aby zmienić atrybuty stylów CSS. Aby nie odkrywać koła na nowo, wiele z tych modyfikacji jest wykonywanych za pomocą biblioteki obsługującej mechanizm przeciągnij i upuść opisanej w rozdziale 6. Dzięki wykorzystaniu gotowej biblioteki znajdującej się w pliku `util.js` działania takie jak modyfikacja lokalizacji, szerokości i wysokości ramek mashupa są obsługiwane za pomocą gotowych funkcjonalności. Gdy klient jest uruchomiony w systemie OS X, również przesuwanie wyświetlanej strony w ramce mashupa jest wykonywane za pomocą przeciągania i upuszczania, a nie za pomocą pasków przewijania jak w Windows.

Poza tym aby zrozumieć, co dzieje się w bibliotece `mashup`, trzeba wiedzieć, że każda ramka mashupa składa się z dwóch głównych części — ramki `iframe` wyświetlającej żadaną stronę oraz `div` przechowującego tę ramkę oraz mającego styl CSS `overflow` ustawiony na `hidden`. Aby pomóc w zrozumieniu, jak to działa, utworzono stronę `mashupBasics.html` niekorzystającą z biblioteki `przeciągnij i upuść`. Strona ta, pokazana na rysunku 11.4, umożliwi użytkownikowi modyfikowanie tych samych atrybutów CSS co biblioteka `przeciągnij i upuść`, ale za pomocą pól do wprowadzania danych.

**Rysunek 11.4.**

Prosta strona  
typu mashup



Na podstawie tego przykładu, umożliwiającego bezpośrednie modyfikowanie atrybutów stylu CSS zamiast użycia biblioteki przeciągnij i upuść, można zrozumieć leżącą u podstaw prostotę wbudowywania ramek mashupa.

Plik *mashupBasics.html* pokazany w poniższym kodzie można pobrać z <ftp://ftp.helion.pl/przyklady/ordatw.zip>. Zawiera on głównie kod HTML służący do definiowania wyświetlania. Funkcjonalność JavaScript składa się z dwóch funkcji, `updateEmbeddedPage` oraz `getUpperLeftPoint`. Ważne jest, by zrozumieć, że funkcja `updateEmbeddedPage` nie jest wykorzystywana w bibliotece mashup, ale została utworzona, aby umożliwić takie same modyfikacje jak ta biblioteka za pomocą biblioteki przeciągnij i upuść.

```
<html>
<head>
<title>Podstawowy mashup</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<style>
#containerDiv{
    width: 1px;
    height: 1px;
}
#maskDiv{
    width: 350px;
    height: 220px;
    border: solid;
    position: absolute;
    top: 150px;
    left: 10px;
    overflow: hidden;
    background-color: white;
}
#display{
    width: 1000px;
```

```

    height: 5000px;
    border: none;
    position: absolute;
}
</style>
<script>
function updateEmbeddedPage(){
    var basePoint = getUpperLeftPoint(document.getElementById('containerDiv'));
    var aURL = document.getElementById('url').value || 'http://www.byui.edu/CIT';
    var locX = document.getElementById('locX').value || 0;
    var locY = document.getElementById('locY').value || 0;
    var visX = document.getElementById('visX').value || 0;
    var visY = document.getElementById('visY').value || 0;
    var width = document.getElementById('width').value || 350;
    var height = document.getElementById('height').value || 220;
    //ustaw URL ramki iframe
    if(document.getElementById('display').src != aURL){
        document.getElementById('display').src = aURL;
    }
    //ustaw położenie góry i lewego brzegu sekcji div z zawartością
    var maskDiv = document.getElementById('maskDiv');
    maskDiv.style.top = (basePoint.topValue+(locY*1))+ 'px';
    maskDiv.style.left = (basePoint.leftValue+(locX*1))+ 'px';
    //ustaw szerokość i wysokość sekcji div z zawartością
    maskDiv.style.width = width+'px';
    maskDiv.style.height = height+'px';
    //ustaw położenie górnego i lewego brzegu ramki iframe
    var display = document.getElementById('display');
    display.style.top = visY+'px';
    display.style.left = visX+'px';
}
//ustal odległość elementu w pikselach od góry i lewej strony,
//gdzie 0,0 jest lewym górnym rogiem strony,
//a nie innego elementu nadrzędnego
function getUpperLeftPoint(aNode){
    var aPoint = new Object()
    aPoint.leftValue = aNode.offsetLeft;
    aPoint.topValue = aNode.offsetTop;
    while((aNode = aNode.offsetParent) != null){
        aPoint.leftValue += aNode.offsetLeft;
        aPoint.topValue += aNode.offsetTop;
    }
    return aPoint;
}
</script>
<body>
<table>
<tr>
<td>URL: </td>
<td><input id='url' value='http://www.byui.edu/CIT'</td>
</tr>
<tr>
<td>Współrzędna X: </td>
<td><input id='locX' value='0'></td>
<td>Wsp. widoku X: </td>
<td><input id='visX' value='0'></td>
</tr>

```

```

<tr>
  <td>Współrzędna Y: </td>
  <td><input id='locY' value='0' /></td>
  <td>Wsp. widoku Y: </td>
  <td><input id='visY' value='0' /></td>
</tr>
<tr>
  <td>Szerokość: </td>
  <td><input id='width' value='350' /></td>
  <td>Wysokość: </td>
  <td><input id='height' value='220' /></td>
</tr>
</table>
<input type='button' value='Update Display' onclick='updateEmbeddedPage()';
<hr/>

<div id='containerDiv'>
  <div id='maskDiv'>
    <iframe id='display' src="http://www.byui.edu/CIT" ></iframe>
  </div>
</div>
<div style='position: absolute; top: 500px;'>
</div>
</body>
</html>

```

Funkcja `updateEmbeddedPage` rozpoczyna się od pobrania wszystkich wartości z wejścia strony i zapisania ich w zmiennych wewnątrz JavaScript. Jeśli użytkownik nie wprowadził odpowiednich wartości, do zmiennych przypisywane są domyślne wartości. Gdy zostanie to wykonane, rozpoczyna się proces modyfikacji stylów CSS.

Jak zostało powiedziane wcześniej, istnieją dwa główne elementy: zawierający ramkę element `div` nazwany `maskDiv` oraz ramka `iframe` nazwana `display`. Atrybut `iframe src` przechowujący lokalizację strony do wyświetlenia jest ustawiony na wartość odnalezioną w polu URL. Jak w przypadku każdej ramki `iframe`, po wprowadzeniu modyfikacji strona jest odświeżana.

Góra i lewy brzeg strony wyświetlanej w ramce `iframe` są ustawiane względem górnego, lewego rogu całej strony. Umożliwia to umieszczanie `maskDiv` w dowolnym miejscu. Może to sugerować, że zawartość przemieszczana jest w elemencie `maskDiv`, ale w rzeczywistości przesuwana jest po całej stronie. Z tego powodu ramka umieszczona jest w `maskDiv`, z wartością `hidden` ustawioną w atrybucie `overflow` stylu CSS elementu `maskDiv`. Dzięki temu cała zawartość `iframe` znajdująca się poza `maskDiv` jest ukrywana. Takie zachowanie powodujące, że zawartość strony poza `iframe` jest ukrywana, daje ramce mashupa szczególne właściwości.

Plik `mashupBasics.html` umożliwia eksperymentowanie z pozycją i rozmiarami elementów, co pozwala na zrozumienie, co komponenty biblioteki przeciągnij i upuść zastosowane w bibliotece mashup muszą robić. Przygotowuje to też do samodzielnego analizowania samej biblioteki mashup.

## Podsumowanie

Dzięki wykorzystaniu biblioteki mashup oraz obiektów sesji po stronie serwera i ich zapisywania z rozdziału 10. wstawianie ramek mashupa do aplikacji obsługującej dane medyczne, jak też i umieszczanie ich na innych stronach lub w innych aplikacjach jest łatwe. Umożliwia to użytkownikowi dołączanie dodatkowych danych, które są mu potrzebne, do aplikacji bez zmiany jej kodu. Robiąc to, użytkownik staje się partnerem w większym stopniu niż kiedykolwiek wcześniej w historii aplikacji sieciowych i uzyskuje pewnego rodzaju poczucie współuczestnictwa w tworzeniu aplikacji bez konieczności zapoznawania się z kodem czy nawet poświęcania na to minimalnej ilości czasu. Dołączenie mashupów do aplikacji zmniejsza obciążenie jej twórców i wyraźnie zwiększa ilość pozytywnych doświadczeń użytkowników.