

XII Seminarium PLOUG
Warszawa
Marzec 2006

AJAX – rewolucja w tworzeniu aplikacji internetowych

Marek Wojciechowski
e-mail: marek@cs.put.poznan.pl

Politechnika Poznańska, Instytut Informatyki

Abstrakt. Asynchronous JavaScript And XML (AJAX) to nowa technika tworzenia aplikacji internetowych, pozwalająca na uzyskanie niespotykanego dotychczas w tego typu aplikacjach poziomu interaktywności interfejsu użytkownika. AJAX nie stanowi nowej samodzielnej technologii, a jedynie systematyzuje sposób tworzenia interaktywnych aplikacji internetowych w oparciu o języki JavaScript i CSS, obiektowy model dokumentów DOM i obiekt XMLHttpRequest. Celem artykułu jest przedstawienie motywacji dla techniki AJAX, jej założeń, zalet i wad oraz omówienie sposobu tworzenia aplikacji AJAX i ról poszczególnych technologii składowych.

Informacja o autorze. Marek Wojciechowski jest adiunktem w Instytucie Informatyki Politechniki Poznańskiej. Jest autorem ponad 40 publikacji naukowych o zasięgu międzynarodowym, głównie z dziedziny eksploracji danych. Prowadzi wykłady i szkolenia z eksploracji danych, systemów baz danych i technologii internetowych.

1. Wprowadzenie

W klasycznym modelu aplikacji internetowych interakcja użytkownika z aplikacją pracującą na serwerze aplikacji odbywa się w trybie żądanie-odpowiedź. Interfejs użytkownika ma postać powiązanych ze sobą regułami nawigacji stron WWW, w całości lub częściowo generowanych programowo przez aplikację po stronie serwera. Praca użytkownika z daną stroną interfejsu aplikacji kończy się wybraniem łącza prowadzącego do kolejnej strony aplikacji lub zatwierdzeniem formularza do wprowadzania danych za pomocą przycisku na stronie. W obu przypadkach interakcja użytkownika ze stroną kończy się wysłaniem żądania HTTP do serwera, a serwer w odpowiedzi wysyła do przeglądarki kolejną stronę aplikacji.

Wspomniany wyżej klasyczny model aplikacji internetowych jest konsekwencją pierwotnego zastosowania usługi WWW, jakim było publikowanie w sieci Internet dokumentów hipertekstowych, powiązanych ze sobą linkami umożliwiającymi nawigację z jednego dokumentu do drugiego. Przez długie lata byliśmy świadkami ignorowania faktu, że architektura WWW odpowiednia dla udostępniania treści niekoniecznie jest optymalna dla udostępniania aplikacji. Podstawowym problemem klasycznych aplikacji internetowych jest oczywiście ich niski stopień interaktywności w porównaniu z aplikacjami desktopowymi, uruchamianymi na komputerze użytkownika. W trakcie gdy użytkownik np. wprowadza dane do formularza, serwer nie zauważy żadnej aktywności użytkownika aż do momentu zatwierdzenia formularza. Z kolei gdy serwer generuje i przesyła kolejną stronę do przeglądarki, użytkownik musi poczekać aż nowa strona zostanie w całości odebrana przez przeglądarkę. W przypadku skomplikowanych aplikacji wymagających np. dostępu do bazy danych i obciążonego serwera aplikacji, czasy przestoju mogą być znaczące. Dodatkowym argumentem skłaniającym do poszukiwania nowych rozwiązań jest też z pewnością fakt, że często kolejna strona wysłana przez serwer do przeglądarki niewiele się różni od poprzedniej, a mimo to generowana jest od początku i przesyłana w całości. Mamy więc w tym wypadku do czynienia z marnowaniem czasu serwera i przepustowości sieci.

Odpowiedzią na niewątpliwe wady klasycznego modelu aplikacji internetowych jest robiąca w ostatnich miesiącach oszałamiającą karierę koncepcja AJAX, czyli Anynchronous JavaScript and XML. Podstawowym założeniem AJAX jest stwierdzenie, że przeglądarka uruchamia aplikację, a nie tylko prezentuje strony stanowiące interfejs użytkownika. Logika aplikacji pracująca po stronie klienta (przeglądarki) jest zaimplementowana w języku JavaScript, a odwołania do serwera w celu pobrania potrzebnych danych są realizowane asynchronicznie bez przerywania pracy użytkownika i bez konieczności przeładowania strony w przeglądarce. Stanowiącą podstawę funkcjonowania aplikacji AJAX technika wymiany danych między aplikacją w przeglądarce a serwerem bez konieczności przeładowania całej strony jest określana jako remote scripting.

AJAX nie jest nową technologią, ale nowym sposobem wykorzystania kombinacji istniejących technologii takich jak m.in. JavaScript, DOM i CSS, które w ostatnich latach osiągnęły dojrzałość i uzyskały powszechne wsparcie w przeglądarkach. Termin AJAX zaproponował Jesse James Garrett [6], jako niewątpliwie chwytliwy akronim opisujący technikę tworzenia interaktywnych aplikacji w dużym stopniu polegających na kodzie JavaScript po stronie klienta, realizującym asynchroniczne odwołania do serwera. Bez wątpienia AJAX stanowi rewolucję w sposobie myślenia o tworzeniu aplikacji internetowych, pozwalając na uzyskanie stopnia interaktywności zarezerwowanego dotychczas dla aplikacji desktopowych. Przykładowo, internetowy interfejs serwisu pocztowego GMail udostępnianego przez Google (<http://gmail.google.com>) pozwala na przeglądanie wielu wiadomości jednocześnie, odświeża prezentowaną zawartość skrzynki w trakcie gdy użytkownik edytuje wiadomość, itp.

Celem niniejszego artykułu jest przedstawienie zasady działania aplikacji AJAX, sposobu ich tworzenia, a także omówienie roli poszczególnych technologii składowych. Sposób tworzenia aplikacji AJAX zostanie zilustrowany na przykładzie prostej aplikacji. Przedstawione będą również dwa przykłady rzeczywistych aplikacji AJAX dostępnych w Internecie: Google Suggest i Google Maps,

które zyskały popularność m.in. właśnie dzięki zaoferowaniu stopnia interaktywności niespotykanego wcześniej w aplikacjach internetowych.

2. Technologie składowe AJAX

Podstawowy zestaw technologii wykorzystywany w aplikacjach AJAX to:

- HTML (XHTML) oraz CSS
- JavaScript
- DOM
- XML
- XMLHttpRequest.

HTML (a najlepiej nowsza, zgodna z regułami XML jego wersja - XHTML) [13] oraz arkusze stylów CSS (Cascading Style Sheets) [10] służą do prezentacji danych. Wszelkie formatowanie zawartości dokumentów, zgodnie z zasadami obowiązującymi od wersji HTML 4, powinno być oczywiście realizowane przez reguły CSS. CSS umożliwia zarówno pozycjonowanie elementów na stronie jak i definiowanie ich stylu (koloru, czcionki, itp.). Charakterystyczną cechą dokumentów HTML, które mają być programowo modyfikowane przez towarzyszący im kod JavaScript (co ma miejsce w aplikacjach AJAX), jest wykorzystanie w nich znaczników `<div>` i ``, które służą do strukturalizacji zawartości dokumentu. XHTML jest najnowszą wersją języka HTML, którą można najkrócej opisać jako HTML sformułowany w postaci aplikacji XML. Dokumenty XHTML są łatwiejsze w programowym przetwarzaniu i pielęgnacji niż dokumenty HTML.

Język JavaScript to zorientowany obiektowo język skryptowy ogólnego przeznaczenia, ale przeznaczony głównie do zagnieżdżania w dokumentach HTML. JavaScript został opracowany przez firmę Netscape i został udostępniony w roku 1996 wraz z pojawieniem się przeglądarki Netscape 2. Miał on służyć przede wszystkim do manipulowania po stronie klienta danymi wprowadzanymi do formularzy HTML. JavaScript został ustandaryzowany pod nazwą ECMAScript [5]. Firma Microsoft opracowała dla swojej przeglądarki podobny do JavaScript język skryptowy o nazwie JScript. JavaScript znalazł szereg typowych zastosowań, takich jak weryfikacja poprawności danych wprowadzanych do formularzy, programowe otwieranie okien, graficzne menu, dynamiczne wypełnianie zawartości list rozwijanych, itp. Mimo niewątpliwej użyteczności, był też JavaScript źródłem niekompatybilności aplikacji i problemów ze starszymi przeglądarkami. Obecnie języki JavaScript i JScript można traktować jako alternatywne implementacje ECMAScript i mimo w dalszym ciągu występujących pewnych różnic w zakresie obsługi JavaScript w poszczególnych środowiskach, można z powodzeniem tworzyć w tym języku aplikacje, które będą działać we wszystkich nowoczesnych przeglądarkach.

Trzecim kluczowym elementem AJAX jest obiektowy model dokumentu DOM (Document Object Model) [11]. Specyfikacja DOM określa standardowy sposób udostępnienia obiektowej (hierarchicznej) struktury dokumentu skryptom uruchamianym w środowisku przeglądarki. DOM API jest dostępny m.in. z poziomu języka JavaScript. W starszych wersjach przeglądarek (Netscape 4, Internet Explorer 4) tylko niektóre węzły drzewa dokumentu były dostępne dla kodu JavaScript. Aktualne wersje przeglądarek udostępniają skryptom całe drzewo DOM dokumentu. Korzystając z interfejsu DOM, z poziomu kodu JavaScript można odczytać element wskazując go przez jego identyfikator (`document.getElementById()`) lub nazwę znacznika (`document.getElementsByTagName()`), modyfikować atrybuty elementu (w tym atrybut `style`, zawierający styl CSS elementu), a także dodawać (`document.createElement()`), a następnie dla wskazanego elementu: `appendChild()` i usuwać (`removeChild()`) elementy zmieniając przez to strukturę drzewa.

Powyższe elementy AJAX wykorzystywane były od lat w połączeniu ze sobą stanowiąc tzw. dynamiczny HTML (DHTML) w celu „ożywienia” statycznych stron WWW poprzez animacje i często ciekawe efekty wizualne. Nową jakością cechującą AJAX są tak naprawdę żądania HTTP wysyłane

przez kod JavaScript asynchronicznie do serwera. Po odebraniu odpowiedzi od serwera, aplikacja AJAX modyfikuje wymagające zmiany fragmenty dokumentu za pomocą interfejsu DOM. Najczęściej aplikacje AJAX realizują asynchroniczne odwołania do serwera poprzez obiekt XMLHttpRequest, choć podobną funkcjonalność można osiągnąć w oparciu o zagnieżdżone ramki <iframe>. Historia, interfejs i sposób korzystania z obiektu XMLHttpRequest zostaną omówione szczegółowo w następnym rozdziale.

Ostatnim elementem AJAX jest format przesyłania danych między serwerem i klientem w odpowiedzi na asynchronicznie wysłane żądanie przez kod JavaScript po stronie klienta. Rozwinięcie skrótu AJAX jak i nazwa obiektu wykorzystywanego do komunikacji z serwerem (XMLHttpRequest) są tu nieco mylące, gdyż sugerują iż formatem tym musi być XML. W istocie XML [12] jest wykorzystywany do tego celu najczęściej ze względu na bogactwo wspierających go bibliotek, parserów, itp. Przykładowo, istnieje możliwość wykorzystania transformacji XSLT [14] w celu przetworzenia danych XML pobranych z serwera do postaci fragmentu HTML gotowego do umieszczenia w dokumencie głównym. (Transformacje XSLT są realizowane po stronie klienta w języku JavaScript poprzez obiekt XSLTProcessor, stanowiący część DOM API.) Argumentem nie bez znaczenia jest też w wypadku XML silne wsparcie ze strony dużych korporacji, mających duży wpływ na rozwój technologii informatycznych. Należy jednak podkreślić, że serwer może przysyłać dane nawet czystym tekstem o strukturze ustalonej przez programistę aplikacji. Wadą takiego rozwiązania jest oczywiście konieczność ręcznego oprogramowania przygotowania dokumentu na serwerze oraz jego parsowania po stronie klienta.

Ważną alternatywę dla XML jako formatu przesyłania danych stanowi JSON (JavaScript Object Notation) [7]. Jest to format umożliwiający zapisanie w postaci tekstowej obiektu JavaScript. Zaletą JSON jest łatwość i szybkość jego parsowania po stronie klienta z poziomu kodu JavaScript. Parsowanie sprowadza się do wywołania funkcji `eval()`, przekazując jej jako parametr treść wysłanego przez serwer dokumentu JSON. Wadą JSON jest mniejsze niż w przypadku XML wsparcie przez języki i narzędzia pracujące po stronie serwera oraz fakt, że JSON jest notacją specjalizowaną dla języka JavaScript, a XML jest powszechnie znanym formatem ogólnego przeznaczenia.

3. Obiekt XMLHttpRequest

Spśród technologii i elementów składowych AJAX na szczególne omówienie zasługuje z pewnością obiekt XMLHttpRequest, jako podstawowy mechanizm realizacji kluczowych w AJAX asynchronicznych odwołań do serwera z poziomu aplikacji JavaScript w przeglądarce. Obiekt ten nie jest elementem żadnego oficjalnego standardu, ale ze względu na powszechne wsparcie w przeglądarkach stał się standardem de facto. Jako pierwsza obiekt XMLHttpRequest zaoferowała programistom firma Microsoft w swojej przeglądarce Internet Explorer 5.0 w postaci obiektu ActiveX, dostępnego z poziomu języków skryptowych wspieranych przez przeglądarkę (JScript, VBScript). Wkrótce Mozilla, Apple i Opera Software opracowały swoje implementacje XMLHttpRequest dla swoich przeglądarek. Obecnie można przyjąć, że wszystkie w miarę aktualne wersje przeglądarek udostępniają obiekt XMLHttpRequest, choć sposób jego tworzenia jest odmienny w Internet Explorer niż w pozostałych przeglądarkach. Na szczęście różnice w implementacji XMLHttpRequest kończą się na sposobie tworzenia obiektu i dalsze odwołania do niego mają charakter w pełni przenaszalny. Pewną wadą przeglądarki Internet Explorer jest to, że do korzystania z XMLHttpRequest wymagane w niej jest włączenie oprócz JavaScriptu dodatkowo możliwości uruchamiania komponentów ActiveX. Poniższy fragment kodu przedstawia przykładowy sposób tworzenia obiektu XMLHttpRequest metodą odpowiednią dla wykorzystywanej przeglądarki:

```
if (window.XMLHttpRequest) { // Mozilla, Safari, Opera ...
    requester = new XMLHttpRequest();
} else if (window.ActiveXObject) { // Internet Explorer
    requester = new ActiveXObject("Microsoft.XMLHTTP");
}
```

W celu wysłania żądania HTTP do serwera za pomocą utworzonego obiektu XMLHttpRequest należy kolejno wywołać na jego rzecz dwie metody: `open` i `send`. Metoda `open` inicjalizuje żądanie i wymaga podania jako argumentów metody żądania HTTP (GET, POST, itd.) oraz adresu URL. Metoda `send` aktywuje połączenie i realizuje żądanie. Argumentem `send` w przypadku żądania POST jest łańcuch znaków z zakodowanymi danymi do przesłania, a w przypadku żądania GET `null`. Poniżej przedstawiono przykład wysłania żądania:

```
requester.open("GET", "/skrypt.php");  
requester.send(null);
```

Po wywołaniu metody `send`, obiekt XMLHttpRequest połączy się z serwerem i pobierze z niego dane. Żądanie to będzie zrealizowane w sposób asynchroniczny i czas po którym proces pobierania danych zostanie zakończony nie jest znany. Przypomnijmy, że jest to kluczowa cecha modelu AJAX, dzięki której użytkownik aplikacji po wykonaniu akcji skutkującej odwołaniem do serwera może kontynuować interakcję z aplikacją, nie czekając na zakończenie przesyłania danych przez serwer. Aby aplikacja mogła zareagować na zakończenie pobierania danych z serwera, konieczne jest (przed wywołaniem żądania) zarejestrowanie dla obiektu XMLHttpRequest funkcji nasłuchującej na zmiany stanu jego właściwości o nazwie `readyState`. Zmienna ta informuje o stanie połączenia obiektu z serwerem i może przyjmować następujące wartości liczbowe:

- 0 – uninitialized,
- 1 – loading,
- 2 – loaded,
- 3 – interactive,
- 4 – completed.

Tak naprawdę interesujący jest moment, w którym zmienna `readyState` osiągnie wartość 4, oznaczającą zakończenie obsługi żądania. Ponadto, przed wykorzystaniem danych przesłanych przez serwer, dobrą praktyką jest sprawdzenie statusu HTTP (wartość 200 oznacza pozytywną realizację żądania). Poniżej przedstawiono szkielet funkcji nasłuchującej i sposób jej zarejestrowania:

```
function myHandler()  
{  
    if (requester.readyState == 4)  
    {  
        if (requester.status == 200)  
        {  
            // przetwarzanie przesłanych danych  
        }  
    }  
    return true;  
}
```

```
requester.onreadystatechange = myHandler;
```

Dane pobrane z serwera są dostępne poprzez dwie właściwości obiektu XMLHttpRequest: `responseXML` i `responseText`. Pierwsza z nich zawiera drzewo DOM zawartości XML przesłanej przez serwer, a druga – dane w postaci jednego łańcucha znaków. W przypadku danych przesłanych jako `text/plain` lub `text/html` jedynie właściwość `responseText` zawiera dane. Dla danych `text/xml` `responseText` zawiera alternatywny tekstowy zapis zawartości XML dostępnej w postaci DOM poprzez właściwość `responseXML`.

Obiekt XMLHttpRequest nie jest jedynym sposobem pobierania danych z serwera bez konieczności przeładowywania całej strony. Starszą, ale ciągle niekiedy stosowaną techniką jest wykorzystanie do tego celu niewidocznych zagnieżdżonych ramek `<iframe>` jak w poniższym przykładzie:

```
<iframe id="data"
```

```
name="data"
style="width:0px; height:0px; border: 0px"
src="blank.html"></iframe>

<a href="URL_on_server" target="data">Call the server!</a>
```

Dokument załadowany do zagnieżdżonej ramki w odpowiedzi na kliknięcie łącza może wywołać funkcję JavaScript z dokumentu głównego i w ten sposób zainicjować przetwarzanie danych załadowanych z serwera po zakończeniu pobierania dokumentu. Ramka `<iframe>` ma zerowe wymiary, co powoduje że nie jest widoczna (jawne ustawienie ramki jako niewidocznej za pomocą `style="display: none;"` w niektórych przeglądarkach powoduje, że zawartość ramki w ogóle nie jest pobierana z serwera!).

Obiekt `XMLHttpRequest` jest rozwiązaniem nowszym niż `<iframe>`, a co ważniejsze opracowanym specjalnie do realizacji asynchronicznych żądań HTTP. Wykorzystanie ramek `<iframe>` do tego celu ma raczej charakter „sztuczki” i nie było ich planowanym zastosowaniem. Zalety `XMLHttpRequest` w porównaniu z ramkami `<iframe>` to poza tym m.in.:

- `XMLHttpRequest` najczęściej jest szybszy,
- `XMLHttpRequest` ma „wbudowaną” obsługę XML,
- Załadowanie dokumentu do ramki `<iframe>` jest uwzględniane w historii przeglądarki, co w aplikacji AJAX powoduje nienaturalne z punktu widzenia użytkownika efekty operacji Back i Refresh.

Domyślnym sposobem realizacji asynchronicznego pobierania danych z serwera bez konieczności przeładowania całej strony jest dziś z pewnością obiekt `XMLHttpRequest`. Czynniki które mogą skłonić twórcę aplikacji do wykorzystania niewidocznej ramki `<iframe>` to m.in.:

- Wsparcie dla `<iframe>` w starszych przeglądarkach
- Wymaganie włączenia uruchamiania ActiveX w przeglądarce Internet Explorer w celu umożliwienia skorzystania z `XMLHttpRequest`.

4. Przykład prostej aplikacji AJAX

Pierwszym etapem praktycznego poznawania nowych technologii jest typowo implementacja aplikacji „HelloWorld”. W przypadku aplikacji AJAX tekst do wyświetlenia w przeglądarce będzie pobrany asynchronicznie z serwera w odpowiedzi na żądanie użytkownika, a następnie „wbudowany” w dokument prezentowany w przeglądarce. Poniżej przedstawiono kod dokumentu HTML zawierającego taką właśnie prostą aplikację AJAX:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
  <HEAD>
    <TITLE>AJAX Hello World</TITLE>
    <META HTTP-EQUIV="Content-Type"
      CONTENT="text/html; charset=windows-1250">
    <SCRIPT type='text/javascript'>
      function getData()
      {
        if (window.XMLHttpRequest) {
          requester = new XMLHttpRequest();
        } else if (window.ActiveXObject) {
          requester = new ActiveXObject("Microsoft.XMLHTTP");
        }
        requester.onreadystatechange = myHandler;
```

```

        requester.open("GET", "hello.txt");
        requester.send(null);
    }

    function myHandler()
    {
        if (requester.readyState == 4)
        {
            if (requester.status == 200)
            {
                el = document.getElementById("hello");
                txt =
                    document.createTextNode(requester.responseText);
                el.appendChild(txt);
            }
        }
        return true;
    }
</SCRIPT>
</HEAD>
<BODY>
    <DIV><A href="javascript:getData()">Powitanie</A></DIV>
    <DIV id="hello"></DIV>
</BODY>
</HTML>

```

Kod JavaScript powyższej aplikacji obejmuje funkcję `getData()` do wysłania żądania HTTP do serwera w celu pobrania zawartości pliku tekstowego `hello.txt` oraz funkcję `myHandler()`, która zostanie uruchomiona po zakończeniu pobierania pliku. Działanie funkcji `myHandler()` sprowadza się do umieszczenia za pomocą metod interfejsu DOM tekstu pobranego z serwera we wskazanym identyfikatorze `hello` elemencie `<div>` w ciele dokumentu. Kod odpowiadający za utworzenie i wykorzystywanie obiektu `XMLHttpRequest` zawarty w powyższych dwóch funkcjach został szczegółowo opisany w poprzednim rozdziale. W celu poprawnej obsługi polskich znaków, plik tekstowy zawierający tekst „Witaj świecie!” musiał zostać zakodowany w systemie UTF-8, gdyż takie kodowanie zakłada obiekt `XMLHttpRequest` dla danych, które nie mają postaci XML. Pobieranie danych z serwera jest inicjowane przez użytkownika wybraniem łącza, któremu przypisany został stosowny kod JavaScript. Poniżej przedstawiono efekt działania aplikacji:



Rys. 1. Efekt działania aplikacji „HelloWorld” w AJAX

Kod powyższej aplikacji można nieznacznie uprościć wykorzystując do modyfikacji struktury dokumentu zamiast metod `createTextNode()` i `appendChild()` właściwość `innerHTML` elementów drzewa dokumentu, obsługiwaną przez wszystkie popularne przeglądarki:

```
el.innerHTML+=requester.responseText;
```

Zaletą właściwości `innerHTML` jest to, że umożliwia ona umieszczenie za pomocą jednej instrukcji dowolnie złożonej zawartości HTML we wskazanym elemencie. Zawartość ta może zawie-

rać zagnieżdżone znaczniki HTML. Korzystanie z właściwości `innerHTML` nie jest jednak zalecane w dokumentach XHTML, ze względu na niespójny sposób obsługi tego rozwiązania przez różne przeglądarki.

W kolejnym kroku zmodyfikujemy aplikację tak by pobierała z serwera dane w postaci XML. Serwer w odpowiedzi na asynchroniczne żądanie z poziomu kodu w JavaScript prześle dokument XML o treści podanej poniżej:

```
<?xml version="1.0" encoding="windows-1250" ?>
<powitanie>Witaj świecie!</powitanie>
```

Wartą podkreślenia różnicą w stosunku do przykładu z przesyłaniem danych czystym tekstem jest to, że w wypadku XML obiekt `XMLHttpRequest` akceptuje różne kodowania znaków, biorąc pod uwagę informację o kodowaniu dokumentu zawartą w prologu dokumentu XML. Dostosowanie przykładowej aplikacji do nowego formatu danych przesyłanych przez serwer będzie polegało na wykorzystaniu właściwości `responseXML` obiektu `XMLHttpRequest` zamiast `responseText` oraz wydobyciu z przesłanego dokumentu XML tekstu powitania za pomocą interfejsu DOM. Dodatkowo, ponieważ na razie nasza aplikacja pobiera z serwera statyczny plik, konieczna jest również zmiana adresu URL pobieranego dokumentu. Poniżej przedstawiono zmodyfikowane funkcje `getData()` i `myHandler()` z wyróżnionymi fragmentami, które uległy zmianie:

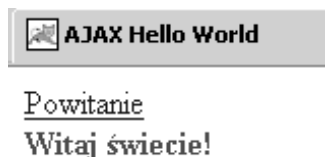
```
function getData()
{
    ...
    requester = new XMLHttpRequest();
    ...
    requester.onreadystatechange = myHandler;
    requester.open("GET", "hello.xml");
    requester.send(null);
}
function myHandler()
{
    if (requester.readyState == 4)
    {
        if (requester.status == 200)
        {
            greeting = requester.responseXML
                .getElementsByTagName("powitanie")[0]
                .firstChild.nodeValue;
            el = document.getElementById("hello");
            txt = document.createTextNode(greeting);
            el.appendChild(txt);
        }
    }
    return true;
}
```

Właściwość `responseXML` udostępnia drzewo DOM pobranego z serwera dokumentu XML. Aby wyłuskać z niego treść powitania należało w pierwszym kroku odnaleźć węzeł `powitanie`. Ponieważ metoda `getElementsByTagName()` wyszukująca węzły o podanej nazwie zwraca tablicę węzłów, konieczne było pobranie z tablicy pierwszego elementu (znamy strukturę dokumentu i wiemy, że zawiera on tylko jeden taki węzeł). Tekst zawarty w danym węźle jest drzewie DOM reprezentowany jako zagnieżdżony węzeł typu tekstowego. Dlatego też w drugim kroku, po znalezieniu węzła `powitanie`, pobrany został z niego węzeł tekstowy (pierwszy i jedyny węzeł potomny), a następnie odczytana wartość tego węzła tekstowego.

Kolejna modyfikacja przykładowej aplikacji polegać będzie na zmianie stylu elementu `div` prezentującego dane pobrane z serwera. Istnieją dwa sposoby zmiany stylu elementu poprzez interfejs

DOM. Pierwszy polega na ustawieniu wartości właściwości `className` węzła, a drugi na bezpośrednim modyfikowaniu właściwości `style`, mającej postać tablicy właściwości CSS. Pierwszy ze sposobów opiera się o arkusze stylów i pozwala na jednoczesne zaaplikowanie do elementu wielu reguł CSS. Drugi sposób jest najczęściej wykorzystywany w połączeniu z pierwszym w celu „dostrojenia” wyglądu elementu poprzez uzupełnienie czy przesłonięcie reguł z arkusza stylów. W naszym przykładzie ograniczymy się wyłącznie do modyfikacji właściwości `style`, aby przykład był możliwie prosty:

```
el.style.color = "red";
el.style.fontWeight = "bold";
```



Rys. 2. Efekt programowej modyfikacji stylu elementu

W dotychczasowych przykładach zawartość pobierana asynchronicznym żądaniem HTTP z serwera miała charakter statyczny tj. serwer przysyłał plik tekstowy lub plik XML odczytany z systemu plików serwera. W rzeczywistych aplikacjach AJAX żądania wysyłane do serwera są sparametryzowane, a zawartość wysyłana w odpowiedzi przez serwer jest generowana przez skrypty, serwlety, itp. Aby zilustrować sposób przesyłania parametrów w asynchronicznych żądaniach HTTP, zmodyfikujemy aplikację przykładową tak, aby tekst pozdrowienia zawierał imię wprowadzone przez użytkownika do formularza. Poniżej przedstawiono zmodyfikowany dokument HTML z wyróżnionymi fragmentami, które uległy zmianie:

```
<!DOCTYPE ...>
<HTML>
  <HEAD>
    ...
    <SCRIPT type='text/javascript'>
      function getData()
      { ...
        requester = new XMLHttpRequest();
        ...
        requester.onreadystatechange = myHandler;
        requester.open("GET", "hello.php?who="
          + document.getElementById("who").value);
        requester.send(null);
      }
      function myHandler()
      {
        ...
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM>
      <INPUT type="text" id="who">
      <INPUT type="button" value="Pozdrów" onclick="getData()" >
    </FORM>
    <DIV id="hello"></DIV>
  </BODY>
```

```
</HTML>
```

Tym razem żądanie do serwera jest wysyłane gdy użytkownik naciśnie przycisk w formularzu. URL żądania wskazuje skrypt PHP. Przed wysłaniem żądania do adresu URL doklejany jest parametr zawierający imię wprowadzone przez użytkownika do pola tekstowego w formularzu. Kod funkcji `myHandler()` przetwarzającej dane przesłane przez serwer nie uległ zmianie. W związku z tym skrypt PHP musi wygenerować zawartość XML w takiej postaci jaką miał we wcześniejszym przykładzie statyczny plik XML. Kod skryptu PHP oraz efekt działania zmodyfikowanej aplikacji przedstawiono poniżej. W kodzie wyróżnione zostały fragmenty odpowiedzialne za poinformowanie przeglądarki o typie przesyłanej zawartości (`text/xml`) oraz za odczyt wartości przesłanego parametru żądania (w naszym przypadku imienia).

```
<?php header("Content-type: text/xml"); ?>
<?xml version="1.0" encoding="windows-1250" ?>
<powitanie>Witaj <?php echo $_GET['who']; ?>!</powitanie>
```

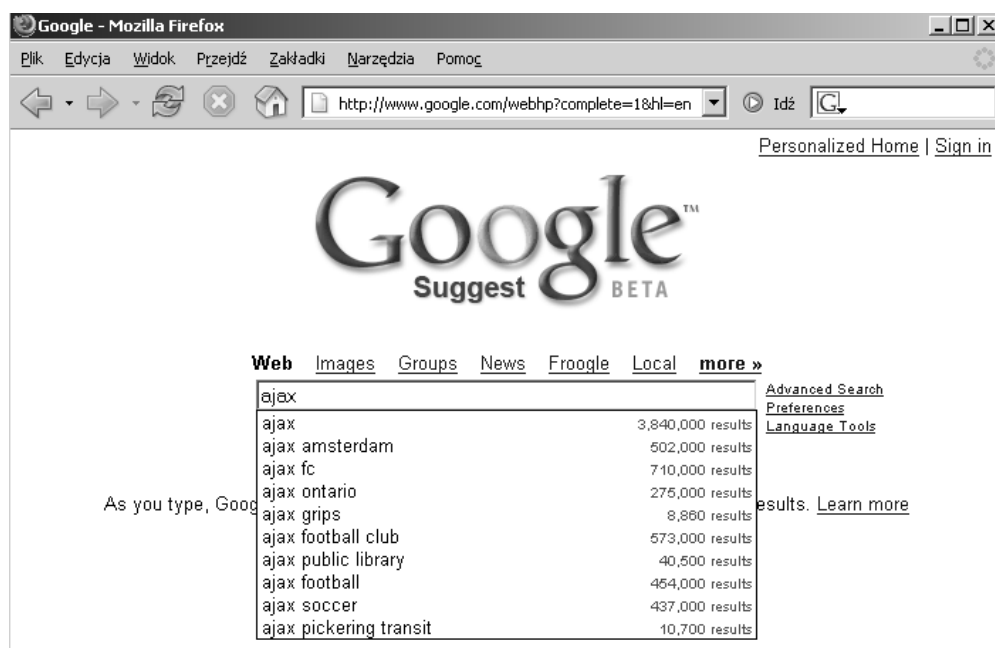


Rys. 3. Efekt działania aplikacji wysyłającej sparametryzowane żądanie do serwera

5. Przykłady istniejących aplikacji AJAX

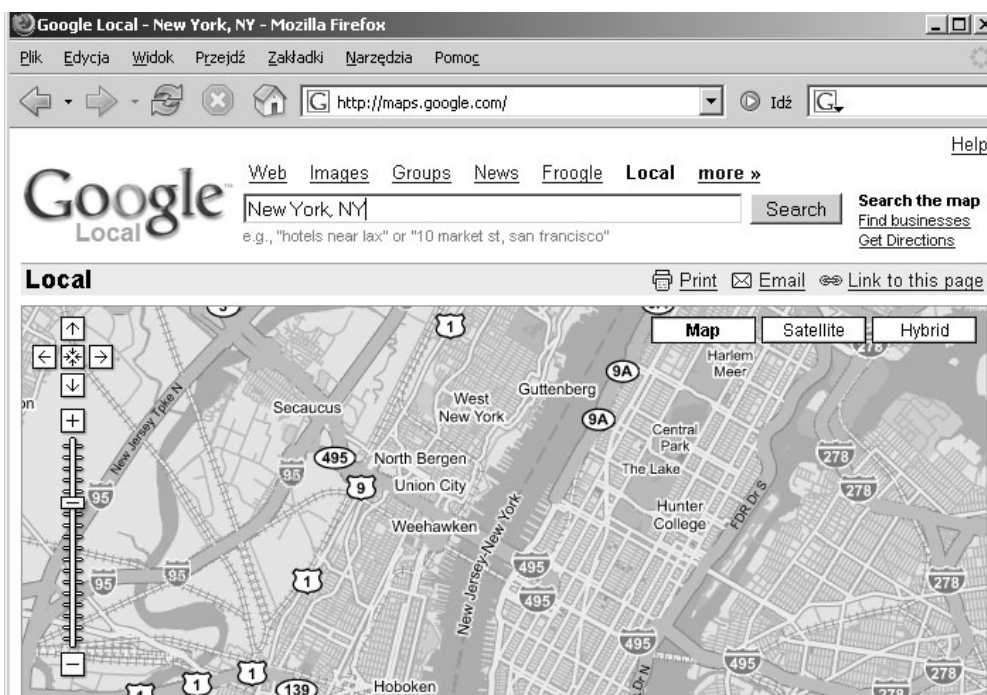
Coraz więcej korporacji m.in. Google, Yahoo i Amazon oferuje serwisy wykorzystujące AJAX. Szczególną rolę w rozwoju i propagowaniu idei AJAX odegrała firma Google, ponieważ to właśnie niespotykana wcześniej interaktywność jej pionierskich serwisów takich jak Google Suggest i Google Maps zwróciła uwagę świata na technikę, dzięki której osiągnięto tak imponujące efekty.

Google Suggest to aplikacja, która wspiera użytkownika w formułowaniu kryteriów wyszukiwania dla wyszukiwarki Google. W trakcie wprowadzania kryteriów przez użytkownika, kod JavaScript odczytuje dotychczas wprowadzone znaki i wysyła je w tle asynchronicznie do serwera. Serwer w oparciu o zgromadzoną historię kryteriów wyszukiwania, odsyła do przeglądarki listę najbardziej popularnych zapytań rozpoczynających się prefiksem wprowadzonym przez użytkownika. Odebrane z serwera dane są przedstawiane użytkownikowi w formie listy rozwijanej.



Rys. 4. Google Suggest w akcji

Google Maps to interaktywna mapa oferująca możliwości pomniejszania i powiększania prezentowanego fragmentu i wyszukiwania obiektów na mapie. Najciekawszym i wykorzystującym AJAX elementem serwisu jest sama mapa. Użytkownik obsługuje mapę na pomocą myszy, „przemieszczając się” w czterech kierunkach, zmieniając skalę oraz przełączając się między trybem mapy i zdjęć satelitarnych. Prezentowany obraz składany jest z kawałków, które w miarę potrzeby są asynchronicznie pobierane z serwera bez konieczności przeładowania całej strony.



Rys. 5. Google Maps w akcji

6. Podsumowanie

AJAX to nowa technika tworzenia aplikacji internetowych oparta o uznane i dojrzałe już technologie takie jak HTML (XHTML), CSS, JavaScript, DOM i XML, wzbogacone o obiekt XMLHttpRequest. Jako rozwiązanie integrujące istniejące rozwiązania, AJAX nie wymaga od programistów uczenia się nowych technologii, ale zmusza ich do zmiany sposobu myślenia o tworzeniu aplikacji internetowych. Twórcy aplikacji zamiast pamiętać o ograniczeniach narzucanych przez architekturę aplikacji internetowych powinni otworzyć się na nowe możliwości jakie oferuje AJAX.

Wielokrotnie wspomnianą w artykule i podstawową zaletą AJAX jest to, że umożliwia on uzyskanie poziomu interaktywności aplikacji dotychczas niedostępnego dla aplikacji internetowych. Nie bez znaczenia jest też ograniczenie ilości danych przesyłanych z serwera do przeglądarki. Dodatkowo AJAX pozwala skrócić czas ładowania się stron aplikacji dzięki możliwości asynchronicznego pobierania z serwera nie tylko danych, ale również kodu JavaScript i reguł stylistycznych CSS dopiero wtedy gdy będą potrzebne. Uzyskanie tego efektu wymaga oczywiście dokładnego przemyślenia sposobu funkcjonowania aplikacji i podziału kodu na części, które muszą być dostępne od początku i te, które można pobrać później. Ważną pozytywną cechą AJAX jest też fakt powszechnego wsparcia jego technologii przez obecne przeglądarki bez konieczności instalowania wtyczek czy bibliotek runtime, w przeciwieństwie do w pewnym stopniu alternatywnych dla AJAX rozwiązań takich jak Java Web Start czy Flash.

AJAX nie jest jednak technologia pozbawiona wad. Z pewnością istotnym mankamentem jest konieczność włączenia obsługi JavaScript w przeglądarce, a w przypadku przeglądarki Internet Explorer również obsługi ActiveX. Innym problemem są w dalszym ciągu jednak występujące opóźnienia w komunikacji sieciowej. Opóźnienia te nie powodują w przypadku AJAX wstrzymania działań użytkownika, ale sprawiają że efekty jego interakcji z aplikacją mogą nie być natychmiastowe. Problem ten można rozwiązać poprzez wyświetlanie na stronie informacji o trwającym pobieraniu danych w przypadku czasochłonnych operacji na serwerze. Na zakończenie należy też przyznać, że implementowanie złożonych aplikacji AJAX jest w chwili obecnej niełatwym zadaniem. Aplikacje AJAX zawierają duże ilości kodu w języku JavaScript, który nie był opracowywany z myślą o dużych aplikacjach. Na szczęście pojawia się coraz więcej wzorców projektowych, bibliotek, szkieletów aplikacji i środowisk IDE (np. najnowsza wersja MyEclipse) wspierających AJAX.

Bibliografia

1. C. Adams, AJAX: Usable Interactivity with Remote Scripting, July 2005, <http://www.sitepoint.com/>
2. Ajax Patterns, <http://ajaxpatterns.org/>
3. R. Asleson, N. T. Schutta, Foundations of Ajax, Apress, 2005
4. D. Crane, E. Pascarello, D. James, Ajax in Action, Manning Publications, 2005
5. ECMA, ECMAScript Language Specification, Standard ECMA-262, 3rd Edition, December 1999
6. J. J. Garrett, Ajax: A New Approach to Web Applications, February 2005
7. JSON (JavaScript Object Notation), <http://www.json.org/>
8. J. Ley, Using the XML HTTP Request object, <http://jibbering.com/2002/4/httprequest.html>, 2002
9. The AJAX Revolution. Join in, <http://www.telerik.com/AJAX/>
10. W3C, Cascading Style Sheets, <http://www.w3.org/Style/CSS/>
11. W3C: Document Object Model (DOM), <http://www.w3.org/DOM/>
12. W3C, Extensible Markup Language (XML), <http://www.w3.org/XML/>
13. W3C, HyperText Markup Language (HTML), <http://www.w3.org/MarkUp/>
14. W3C, XSL Transformations (XSLT), <http://www.w3.org/TR/xslt>