

AJAX und Comet – Web in den Versionen 2.0 und 3.0

The browser is really not much better than a 3270. Recently, folks have been trying to circumvent this simplicity by making browser-based applications more interactive using technologies such as Ajax.

To my mind, this is just a stop-gap until we throw the browser away altogether - **Ajax is just lipstick on a pig.**

Dave Thomas (pragmaticprogrammer.com)



puremedia
Reinsburgstr. 37
70178 Stuttgart

t 0700 / 0078 0079
m 0179 / 90 16 016
f 0700 / 0078 0079

info@puremedia-online.de
www.puremedia-online.de

Internet-Agentur

AJAX / JavaScript

Ruby on Rails

Typo3 / Extension-Programmierung

Kunden



DAIMLERCHRYSLER

Inhalte

- Wann und warum AJAX
- Einkaufszettel für AJAX
- Das X in AJAX
- JavaScript als Sprache
- Anwendungsarchitektur
- Unit-Testing in Javascript
- AJAX aus wirtschaftlicher Sicht
- Erfahrungen aus 4.000 Zeilen AJAX-FatClient
- Comet

Wann und warum AJAX

rich User Interface

- “neue” Interaktionskonzepte (Drag'n'Drop)
- weniger Wartezeit / reaktionsstarke UIs
- weiterarbeiten, während im Hintergrund Elemente aktualisiert werden
- desktop-like

Wann und warum AJAX

Websites vs. Webapplikationen

- früher: Präsentation von Inhalten
 - „Click'n'Wait“
 - Unternehmenspräsentationen, Nachrichten...
- heute: Arbeitsoberfläche / Anwendung
 - „Drag'n'Drop“
 - Google Spreadsheet, Yahoo!Mail, Writely, Flickr...
- klassisches „Desktop-Land“ wurde erobert

Einkaufszettel für AJAX

1. Usability-/Interaction-Designer

Wer macht Usability-/Interaction-Design?

Grafiker???

BE-Entwickler???

Einkaufszettel für AJAX

Hauptsache, einer macht's...

Einkaufszettel für AJAX

2. FE-Entwickler

HTML +

Einkaufszettel für AJAX

2. FE-Entwickler

HTML + JavaScript

Einkaufszettel für AJAX

2. FE-Entwickler

HTML + JavaScript + Architektur-Verständnis

Einkaufszettel für AJAX

2. FE-Entwickler

HTML + JavaScript + Architektur-Verständnis + BE-Sprache

Einkaufszettel für AJAX

2. FE-Entwickler

HTML + JavaScript + Architektur-Verständnis + BE-Sprache

= gar nicht so einfach...

3. gute Libraries

- gute Neuigkeiten: „Das Rad wurde bereits erfunden!!!“
- Prototype (www.prototypejs.org)
- Scriptaculous (www.script.aculo.us)
- Dojo (www.dojotoolkit.org)
- Rico (www.openrico.org)
- jQuery (www.jquery.com)

4. professionelle Tools

- Libraries
- IDE, die JavaScript spricht
- Firebug (Alternativen: Venkman, IE DOM Explorer)

5. KEINE hohen Systemanforderungen

- XMLHttpRequest in jedem halbwegs modernen Browser
- Libraries machen Cross-Browser-Apps einfach (naja... fast...)
- “Degradable AJAX“:

Website läuft auch ohne JavaScript (Fragen: Zielgruppe? Anwendung?)
Webapplikation nicht (GoogleMaps ohne JavaScript????)

Das X in AJAX

...steht eigentlich für XML

Das X in AJAX

...steht eigentlich für XML

...kann man aber gerne weglassen („AJA“ klingt aber nicht so gut)

Das X in AJAX

...steht eigentlich für XML

...kann man aber gerne weglassen („AJA“ klingt aber nicht so gut)

...und durch etwas “Besseres” ersetzt werden: JSON

Das X in AJAX

JSON: reines JS-Datenformat

```
{"addresses": [
  {"email": "tg@puremedia-online.de", "name": "Tobias G\u00f6nther"},
  {"email": "test@test.de", "name": "Test"} ],
"status": {"code": "OK"}}
```

```
<response>
  <addresses>
    <address>
      <email>tg@puremedia-online.de</email>
      <name>Tobias G\u00f6nther</name>
    </address>
    <address>
      <email>test@test.de</email>
      <name>Test</name>
    </address>
  </addresses>
  <status>
    <code>OK</code>
  </status>
</response>
```

Das X in AJAX

- JSON ist weniger “verbose” als XML ==> weniger Daten
- JSON wird deutlich schneller geparkt als XML
- weniger / übersichtlicherer Code durch vereinfachtes Parsing

“Mit JavaScript macht man doch diese Laufschriften, oder?”

JavaScript als Sprache

- “JavaScript ist eine professionelle, ausgewachsene Programmier-Sprache”
(bitte 10 Mal laut wiederholen)
- Objektorientierung (Prototypen-basiert)
- UnitTesting
- Debugging
- Prototype-Library macht JS rund

Anwendungsarchitektur

- Service-orientierte Architekturen (SOA) + AJAX = sehr gutes Team
- Hauptmerkmal eines Webservice für AJAX:
 - fein-strukturierte Response (XML, JSON, kurzer String, HTML-Snippet)
 - NICHT: komplette HTML-Seite

Zum Beispiel:

- REST
- RPC...

Stichwort: Trennung Datenhaltung/Business-Logik <===> Präsentation

Bis hin zu... AJAX-Fat-Clients:

- Client (Browser) nimmt dem Server soviel Logik wie möglich ab
- Server wird nur dann einbezogen...
 - wenn der Client nicht weiter weiß
 - wenn der Client Daten braucht
- Jeder macht das, was er am Besten kann:
 - Server: Datenhaltung
 - Client: Datenrepräsentation
- Vorteile:
 - desktop-like UIs, sehr reaktionsschnell
 - Last wird auf den Client verlagert
- Beispiele: GoogleSpreadsheet, YahooMail!...

"JavaScript schön und gut - aber
man kann es nicht testen!"

“JavaScript muss man doch nicht testen...”

UnitTesting in JavaScript

Ausreden.

UnitTesting in JavaScript

```
new Test.Unit.Runner({

  setup: function(){
    datepicker = new G1X.DatePicker(['year_field', 'month_field', 'day_field'], ['hour_field', 'minute_fiel
    calendar = new G1X.Calendar(['year_field', 'month_field', 'day_field', 'hour_field', 'minute_field'], {ye
  },

  teardown: function(){
    //datepicker = null;
    //calendar = null;
  },

  test_calendar_namespace_should_exist: function() { with(this) {
    assertInstanceOf(Object, G1X);
  }},

  test_should_display_hide_datepicker_on_focus_of_year_month_day_fields: function() { with(this){
    assertNotVisible('date_picker');
    $('year_field').focus();
    assertVisible('date_picker');
    $('year_field').blur();
    wait(200, function(){assertNotVisible('date_picker')});
  }},

  test_should_change_datepicker_highlight_on_yearmonth_change: function() { with(this){
    $('year_field').focus();
    assertVisible('date_picker');
  }},
```

UnitTesting in JavaScript

Test of G1X calendar.js

8 tests, 17 assertions, 4 failures, 0 errors

Status	Test	Message
passed	test_calendar_namespace_should_exist	1 assertions, 0 failures, 0 errors
passed	test_creation_of_calendar_and_datepicker	2 assertions, 0 failures, 0 errors
passed	test_should_display_hide_datepicker_on_focus_of_year_month_day_fields	8 assertions, 0 failures, 0 errors
passed	test_should_change_datepicker_highlight_on_yearmonth_change	4 assertions, 0 failures, 0 errors
passed	test_should_highlight_selected_month_in_yearmonth_cal_on_yearmonth_change	0 assertions, 0 failures, 0 errors
failed	test_should_display_day_hour_minute_on_select_change	1 assertions, 2 failures, 0 errors Failure: assertNotEqual: got "" Failure: assertMatch : regex: "<td class=\"selected\">3</td>" did not match: ""
failed	test_should_not_select_day_if_in_the_past	0 assertions, 1 failures, 0 errors Failure: assertMatch : regex: "<td class=\"past\">1</td>" did not match: ""
failed	test_should_change_select_field_value_on_click_on_day_table	1 assertions, 1 failures, 0 errors Failure: assertMatch : regex: "<td class=\"selected\">3</td>" did not match: ""

Year: Month: Day: Minute: Minute:

« January 2007 »

Sun Mon Tue Wed Thu Fri Sat

1 2 3 4 5 6

AJAX aus wirtschaftlicher Sicht

Wo kann AJAX Kosten sparen?

- Wartezeiten
- Zeit zur Aufgabenerledigung
- Bandbreite / technische Infrastruktur
- vertraute UIs (Desktop)
- Browser-basierte Software (Wartung/Installation, Lizenzen, Support)

AJAX aus wirtschaftlicher Sicht

Vergleich AJAX – klassisch

- gleiche Anwendung (mit und ohne AJAX)
- Transaktions-Verwaltungs-Tool (Kunden und Transaktionen)

(Quelle <http://www.developer.com/xml/article.php/3554271>)

AJAX aus wirtschaftlicher Sicht

Vergleich AJAX – klassisch

	Traditionell	AJAX	Performance +
Datenvolumen (bytes)	1.737.607	460.799	73,00%
Zeit (Sek.)	115	78	32,00%

Kostenreduzierung = Arbeitskosten pro Stunde \times (Gesparte Sekunden pro Transaktion \times Anzahl Transaktionen pro Jahr) / 3600

Bei einer Zeitersparnis von 36 Sek. pro Transaktion, 50.000 dieser Transaktionen pro Jahr und angenommenen €20/Std.

==> Ersparnis von €10,000 oder 500 Std. pro Jahr (nur für diese Transaktion!)

Erfahrungen aus 4.000 Zeilen AJAX-FatClient

Browser-Side-Cache

- JavaScript-eigener Caching-Mechanismus
- sehr flexibel und individuell
- z.B.: JS-Objekt, das Query-URLs und Return-Werte hält

Erfahrungen aus 4.000 Zeilen AJAX-FatClient

Event-Handler

- erhältlich in den Editionen

„DOM0“ JS: meinLink.onclick = meine_funktion

HTML:

„W3C“ JS: addEventListener / attachEvent

wenn mögl. W3C --> unterstützt mehrere Listener gleichzeitig

- Memory Leaks

- Keyboard-Events

Erfahrungen aus 4.000 Zeilen AJAX-FatClient

Objektorientiertes JavaScript

- macht früher Sinn als man glaubt...
- bei großen Applikationen sehr zu empfehlen

Erfahrungen aus 4.000 Zeilen AJAX-FatClient

Naming Conventions

- Namensräume auch in JS
- HTML-IDs sind das A und O
 - macht unabhängiger von strukturellen HTML-Änderungen
 - angenehmer / sicherer als Child/Parent-Traversal des DOM
- Funktions-Parameter als Objekte

```
// Beim Methoden-Aufruf:
```

```
test_function('Wert Param1', { option1: 'wert1', option2: 'wert2' });
```

```
// In der Methode:
```

```
test_function: function(param1, options){
```

```
    var parameter1 = param1;
```

```
    var option1 = options.param1;
```

```
    var option2 = options.param2;
```

```
};
```

Erfahrungen aus 4.000 Zeilen AJAX-FatClient

Performance-tuning außerhalb des Codes

- wenige große Files statt vieler kleiner
- Minimizer
- JS on-demand laden

"Was kommt nach AJAX?"

"Wie können Web-Anwendungen
noch aktuellere Daten anzeigen?"

AJAX = reaktionsstarkes UI für EINEN User

Probleme

- veraltete Daten / UI
- aktuelle Daten müssen explizit angefragt werden, durch:
 - user action
 - polling

Comet

Preisfrage:

Woher soll der Client wissen, dass neue Daten da sind?

Comet

Antwort:

Kann er nicht wissen.

Ach... wenn uns doch nur der
Server sagen könnte, wenn er neue
Daten hat...

Comet

AJAX = pull

Comet = push

Einsatzbeispiele:

StockQuote Demo: <http://app.lightstreamer.com/DojoDemo/>

Google Image Labeler: <http://images.google.com/imagelabeler/>

Chat: <http://www.chabotc.nl:2001/chat.html>

Anslive: http://www.refwell.com/main/anslive_video_tour

Comet

Comet = reaktionsstarkes UI für ALLE User einer Applikation

- push statt pull
- low latency
- low bandwidth
- UI veraltet nicht

Comet

Einsatzbereiche:

- multi-user (collaborative) Systeme
- Applikationen mit sehr aktuellen u. veränderlichen Daten

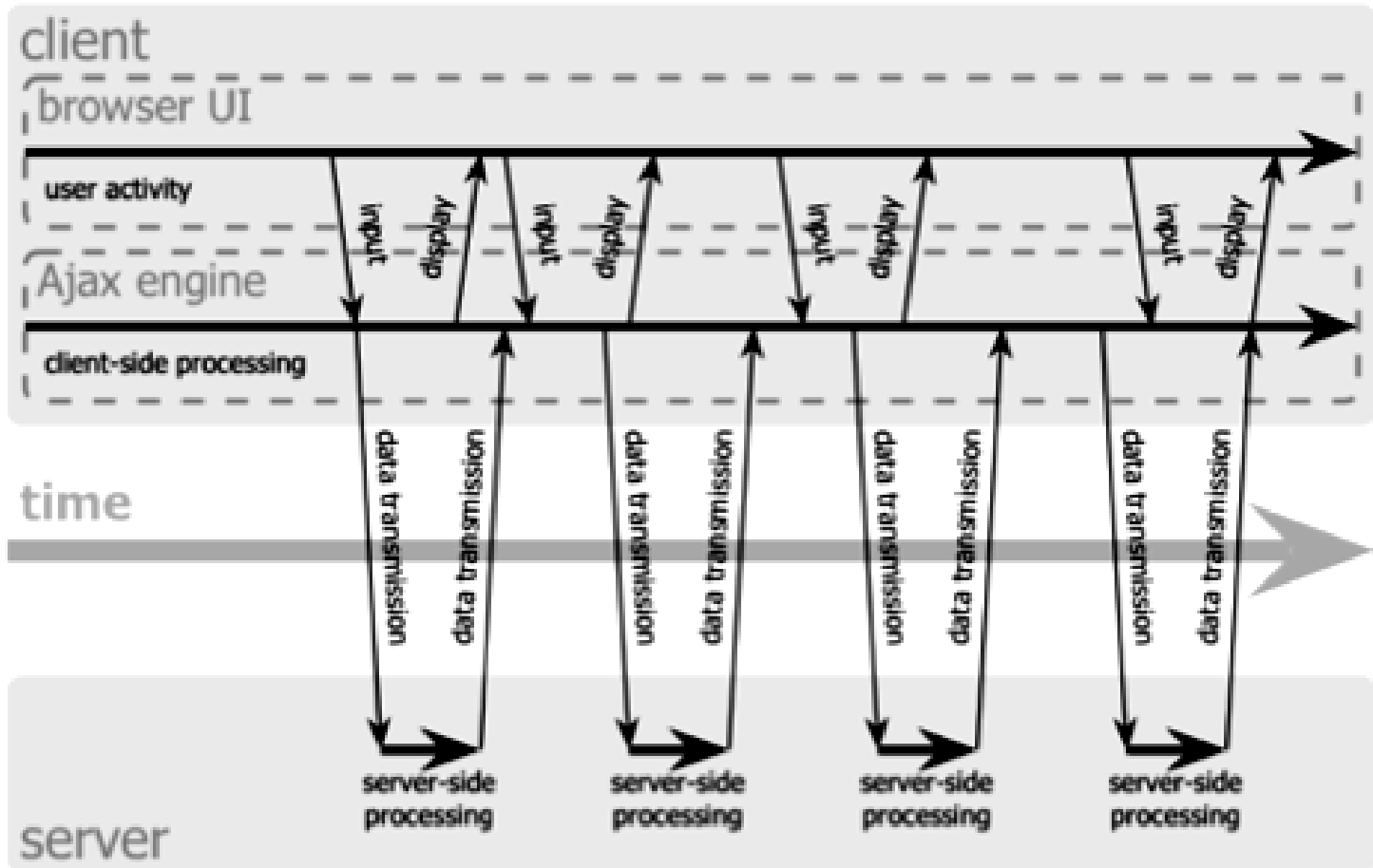
Comet

Wie funktioniert Comet?

Kerngedanke: "long-lived HTTP-Connection"

Wie funktioniert AJAX?

Ajax web application model (asynchronous)



Schwierigkeiten bei Comet

- [Begriff "Comet": Konzept / Paradigma, nicht: konkrete Implementierung!]
- Ports / Firewalls (z.B. Juggernaut über Flash-Socket)
- Proxies (schließen Connections teilw. von selbst...)
- Web-Server (haben Probleme mit long-lived connections)

Stand der Dinge

- viele Webserver kommen noch nicht mit long-lived connections klar
- vielversprechendes Projekt: „Cometd“ (Alex Russell, Dojo Foundation)

"Ist Comet das Web 3.0 ?"

Antwort & Gegenfrage...

Antwort & Gegenfrage...

Antwort: für einige wenige Projekte ja – für das gesamte Web nicht

Antwort & Gegenfrage...

Gegenfrage: geht es vielleicht auch mit AJAX ?



puremedia
Reinsburgstr. 37
70178 Stuttgart

t 0700 / 0078 0079
m 0179 / 90 16 016
f 0700 / 0078 0079

info@puremedia-online.de
www.puremedia-online.de