assono

Ajax in Domino-Web-Anwendungen

22. Februar 2006

Thomas Bahn
assono IT-Consulting & Solutions
tbahn@assono.de
http://www.assono.de
+49/4307/900-401

assono

Agenda

- Motivation
  - Arten von Anwendungen
  - Lücke und Lösung
  - Neue Möglichkeiten
- Grundlagen
  - Was heißt Ajax?
  - XMLHttpRequest
  - Historie
- Praxis in fünf Schritten

Agenda (forts.)

- Demos
  - 1. Institut holen (mit Agenten)
  - 2. Bankleitzahlen vorschlagen (mit Agenten)
  - 3. Postleitzahlen vorschlagen (mit ?ReadViewEntries)
  - 4. Ort holen (mit ?ReadViewEntries)
  - 5. Orte vorschlagen (mit ?ReadViewEntries)
  - 6. Abhängige Auswahllisten nachladen
  - 7. Eindeutigkeit einer ID prüfen
  - 8. Fortschrittsanzeige
  - 9. In-View-Edit
  - 10. JavaScript-Fehler auf Server protokollieren

Agenda (forts.)

- Herausforderungen
- Alternativen
  - Transport
  - Datenformat
- Werkzeuge
  - Arten von Frameworks
  - Frameworks für Domino-Web-Anwendungen
  - Clientseitige Frameworks
  - Entwicklungshilfen
  - Quellen

Organisatorisches

- Wer bin ich?
  - Thomas Bahn, IT-Berater, Dipl.-Math., 35 Jahre, verheiratet
  - Mitgründer und -inhaber der assono IT-Consulting & Solutions
  - seit 1997 entwickle ich mit Java und RDBMS, z. B. Oracle (OCP)
  - seit 1999 mit IBM Lotus Notes/Domino (PCLP Designer R4 - 6.5)
  - Mein Schwerpunkt liegt auf Anwendungen mit Schnittstellen zu anderen Systemen, z. B. RDBMS, SAP R/3, und interaktiven Web-Anwendungen
- bitte zum Schluss Bewertungsbögen ausfüllen
- Zwischenfragen erwünscht!

Motivation – Arten von Anwendungen

- Desktop-Anwendungen
  - lokal installiert, konfiguriert, administriert
    ⇒ geringe Reichweite
  - komfortable Ein-/Ausgabe-Elemente, schnelle Reaktionen
    ⇒ reichhaltige Benutzerschnittstelle

- Web-Anwendungen
  - zentral installiert, konfiguriert, administriert, stets aktuell
    ⇒ hohe Reichweite
  - einfachere Benutzerschnittstelle, immer kompletter Bildaufbau, Reaktion bei nächstem Request-Response-Zyklus (wenn Daten vom Server benötigt werden)
    ⇒ „ärmere" Benutzerschnittstelle

Motivation – Arten von Anwendungen (forts.)

- Mit JavaScript, CSS und Manipulationen am DOM kann man die Benutzerschnittstelle von Web-Anwendungen komfortabler gestalten:
  - Validierungen auf den eingegebenen Daten (und den ausgelieferten, z. B. in versteckten Feldern)
  - Elemente verstecken und nur bei Bedarf sichtbar machen
  - durch geschickte Kombination auch komplexere Ein- und Ausgabe-Elemente, z. B. Dialogliste mit neuen Werten

Motivation – Lücke und Lösung

- **Aber**:
  - Es gibt keine Möglichkeit, dabei auf Server-Daten zuzugreifen
  - Neue Informationen erfordern einen Server-Roundtrip und die Seite wird im Browser komplett neu aufgebaut (und flackert)

- Mit Ajax
  - können – während eine Seite im Browser dargestellt wird – neue Informationen vom Server geholt werden.
  - Diese können mit DOM-Manipulationen in die dargestellte Seite eingebaut werden, ohne dass sie neu aufgebaut werden muss
    ⇒ kein Flackern

Motivation – Neue Möglichkeiten

- Wozu braucht man das?
  - Master-Detail-Beziehungen, z. B.
    - mehrstufige, abhängige Auswahllisten
    - Vorschaufenster für Dokumente in Ansichten
    - Navigation in großen Bäumen
      (wie in kategorisierten Ansichten)
  - Validierungen auf großen Datenmengen, z. B.
    - gültige Postleitzahl, gültige Bankleitzahl
    - eindeutiger Benutzername, eindeutige ID, Seriennummern
      (Daten bleiben auf dem Server)
  - Suche nach freien Benutzerkennungen

Motivation – Neue Möglichkeiten (forts.)

- Wozu braucht man das? (forts.)
  - automatische Eingabe-Vervollständigung oder Vorschläge (wie bei Google Suggest)
  - automatische Aktualisierung, z. B. aktuell angemeldete Benutzer in Online-Chats
  - Aufruf von Web Services (über Proxy, s. a. Herausforderungen)
  - Vorladen von vielleicht benötigten Daten
  - ...

assono

Grundlagen – Was heißt Ajax?

- Ajax steht für „**A**synchronous **J**ava**S**cript and **X**ML“
- Es werden mit JavaScript Anfragen an einen Server geschickt.
- Nach dem Versenden der Anfrage kann der Benutzer weiterarbeiten (aber auch synchrone Anfragen möglich).
- Schickt der Server zu einem späteren Zeitpunkt eine Antwort, so wird eine JavaScript-Prozedur aufgerufen, die auf die Antwort reagieren kann, indem sie die angezeigte Seite verändert.
- Anfragen und Antworten können in XML kodiert sein.

Grundlagen – XMLHttpRequest

- Spricht man heute von Ajax, meint man fast immer einen
  XMLHttpRequest
- Das ist eine Klasse zum Senden von HTTP-Request.
  Sie existiert als JavaScript-Klasse in vielen aktuellen Browsern, z. B. Mozilla 1.0+, Firefox 1.0+, Netscape 7.0+, Safari 1.2+, Opera 7.6+, Konqueror 3.3+
- Im Internet Explorer gibt es XMLHttpRequest als ActiveX-Objekt seit Version 5.0; ab Version 7.0 auch als JavaScript-Klasse
- XMLHttpRequest ist leider (noch) nicht standardisiert, aber lässt sich auf den o. a. Browsern weitgehend gleich nutzen.

Grundlagen – Historie

- Microsoft hat XMLHttpRequest mit dem IE 5.0 eingeführt.
  Es entstand im Rahmen der Entwicklung von Outlook Web Access.

- Es wurde dann vom Mozilla-Projekt (und später anderen Browsern) übernommen und als JavaScript-Klasse nachimplementiert.

- Der Begriff „Ajax" wurde im Februar 2005 durch ein Essay von Jesse James Garret von Adaptive Path geprägt: „Ajax: A New Approach to Web Applications"
  (http://www.adaptivepath.com/publications/essays/archives/000385.php)

Praxis

- 5 Schritte
  - XMLHttpRequest-Objekt erstellen
  - Verbindung „öffnen“
  - Request vorbereiten
  - Request absenden
  - Rückgabe behandeln

Praxis – 1. Schritt

- XMLHttpRequest-Objekt erstellen
  - alle außer Internet Explorer

```
var request = new XMLHttpRequest();
```

  - Internet-Explorer

```
var request = false;
try {
   request = new ActiveXObject('Msxml2.XMLHTTP');
} catch (e) {
   try {
       request = new ActiveXObject('Microsoft.XMLHTTP');
   } catch (e2) {}
}
```

Praxis – 1. Schritt (forts.)

- browserneutral

```
var xmlHttp = false;
/*@cc_on @*/
/*@if (@_jscript_version >= 5)
try {
    xmlHttp = new ActiveXObject('Msxml2.XMLHTTP');
} catch (e) {
    try {
          xmlHttp = new ActiveXObject('Microsoft.XMLHTTP');
    } catch (e2) {
          xmlHttp = false;
    }
}
@end @*/
if (!xmlHttp && typeof XMLHttpRequest != 'undefined') {
    xmlHttp = new XMLHttpRequest();
}
```

Praxis – 1. Schritt (forts.)

- oder XMLHttpRequest im IE nachbilden

```
if (!window.XMLHttpRequest && window.ActiveXObject) {
   window.XMLHttpRequest = function() {
        // erst die neueren, dann die älteren Versionen probieren
        var versions = ['MSXML2.XMLHTTP.7.0', 'MSXML2.XMLHTTP.6.0',
                'MSXML2.XMLHTTP.5.0', 'MSXML2.XMLHTTP.4.0',
                'MSXML2.XMLHTTP.3.0', 'MSXML2.XMLHTTP',
                'Microsoft.XMLHTTP'];
        for (var i = 0; i < versions.length; i++) {
                try {
                        return new ActiveXObject(versions[i]);
                } catch(e) {}
        }
        return undefined;
   }
}
```

Praxis – 2. Schritt

- Verbindung „öffnen“ mit

```
request.open(request-typ, url, async, benutzer, passwort);
```

  - *request-typ*: GET, POST, HEAD oder andere HTTP-Request-Methode
  - *url*: aufzurufende URL, ggf. mit Parametern (GET) auch SSL (https://...) möglich
  - *async* (optional): soll der Aufruf asynchron erfolgen? Vorgabe ist `true`, trotzdem immer angeben
  - *benutzer* (optional): Benutzername auf dem Server
  - *passwort* (optional): Passwort des Benutzers

Praxis – 2. Schritt (forts.)

- POST verwenden, wenn Server-Status oder Daten verändert werden, sonst GET
- open() verursacht noch keine Netzwerk-Operation, die Verbindung zum wird noch nicht wirklich geöffnet
- Beispiel:

```
var url = '/myDB.nsf/myAgent?OpenAgent&Suche=' + escape(suche);
   request.open('GET', url, true);
```

Praxis – 3. Schritt

- Request vorbereiten
  - Callback-Methode festlegen (für asynchronen Aufruf)

```
request.onreadystatechange = callbackMethode;
```

    - *callbackMethode*: parameterlose JavaScript-Funktion
  - ggf. Request-Header mitgeben (POST-Request erfordert festen MIME-Typ)

```
request.setRequestHeader('Content-Type',
   'application/x-www-form-urlencoded');
```

  - gegen das Caching hilft (meistens)

```
request.setRequestHeader('Cache-Control', 'no-cache');
```

Praxis – 4. Schritt

- Request absenden

```
request.send(body);
```

  - *body*:
    - bei GET: null
    - bei POST: durch & getrennte Name=Wert-Paare

  - Beispiel:

```
request.send('Passwort=' + escape(passwort));
```

Praxis – 5. Schritt

- Rückgabe behandeln
  - Callback-Methode wird mehrmals aufgerufen, nämlich einmal für jeden Statuswechsel. `request.readystate` ist
    - 0 (`UNINITIALIZED`): open() wurde noch nicht aufgerufen.
    - 1 (`LOADING`): send() wurde noch nicht aufgerufen.
    - 2 (`LOADED`): send() wurde aufgerufen, Header und Status sind verfügbar.
    - 3 (`INTERACTIVE`): Die Antwort wird gerade heruntergeladen, `response.responseText` enthält die bereits empfangenen Daten.
    - 4 (`COMPLETED`): Alle Operationen sind abgeschlossen.

Praxis – 5. Schritt (forts.)

- Rückgabe behandeln (forts.)
  - HTTP-Status des Requests (`request.status`) prüfen:
    - 200 OK
    - 400 Bad Request
    - 401 Unauthorized
    - 403 Forbidden
    - 404 Not Found
    - 500 Internal Server Error
    - u. v. m.
  - Der Status des Request als Text steht in `request.statusText`.

Praxis – 5. Schritt (forts.)

- Rückgabe behandeln (forts.)
  - Redirection-Responses (Status 301 oder 302) werden transparent verfolgt und die Antwort von der neuen URL geholt.
  - Die Rückgabe steht in `request.responseText` als Text.
  - Wurde vom Server der MIME-Typ der Rückgabe auf `"text/xml"` gesetzt (und ist sie ein gültiges XML-Dokument), dann enthält `request.responseXML` ihre XML-Repräsentation.

Demos – 1. Institut holen

- Nach Eingabe einer Bankleitzahl das Institut holen
- Bankleitzahl.onchange:

```
var url = '/' + $('WebDBName_').value +
   '/HoleInstitut?OpenAgent&BLZ=' + escape($('Bankleitzahl').value);
var statusOKHandler = createSetFieldValueHandler('Institut');
callServer('GET', url, false, 0, statusOKHandler); // synchron
```

- Agent sucht passendes Dokument und gibt Institut zurück

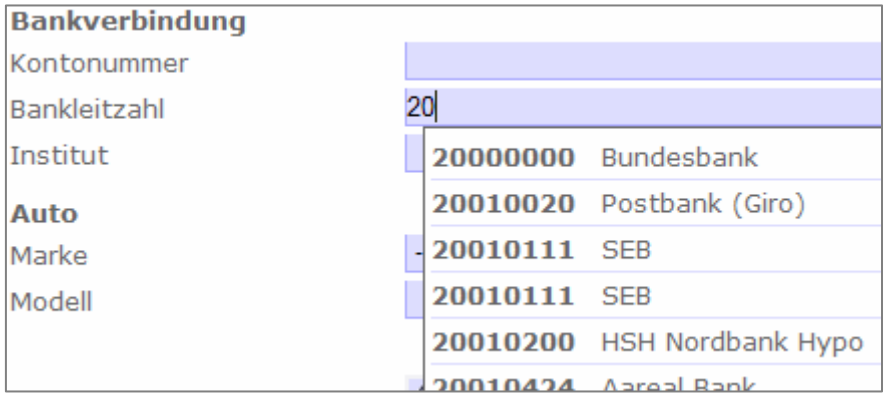
Demos – 2. Bankleitzahlen vorschlagen

- Nach Teileingabe passende Bankleitzahlen vorschlagen

- Bankleitzahl.onkeyup:

```
var url = '/'+$('WebDBName_').value+'/SchlageBankleitzahlenVor?OpenAgent&BLZ='
   + escape($('Bankleitzahl').value);
callServer('GET', url, true, 0,
   createCallFunctionWithTextHandler('fillAndShowBLZVorschlaege'));
```

- Agent sucht passende BLZen und gibt Liste zurück
- Rückgabe in JSON
- JavaScript baut daraus mit Event-Handlern

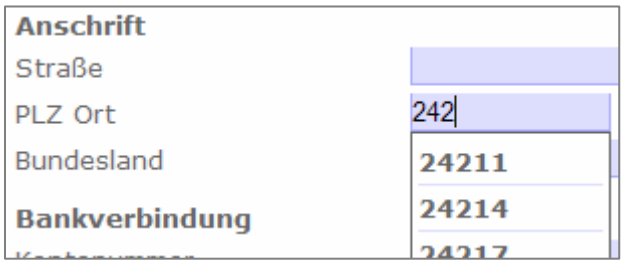


Demos – 3. Postleitzahlen vorschlagen

- Nach Teileingabe passende Postleitzahlen vorschlagen

- PLZ.onkeyup:

```
aktuellePLZ = $('PLZ').value;
if (aktuellePLZ == '') {
   emptyPLZVorschlaege();
   hidePLZVorschlaege();
} else { ... }
```



Demos – 3. Postleitzahlen vorschlagen (forts.)

- PLZ.onkeyup (forts.):

```
...
} else { // hole Vorschlagsliste vom Server
   var letzteStelle = aktuellePLZ.charCodeAt(aktuellePLZ.length - 1);
   letzteStelle = String.fromCharCode(letzteStelle + 1);
   var naechstePLZ = aktuellePLZ.substr(0, aktuellePLZ.length - 1) +
        letzteStelle;
   var url = '/' + $('WebDBName_').value + '/PostleitzahlenNachPLZ, +
        '?ReadViewEntries&CollapseView&Count=10&StartKey=' +
        escape(aktuellePLZ) + '&UntilKey=' + escape(naechstePLZ);
   callServer('GET', url, true, 0,
        createCallFunctionWithXMLHandler('fillAndShowPLZVorschlaege'));
}
```

- ohne Agent allein mit ?ReadViewEntries-URL!

Demos – 4. Ort holen

- Nach PLZ-Eingabe (ersten) passenden Ort holen

- PLZ.onchange:

```
var url = '/' + $('WebDBName_').value +
   '/PostleitzahlenNachPLZ?ReadViewEntries&RestrictToCategory=' +
   escape($('PLZ').value);
callServer('GET', url, false, 0, getErstenOrtMitPLZ); // synchron
```

- ebenfalls mit ?ReadViewEntries

Demos – 5. Orte vorschlagen

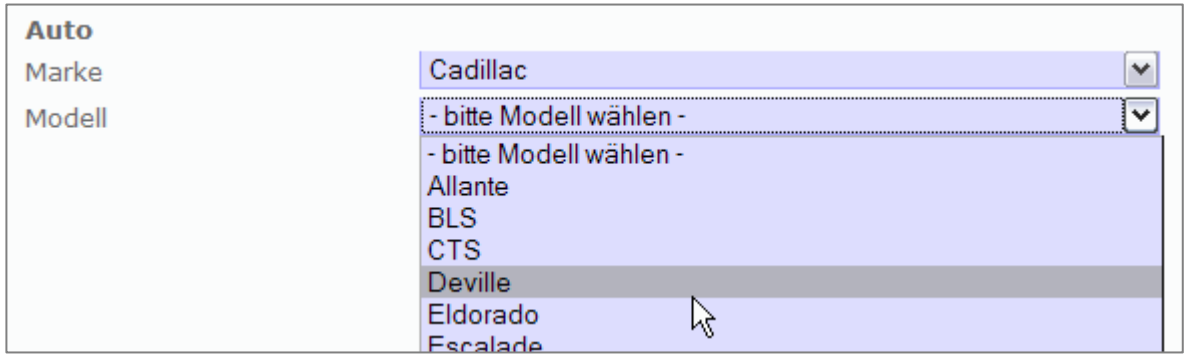
- Nach PLZ-Eingabe (ersten) passenden Ort holen

- Ort.onkeyup:

```
var url = '/' + $('WebDBName_').value + '/PostleitzahlenNachPLZ' +
   '?ReadViewEntries&RestrictToCategory=' + escape(aktuellePLZ);
callServer('GET', url, true, 0,
   createCallFunctionWithXMLHandler('fillAndShowOrtVorschlaege'));
```

Demos – 6. Abhängige Auswahllisten nachladen

- Nach Auswahl der Marke die passenden Modelle in zweite Auswahlliste eintragen
- Zu viele Daten, um alle Modelle für alle Marken im Voraus zu laden



Demos – 6. Abhängige Auswahllisten nachladen

- Marke.onchange:

```
var markeSelect = $('Marke');
aktuelleMarke = markeSelect[markeSelect.selectedIndex].innerHTML;
var url = '/' + $('WebDBName_').value +
   '/Automarken?ReadViewEntries&count=1&startKey=' +
   escape(aktuelleMarke);
callServer('GET', url, true, 0,
   createCallFunctionWithXMLHandler('setModellauswahl'));
```

Demos – 7. Eindeutigkeit einer ID prüfen

- Nach Eingabe der ID ihre Eindeutigkeit überprüfen

- ID.onchange:

```
var url = '/' + $('WebDBName_').value + '/KundenNachID' +
   '?ReadViewEntries&count=1&startKey=' + escape($('ID').value);
callServer('GET', url, true, 0, checkIDUniqueness);
```

Demos – 7. Eindeutigkeit einer ID prüfen (forts.)

- Nach Eingabe der ID ihre Eindeutigkeit überprüfen (forts.)

- JavaScript:

```
checkIDUniqueness(request)
   var firstText =
        request.responseXML.getElementsByTagName('text')[0];
   if (firstText) {
        if (firstText.firstChild.nodeValue == $('ID').value) {
                showIDNotUnique();
        } else {
                showIDUnique();
        }
   ...
```

Demos – 8. Fortschrittsanzeige

- Den Fortschritt einer lang laufenden Aktion als wachsenden Balken anzeigen
- Analysieren startet lang laufenden Prozess
- window.setInterval um regelmäßig Fortschrittsanzeige zu aktualisieren

Demos – 9. In-View-Edit

- Daten direkt **in der Ansicht** bearbeiten
- Nach dem Anklicken Text durch Textfeld ersetzen
- Nach Tab oder Return Änderungen sofort speichern

Kunden

	ID	Nachname	Anrede	Titel	Vorname	Straße	PLZ	Ort
	lbahn	Bahn	Frau	Dipl.-Math.	Lydia	Lise-Meitner-Straße 1-7	24223	Raisdorf
	tbahn	Bahn	Herr	Dipl.-Math.	Thomas	Lise-Meitner-Straße 1-7	24223	Raisdorf

Demos – 10. JavaScript-Fehler auf Server protokollieren

- Tritt auf der Seite ein JavaScript-Fehler auf, wird er zum Server gesendet und dort protokolliert

```
function sendFehlerbericht(jsFehler) {
   var url = '/, + $('WebDBName_').value +
        '/SpeichereFehlerbericht?OpenAgent';
   var postArguments = 'Fehler=' + escape(jsFehler.name) +
        '&Nachricht=' + escape(jsFehler.message) + '&Ort=' +
        escape(jsFehler.location)+ '&HRef=' + escape(location.href);
   callServer('POST', url, true, 5000,
        createCallFunctionWithTextHandler('fehlerberichtGeloggt'),
        undefined, undefined, 'application/x-www-form-urlencoded',
        undefined, postArguments); // asynchron, 5 s Time-Out
}
```

Herausforderungen

- Anforderungen werden nicht erfüllt
  - älterer Browser ohne XMLHttpRequest
  - Browser auf mobilen Geräten
  - Spezial-Browser für Behinderte (z. B. Screenreader)
  - JavaScript deaktiviert
  - ActiveX deaktiviert (nur IE)

- Benutzerfreundlichkeit
  - Zurück/Vorwärts-Buttons
  - URL für Lesezeichen oder zum Verschicken
  - fehlendes Feedback (Browser zeigt sonst Nachladen an)
  - Unterbrechung des „normalen" Flusses wegen Asynchronität
  - Zugänglichkeit (für Behinderte)

Herausforderungen (forts.)

- Erhöhte Serverlast, viele kleine Anfragen
- Unzuverlässiges oder langsames Netzwerk
  - Hohe Latenz
  - Pakete gehen verloren
  - Pakete kommen in falscher Reihenfolge zurück (auch wenn der Server unterschiedlich lange braucht)
- Testbarkeit mit automatisierten Tests
- XMLHttpRequests nur möglich an die gleiche Domäne, von der die Seite geladen wurde

Herausforderungen (forts.)

- Drucken
- Suchmaschinen
- Browser-Inkompatibilitäten
- Rückkehr des „Fat-Client“
- Technik um der Technik Willen: übermäßiger Gebrauch

Alternativen – Transport

- Alternativen beim Transport der Daten
  - unsichtbarer FRAME/IFRAME (0 Pixel breit/hoch)
  - unsichtbares Java-Applet
  - Load in DOM Level 3
  - Cookies + IMG.src (RS Lite), „GIF-Pipe“
  - SCRIPT-Tag per DOM-Manipulation hinzufügen, „JS-Pipe“
  - XML islands (nur IE ab 5.0, aber ohne ActiveX)

```
<xml id="test"></xml>
...
test.async = true;
erfolg = test.load(url);
```

  - ...

Alternativen – Datenformat

- Alternativen beim Format der Daten
  - XML
    - „einfaches“ XML
    - SOAP, insgesamt also Web Services
  - Text
    - „einfacher“ Text
    - JSON (JavaScript Object Notation)
      Untermenge von JavaScript (!)
      Beispiel: `{"name" : "tbahn", "feld" : ["1", "2", "3"]}`
      http://json.org
    - CSV, |-separiert, feste Breite, ...

Werkzeuge – Arten von Frameworks

- Frameworks kann man nach ihrem Einsatz-"Ort" unterscheiden
  - Client-seitige Frameworks sind JavaScript-Bibliotheken, die nur auf dem Client laufen.
  - Client- und serverseitige Frameworks bestehen einerseits auf JavaScript-Bibliotheken für den Client, als auch aus entsprechenden Gegenstücken für den Server. Sie sind dort natürlich sprach- oder plattformspezifisch (.net).
  - Rein serverseitige Frameworks erzeugen das notwendige JavaScript für den Client automatisch beim Ausliefern der HTML-Seite. Sie versprechen häufig, ohne JavaScript-Kenntnisse einsetzbar zu sein und werden durch z. B. entsprechende Tags durch den Web-Designer in die HTML-Seite eingebaut.

Werkzeuge – Frameworks für Domino-Anwendungen

- Ich kenne bisher kein server-seitiges Framework für Domino-Web-Anwendungen, aber Facelift für Domino geht in diese Richtung (http://www.faceliftfordomino.com/)

- Clientseitige Frameworks lassen sich natürlich gut einsetzen. Üblicherweise braucht man nur die js-Dateien auf den Server zu legen – oder als JavaScript-Bibliothek oder Page oder ... in die Datenbank einzubinden – und in die Gestaltungselemente entsprechende Aufrufe einzubauen.

Werkzeuge – Clientseitige Frameworks

- Bekannte kostenlose clientseitige Frameworks sind z. B.
  - Ajax Client Engine (http://www.lishen.name/)
  - Bindows (http://www.bindows.net/)
  - Dojo (http://dojotoolkit.org/)
  - MochiKit (http://mochikit.com/)
  - Prototype (http://prototype.conio.net/)
  - RSLite (http://www.ashleyit.com/rs/rslite/)
  - Sarissa (http://sarissa.sourceforge.net/)

Werkzeuge – Entwicklungshilfen

- Bei der Entwicklung von Web-Anwendungen mit Ajax helfen natürlich die alten Bekannten
  - Browser-Erweiterungen für Mozilla/Firefox:
    - DOM Inspector
    - IE View
    - Live HTTP Headers
    - Venkman (JavaScript-Debugger)
    - Web Developer
  - Browser-Erweiterungen für Internet Explorer
    - Microsoft Script Debugger
    - Developer Toolbar
    - IE DOM Explorer

Werkzeuge – Entwicklungshilfen (forts.)

- Weitere alte Bekannte:
  - HTML-, CSS- und JavaScript-Editoren (wenn der Domino Designer nicht reicht ;-)
  - HTML-Validatoren
- Für die Ajax-Entwicklung sind zusätzlich sinnvoll:
  - XML-Editor
  - Netzwerk-Monitor
    - FiddlerTool (http://www.fiddlertool.com/),
    - IBM Page Detailer (http://www.alphaworks.ibm.com/tech/pagedetailer)
    - Packetyzer (http://www.networkchemistry.com/products/packetyzer.php)
    - Ethereal (http://www.ethereal.com/)

Werkzeuge – Entwicklungshilfen (forts.)

- Eine neue Firefox-Erweiterung möchte ich euch ganz besonders ans Herz legen:
  - FireBug (http://www.joehewitt.com/software/firebug/)

    „FireBug is a new tool for Firefox that aids with debugging Javascript, DHTML, and Ajax. It is like a combination of the Javascript Console, DOM Inspector, and a command line Javascript interpreter. "

    und zeigt die XMLHttpRequests und die Server-Antworten an.

Werkzeuge – Quellen

- Bücher:
  - gibt es noch ziemlich wenig...
  - AJAX - frische Ansätze für das Web-Design (deutsch, online)
    http://www.teialehrbuch.de/kurse/ajax/
  - Ajax in Action (Dave Crane, Eric Pascarello, Darren James)
    ISBN: 1-932394-61-3
    http://manning.com/books/crane/

Werkzeuge – Quellen (forts.)

- Online:
  - Alles in englisch...

http://ajaxpatterns.org/
http://www.ajaxmatters.com/
http://www.xulplanet.com/references/objref/XMLHttpRequest.html
http://www.xml.com/pub/a/2005/02/09/xml-http-request.html
https://blueprints.dev.java.net/ajax-faq.html
http://www.adaptivepath.com/publications/essays/archives/000385.php
http://searchdomino.techtarget.com/generic/0,295582,sid4_gci1147220,00.html

und unter http://www-128.ibm.com/developerworks/ nach Ajax suchen

Werkzeuge – Quellen (forts.)

- Ajax und Domino:

http://www.rhs.com/poweroftheschwartz/htdocs/LotusDominoAjax.htm

Die Übersicht über Lotus Domino und Ajax!

http://www.faceliftfordomino.com/

http://www.openntf.org/Projects/pmt.nsf/ProjectLookup/Domino%20Web%20Tools

- Demos und Beispiele

http://www.workflowstudios.com/camtasia/LSPN-6FFGVH/AjaxDomino1.html

http://bob-obringer.com/A557B7/blog.nsf/dx/04272005071321PMBOBV8U.htm

http://bob-obringer.com/A557B7/blog.nsf/dx/09142005163216DOMT9Q.htm

http://www.nsftools.com/tips/NotesTips.htm#notessuggest

http://www.11tmr.com/11tmr.nsf/d6plinks/MWHE-6FAQYK

http://www.rhs.com/web/blog/poweroftheschwartz.nsf/d6plinks/RSCZ-6CATX6

Zum guten Schluss...

- Fragen?
  - jetzt stellen oder...
  - später:

    E-Mail: tbahn@assono.de

    Tel.: 0 43 07/900-401
  - Folien und Beispiele unter http://www.assono.de/ajax.html