JavaScript, Ajax & JavaScript-Frameworks

Jonas Schlaak

Institut für Architektur von Anwendungssystemen

1 Einleitung

1.1 JavaScript

JavaScript ist eine Skriptspache, die für den Einsatz in Webbrowsern verwendet wird. Sie wurde 1995 unter dem Namen LiveScript von der Firma Netscape Communications entwickelt. Die Syntax von JavaScript ist an die der C-Programmiersprache angelehnt und folgt objektorientierten Programmierparadigmen. Mittlerweile ist JavaScript durch die Ecma International standardisiert (ECMA 262)[7].

1.2 Ajax

Der Beriff „Ajax“ stammt von Jesse Garrett, der ihn erstmals in einem Artikel von 2005[2] verwendete. Unter „Ajax“ oder „AJAX“ verstand er ursprünglich eine Abkürzung für „**A**synchronous **J**avaScript **a**nd **X**ML“. Seitdem wird Ajax als Begriff für eine Sammlung von Technologien oder als ein Konzept verwendet, die bei der Entwicklung von dynamischen, an Desktopanwendungen erinnernde, Webseiten zum Einsatz kommen. Anders als die ursprüngliche Bedeutung von Ajax suggeriert, muss dabei keinesfalls XML oder Asynchronität verwendet werden.

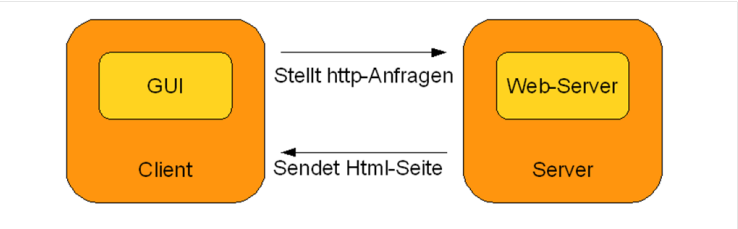

Abbildung 1. Das Schema von herkömmlichen Webanwendungen

Mit Ajax wird die Zustandslosigkeit des Http-Protokolls und damit der Webseiten scheinbar umgangen. Traditionell muss der Webclient (Browser) bei jeder Änderung einer Seite oder Aktion durch den Benutzer erneut eine Anfrage an den Server schicken. Die Änderungen oder Aktionen werden vom Server verarbeitet

und eine entsprechend veränderte Seite wird an den Client zurückgeschickt (siehe Abbildung 1). Gegenüber herkömmlichen Desktopanwendungen hat dies den Nachteil, dass der Aktionsfluss des Benutzers unterbrochen wird oder, abhängig von der Verbindung, Wartezeiten und Unterbrechungen entstehen.

Garrett beschreibt in seinem Artikel[2] eine „Ajax-Engine“, eine clientseitige Komponente, die zwischen dem Server und dem eigentlichen Client, mit dem der Benutzer interagiert, zwischengeschaltet wird (siehe Abbildung 2). Die Ajax-Engine umgeht die Nachteile traditioneller Webanwendungen, indem die Anfragen des Clients zunächst an die Engine gehen. Die Engine wiederum ist in der Lage, gezielt Anfragen nach Daten, die benötigt werden, an den Server zu stellen. Aktionen des Benutzers, die keine Daten vom Server benötigen, wie beispielsweise die Validierung von Daten, können von der Engine selbstständig verarbeitet werden. Die Daten, die der Server als Antwort auf die Anfragen der Engine liefert, können per JavaScript in die Webseite eingebunden werden. Die Interaktion des Benutzers mit der Webseite wird also nicht von der Kommunikation mit dem Server beeinflusst. Ein Beispiel für diesen Vorgang liefert das folgende Szenario: Ein Benutzer sucht nach Einträgen in einem Telefonbuch. Er grenzt seine Anfrage durch die Eingabe der ersten Buchstaben des Nachnamens und der Postleitzahl ein. Die Ajax-Engine prüft, ob die eingegebene Pstleitzahl nur aus Ziffern besteht. Gleichzeitig sendet sie eine entpsrechende Suchanfrage an den Server. Die Validierung und die Anfrage werden durch die Eingabe des Benutzers ausgelöst, also ohne explizites Wissen des Benutzers. Die gefundenen Einträge des Servers werden durch die Engine verarbeitet und als Tabelle auf der Webseite angezeigt. Dieser Vorgang widerholt sich bei jeder neuen Eingabe des Benutzers. Die Suchergebnisse werden also dynamisch aktualisiert. Der Benutzer nimmt nicht explizit war, dass mit einem Server kommuniziert wird.

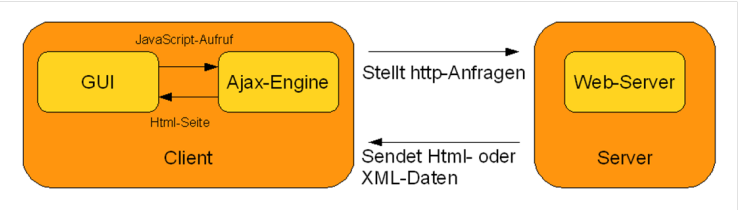

Abbildung 2. Das Schema von Ajax-Webanwendungen

2 Das `XMLHttpRequest`-Objekt

In Abschnitt 1.2 wurde das Ajax-Interaktionkonzept erläutert, das vorsieht, die Serveranfragen und das Nachladen von Daten ohne Unterbrechung der Benutzeraktionen durchzuführen. Im Folgenden wird auf die technische Verwirklichung eingegangen. Das unterbrechungsfreie Anfragen und Nachladen wird duch

das `XMLHttpRequest`-Objekt ermöglicht. Das `XMLHttpRequest`-Objekt wurde ursprünglich von Microsoft als Bestandteil von ActiveX mit dem Internet Explorer 5 eingeführt. Mittlerweile ist es bei allen gängigen Browsern ein natives JavaScript-Objekt.

2.1 Erzeugen und Absenden von Serveranfragen

Zunächst muss ein neues `XMLHttpRequest`-Objekt erzeugt werden:

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

Listing 1.1.

Das Objekt selbst stellt die Methoden `open()` und `send()` zur Verfügung, um Anfragen an den Server vorzubereiten und abzusenden.

```
request.open("get", "webpage.html", true);
request.send();
```

Listing 1.2.

Im vorhergehenden Beispiel wird mit `open()` eine GET-Anfrage nach der Web-Ressource `webpage.html` an den Server vorbereitet und anschließend mit `send()` verschickt. Das dritte Argument `true` gibt an, dass der nachfolgende JavaScript-Code ausgeführt wird, ohne auf eine Antwort des Servers zu warten. Durch diese asynchrone Kommunikation kann der Benutzer weiterhin mit der Webanwendung interagieren, ohne dass er durch das Laden der Daten vom Server unterbrochen wird.

2.2 Das Attribut `readyState`

Das `XMLHttpRequest`-Objekt besitzt das Attribut `readyState`, um den Zustand einer Serveranfrage und den einer eventuellen Antwort abfragen zu können. Das Attribut kann, wie in Tabelle 1 beschrieben, fünf verschiedene Integer-Werte annehmen. Mit Hilfe des Attributs `onreadystatechange` ist es möglich, eine

Wert	Bezeichnung	Bedeutung
0	UNSENT	Standardzustand des `XMLHttpRequest`-Objekts
1	OPENED	Die `open()`-Methode wurde aufgerufen
2	HEADERS_RECEIVED	Die Header der Serverantwort wurden empfangen
3	LOADING	Der Body der Serverantwort wird empfangen
4	DONE	Die Serverantwort wurde vollständig empfangen

Tabelle 1. Die möglichen Werte des Attributs `readyState`

Funktion zu definieren, die bei jeder Änderung von `readyState` aufgerufen wird. Innerhalb der Funktion kann dann durch Fallunterscheidung definiert werden, welche Aktionen für den jeweiligen Zustand ausgeführt werden sollen:

```
request.onreadystatechange = example();

function example() {
  if (request.readyState == 4) {
    /* Die Antwort des Servers ausgeben */
    document.write(request.responseText);
  }
}
```

Listing 1.3.

3 JavaScript Object Notation (JSON)

JSON ist ein strukturiertes Datenformat für den Datenaustausch. Es wurde mit dem Ziel entwickelt, sowohl menschenlesbar als auch einfach zu parsen sein. JSON ist als Teilmenge von JavaScript Bestandteil der ECMA-262-Spezifikation[7], [4]. In einer Ajax-Webanwendung dient JSON als Alternative zum XML-Format.

3.1 Objekte und Arrays

In JSON gibt es zwei wesentliche Elemente: Objekte und Arrays.

Ein Objekt ist eine unsortierte Menge von Schlüssel-Wert-Paaren. Ein Objekt beginnt mit { und endet mit }. Der Schlüssel und der Wert eines Paares wird durch : getrennt. Schlüssel-Wert-Paare eines Objekts werden durch ein Komma getrennt. Ein Beispiel:

```
var json_objekt = {"Schlüssel_1":"Wert_1","Schlüssel_2":"Wert_2"};
```

Listing 1.4.

Ein Wert kann ein String, eine Zahl, ein boolscher Wert oder wiederum ein Objekt oder ein Array sein.

Ein Array ist eine sortierte Liste von Werten. Ein Array beginnt mit [und endet mit]. Jeder Wert wird durch ein Komma getrennt. Ein Beispiel:

```
var json_array = ["Wert_1", "Wert_2", "Wert_3"];
```

Listing 1.5.

3.2 Die `eval()`-Funktion

Der Vorteil von JSON liegt in der einfachen Repräsentation von Objekten und Arrays als Zeichenkette. Die (De-)Serialisierung von Daten ist somit einfach zu bewerkstelligen. Um eine Zeichenkette wieder in ein JavaScript-Objekt oder -Array umzuwandeln, wird die `eval()`-Funktion benötigt. Mit ihr ist es möglich, eine Zeichenkette als Parameter zu übergeben, die als JavaScript-Code ausgeführt wird. Im folgenden Beispiel wird eine Zeichenkette im JSON-Format vom Server zurückgegeben:

```
{"Name":"Max Mustermann", "Wohnort":"Stuttgart"}
```

Listing 1.6.

Sei `request` das `XMLHttpRequest`-Objekt (siehe Abschnitt 2.2), welches die Anfrage geschickt hat.

```
request.onreadystatechange = example();

function example() {
  if (request.readyState == 4) {
    /* Die Antwort des Servers in ein JSON-Objekt umwandeln */
    var json_objekt = eval("(" + request.responseText + ")");
    document.write(json_objekt.Name + " wohnt in "
        + json_objekt.Wohnort + ". ");
  }
}
```

Listing 1.7.

Das obige Code-Beispiel würde zur Ausgabe „Max Mustermann wohnt in Stuttgart.“ führen.

4 Frameworks

Um die Erzeugung von und den Umgang mit interaktiven Elementen einer Webseite zu erleichtern, gibt es eine Reihe von JavaScript-Frameworks, die bereits fertige Funktionen für den Webprogrammierer zur Verfügung stellen. Im Folgenden werden ein paar dieser Frameworks behandelt, speziell wird dabei auf den Aspekt eingegangen, ob und wie graphische Objekte handzuhaben und zu manipulieren sind. Dies wird am Beispiel der Erzeugung eines Rechtecks demonstiert, das verschiebbar und in der Größer veränderbar sein soll.

4.1 jQuery

jQuery wurde von John Resig entwickelt und im Januar 2006 vorgestellt[3]. Es ist wahlweise unter der MIT- oder GNU-Lizenz erhältlich.

Einbindung Das komplette jQuery-Framework wird als einzelne JavaScript-Datei `jquery.js` ausgeliefert. Um das Framework zu laden, muss folgende Zeile in einem HTML-Dokument vorhanden sein:

```
<script type="text/javascript" src="jquery.js"></script>
```

Listing 1.8.

Benutzung Ein Beispiel für den Aufbau eines HTML-Dokuments bei der Benutzung von jQuery:

```
<html>
<head>
  <script type="text/javascript" src="jquery.js"></script>
  <script>
  $(document).ready(function(){
    $("a").click(function(){
      alert("Hallo Welt!");
    });
  });
  </script>
</head>
<body>
  <a href="">Link</a>
</body>
</html>
```

Listing 1.9.

Das $-Zeichen ist ein Alias für die jQuery-Klasse. Damit ist `$()` ein Konstruktor. Übergibt man diesem Konstruktor ein Objekt als Parameter, wird das Objekt als ein jQuery-Objekt behandelt. In der `ready`-Methode wird der Code angegeben, der ausgeführt werden soll, wenn die Webseite fertig geladen worden und bereit für Interaktion mit dem Benutzer ist. In diesem Beispiel werden alle `a`-Objekte (HTML-Anker-Objekte) ausgewählt. Mit der `click`-Methode wird ein Mausklick-Event mit allen Ankerobjekten verknüpft. Die nachfolgende Funktion spezifiziert, was bei einem Mausklick passieren soll. In diesem Fall wird „Hallo Welt!“ in einem Popup-Fenster ausgegeben.

Grafik und Interaktion jQuery bietet von sich aus keine Möglichkeiten, eigene graphische Objekte zu erzeugen. Vielmehr können vorhandene graphische Objekte (z.B. eine farbige `div`-Fläche) bestimmte Interkationsmöglichkeiten zugeschrieben werden. Diese Anzahl der Möglichkeiten ist beschränkt, es gibt momentan zwei wesentliche Eigenschaften:

- „draggable“, sorgt dafür, dass ein Objekt verschoben werden kann
- „resizable“, sorgt dafür, dass ein Objekt in seiner Größe verändert werden kann

Ein Beispiel für die Erzeugung eines verschiebbaren Rechtecks:

```
<html>
<head>
  <script src="http://code.jquery.com/jquery-latest.js"
  </script>
  <script src="http://ui.jquery.com/latest/ui/ui.core.js">
```

```
  </script>
  <script src="http://ui.jquery.com/latest/ui/ui.draggable.js">
  </script>

  <script>
  $(document).ready(function(){
    $(".block").draggable();
  });
  </script>
  <style>
  .block {
    border: 1px solid #000000;
    background-color: #0000FF;
    width: 160px;
    height: 80px;
    margin: 10px;
  }
  </style>
</head>
<body>
  <div class="block"></div>
</body>
</html>
```

Listing 1.10.

Das obige Beiscpiel beschreibt die Erzeugung eines `<div>`-Elements, dessen Aussehen über den `<style>`-Block definiert wird. Im jQuery-Block wird dieses Element selektiert und mit der Methode `draggable()` verschiebbar gemacht. Da sich diese graphischen Features noch in Entwicklung befinden, verweisen die Importe auf die neuesten Versionen von jQuery.

4.2 script.aculo.us

script.aculo.us ist ein Aufsatz für das Prototype-Framework und wurde von Thomas Fuchs entwickelt[6]. Es wurde im Juni 2005 vorgestellt und ist unter der MIT-Lizenz erhältlich.

Einbindung Das script.aculo.us-Framework besteht aus sechs einzelnen JavaScript-Dateien. Analog wie im Abschnitt „Einbindung“ von 4.1 müssen die Dateien über den `<script>`-Befehl importiert werden.

Benutzung Ähnlich wie bei jQuery wird script.aculo.us über das $-Zeichen angesprochen. Als Parameter wird das zu manipulierende Objekt übergeben. Über eine jeweilige script.aculo.us-Methode kann das Objekt manipuliert werden. Ein Beispiel für einen Einblendeffekt eines `<div>`-Objekts wird im folgenden Listing beschrieben:

```
<script type="text/javascript" language="javascript">
  $('block').appear();
</script>
<body>
  <div id="block" style="width:80px;
      height:80px; background:#ffffff>
  </div>
  ...
</body>
```

Listing 1.11.

Im Gegensatz zu jQuery besteht alternativ die Möglichkeit, das Verhalten direkt bei den Eventhandlern des `<div>`-Objekts anzugeben. Ein Beispiel hierfür:

```
<div id="block" style="width:80px; height:80px;
    background:#ffffff onlick="$(this).appear()">
</div>
```

Listing 1.12.

Grafik und Interaktion script.aculo.us bietet keine Möglichkeit, eigene graphische Objekte zu erzeugen. Im Gegensatz zu jQuery gibt es allerdings eine Reihe von Möglichkeiten, verschiedene Objekte zu manipulieren. Ein unvollständiger Auszug aus der Liste der Möglichkeiten:

- draggable
- droppable(),
- appear()
- fade()
- grow()
- shrink()
- scale()
- ...

Es folgt ein Beispiel für die Erzeugung eines verschiebbaren Rechtecks:

```
<script type="text/javascript">
  new Draggable('drag_demo_1');
</script>
<div id="block" style="width:160px; height:80px;
    background:#ffffff;">
</div>
```

Listing 1.13.

Dieses Reckteck kann durch direkte Manipulation mit der Maus verschoben werden. Im Gegensatz zu jQuery kann ein in der Größe veränderbares Rechteck nicht erzeugt werden. Vielmehr lassen die Methoden `grow()`, `shrink()` und `scale()` nur Größenanderungen mit statischen Werten zu. Eine direkte Manipulation durch Interaktion mit der Maus ist damit nicht möglich.

4.3 draw2d

draw2d ist ein Aufsatz für das Mootools-Framework und wurde von Andreas Herz entwickelt[1]. Es wurde Ende 2007 vorgestellt und ist unter der LGPL-Lizenz erhältlich. Es wurde entwickelt, um eine graphische Zeichenkomponente für Workflows in einem Browser zu ermöglichen.

Einbindung Das draw2d-Framework besteht aus vier einzelnen JavaScript-Dateien. Analog wie im Abschnitt „Einbindung“ von 4.1 müssen die Dateien über den `<script>`-Befehl importiert werden:

```
<script src="wz_jsgraphics.js" type=text/javascript>
</script>
<script src="mootools.js" type=text/javascript>
</script>
<script src="moocanvas.js" type=text/javascript>
</script>
<script src="draw2d.js" type=text/javascript>
</script>
```

Listing 1.14.

Benutzung Um mit draw2d zeichnen zu können, muss zunächst ein Zeichenbereich definiert werden, in dem anschließend die graphischen Objekte gezeichnet werden. Das wird durch ein `<div>`-Objekt erreicht:

```
<div id="canvas" style="width:1000px;height:750px"></div>
```

Listing 1.15.

Die Benennung in „canvas“ ist dabei willkürlich. Das eigentliche Objekt, in das später gezeichnet wird, ist das `workflow`-Objekt. Das muss mit dem Zeichenbereich assoziiert werden:

```
<script>
   var workflow = new Workflow("canvas");
   ...
</script>
```

Listing 1.16.

Grafik und Interaktion Um ein einfaches, verschiebbares und in der Größe veränderbares Rechteck zu erzeugen, muss im Gegensatz zu den vorherigen Frameworks nicht auf ein `<div>`-Objekt zurückgegriffen werden. Vielmehr stellt draw2d dafür eigene Objekte zur Verfügung.

```
<script>
  ...
  var rechteck = new Rectangle();
  rechteck.setDimension(160,80);
  workflow.addFigure(rechteck,250,300);
</script>
```

Listing 1.17.

Mit der `addFigure`-Methode wird das erzeugte Rechteck dem Workflow hinzugefügt und kann so manipuliert werden.

5 Prototyp: Draw2d und Serialisierung zu XML

Im Rahmen des Seminars soll ein Prototyp entwickelt werden, mit dem man einen rudimentären Workflow-Graphen zeichnen kann. Der Workflow-Graph soll anschließend in ein XML-Format serialisiert werden. Der Prototyp wurde mit draw2d verwirklicht, da draw2d mehr Möglichkeiten zur graphischen Manipulation bietet als die anderen, in dieser Arbeit vorgestellten, Frameworks. Außerdem bietet draw2d Objekte und Methoden an, um einen vollständigen Graphen, bestehend aus Knoten und Kanten, zu modellieren. Draw2d bietet in seiner aktuellen stabilen Version 0.9.14 von sich aus keine Möglichkeit an, die gezeichneten Objekte in irgendein Datenformat zu serialisieren. Für zukünftige Versionen ist eine solche Möglichkeit angekündigt. Auf der Webseite von draw2d findet sich jedoch eine Demo, die einen rudimentären XML-Serialisierer zur Verfügung stellt. Auf Basis dieses Serialisierers wurde für den Prototypen eine Erweiterung enwickelt. Die Modellierung eines Workflows als Graph und die anschließende Serialisierung zu XML werden in den folgenden zwei Abschnitten näher erläutert.

5.1 Die Klasse `FlowNode.js`

Um mit Hilfe von draw2d in Ansätzen Workflows bzw. vereinfacht Graphen modellieren zu können, wurde ein neue draw2d-Klasse `FlowNode` erstellt, die Objekte vom Typen `Node` erstellt und zeichnet. `Node` respräsentiert einen Knoten. Der Einfachheit halber kann ein Knoten nur eine eingehende Kante und eine ausgehende Kante besitzen. Das wird durch sogenannte Input- bzw. Output-Ports verwirklicht. Code-Ausschnitt von `FlowNode.js`:

```
/* Node wird erstellt */
var node = new Node();

/* Input-Port, eingehende Kante muendet hier */
var ip = new InputPort();
node.addPort(ip, node.width / 2, 0);

/* Output-Port, ausgehende Kante faengt hier an */
var op = new OutputPort();
```

```
node.addPort(op, node.width / 2, node.height);
```

Listing 1.18.

5.2 XML-Serialisierer

Der draw2d-Workflow bietet über die Methode `getDocument()` Zugriff auf seine DOM-Struktur. Diese Struktur bietet weitere Methoden, um auf die einzelnen Zeichenobjekte des Workflows zuzugreifen. Die Methode `getFigures()` liefert beispielsweise eine ArrayList alle Zeichenobjekte eines Workflows zurück. Die Zeichenobjekte bieten wiederum Methoden an, um auf ihre Struktur zuzugreifen. Im Folgenden ein Ausschnitt des XML-Serialisiers (die Variable `xml` ist ein String, der zum Schluss das vollständige XML-Dokument repräsentiert):

```
var xml = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n";
xml = xml + "<form>\n";

/* alle Zeichenobjekte in eine ArrayList holen */
var figures = document.getFigures();
for (var i = 0; i < figures.getSize(); i++) {

    /* Jedes Zeichenobjekt einzeln ansprechen */
    var figure = figures.get(i);
    xml = xml + "<" + figure.type + "\" id=\"" +
        figure.getId() + "\">\n";

    /* Falls ein Objekte Ports hat, werden diese in eine
     * ArrayList geholt
     * /
    if (figure.getPorts().getSize() > 0) {
        var ports = figure.getPorts();
        xml = xml + "<ports>\n"
        for (var j = 0; j < ports.getSize(); j++) {
            /* Auf Ports einzeln zugreifen */
            var port = ports.get(j);
            /* Connections = Kanten ermitteln */
            if (port.getConnections().getSize() > 0) {
                var connections = port.getConnections();
                for (var k = 0; k < connections.getSize(); k++) {
                    var connection = connections.get(k);
                    xml = xml + "<" + port.type + " targetId=\""
                        + connection.getTarget().getParent().getId()
                        + "\">\n";
                }
            }
        }
```

```
            xml = xml + "</ports>\n";
        }

        xml = xml + "</" + figure.type + ">\n";
    }
    xml = xml + "</form>\n";
```

Listing 1.19.

Literatur

1. Draw2D: http://draw2d.org, abgerufen am 5.1.2009.
2. Garrett, J.J. (2005): Ajax: A New Approach to Web Applications, http://adaptivepath.com/ideas/essays/archives/000385.php, abgerufen am 3.1.2009.
3. jQuery: Write less, do more, http://jquery.com/, abgerufen am 5.1.2009.
4. JSON: JavaScript Object Notation, http://json.org, abgerufen am 5.1.2009.
5. Koch, S. (2006): JavaScript - Einführung, Programmierung und Referenz. Dpunkt Verlag, 4. Auflage.
6. script.aculo.us: web 2.0 javascript, http://script.aculo.us/, abgerufen am 5.1.2009.
7. Standard ECMA-262 (1999): ECMAScript Language Specification, http://www.ecma-international.org/publications/standards/Ecma-262.htm, abgerufen am 3.1.2009.
8. Wenz, C. (2006): JavaScript und Ajax. Galileo Press, 7. Auflage.