

Skriptsprachen:

JavaScript

20. November 2003

Bernhard Schneider
schneidb@fmi.uni-passau.de

Gliederung

1.	Einführung	3
1.1.	Zielsetzung, Anwenderkreis und Einordnung der Sprache.....	3
1.2.	Entstehungsgeschichte der Sprache JavaScript	3
1.3.	Bestandteile der Sprache	3
1.4.	Der Sandkasten: was JavaScript darf und was nicht	4
2.	Definition von Syntax, Semantik und Objektmodell	4
2.1.	ECMA- und DOM- Standard: Trennung von Sprache und Objektmodell.....	4
3.	Einbettung von JavaScript- Code in eine HTML- Seite	4
4.	Datenverwaltung: Datentypen und Datenstrukturen	5
4.1.	Typing, Datentypen und Datenstrukturen	5
5.	Sprachkonstrukte: Verzweigungen und Schleifen	5
6.	Funktionen	5
6.1.	Grundlagen: Funktionsarten, Definition und Parameterübergabe	5
6.2.	Funktionen mit Rückgabewerten, rekursive Funktionen.....	6
6.3.	Vordefinierte Funktionen	6
7.	Sprachprinzipien	6
7.1.	Klassenbasierte Sprachen im Gegensatz zu prototyp- basierten Sprachen.....	6
7.2.	Der Umgang mit Objekten: Definition, Instanziierung, Zugriff.....	7
7.3.	Vererbung: Der <i>prototype</i> - Operator	8
7.4.	Hinzufügen und Entfernen von Eigenschaften	8
7.5.	JavaScript und die Paradigmen der Objektorientierung	9
7.6.	Ausführung von JavaScript- Code	9
8.	Vordefinierte Objekte	10
8.1.	Objekte der Core Language.....	10
8.2.	Objekte der Client- Side- Erweiterung	10
9.	Ereignisse	12
10.	Kommunikation zwischen HTML- Seiten	12
10.1.	Kommandoparameter	12
10.2.	Cookies	13
11.	Erweiterung der Sprache JavaScript: LiveWire	13
11.1.	Komponenten der LiveWire- Software	13
11.2.	Server- Side- JavaScript Applikationen.....	14
11.3.	LiveWire- Objekte und vordefinierte Funktionen	14
12.	LiveConnect: Java- JavaScript- Kommunikation.....	14
13.	Sicherheitsaspekte	15
13.1.	Sicherheitslücken	15
13.2.	Sicherheitsmodelle	15
14.	Praktische Arbeit mit JavaScript: Editoren und Entwicklungstools	16
15.	Entwicklungstendenzen	16
16.	Bewertung der Sprache JavaScript	16
17.	Zusammenfassung.....	17
18.	Quellenverzeichnis.....	17

1. Einführung

1.1. Zielsetzung, Anwenderkreis und Einordnung der Sprache

JavaScript wurde entwickelt um HTML- Seiten zu optimieren. Bestehende interaktive Lücken in HTML- Dokumenten sollten damit geschlossen werden.

Angestrebte Verbesserungen bezogen sich auf die Interaktion mit dem Benutzer und mit HTML- Dokumenten, die Kontrolle des Browsers und das Management der Dokumenten- Hierarchie [Har00].

JavaScript- Code sollte plattformunabhängig, schlank und kompakt sein. Er sollte leicht in den Code anderer Applikationen eingebettet werden können, um die Kontrolle über deren Objekte zu übernehmen. Die Sprache sollte leicht zu erlernen und auch von Laien beherrschbar sein. Zum Anwenderkreis sollte auch der „Nichtprogrammierer“ [Hes97] gehören.

JavaScript orientiert sich syntaktisch an Java, stellt aber keine Untermenge von Java dar [Fri00]. JavaScript wurde nicht als *standalone*- Sprache konzipiert [Net98]. Die Sprache ist deshalb als Ergänzung der Sprache HTML und als Erweiterung des Web- Browsers zu sehen [Rüe01]. JavaScript- Code wird auf dem Client interpretiert: alle gängigen Browser verfügen über abschaltbare Interpreter.

JavaScript verwendet eine objektorientierte Semantik, der Programmierer definiert jedoch keine eigenen Klassen. In der Fachliteratur wird JavaScript deshalb oftmals als *objekt-basiert* beschrieben.

Der Code wird unmittelbar während der Laufzeit des Skripts auf dem Client interpretiert, also nicht im Voraus kompiliert. So wird Plattformunabhängigkeit und Einfachheit der Programmierung erreicht, was zur Verringerung der Perfomanz führt.

Die Konkurrenz dieser Web- Scripting- Sprache stellen Microsofts *VBScript* und *JScript* [Mic03], und weitere dar. JavaScript besitzt derzeit einen Marktanteil von 95% unter den Client- basierten Skriptsprachen [Moh03].

1.2. Entstehungsgeschichte der Sprache JavaScript

Im Jahr 1995 stellte Netscape eine Skriptsprache namens *LiveScript* vor. Diese Sprache wurde fester Bestandteil des Internet- Browsers *Netscape Navigator*. Mit ihr war es möglich, in verstärktem Masse Einfluss auf die Gestaltung von HTML- Seiten zu nehmen. Die Syntax lehnte sich sehr stark an die der Sprache *Java* an. Im Zuge der aufkommenden ‚Java- Euphorie‘ beschloss Netscape, *LiveScript* aus marketingtechnischen Gründen in *JavaScript* umzubenennen [Wen02].

Seit Einführung von JavaScript 1.1 besitzt auch Microsoft eine JavaScript-Lizenz. Aus lizenzrechtlichen Gründen musste der Name der unterstützten Sprache geändert werden. So bot Microsofts Internet Explorer ab der Version 3.0 Unterstützung für eine Skriptsprache namens *JScript*. Diese wird von Microsoft weiterentwickelt.

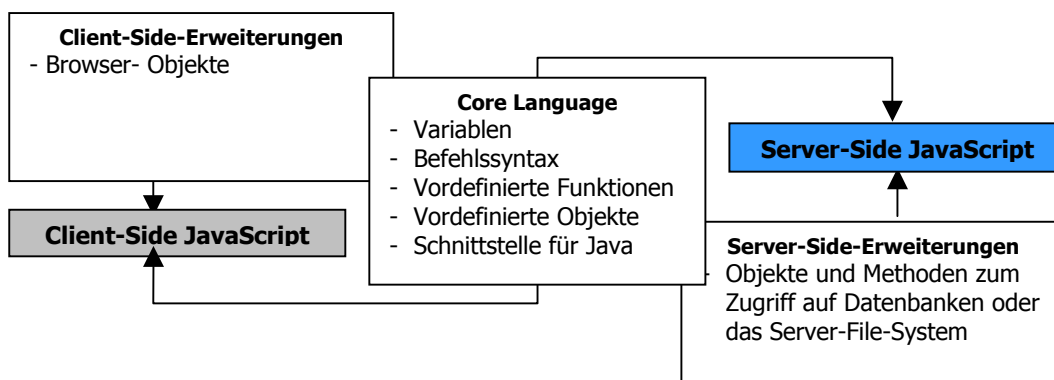
JavaScript wird in Zusammenarbeit von Netscape und Sun weiterentwickelt. In Anlehnung an die Kaffeesorte wird diese Skriptsprache auch *Mocha* genannt, was ihrem internen Projektnamen entspricht [Röt02].

JavaScript liegt gegenwärtig in der Version 1.5 vor. Diese ist ab Version 6.0 im *Netscape Navigator* verfügbar.

[Net00] bietet einen Überblick über die Browser- und JavaScript- Versionen.

1.3. Bestandteile der Sprache

JavaScript kann in die *Core Language*, *Client- Side- Erweiterungen* und *Server- Side- Erweiterungen* unterteilt werden [Net99]:



1.4. Der Sandkasten: Was JavaScript darf und was nicht

JavaScript läuft in einem abgesicherten *Sandkasten* [And97] gänzlich auf dem Client ab. Es kann nur auf die Objekte zugreifen, die der Browser gegenwärtig im Speicher hält. Daher kann es keine Informationen, mit Ausnahme von Cookies, in beliebigen Dateien auf den Festplatten des Benutzers oder eines Servers speichern und keine fremden Fenster schliessen oder deren Inhalt manipulieren.

Nach [And97] ist es aber möglich, die Restriktionen der Sandbox für signierte Javascripts aufzuheben um diesen Sonderrechte einzuräumen.

JavaScript kann eigenen Quelltext nicht verbergen und E- Mails nicht heimlich versenden. Der Zugriff auf Peripheriegeräte ist nicht möglich. Abgesehen vom Browser selbst kann JavaScript keine anderen Applikationen ansprechen oder beenden.

2. Definition von Syntax, Semantik und Objektmodell

2.1. ECMA- und DOM- Standard: Trennung von Sprache und Objektmodell

Die Programmiersprache selbst ist der generische Teil von JavaScript, der unabhängig vom Verwendungszweck ist. Dazu gehören Syntax, Semantik und die Standardbibliotheken für Mathematik, Zeitrechnung, Zeichenkettenmanipulation und weitere.

Dieser Teil wurde unter dem Namen *ECMAScript* von ECMA (European Computer Manufactures Association) und ISO (International Organization for Standards) in den Jahren 1996 und 1997 standardisiert. Seit Version 1.3 (1997) ist JavaScript dem ECMA- 262 Standard (*ECMAScript Edition 3*) [Ecm99] und der ISO- Norm 16262 [Iso03] unterworfen.

Der zweite Bestandteil von JavaScript ist das Objektmodell. Dieser Teil der Sprache wurde unter dem Namen *DOM (Document Object Model)* [Dom03] vom *W3C (World Wide Web Consortium)* standardisiert [Www03]. Das *DOM* ist ein Standard für die Darstellung eines Dokumentes durch Objekte. Es definiert eine Objekthierarchie, die es Programmiersprachen ermöglicht, auf Dokumentbestandteile zuzugreifen. Der *DOM*- Standard liegt gegenwärtig im *Level 3* vor. Netscape unterstützt derzeit *Level 0,1* und ansatzweise *2* [Net03].

3. Einbettung von JavaScript- Code in eine HTML- Seite

Es gibt drei verschiedene Arten, JavaScript- Code in ein HTML- Dokument einzubetten. Eine Möglichkeit stellt das direkte Einbringen des JavaScript- Codes in den HTML- Code dar. Dazu wird die entsprechende Codestelle mit *<script>*-Tags eingefasst:

```
<script language ="JavaScript">
  // hier folgt der JavaScript- Teil
</script>
```

Zusätzlich ist es möglich, eine bestimmte JavaScript- Versionsnummer im *<script>*-Tag anzugeben. So kommt der betreffende Codeabschnitt nur dann zur Ausführung, wenn der Browser JavaScript ab der entsprechenden Version unterstützt.

JavaScript-Code kann auch in einer externen Datei abgelegt und in HTML- Code eingebunden werden.

Die Einbindung in den HTML- Code erfolgt ebenfalls über die *<script>*-Tags, wobei das Attribut *SRC* angegeben wird. Diesem wird der Pfad zur entsprechenden Datei zugewiesen:

```
<script language="JavaScript1.1" SRC="/js/greetings.js">
</script>
```

Damit ist es möglich, eigene Bibliotheken anzulegen.

Die letzte Möglichkeit der Einbettung erlaubt ausschliesslich das Setzen der Werte von HTML-Attributen. Sie wird nur vom Netscape Navigator angeboten. Man bedient sich hierbei der *JavaScript- Entities*: sie beginnen mit *{*, enden mit *}*; dazwischen steht ein JavaScript- Ausdruck:

```
<html>
<form>
  Protokoll: <INPUT TYPE="TEXT" VALUE="&{location.protocol};">
</form>
</html>
```

Das verwendete *JavaScript- Entity* trägt die Bezeichnung *&location.protocol*; Es gibt das verwendete Protokoll der aktuellen Seite vor, also beispielsweise *file* bei lokalen Dateien und *http* bei Dateien aus dem World Wide Web.

4. Datenverwaltung

4.1. Typing, Datentypen und Datenstrukturen

JavaScript verwaltet intern 3 Datentypen: *Number*, *String*, *Boolean*. Mit Hilfe des Datentyps *Number* lassen sich Integerwerte im Dezimal-, Oktal- und Hexadezimalsystem darstellen. Zudem sind Fließkommawerte darstellbar. Daneben existiert das Schlüsselwort *null*. Es stellt einen Nullwert dar. Selbstdefinierte Typen lassen sich mit JavaScript nicht erstellen.

Bei der Deklaration von Variablen werden keine Typen angegeben. JavaScript unterscheidet anhand des Kontextes selbst, welcher Typ benötigt wird [Koc97]. Dieses Konzept wird mit dem Begriff *loose typing* umschrieben. Die Deklaration einer lokalen Variable erfolgt mit dem Schlüsselwort *var*: `var a = 12;` Ihr Gültigkeitsbereich ist der Block, in dem sie definiert wurde. Bei Deklaration einer globalen Variable entfällt das Schlüsselwort. Deklarationen können mit oder ohne Initialisierung erfolgen. Bei nicht- initialisierender Deklaration erhält die betreffende Variable den Wert *null* zugewiesen.

JavaScript bietet die Möglichkeit, Daten in Arrays zu arrangieren. In JavaScript stellt ein Array keine homogene Datenstruktur dar. Arrays sind dynamisch und können assoziativ angelegt werden. Man erhält sie durch Instanziierung eines vordefinierten Array- Objekts [vgl. 8.1. Vordefinierte Objekte].

Das *loose typing* erfordert Konvertierung von Datentypen. In Ausdrücken, die numerische Werte und Zeichenketten mit einem '+'- Operator verbinden, wird der numerische Wert in einen String konvertiert. Kommen andere Operatoren zum Einsatz, erfolgt diese Konvertierung nicht:

```
"37" + 7; // Ergebnis: 377
```

```
"37" - 7; // Ergebnis: 30
```

5. Sprachkonstrukte: Verzweigungen und Schleifen

Die Kontrollstrukturen von JavaScript entsprechen denen der Sprache Java. Fallunterscheidungen und Verzweigungen werden mittels des *if* Konstrukts realisiert. Geschachtelte *if* Abfragen sind auch möglich. Innerhalb der Bedingungen sind die selben unären und binären Operatoren erlaubt, die auch in der Sprache Java zur Verfügung stehen.

Zur wiederholten Ausführung von Programmteilen stehen die *for*- Schleife und die *while*- Schleife zur Verfügung. Die syntaktischen Strukturen dafür entsprechenden denen der Sprache Java.

Bei Benutzung der *for*- Schleife erfolgt die Deklaration der Zählvariable ob des *loose typings* ohne Typangabe mit Hilfe des Schlüsselwortes *var*:

```
for (var i = 0; i < 100; i++){           while(Bedingung){
    // JavaScript- Anweisungsfolge      // JavaScript- Anweisungsfolge
    if (Abbruchbedingung) break;        if (Bedingung) continue;
}
```

Mittels der *break*- Anweisung im Schleifenkörper ist es möglich, die Wiederholungssequenz einer *for*- oder *while*- Schleife zu beenden, falls eine Abbruchbedingung erfüllt ist.

Durch bedingungsabhängige Ausführung der Anweisung *continue* werden alle, im Schleifenrumpf auf die *continue*- Anweisung folgenden Befehle für den aktuellen Schleifendurchlauf übersprungen.

Zusätzlich zur *for*- Schleife existiert ein *for-in*- Konstrukt. Hier wird über die Indexzahl eines Objektes iteriert. Existiert beispielsweise innerhalb einer Webseite ein Formular *test* mit 4 Eingabezeilen, so würde das Konstrukt `for (i in test){Anweisung}` die Durchführung der Anweisung in jeder der 4 Eingabezeilen erzwingen.

6. Funktionen

Funktionen sind zentraler Bestandteil der Sprache JavaScript. Sie werden selbst als Objekte angesehen. In prototyp- basierten Sprachen nehmen sie den Platz von Klassen in objektorientierten Sprachen ein.

6.1. Grundlagen: Funktionsarten, Definition und Parameterübergabe

Es gibt zwei verschiedene Arten von Funktionen: Methoden und Konstruktorfunktionen. Methoden dienen der Bündelung von oftmals benötigten Anweisungsfolgen. Sie können Bestandteile von Objekten sein, oder unmittelbar aus dem HTML- Kontext aufgerufen werden. Methoden können Rückgabewerte liefern.

Konstruktorfunktionen dienen einzig der Erzeugung von neuen Objekten. Eine Funktion wird erst dann zur Konstruktorfunktion, wenn sie dazu benützt wurde mittels des *new*- Operators ein neues Objekt anzulegen.

Funktionen können in jedem JavaScript- Bereich eines HTML- Dokuments definiert werden. Ihre Definition besitzt in JavaScript den folgenden Aufbau:

```
function Funktionsname (Liste_der_formalen_Parameter){
    JavaScript- Anweisungen
}
```

Es ist erlaubt, Funktionen mit variabler Anzahl von Übergabeparametern zu definieren. In diesem Fall entfällt die Liste der formalen Parameter. Hier könnte ohne Annotation ein dynamisches Array übergeben werden, über dessen Größe man im Funktionsrumpf iterieren kann.

In JavaScript werden Variablen über die *Call- by- Value*- Strategie an Funktionen übergeben [Koc97]. Dabei wird der Wert der Variable zum aktuellen Zeitpunkt übergeben, nicht die Referenz auf die Variable. Der Wert des aktuellen Parameters wird im Funktionsrumpf also nicht verändert.

Es ist möglich, globale Variablen innerhalb einer Funktion zu definieren. Dazu wird auf das *var*- Schlüsselwort verzichtet. Diese Variablen sind erst nach dem ersten Aufruf der Funktion auch außerhalb des Funktionsrumpfes bekannt.

Funktionen sollten immer im Header einer HTML- Datei definiert werden: der Benutzer kann erst dann auf eine HTML- Seite Einfluss nehmen, wenn der Kopfteil vollständig im Speicher liegt. So sind alle Funktionen geladen, bevor der Benutzer eine Aktion tätigen kann, die den Aufruf einer noch nicht bekannten Funktion nach sich ziehen würde.

6.2. Funktionen mit Rückgabewerten, rekursive Funktionen

Die Definition von Funktionen mit Rückgabewerten folgt dem Standardschema. Der Rückgabewert wird im Funktionsrumpf durch das *return*- Schlüsselwort markiert: es kann sich dabei um einen beliebigen Ausdruck handeln:

```
function multiply (x,y){
    return x*y;
}
```

JavaScript unterstützt Rekursion: lineare, repetitive, kaskadenartige und verschränkte Rekursion sind möglich.

6.3. Vordefinierte Funktionen

JavaScript bietet einige vordefinierte Funktionen, die zur *Core Language* gehören: *parseInt()*, *parseFloat()*, *isNaN()*, *escape()*, *unescape()* und *eval()*.

Mit Hilfe der Funktionen *parseInt()* und *parseFloat()* ist es möglich, eine Ganzzahl bzw. eine Fließkommazahl aus einem String zu filtern. Diese Methoden werden vor allem zum Filtern von Benutzereingaben eingesetzt.

Die Funktion *isNaN()* (is not a number) liefert einen booleschen Wert: das Ergebnis aus den Test, ob sein Argument vom internen Typ *Number* ist oder nicht.

Die Funktionen *escape()* und *unescape()* dienen der Ver- und Entschlüsselung von Eingaben. Einsatzmöglichkeiten werden im Abschnitt *Kommunikation zwischen HTML- Seiten* beschrieben.

Die Funktion *eval()* kann JavaScript- Code interpretieren. Mittels des folgenden Konstruktes kann eine Eingabe des Benutzers ausgewertet werden: `x = eval(„34.65+27.564“);`

7. Sprachprinzipien

7.1. Klassenbasierte Sprachen im Gegensatz zu Prototyp- basierten Sprachen

Klassenbasierte, objektorientierte Sprachen bieten zwei grundlegende Bestandteile: Klassen und Objekte. In einer Klasse werden Attribute und Methoden definiert. Eine Klasse ist ein abstraktes Konstrukt. Ein Objekt wird durch Instanziierung einer Klasse erzeugt. Es verfügt so über einen festgelegten Satz von Attributen und Methoden.

JavaScript ist dagegen eine *prototyp-basierte* Sprache. In derartigen Sprachen existieren nur Objekte. Darunter gibt es *prototypische Objekte*. Sie dienen als Vorlage für neue Objekte, definieren eine Vererbungshierarchie und ermöglichen das Hinzufügen von Eigenschaften zur ihren Instanzen noch zur Laufzeit. Prototypen gehören jeweils zu genau einem Funktionsobjekt. Alle Eigenschaften des Funktionsobjekts sind im zugehörigen Prototyp sichtbar. Über den Prototyp und die zugehörige Konstrukturfunktion erhalten die Instanzen alle vorhandenen Eigenschaften. Diese werden über das *this*- Schlüsselwort in der Konstrukturfunktion mit Initialwerten belegt. Jedes Objekt kann in JavaScript als Prototyp für neue Objekte dienen [Net98].

Zusätzlich zu den geerbten Eigenschaften des Prototyps kann jedes instanziierte Objekt zusätzlich seine eigenen Eigenschaften definieren. Das kann entweder schon bei seiner Erzeugung oder aber erst zur Laufzeit geschehen.

7.2. Der Umgang mit Objekten: Definition, Instanziierung, Zugriff

In klassenbasierten Sprachen wird eine Klasse in einer separaten Klassendefinition mit Konstruktormethode definiert. Mit ihrer Hilfe lassen sich Instanzen der Klasse mittels des *new*- Operators erzeugen.

In JavaScript definiert man ein Funktionsobjekt als Vorlage für ein prototypisches Objekt, aus dem weitere Objekte mit einem Satz von Initialwerten generiert werden. Jede gewöhnliche JavaScript- Funktion kann zur Objekterzeugung benützt werden.

Zur Objekterzeugung wird in JavaScript auf die *prototype*- Eigenschaft von Funktionsobjekten zurückgegriffen. Sie wird mit *prototype* bezeichnet. Diese Eigenschaft ist in allen Funktionsobjekten enthalten und verweist auf das, zu dem Funktionsobjekt gehörende prototypische Objekt. Existiert kein derartiges Objekt, so ist der Wert dieser Eigenschaft für das betreffende Funktionsobjekt gleich *null*. In allen erzeugten Objekten ist diese Eigenschaft ebenfalls vorhanden, hier wird sie *_proto_* bezeichnet.

Zur Fragestellung, wann ein prototypisches Objekt zu einem Funktionsobjekt angelegt wird, gibt es zwei Ansätze. Bis JavaScript 1.1 wurde ein prototypisches Objekt nur dann angelegt, wenn tatsächlich eine Anwendung des *new*- Operators auf das betreffende Funktionsobjekt erfolgte. Ab JavaScript 1.1 wird zu jedem definierten Funktionsobjekt sofort ein prototypisches Objekt angelegt.

Zur Objekterzeugung wird der *new*- Operator auf ein Funktionsobjekt angewandt: Die betreffende Funktion wird dadurch zu einer Konstrukturfunktion. Zu jeder Konstrukturfunktion existiert in JavaScript ein prototypisches Objekt. Das prototypische Objekt ist das Objekt, auf das die *prototype*- Eigenschaft eines Funktionsobjektes verweist. Es verfügt über alle Eigenschaften des zugehörigen Funktionsobjektes.

Der Prozess der Erzeugung von Objekten soll am folgenden Beispiel verdeutlicht werden:

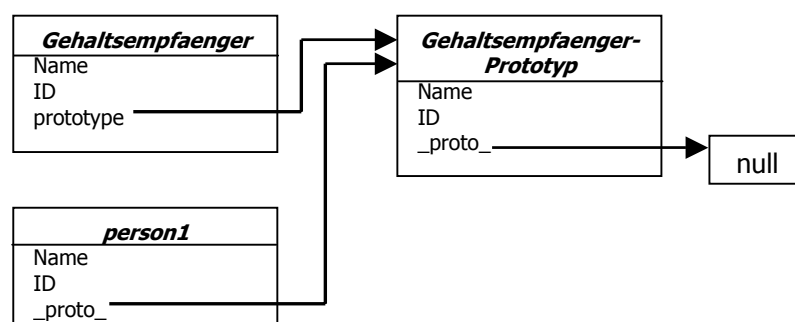
```
function Gehaltsempfaenger(name,id){ // die Funktion als Objekt
    this.Name = name || "John Doe";
    this.ID    = id || -1;
}
```

Durch Anwendung des *new*- Operators auf das Funktionsobjekt *Gehaltsempfaenger* wird die zugrunde liegende Funktion zur Konstrukturfunktion. Die Instanziierung erfolgt über das Funktionsobjekt *Gehaltsempfaenger* :

```
person1 = new Gehaltsempfaenger(); // Anlegen eines neuen Objektes mit Initialwerten
```

Nun verweist die *_proto_* - Eigenschaft des Objekts *person1* auf den Prototypen des Funktionsobjekts *Gehaltsempfaenger*. Es gilt folgende Beziehung:

```
person1._proto_ = Gehaltsempfaenger.prototype;
```



Der Zugriff auf Eigenschaften der Objekte erfolgt nach bekanntem Schema: *Objektname.Eigenschaft*
 Das Schlüsselwort *with* gestaltet den Zugriff auf mehrere Bestandteile eines Objektes komfortabler:

```
with (Objekt){
    Attribut_1 = Wert_1;...
    Attribut_n = Wert_n;
}

with (document){
    writeln("Erste Zeile");
    writeln("Zweite Zeile");
}
```

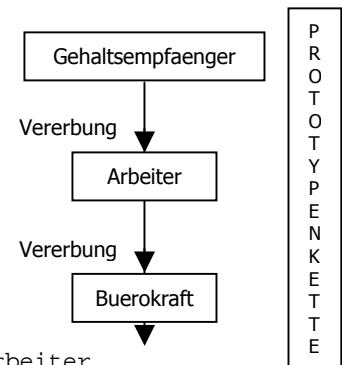
In JavaScript existieren neben den selbstdefinierten Objekten auch spezifische Objekte des Browsers und vordefinierte Objekte [vgl. 8. Spezielle Objekte].

7.3. Vererbung: Der *prototype* - Operator

In klassenbasierten Sprachen erzeugt man über das Konzept der Vererbung mittels eigens definierter Unterklassen eine Klassenhierarchie. Dabei erbt die Unterklasse alle Eigenschaften der Superklasse. JavaScript implementiert das Konzept der Vererbung auf anderem Weg: es ist möglich, ein prototypisches Objekt mit jeder beliebigen Konstrukturfunktion zu assoziieren. Im Gegensatz zur Generalisierungshierarchie in klassenbasierten Sprachen entsteht so eine *Prototypenkette*:

```
function Arbeiter(){
    this.Abteilung = "";
}
Arbeiter.prototype = new Gehaltsempfaenger;
// Arbeiter- Objekte erben Attribute von Gehaltsempfaenger

function Buerokraft(){
    this.ZimmerNummer = "";
    this.Aufgabenbereiche = [];
}
Buerokraft.prototype = new Arbeiter;
// Buerokraft- Objekte erben Attribute von Gehaltsempfaenger und Arbeiter
jim = new Buerokraft();
```



Im obigen Beispiel wird ein prototypisches Objekt zum Funktionsobjekt *Arbeiter* erstellt. Dieses wird mit dem prototypischen Objekt des Funktionsobjekts *Gehaltsempfaenger* assoziiert. Die Realisierung erfolgt über die *prototype*- Eigenschaft des *Arbeiter*- Funktionsobjektes. Im nächsten Schritt wird die Konstrukturfunktion *Buerokraft* mit dem prototypischen Objekt des Funktionsobjekts *Arbeiter* assoziiert, welches mit dem *Gehaltsempfaenger*- Prototypen verbunden ist.

Betrachtet man die *prototype*- und *_proto_*- Eigenschaften der beteiligten Objekte, so gelten in der vorliegenden Prototypenkette folgende Beziehungen:

```

jim._proto_ = Buerokraft.prototype;
jim._proto_._proto_ = Arbeiter.prototype; // Arbeiter ist ein Funktionsobjekt
jim._proto_._proto_._proto_ = Gehaltsempfaenger.prototype; // Gehaltsempfaenger auch
jim._proto_._proto_._proto_._proto_ = Object.prototype; // höchste Ebene
jim._proto_._proto_._proto_._proto_._proto_ = null;
  
```

Das Verhalten eines Objekts kann also von mehreren Definitionen abhängen. Ein Objekt wird aber unmittelbar nur aus einem Prototyp abgeleitet. Deshalb unterstützt JavaScript keine Mehrfachvererbung [Est01].

Wird lesend auf eine Objektkomponente zugegriffen, so wird die entsprechende Eigenschaft zuerst lokal im betreffenden Objekt gesucht. Bei negativem Suchergebnis wird die Prototypenkette anhand der *_proto_* - Eigenschaft der assoziierten Objekte bis zum Anfangsglied durchsucht.

7.4. Hinzufügen und Entfernen von Eigenschaften

In klassenbasierten Sprachen werden Klassen gewöhnlich zur Compile- Zeit erzeugt. Instanzen der Klasse werden entweder schon zur Compile- Zeit oder erst zu Laufzeit erzeugt. Zur Laufzeit ist es nicht mehr möglich, die Definition der Klasse zu ändern.

In JavaScript lassen sich noch zur Laufzeit Objektbestandteile von Objekten entfernen zu diesen hinzufügen. Eine Ausnahme stellen prototypische Objekte dar. Ihre vorhandenen Eigenschaften können nicht entfernt werden. Hinzufügen von Eigenschaften zum Prototypen erfolgt über folgendes Schema:

```
bestehendesFunktionsobjekt.prototype.neue_Eigenschaft = eine_neue_Eigenschaft;S
```

Fügt man ein Attribut oder eine Funktion zu einem prototypischen Objekt hinzu, aus dem eine Menge von Objekten hervorgegangen ist, so erhalten diese Objekte die neuen Eigenschaften hinzu:

```
person2 = new Buerokraft();
...
Gehaltsempfaenger.prototype.Alter = "18";
// person2 erbt das, zum Gehaltsempfaenger-Prototypen hinzugefügte Attribut.
```

Löschen von Objekteigenschaften ist über das *delete* - Schlüsselwort möglich.

7.5. JavaScript und die Paradigmen der Objektorientierung

„Aggregation ist in JavaScript (...) implementiert“ [Bon02]. Objekte können neben Attributen auch Methoden erhalten. In JavaScript stellen auch Funktionen Objekte dar. Somit lassen sich Objekte in anderen Objekten ablegen:

Wird in JavaScript ein Host- Objekt gelöscht, so zieht das die Löschung aller enthaltener Objekte nach sich.

Auch das Konzept der Assoziation ist in JavaScript vorhanden. „(Sie) kann als Grundprinzip der Sprache betrachtet werden“ [Bon02]. Objekte können auf andere Objekte verweisen.

```
function Name(fName, lName){
    this.firstName = fName;
    this.lastName = lName;
}

function Manager(dept, fName, lName){
    this.mydept = dept || "NODEPT";
    this.myName = new Name(fName, lName);
    this.print = printManager;
}

function printManager(){
    var display = this.mydept + „: “ + this.myName.firstName + “ “ + this.myName.lastName;
    document.writeln(display);
}

NellyF = new Manager ("LOGISTIK", "Nelly", "Franco");
NellyF.print();
```

Der Aspekt der Kapselung wird von JavaScript unterstützt. Daten und Operationen können zu einer Einheit zusammengefasst werden: einem Objekt. Am Quelltext wird dies durch den Punktoperator ersichtlich.

Laufzeit- Polymorphie ist ein Grundprinzip der Sprache, die durch Unterstützung des Konzeptes des *dynamic binding* gewährleistet wird [Bon02].

JavaScript bietet momentan keine Möglichkeit, die Sichtbarkeit, also den Zugriff auf Attribute oder Methoden eines Objekts von außen zu beschränken. Das von Parnas postulierte Geheimnisprinzip (Information Hiding) wurde nicht umgesetzt, da es kein Konzept zur Einschränkung des Zugriffs auf Eigenschaften von Objekten existieren.

JavaScript unterstützt also das Konzept der Kapselung und Aggregation, ohne dabei jedoch einem Mechanismus für Information Hiding bereit zu stellen. Assoziation wird uneingeschränkt unterstützt, Laufzeit- Polymorphie zählt zu den Grundprinzipien der Sprache. JavaScript unterstützt Vererbung mittels des Konzeptes der Prototypenkette. Somit werden alle relevanten Paradigmen der Objektorientierung erfüllt: JavaScript darf als objektorientierte Sprache bezeichnet werden.

7.6. Ausführung von JavaScript- Code

JavaScript- Code wird in HTML- Code eingebettet. Im Gegensatz zu Java- Applets, die in kompiliertem Zustand geladen werden, erfolgt bei JavaScript- Code kein Übersetzungsvorgang. Die JavaScript- Programmteile sind als solche innerhalb der HTML- Seite gekennzeichnet und werden vom Browser zur Laufzeit interpretiert. Damit ergibt sich ein Nachteil in der Ausführungsgeschwindigkeit gegenüber kompilierten Sprachen: Berechnungsvorgänge laufen in JavaScript um den Faktor 100 langsamer ab als in C.

8. Vordefinierte Objekte

8.1. Objekte der Core Language

In JavaScript gibt es vordefinierte Objekte: *Date*, *Math*, *String*, *Array*, *Boolean*, *Number*, *Function*, *Image*, und weitere. Sie gehören zur *Core Language*.

[Eis97] gibt einen Überblick über die Methoden des *date*-Objekts.

Das *Math*-Objekt ermöglicht Zugriff auf wichtige mathematische Funktionen. Auf die Methoden des *Math*-Objekts kann direkt zugegriffen werden: `x = Math.sin(12) + Math.PI`

Die Attribute und Methoden des *Math*-Objekts sind in [Eis97] vollständig aufgelistet.

Das *String*-Objekt muss ebenfalls über die Konstrukturfunktion instanziiert werden. Es stellt Methoden zur Manipulation von Zeichenketten und zur Formatierung bereit.

Mit Ausnahme der Methoden zur Manipulation von Zeichenketten wie *charAt()* oder *indexOf()* sind alle anderen Methoden des *String*-Objekts auch in HTML realisierbar. Beispielsweise ist der JavaScript-Ausdruck

```
„Home“.link(http://www.aHomepage.de)  gleichbedeutend mit der HTML Anweisung
<a href =“ http://www.aHomepage.de“> .
```

[Eis97] zeigt eine vollständige Auflistung der Methoden und Attribute des *String*-Objektes.

Die Datenstruktur *Array* wird in JavaScript durch das *Array*-Objekt repräsentiert. Durch Instanziierung wird ein Array mit der angegebenen Grösse angelegt: `myArray = new Array(100)`. Es ist möglich, ein Array bei seiner Deklaration zu initialisieren: `myArray = new Array(„Name1“, „Name2“)`; In diesem Fall wird die Grösse des Arrays über die Anzahl der Initialisierungsattribute definiert.

JavaScript erweitert Arrays dynamisch. Im folgenden Beispiel wird das Array zur Laufzeit erweitert:

```
myArray = new Array(10);
...
myArray[67] = Math.PI;    // keine Fehlermeldung beim Zugriff auf myArray[67]
```

In JavaScript stellen Arrays keine homogene Datenstruktur dar. Es ist möglich, Daten verschiedener interner Typen im selben Array unterzubringen. So können Integerwerte, Fließkommawerte, Strings oder auch Objekte in einem gemeinsamen Array abgelegt werden. Mit Hilfe des *typeof*-Operators kann in solchen Fällen der interne Typ einer Variable festgestellt werden: `var aType = typeof myArray[0];`

Es ist möglich, ein Feld nicht nur über Integer- Werte anzusprechen, sondern gespeicherte Werte an beispielsweise Strings zu koppeln. Derartige Arrays werden *assoziative Arrays* genannt [Koc97]:

```
var myArray = new Array();
myArray[„Car“] = „Yellow“;
```

Wegen des *loose typings* macht es in JavaScript prinzipiell keinen Unterschied, ob Variablen oder Objekte verwendet werden. Dennoch existieren Objekte *Boolean* und *Number*, denen explizit boolesche bzw. numerische Werte zugewiesen werden können.

8.2. Objekte der Client- Side- Erweiterung

Sobald eine Webseite von einem JavaScript- fähigen Browser geladen wurde, sind automatisch folgende Objekte vorhanden: *window*, *document*, *location* und *history*. Diese gehören nicht zur *Core Language*, sondern entstammen der *Client- Side*- Erweiterung [vgl. 1.3. Bestandteile der Sprache].

Das *window*-Objekt repräsentiert das Fenster, in dem eine Seite geladen wurde. Da jede HTML- Seite ein Fenster voraussetzt, ist dieses Objekt immer vorhanden. Folgende Liste ist ein Auszug der wichtigsten Methoden: *open()*, *close()*, *alert()*, *confirm()* und *prompt()*.

Die Verwendung der Methoden *open()* und *close()* soll kurz am folgenden Beispiel erläutert werden:

```
function oeffneFenster(){           // öffnet die angegebene Webseite in einem neuen Fenster
    neuesFenster = open(http://www.fmi.uni-passau.de);
}

function schliesseFenster(){       // schliesst die Seite
    neuesFenster.close();
}
```

Das *window*- Objekt muss nicht instanziiert werden, da es nach dem Laden der Seite bereits vorliegt. Das zu schliessende Fenster muss mit seinem Namen angesprochen werden, da der Aufruf der Funktion vom *Stammfenster* aus erfolgt. Mittels der formalen Parameter der *open()*- Methode ist es möglich, Einfluss auf die Erscheinung des zu öffnenden Fensters zu nehmen [Koc97].

Die Methoden *alert()*, *confirm()* und *prompt()* tragen einer fundamentalen Anforderung der Sprache JavaScript Rechnung: Verwirklichung einer komfortablen Interaktion mit dem Benutzer.

Die Methode *alert()* öffnet ein Hinweisfenster mit einer *OK*- Schaltfläche, *confirm()* lässt dem Benutzer eine Auswahlmöglichkeit zwischen *Ok* und *Abbrechen* welche den booleschen Rückgabewert bestimmt. Ein *prompt()*- Aufruf fordert den Benutzer zu einer Eingabe auf und liefert die Eingabe des Benutzers zurück. Es ist möglich, diese Eingabe mit einem Defaultwert vorzubeseetzen:

```
alert(„Hinweis:\n Falsche Formulareingabe. Bitte wiederholen.“);
userdecision = confirm(„Formulareingaben wirklich löschen?“)
usereingabe = prompt(„Bitte geben Sie die Grösse des neuen Fensters an:“,“400x400“);
```

Anmerkung: jedes *window*- Objekt verfügt über eine Liste seiner Frames: Diese können unter Angabe des Indexwertes angesprochen werden: `window.frames[Index]`

Das *document*- Objekt repräsentiert die aktuelle HTML- Seite. Es beinhaltet Methoden zur Gestaltung des Seiteninhalts. Dazu zählt die Methode *write()*, die eine textuelle Ausgabe in die aktuelle Seite schreibt. Mit dem *document*- Objekt, kann auf alle Komponenten einer HTML- Seite zugegriffen werden. Jede Komponente ist durch ein eigenes Objekt vertreten: es existieren Objekte für Schalter, Buttons, Bilder, Formulare, die Titelzeile oder die Hintergrundfarbe der Seite.

HTML- Seiten können *on- the- fly* erzeugt werden. Hier kommen die *window*- und *document*- Objekte zum Einsatz. Die neue Seite wird ganzheitlich auf dem Client erstellt, es entfällt Datenübertragung zwischen Client und Server.

Über das *location*- Objekt hat der Programmierer Zugriff auf vom Browser gespeicherte Informationen über URL, Server und Protokoll der aktuellen HTML-Seite. Eine Internet- Adresse wird dabei durch folgendes Schema dargestellt: *protocol // hostname : port pathname search hash*

Dabei beschreibt das Element *search* einen optionalen Suchstring, der an die eigentliche Adresse angehängt wird. Im Element *hash* kann ein Anker angegeben werden. Diese Elemente stellen die Attribute des *location*-Objektes dar. Zusätzlich kommen die Attribute *host* und *href* hinzu. Das *host*-Attribut stellt die Verbindung zwischen *hostname* und *port* dar, das *href*- Attribut beinhaltet die komplette Adresse des Dokuments.

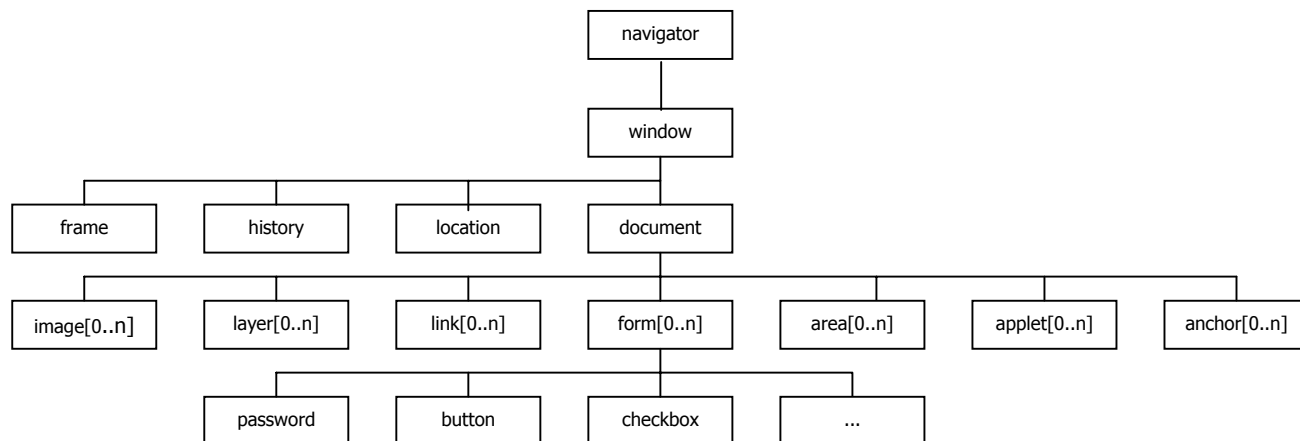
Das *location*- Objekt kommt vor allem dann zum Einsatz, wenn eine neue Webseite im selben Fenster geladen werden soll. Dazu wird dem *href*- Attribut des *location*- Objektes eine neue Adresse zugewiesen:

```
location.href = http://www.google.de;
```

Im *history*- Objekt sind alle WWW-Seiten chronologisch aufgeführt, die der Benutzer besucht hat. Dieses Objekt stellt die Grundlage für die Funktionalität der *Vorwärts*- und *Zurück*- Buttons des Browsers dar. JavaScript kann über die Methoden *back()*, *forward()* und *go()* auf diese Information zugreifen:

```
history.back(); ist hierbei semantisch äquivalent zu history.go(-1) .
```

Das *navigator*- Objekt stellt Information über den verwendeten Browser bereit. Über die Attribute *appName*, *appVersion* und *userAgent* lassen sich Bezeichnung, Hersteller, Versionsnummer und Information über den Client- Rechner in Erfahrung bringen um beispielsweise die Seitendarstellung individuell anzupassen. Zusammenfassend präsentiert sich die Client- seitige Objekt-Hierarchie in JavaScript wie folgt [Koc97]:



JavaScript ermöglicht dabei den Zugriff auf jedes Element in dieser Hierarchie. [Lam99] bietet einen genauen Überblick über die Objekte des Browsers.

9. Ereignisse

Zur Reaktion auf Benutzeraktionen existiert ein Modell zur Wahrnehmung, Differenzierung und Einleitung entsprechender Maßnahmen. Es basiert auf *Events* und *Eventhandlern*.

Jedes Event ist an bestimmte Objekte innerhalb der HTML- Seite gebunden. Wird beispielsweise ein Button einer HTML- Seite betätigt, so sendet der Browser eine Nachricht an das behandelnde Skriptsprachen- Programm. In dieser Nachricht ist die Information enthalten, welche Aktion an welchem Objekt der HTML-Seite ausgelöst wurde. Nun reagiert der entsprechende Eventhandler im Programm.

Exemplarisch soll das Event *click* genauer betrachtet werden.

Das Event *click* wird immer dann ausgelöst, wenn der Benutzer eine der Maustasten drückt. Es kann mit folgenden Objekten der Objekthierarchie in Verbindung stehen: *button*, *checkbox*, *radio*, *link*, *reset*, *submit*. Der angesprochenen Eventhandler ist für diesen Fall *onClick*.

Im Folgenden wird bei Klick eines Buttons eine Hinweismeldung ausgegeben:

```
<html>
  <input type = "button" value = "Press" onClick= "alert('Button pressed!')">
</html>
```

Der Eventhandler wird dabei ohne Definition eines JavaScript-Bereiches in den entsprechenden HTML- Statement eingebunden.

Anmerkung: der Internet Explorer besitzt seit Version 4 ein Ereignismodell, das fast vollständig inkompatibel zur Netscape- Variante ist. Daher kann nicht immer davon ausgegangen werden, dass alle Ereignisse auf beiden Browsern gleich abgehandelt werden [Wen02].

10. Kommunikation zwischen HTML- Seiten

Es gibt Problemstellungen, die einen gewissen Grad an Kommunikation zwischen verschiedenen HTML- Seiten erfordern. Möchte man beispielsweise die Mailadresse eines Besuchers in Erfahrung bringen, so kann man dies mittels eines Formulars und der entsprechenden Eingabe erreichen. Diese Information ist aber nur auf der Seite verfügbar, auf der die Eingabe getätigt wurde.

Um über derartige Informationen über längeren Zeitraum hinweg *global* verfügen zu können, also über alle Seiten der Dokumentenhierarchie hinweg, gibt es die folgenden beiden Möglichkeiten.

10.1. Kommandoparameter

Information lässt sich an die URL einer Seite anhängen und versenden. Suchmaschinen, hängen den Suchstring mit einem *?* an die Adresse an. In JavaScript bedient man sich dazu des *search*- Strings [vgl. 8.1. Objekte des Browsers].

Bestimmte Zeichen, wie Leerzeichen, dürfen in einer Adresse nicht vorkommen. Abhilfe schaffen die aus [6. Funktionen] bekannten Vertreter *escape()* zum Verschlüsseln unerlaubter Zeichen auf der versendenden Seite und *unescape()* zum Wiederherstellen derselben in der empfangenden Seite.

Die Funktion zum Übermitteln der Information von Seite *page1.htm* an die Seite *page2.htm* präsentiert sich wie folgt:

```

page1.htm:    function sendData(uebergabe){
                Location.href = „page2.htm?“ + escape(uebergabe);
            }

page2.htm:    function getData(){
                uebergabe = location.search // Daten aus dem search- String holen
                uebergabe = uebergabe.substring(1, uebergabe.length); // ? wegschneiden
                document.write(unescape(uebergabe)); // entschlüsseln
            }

```

Kommandozeilenparameter werden bei der Übergabe kurzer Botschaften aufgrund der einfachen Realisierung bevorzugt. Bei komplexeren Sachverhalten wird auf *Cookies* zurückgegriffen.

10.2. Cookies

Cookies sind Daten, die von www- Seiten auf dem Client- Rechner gespeichert werden dürfen. Alle Cookies werden in einer Datei abgelegt. Sie sind die einzigen Daten, die JavaScript auf dem Client- Rechner speichern darf. Ein Cookie muss mindestens aus einem Namen zur Identifizierung und einem Wert bestehen.

Um mit Cookies zu arbeiten benötigt man Zugriff auf das *cookie*- Objekt. Angelegt wird ein Cookie mit folgendem Konstrukt: `document.cookie = Name_des_Cookies = Wert;`

Name und Wert des Cookies lassen sich unter Verwendung der *unescape()* Funktionen gewinnen:

```
return (unescape(document.cookie.substring(position, document.cookie.length)));
```

Das Attribut *position* identifiziert dabei den Index, ab dem der Wert des Cookies beginnt.

Ein Server kann bis zu 20 Cookies auf einem Client speichern. Deshalb muss das gesuchte Cookie mittels seines Namens identifiziert werden. Es gibt diverse Funktionen zum Umgang mit Cookies, die als Public Domain veröffentlicht im Internet verfügbar sind, wie in [Jup03]. Hieraus wird die genaue Vorgehensweise ersichtlich.

Cookies besitzen ein internes Haltbarkeitsdatum. Dieses wird unter Verwendung des *expires*- Wortes angegeben:

```
document.cookie="Name_des_Cookies = Wert" + "expires=Fri, 31 Dec 2003 00:00:00 GMT";
```

Wird dieses Konstrukt nicht in den Wert des Cookies eingebunden, so bleibt es nur bis zum Verlassen der Webseite erhalten. Generell sind Cookies nur von dem Server lesbar, der das Cookie auch gesetzt hat.

11. Erweiterung der Sprache JavaScript: LiveWire

Die bisher vorgestellten Objekte und Konstrukte entstammen dem *Client- Side- JavaScript*. Damit kann Code erstellt werden, der auf dem Client ausgeführt wird.

JavaScript- Programme können auch auf einem Server ausgeführt werden: *Server- Side- JavaScript*.

Server- Side- JavaScript ist bislang auf dem *FastTrack*- [Nfs00] und *Enterprise*-Server [Nes00] von Netscape lauffähig [Koc97]. Zudem benötigt man Erweiterungspakete: *LiveWire*.

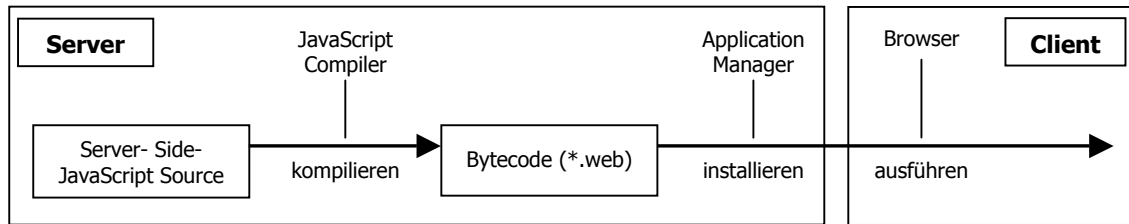
11.1 LiveWire- Erweiterungen

Das *LiveWire*- Paket beinhaltet einen Compiler, der JavaScript- Programme und HTML- Seiten mit eingebettetem JavaScript- Code in plattformunabhängigen Bytecode umwandelt, eine Bibliothek um SQL- Zugriff auf die gängigen Datenbanksysteme zu ermöglichen und verschiedene Werkzeuge zum Erstellen und Verwalten von Webseiten. Optional ist das Datenbanksystem *Informix-Online Workgroup Database* verfügbar.

Die *LiveWire Server Extension* wird zudem benötigt. Sie stellt Objekte zur Erstellung von *LiveWire*- Applikationen bereit.

11.2. Server- Side- JavaScript Applikationen

Folgende Arbeitsschritte fallen an, um *Server- Side- JavaScript*- Applikationen ausführen zu können:



Fertige Applikationen können auf dem Server über ein Tool aus dem *LiveWire*- Paket, den *Application Manager*, oder direkt unter Angabe der entsprechenden URL gestartet werden. Der Benutzer ruft dazu mit seinem Browser die Applikation auf. Der Server erstellt eine HTML- Seite, wobei er nur den server- seitigen JavaScript- Code ausführt.

Vorhandener client- seitiger Code kommt auf dem Server nicht zur Ausführung. Er wird in das HTML- Dokument eingebunden. Dieses wird an den Client gesandt. Hier wird eine Seite mit den HTML-Tags und dem client- seitigen Code aufgebaut.

Server- Side- JavaScript- Code wird durch Einfassen in `<server>`- Tags oder mittels *Backquotes* in den HTML- Code eingebettet:

```

<server>                                <form>
    write(„Server-Side Code!“)          <input type="text" value='request.agent'>
</server>                                </form>
// server- Tags                          // Backquotes
  
```

11.3. *LiveWire* - Objekte und vordefinierte Funktionen

Server- Side- JavaScript benötigt neue Objekte: *server*, *project*, *client* und *request*.

Das *server*- Objekt ist nur einmalig vorhanden, bis der Server gestoppt wird. Es ermöglicht den Datenaustausch zwischen verschiedenen Applikationen und Clients.

Für jede Applikation existiert ein *project*- Objekt. Es ist für den Datenaustausch zwischen verschiedenen Clients verantwortlich.

Für jeden Client, der mit dem Server in Verbindung steht existiert ein *client*- Objekt, das Daten über den spezifischen Client bereitstellt.

Das *request*- Objekt wird erzeugt, nachdem eine Anfrage an den Server gestellt wurde. So ist beispielsweise die Übertragung von Formulardaten vom Client zum Server möglich.

Neben diesen existiert das *File*- Objekt, mit dem der lesende und schreibende Zugriff auf Dateien des Servers realisiert wird und weitere.

Neben den neuen Objekten wird auch ein Satz neuer vordefinierter Funktionen bereitgestellt. Damit wird der Zugriff auf Datenbanksysteme oder das Server- File- System möglich.

Trotz der Erweiterungen hat sich *Server- Side- JavaScript* im Wettbewerb nicht durchgesetzt.

12. LiveConnect: Java- JavaScript- Kommunikation

LiveConnect ist eine Erweiterung des Browsers zur Kommunikation zwischen Java und JavaScript. Ab Version 3.0 des Netscape Navigator ist *LiveConnect* verfügbar.

Von JavaScript aus hat man über das folgende Schema vollständigen Zugriff auf alle vordefinierten Java- Packages und Klassen: `Packages.vollstaendiger_Name_der_Klasse`, also z.B. `Packages.java.lang.System`

Bei Verwendung der Packages *java*, *sun* oder *netscape* ist der Bezeichner *Packages* optional. In der Praxis empfiehlt es sich, den vollständigen Klassennamen in einer Variablen abzulegen:

```
function toWrite(name){
    System = java.lang.System;
    System.out.println(„Name: “ +name);
}
```

Durch Import des Packages *netscape.javascript.** kann von Java auf JavaScript- Objekte zugegriffen werden. Die wichtigsten Methoden finden sich in der Klasse *netscape.javascript.JSObject*. Über das *JSObject* und seine Methoden lassen sich einzelne Objekte der HTML- Seite manipulieren.

JavaScript und Java variieren in ihren Datentypen und Objekten stark. Deshalb kann nicht immer für eine verlustfreie Umwandlung gebürgt werden [Koc97].

Zugriffsmöglichkeit aus JavaScript heraus auf die Methoden von Java- Applets besteht über die Objekthierarchie: alle Applets einer Seite sind unter dem *document.applets*- Objekt ansprechbar. Existiert ein Applet mit dem Namen *anApplet*, welches eine Methode *aMethod* enthält, so wird diese wie folgt angesprochen:

```
document.anApplet.aMethod();
```

Ein Applet kann auf JavaScript- Objekte genau dann zugreifen, wenn das Applet im *<applet>*-Tag mit dem Attribut *mayscript* gekennzeichnet wird.

So wird der Zugriff auf JavaScript- Objekte der Seite ohne Wissen des Programmierers unterbunden.

13. Sicherheitsaspekte

13.1. Sicherheitslücken

In frühen JavaScript- Versionen bestanden schwerwiegende Sicherheitslücken. So war lesender Zugriff auf das File- System des Clients im Navigator 2.0 möglich. E- Mail- Adressen konnten ausgespäht und benützt werden. Auch die history- Liste konnte ausgelesen werden. Die Einführung des Navigator 3.0 (JavaScript 1.1) beseitigte diese Sicherheitslücken.

Im Laufe der Zeit traten zahlreiche neue Fehler auf. Die in [Nsn03] genannten Lücken betreffen vor allem das Tracking der Bewegungen eines Benutzers im Netz und das Abfangen von Eingaben in Formularen.

Über behobene und bestehende Sicherheitslücken sind Pressemitteilungen verfügbar. Diese beziehen sich zumeist auf die über die Zeit weiter bestehenden Gefahren, denen sich der Anwender bei JavaScript-Aktivierung ausgesetzt sieht. Dazu zählen die Möglichkeit, beliebig viele Fenster mit *alert*- Meldungen zu öffnen, der Ausgangspunkt für *Denial- of- Service*- Attacken, ebenso die Möglichkeit, Eingabefenster zu simulieren und so Authentifizierungsdaten des Benutzers zu erschleichen.

Deshalb existieren Empfehlungen zur Deaktivierung von JavaScript. Diese wurden ausgegeben vom *Bundesministerium für Wirtschaft und Technologie*, *Bundesministerium der Innern*, *Bundesamt für Sicherheit in der Informationstechnik* und der *Regulierungsbehörde für Telekommunikation und Post*.

13.2. Sicherheitsmodelle

Es existieren verschiedene Sicherheitsmodelle, um die bei Ausführung von JavaScript- Code entstehen Gefährdungen zu beherrschen. Diese sollen in die künftigen Entwicklungsstufen von Browser und Sprache miteinbezogen werden:

Die *Same Origin Policy* erzwingt, dass ein von einer Quelle geladenes JavaScript- Programm bestimmte Zugriffe innerhalb des Browsers nicht ausführen darf.

Durch *Data Tainting* ist es möglich, dass Fenster auf Objekte anderer Fenster zugreifen können, unabhängig von der Quelle der Fenster. Dabei sollen die, dem Zugriff ausgesetzten Objekte vom Autor als privat gekennzeichnet werden können. Zugriffe werden dann nur nach Rückfrage gestattet.

Die *Signed Script Policy* basiert auf der Möglichkeit, Objekte digital signieren zu lassen. Ein signiertes JavaScript- Programm kann erweiterte Zugriffsrechte auf Ressourcen des lokalen Rechners erhalten.

14. Praktische Arbeit mit JavaScript: Editoren und Entwicklungstools

Im Gegensatz zu HTML gibt es bei der JavaScript-Programmierung keinen großen Markt für Editoren. Viel genutzt sind einfache Texteditoren. Alternativ dazu gibt es eine kleine Menge von Shareware- Editoren. Die geläufigsten Vertreter sind *Ultraedit* [Ult03], *Programmer's File Editor* [Phi03], *NoteTab* [Foo03] und *BBEEdit* [Bar03].

Daneben existieren Netscape- eigene Entwicklungstools wie *Visual JavaScript 1.0* [Nvj99], der *Netscape JavaScript Debugger* [Ndj97] oder die im Netscape Navigator enthaltenen *JavaScript Console*. Microsoft setzt das, im Internet Explorer enthaltene *Web Development Environment* dagegen.

15. Entwicklungstendenzen

Zu den, im Browser interpretierten Sprachen nach *ECMA- 262- Standard*, haben sich in den letzten Jahren einige weitere Sprachen gesellt. Vertreter sind beispielsweise die kostenlose Variante *DMDScript* [Dig02] und die kommerzielle Variante *ScriptEase* [Nom02].

Ihre Besonderheit besteht darin, JavaScript- Code in einer separaten Datei abzulegen und ihn unabhängig von einem Browser in einer Textkonsole auszuführen.

Genauso ist es aber auch möglich, deren Interpreter als Standardinterpreter des Webbrowsers einzurichten. Für die Sprachen existieren Entwicklungsumgebungen, ähnlich denen für die Sprachen Java oder C++.

Gegenwärtig arbeitet das *ECMAScript technical committee* an einer Überarbeitung des gegenwärtigen Standards: *ECMAScript Edition 4*. Dieser wird getypte Variablen und ein Klassen- Konzept beinhalten.

Von Netscape existiert ein Vorschlag zur Umsetzung unter dem Titel *ECMAScript 4 Netscape Proposal*, der in eine neue JavaScript Version (2.0) münden wird. Microsoft arbeitet an einer Implementierung dessen unter dem Namen *JScript.NET*.

16. Bewertung der Sprache JavaScript

JavaScript eröffnet zahlreiche neue Möglichkeiten für die Gestaltung von Webseiten. Die Interaktion des Benutzers mit einer Webseite wird wesentlich komfortabler. Sehr leicht erlangt der Programmierer die Kontrolle über alle Seitenelemente. Mit den Methoden zum Informationsaustausch zwischen verschiedenen Seiten werden die gestellten Anforderungen an die Sprache erfüllt.

JavaScript verfügt über ein schlankes Objektmodell. Der prototyp- basierte Ansatz ist mächtig, ermöglicht aber nicht die gesamte Funktionalität vollständiger klassenbasierter und objektorientierter Sprachen. Der eingeschränkte Verwendungszweck der Sprache rechtfertigt diese Abstriche. Die für die Umsetzung des Prototypen- Konzeptes notwendige schwache Typisierung im Zusammenspiel mit der Einschränkung auf drei intern verwaltete Datentypen können den Verlust von Information in der Zusammenarbeit mit Java nach sich ziehen.

Unterschätzt wird der Umfang der Sprache: für wenig versierte Programmierer dürfte es einige Zeit in Anspruch nehmen, die Grundlagen der Sprache zu erlernen. Der „Nichtprogrammierer“ [Hes97] kann somit nicht zum Anwenderkreis der Sprache gezählt werden.

Über die zahlreichen JavaScript- durchsetzten Webseiten hinweg ist zu beobachten, dass der Einsatz von JavaScript oft in technisch anspruchsvolle, aber wenig informative Seiten mündet. Die effiziente Präsentation von Information als eigentliche Aufgabe einer Webseite darf nicht vernachlässigt werden.

Letztlich besteht eine Gratwanderung zwischen Sicherheit und Komfort: es liegt im Auge des Betrachters, und ist abhängig vom Verwendungszweck, ob das Risiko von Sicherheitslücken, die durch den Einsatz von JavaScript entstehen, durch seine vielfältigen, HTML- bereichernden Möglichkeiten aufgewogen wird.

17. Zusammenfassung

JavaScript ist eine objektorientierte, prototyp- basierte, Skriptsprache mit schwacher Typisierung, deren Verwendungszweck in der Übernahme von Kontroll- und Kommunikationsfunktionen innerhalb einer HTML- Seite liegt. Dazu stellt JavaScript Zugriffsmöglichkeiten auf alle Objekte einer HTML- Seite bereit. Selbst verfügt die Sprache zwar nur über ein kleines Objektmodell mit Vererbung und Kapselung, jedoch ohne Realisierung des Geheimnisprinzips. JavaScript kann vom Browser auf dem Client interpretiert werden, hier spricht man von *Client- Side- JavaScript*, oder auch in Form von Bytecode als Applikation auf einem Server installiert und ausgeführt werden. Hier ist die Rede von *Server- Side- JavaScript*.

JavaScript besitzt keine Grafikfähigkeit, es kann selbst keine geometrischen Objekte zeichnen. JavaScript verfügt nicht über Multithreading oder Netzwerkkapazitäten.

Syntaktisch orientiert sich die Sprache stark Java, dennoch ist sie keine Untermenge davon. Die Kommunikation zwischen beiden Sprachen wird über eine browsereigene Erweiterung ermöglicht.

Die Vorzüge dieser Sprache liegen in der flexibleren Interaktion mit dem Benutzer, der Animation von Bildschirminhalten, Formularüberprüfungen, und der Möglichkeit, kleine Anwendungen innerhalb einer HTML- Seite zur Verfügung zu stellen. Damit werden gleichzeitig Sicherheitslücken eröffnet. Aus diesem Grunde wird vielfach vor dem Einsatz der Sprache gewarnt.

Obwohl JavaScript als eingebettete Sprache konzipiert wurde, existieren Interpreter, die eine Ausführung von allein stehendem JavaScript- Code in einer Textkonsole ermöglichen. Sie spielen jedoch bislang keine Rolle auf dem Markt.

Trotz aller Nachteile erfreut sich JavaScript großer Beliebtheit vor allem unter privaten Web- Entwicklern. Es stellt die, im Moment meistgenutzte Web- Skriptsprache dar.

18. Quellenverzeichnis

- [And97] Andreessen Marc: JavaScript comes of age. o.V., o.O 30.04.1997. Abrufbar im Internet:
URL: <http://wp.netscape.com/comprod/columns/techvision/javascript.html>. Stand: 13.10.2003.
- [Bar03] Bare Bones Software: Bare Bones Software :: Products :: BBEdit. 2003.
URL: <http://www.barebones.com/products/bbedit/index.shtml>. Stand: 14.10.2003.
- [Bon02] Bongers, Frank: Referenz: Objekte. 2002.
URL: <http://home.pfaffenhofen.de/internetschule/javascript/referenz/objekte-referenz.html>
Stand: 23.09.2003.
- [Dig02] Digital Mars: DMDScript. 2002. URL: <http://www.digitalmars.com/dscript/>. Stand: 10.10.2003.
- [Dom03] Netscape: DOM Central. 2003. URL: <http://devedge.netscape.com/central/dom/>. Stand: 10.09.2003.
- [Ecm99] European Computer Manufactures Association: ECMAScript Language Specification. 1999.
URL: <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>. Stand: 13.08.2003.
- [Eis97] Eisenmenger, Richard: JavaScript. Pearson Education, München 1997, S.109ff.
- [Est01] Esterline, Albert C.: Internet Technologies. o.V., NC, USA 2001.
URL: <http://www.ncat.edu/~esterlin/c600s01/Notes/JSObjectModel.pdf>. Stand: 27.09.2003.
- [Fri00] Friedrich, Bernhard: JavaScript. 30.09.2000.
URL: http://www20.pair.com/doozer/guides/javascript/js_003.html. Stand: 28.07.2003.
- [Foo03] Fookes Software: Award-Winning NoteTab Text Editors and HTML Editors. 23.05.2003.
URL: <http://www.notetab.ch>. Stand: 13.10.2003.
- [Har00] Harrison, Graham Paul: Dynamic Web Programming. Prentice Hall Inc., New Jersey, NJ, USA 2000, S.51.

- [Hes97] Hesser, Mike: JavaScript. 1997.
URL: http://www-vs.informatik.uni-ulm.de/Lehre/Seminar_Java/ausarbeitungen/JavaScript/index.htm.
Stand: 28.07.2003.
- [Iso03] International Organization for Standards: ISO - International Organization for Standardization.
03.10.2003. URL: <http://www.iso.ch/iso/en/ISOOnline.frontpage>. Stand: 14.10.2003.
- [Jup03] Jupitermedia Corporation: The JavaScript Source: Cookie Scripts. 2003.
URL: <http://javascript.internet.com/cookies/>. Stand: 28.09.2003.
- [Koc97] Koch, Stefan: JavaScript. dPunkt Verlag, Heidelberg¹ 1997, S. 101ff.
- [Lam99] Lamprecht, Stephan: Programmieren für das World Wide Web. Carl Hanser Verlag,
München, Wien 1999, S. 201ff.
- [Mic03] Microsoft: Windows Script. 2003. URL: <http://msdn.microsoft.com/scripting/>. Stand: 13.10.2003.
- [Moh03] Mohr o.A.: Überbetriebliche Datenkommunikation. 10.06.2003.
URL: http://www.winfo.rwth-aachen.de/inhalte/archiv/download/material/UEDK/UEDK_2003_07_10.pdf. Stand: 10.08.2003.
- [Nes00] Netscape: Netscape Enterprise Server. 2000.
URL: <http://wp.netscape.com/enterprise/v3.6/>. Stand: 14.10.2003.
- [Net00] Netscape Communications Corporation: Core JavaScript Guide 1.5: About this Book. 28.09.2000.
URL: <http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/preface.html>.
Stand: 20.08.2003
- [Net03] Netscape: W3C Standards Support in IE and the Netscape Gecko Browser Engine. 2003.
URL: <http://wp.netscape.com/browsers/future/standards.html>. Stand: 10.09.2003.
- [Net98] Netscape : Core JavaScript Guide. 30.10.1998.
URL: <http://devedge.netscape.com/library/manuals/2000/javascript/1.5/guide/intro.html>.
Stand: 20.08.2003.
- [Net99] Netscape Communications Corporation: JavaScript Overview. 27.05.1999.
URL: <http://devedge.netscape.com/library/manuals/2000/javascript/1.3/guide/intro.html>. Stand: 20.08.2003.
- [Nfs00] Netscape: Netscape FastTrack Server. 2000.
URL: <http://wp.netscape.com/fasttrack/v3.0/>. Stand: 14.10.2003.
- [Njd97] Netscape: Netscape JavaScript Debugger. 1997.
URL: <http://wp.netscape.com/eng/Tools/JSDebugger/relnotes/relnotes11.html>. Stand: 14.10.2003.
- [Nom02] Nombas, Inc.: Nombas, Inc. - Technology Leaders in JavaScript and ECMAScript. 2002.
URL : <http://www.nombas.com/us/>. Stand : 14.10.2003.
- [Nsn03] Netscape: Netscape Security News Archive. 2003.
URL: <http://wp.netscape.com/security/notes/index.html>. Stand: 14.10.2003.
- [Nvj99] Netscape: Netscape Visual JavaScript and Visual JavaScript Pro. 1999.
URL: <http://wp.netscape.com/enterprise/vjs/>. Stand: 14.10.2003.
- [Phi03] Phillips, Allan: Programmer's File Editor Home Page. 2003.
URL: <http://www.lancs.ac.uk/people/cpaap/pfe/>. Stand: 10.10.2003.
- [Röt02] Röttgers, Janko: Mocha oder Espresso?. 15.08.2002.
URL: <http://www.heise.de/tp/deutsch/inhalt/te/12905/1.html>. Stand: 14.11.2003.
- [Rüe01] Rüegg, Michael: Security - Java, JavaScript und ActiveX. 2001.
URL: <http://www.irongate.ch/security/java.htm>. Stand: 04.08.2000.
- [Ult03] IDM Computer Solutions, Inc: Text Editor - HEX Editor - HTML Editor - Programmers Editor – UltraEdit. 2003.
URL: <http://www.ultraedit.com>. Stand: 14.10.2003.
- [Wen02] Wenz, Christian: JavaScript. Galileo Press, o.O⁴ 2002. Abrufbar im Internet:
URL: <http://www.galileocomputing.de/openbook/javascript/>. Stand: 03.08.2003. Kap. 15.
- [Www03] W3C DOM Working Group: Document Object Model DOM. 11.08.2003.
URL: <http://www.w3.org/DOM/>. Stand: 13.08.2003.