Das Common–Gateway–Interface (CGI)

Seminar

„Web–Based Programming“

an der

TU Berlin

von Thorsten Schultzke

und Lars Oergel

28. Februar 1999

Inhaltsverzeichnis

1 Einleitung

Zu Beginn der Entwicklung des WWW hatte man lediglich die Möglichkeit, statisch in der Sprache HTML geschriebene Seiten über das Hypertext–Transfer–Protokoll (HTTP) abzurufen. Bald zeigten sich jedoch die Einschränkungen dieses Konzeptes: Das WWW stellte eine Einbahnstraße für Daten dar, es gab keinerlei Interaktionsmöglichkeit mit den Benutzern.

Mit dem *Common Gateway Interface* (CGI) wurde ein Standard geschaffen, mit dessen Hilfe sog. CGI–Programme oder -Skripte, die auf dem gleichen Rechner laufen wie der Web–Server, Daten von einem Web–Browser entgegennehmen und Ergebnisse an ihn zurückliefern können. Dadurch wurden erstmals echte interaktive Anwendungen im WWW realisierbar, z.B.

- Suchmaschinen,
- Web–Kataloge,
- Anwendungen mit Datenbankzugriffen (z.B. Produktkataloge),
- E-Commerce–Anwendungen,
- Gästebücher und
- Zugriffszähler.

Bei der Verarbeitung von Daten mit Hilfe eines CGI–Programmes ergibt sich folgender Ablauf (s. Abbildung 1):

1. Der Client, d.h. ein Web–Browser, schickt z.B. Formulardaten, die der Benutzer in ein HTML–Formular eingegeben hat, an den Web–Server.
2. Der Web–Server startet das CGI–Skript und übergibt diesem die Eingabedaten als Parameter.
3. Das CGI–Programm verarbeitet die Daten, tätig evtl. Datenbankabfragen oder dergleichen und liefert die Ausgabe an den Web–Server zurück.
4. Der Web–Server reicht das Ergebnis an den Client weiter.

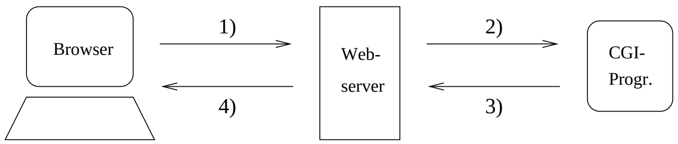

Abbildung 1: Ablauf eines CGI–Programmaufrufes

Wir betrachten im Folgenden die einzelnen Aspekte von CGI–Programmen in der „natürlichen", chronologischen Reihenfolge: Aufruf, Übermittlung der Eingabedaten, Implementierung und Rücklieferung der Ergebnisse. Im Anschluß machen wir uns ein paar Gedanken über die Sicherheit eines Systems im Zusammenhang mit CGI–Programmen und stellen kurz die Konzepte von Server Side Includes (SSI) und den Personal Homepage Tools (PHP) vor, bevor wir mit einer zusammenfassenden Sammlung der Vor- und Nachteile der CGI–Programmierung schließen.

2 Aufruf von CGI–Programmen

Wir wollen nun die verschiedenen Möglichkeiten besprechen, wie man ein CGI–Programm aufrufen kann.

2.1 Hyper–Links

Man kann den Aufruf eines CGI–Skriptes einfach in einen Hyperlink „verpacken". Dies geschieht folgendermaßen:

```
<a href="http://server/cgi-bin/cgi-prog">Starte CGI-Programm!</a>
```

Klickt der Benutzer diesen Link an, wird das CGI–Programm gestartet, und der Anwender bekommt das vom Programm ausgegebene Resultat als neue Webseite angezeigt.

2.2 Grafikverweise

Da CGI–Programme u.a. auch Bilder als Ergebnis liefern können[1], kann man sie anstelle einer Bilddatei aufrufen. Dies ist eine der gebräuchlisten Aufrufarten von CGI–Programmen, da hiermit die beliebten Seitenzähler realisiert werden können: Das Programm verwaltet einen internen Zähler (z.B. in einer Datei), der bei jedem Aufruf um eins erhöht wird. Aus dem Zählerstand wird eine kleine Grafik berechnet, die oftmals optisch einem analogen Bandzählwerk nachempfunden ist. Diese Grafik wird als Ergebnis des CGI–Skriptes zurückgeliefert:

```
<img src="/cgi-bin/counter">
```

Dabei ist allerdings zu beachten, daß der Seitenzähler nur aktualisiert wird, wenn der Browser auf automatisches Laden von Bildern eingestellt ist, da ja ansonsten dieser Verweis ignoriert und somit das CGI–Programm überhaupt nicht aufgerufen wird.

2.3 Automatisches Laden via Meta–Statement

HTTP definiert einen Mechanismus zur Weiterleitung des Browsers an eine andere als die angeforderte URL. Dabei kann statt auf eine „normale" Webseite auch auf ein CGI–Programm verwiesen werden:

```
<meta http-equiv="refresh" content="0; URL=/cgi-bin/welcome">
```

Die „0" steht dabei für die Wartezeit in Sekunden, nach der der Browser automatisch die angegebene Seite lädt bzw. das CGI–Skript aufruft.

[1] s. dazu auch Kapitel 5

2.4 Server Side Includes

Moderne Web–Server bieten die Ausführung sog. *Server Side Includes* an, mit denen ebenfalls CGI–Programme aufgerufen werden können. So würde das Seitenzählerbeispiel von eben mit Hilfe von Server Side Includes folgendermaßen aussehen:

```
<!--#exec cgi="/cgi-bin/counter" -->
```

Weitere Informationen zu Server Side Includes befinden sich im Kapitel 7.

2.5 HTML–Formulare

Die komfortabelste Art, um ein CGI–Programm aufzurufen und ihm auch gleich Parameter mitgeben zu können, stellen HTML–Formulare dar. Innerhalb eines HTML–Formulares kann man unterschiedliche Arten von Feldern zur Dateneingabe definieren, außerdem läßt sich mit dem *action*–Tag festlegen, ob die Daten per E–Mail verschickt werden sollen (z.B. `action="mailto:lobyte@cs.tu-berlin.de"`) oder ob ein CGI–Programm aufgerufen werden soll (z.B. `action="/cgi-bin/cgi-prog"`), das die eingegebenen Daten direkt verarbeiten und ein entsprechendes Ergebnis umgehend zurückliefern kann. Die E–Mail–Alternative ist insbesondere dann sinnvoll, wenn der Webspace–Provider z.B. aus Sicherheitsgründen[2] die Ausführung eigener CGI–Programme verbietet.

Da eine detaillierte Vorstellung mit Syntaxerklärung zur Erstellung eines Formulares den Rahmen dieser Ausarbeitung sprengen würde, beschränken wir uns hier auf die Vorstellung der Konzepte und Möglichkeiten von Formularen[3]. Allen Eingabefeldern ist gemein, daß sie einen formularweit eindeutigen Namen zu bekommen haben, den der Ersteller des Formulares vorgeben muß. Der in ein bestimmtes Formularfeld eingebene Text oder die dort getroffene Auswahl wird dann zur Übermittlung der Daten dem jeweiligen Formularfeldnamen zugeordnet, so daß sich dadurch die sog. *Namens-/Wertepaare* ergeben[4].

Als Eingabefelder stehen im einzelnen zur Verfügung:

- Einzeilige Eingabefelder

  Diese erlauben die Eingabe einer einzigen Zeile Text, deren maximale Länge im Formular festgelegt werden kann. Zudem kann ein Defaulttext angegeben werden, der sich bei Anzeige des Formulares bereits im Textfeld befinden soll.

- Eingabefelder für Paßwörter

  Eine Variante von einzeiligen Eingabefeldern sind Eingabefelder für Paßwörter. Der in einem solchen Feld eingebene Text wird nicht direkt angezeigt, stattdessen erscheinen lediglich Platzhalter z.B. in Form von Sternchen. Dabei ist allerdings zu beachten, daß diese Felder nur einen Sichtschutz gewähren, der eigentliche Transport über das Netz erfolgt nach wie vor unverschlüsselt!

- Mehrzeilige Eingabefelder

  Soll der Benutzer die Möglichkeit haben, auch etwas längere Mitteilungen (z.B. für ein Gästebuch) eingeben zu können, kann dieses mit mehrzeiligen Eingabefeldern realisiert werden. Es ist zwar eine horizontale und vertikale

[2] s. dazu auch Kapitel 6
[3] Eine detaillierte Einführung in HTML–Formulare findet sich z.B. in [1].
[4] s. dazu auch Kapitel 3

  Größenangabe möglich, diese bezieht sich jedoch nur auf das angezeigte Textfenster. I.d.R. wird ein Browser Scrollbalken in beide Dimensionen vorsehen, so daß auch Texte erfaßt werden können, die die vorgegebenen Grenzen überschreiten. Auch ein mehrzeiligen Eingabefeld kann mit einem Text vorbelegt werden, außerdem kennen einige Browser weitere Features wie z.B. eine Kontrolle des automatischen Zeilenumbruchs.

- Auswahllisten mit Einfach- und Mehrfachauswahl

  Auswahllisten stellen den Benutzer vor die Möglichkeit, aus einer vordefinierten Liste ein oder mehrere Elemente auswählen zu können (z.B. ein Fachgebiet einer Universität). Wurde die Liste auf Einfachauswahl konfiguriert, löst das Auswählen eines Eintrages eine evtl. vorher getroffene Auswahl auf. Bei Mehrfachauswahl können im Zweifelsfall auch sämtliche Einträge gleichzeitig angewählt werden.

- Radio–Buttons und Check–Boxen

  Es kann auch eine Gruppe von Radio–Buttons oder Check–Boxen zur Verfügung gestellt werden. Bei den Radio–Buttons ist wiederum sichergestellt, daß immer nur eine Auswahl getroffen werden kann, Check–Boxen erlauben dagegen eine Mehrfachauswahl. Auch hier kann eine Vorauswahl getroffen werden, z.B. damit sichergestellt ist, daß auch wirklich ein Radio–Button ausgewählt wurde.

- Buttons zum Absenden und Abbrechen

  Diese zwei Standard–Knöpfe dienen zum Bestätigen der Formulardaten und Auslösen der mittels *action*–Tag festgelegten Aktion einerseits und zum Löschen bzw. auf Defaultwert setzen sämtlicher Formularfelder andererseits.

- Freie Buttons und Datei–Buttons

  Freie Buttons haben keine eingebaute Funktion, sondern können mit JavaScript–Funktionsaufrufen belegt werden. Datei–Buttons erlauben den Datei–Upload von Dateien mit vorbestimmtem Mime–Inhalt.

- Versteckte Elemente

  Benötigt man bei einem mehrstufigen Dialog (d.h. mehrere Formulare hintereinander) ein Datum, daß eine Verbindung zwischen den einzelnen Skriptaufrufen, die ja zunächst an sich unabhängig voneinander sind, kann man mittels versteckter Elemente z.B. eine Kundennummer von Formular zu Formular übernehmen, ohne daß der Benutzer sie jedesmal neu eingeben muß oder auf dem Bildschirm sieht.

Neben den freien Buttons kann man auch andere Formularfelder mit JavaScript–Funktionen koppeln, um „intelligente“ Formulare realisieren zu können. So besteht z.B. die Möglichkeit, eine JavaScript–Funktion aufzurufen, sobald ein bestimmtes Eingabefeld den Fokus bekommen hat, der Benutzer eine bestimmte Auswahl aus einer Auswahlliste getroffen hat oder einen bestimmten Radio–Button oder eine Check–Box aktiviert hat. Innerhalb einer JavaScript–Funktion kann man dann z.B. den Inhalt logisch zusammenhängender Eingabefelder konsistent halten.

2.6 Semantik der einzelnen Aufrufmöglichkeiten

Zur Semantik der einzelnen Aufrufmöglichkeiten ist zu bemerken, daß der Browser bei Hyper–Links, Grafikverweisen und dem automatischen Laden via Meta–Statement überhaupt nichts davon mitbekommt, daß er gerade ein CGI–Programm aufruft! Für ihn sieht der Aufruf wie eine ganz „normale" URL aus.

Dieses spielt z.B. bei der Übertragung der Eingabedaten eine Rolle. Da der Browser nicht weiß, daß er ein Programm startet, kann er auch keine Eingabedaten explizit übermitteln. Es gibt jedoch die Möglichkeit, bereits in der URL zum Aufruf des Skriptes Daten übergeben zu können[5], so daß feststehende Eingabedaten z.B. in einen Hyperlink mit eingebaut werden können. Wenn man z.B. einen Link realisieren will, der stets eine Suchmaschine mit dem feststehenden Suchbegriff „CGI" aufruft, könnte man dieses mittels

```
<a href="http://server/cgi-bin/such-prog?suche=cgi">CGI suchen!</a>
```

bewerkstelligen.

Bei Server Side Includes bemerkt der Browser ebenfalls nichts von dem CGI–Programmaufruf, er sieht ihn ja nicht einmal, da er bereits vom Server ausgeführt wird. Der Browser sieht den Ergebnistext des Programmaufrufes als integralen Bestandteil einer „normalen", statischen Webseite. Auch hier können aber natürlich Daten bereits in die URL einkodiert werden.

Einzig und allein bei Formularen wird dem Browser „bewußt", daß er ein CGI–Programm aufruft (oder zumindest eine Mail verschickt) und daher vorher mit Hilfe des Formulares die zu übermittelnden Daten vom Benutzer zu erfragen hat.

3 Übermittlung der Eingabedaten

Um eine Anfrage abzusetzen, gibt es die Methoden *GET* und *POST*. Ihre genaue Arbeitsweise soll später erläutert werden. Beiden gemeinsam ist jedoch das Format der Datenübermittlung. So werden die Informationen durch *Namens- und Wertepaare* übertragen. Auch hier hat die Regel eine Ausnahme, nämlich die *ISINDEX*–Abfrage. Diese Abfrageart ist allerdings nur eine Variante der GET–Methode und soll deshalb erst später erläutert werden.

3.1 Namens- und Wertepaare

Nun aber zurück zu den Namens- und Wertepaaren. Die Daten werden einer oder mehreren „Variablen" zugeordnet. Die Übermittlung hat folgendes Format: „Name=Wert". Zum Beispiel sieht die Übermittlung des Namens „Max Mustermann" mittels zweier Formularfelder, „Nachname" und „Vorname", so aus:

`nachname=Mustermann&vorname=Max`.

Dabei sieht man auch, daß mehrere Namens- und Wertepaare durch ein „&"[6] getrennt werden.

[5] s. dazu auch die Beschreibung der GET–Methode im Kapitel 3

[6] Ein Semikolon kann auch als Trenner verwendet werden, es findet jedoch nur in statischen Aufrufen in HTML–Seiten Anwendung, sonst ist es unüblich.

3.2 URL–Encoding

So ohne weiteres lassen sich jedoch nicht alle Werte in dieses Namens- und Wertepaareschema pressen. So werden alle Sonderzeichen durch den Hexadezimalwert der ASCII–Tabelle kodiert. Die Ausnahme macht hier das Leerzeichen, es wird durch ein „+" kodiert. So wird der String: „Das ist ein Beispiel!" durch

`einstring=Das+ist+ein+Beispiel%21`

kodiert. Diese „URL Encoding" genannte Kodierung hat leider eine große Schwäche, sie ist nicht eindeutig! Besonders im weltweiten Internet, in dem ja CGI hauptsächlich benutzt wird, ist nicht sichergestellt, daß sich hinter einer Hexadezimalzahl auch auf der ganzen Welt das gleiche Zeichen verbirgt. So können z.B. länderspezifische Sonderzeichen wie die deutschen Umlaute dafür verantwortlich sein, daß sich Programme nicht wie gewünscht verhalten.

Ein CGI–Programm wird durch `http://www.foo.com/cgi-bin/test-cgi` aufgerufen. Zusätzlich kann eine Pfadinformation, die sog. *Extra Path Information*, hinter dem Programmnamen übergeben werden, falls das CGI–Programm als Parameter einfach einen Dateinamen haben möchte. Beizeiten sieht man auch WWW–Proxies, die die zu liefernde URL als Pathinfo geliefert bekommen wollen. Die URL sieht dann wie folgt aus:

`http://www.foo.com/cgi-bin/test-cgi/path_info`.

test–cgi ist immer noch das Programm, doch wurde der Pfad „path_info" zusätzlich übergeben. Als Kennzeichnung, daß eine „Extra Path Information" folgt, dient der „/" nach dem CGI-Programmnamen.

3.3 Die GET–Methode

Das Charakteristische an der *GET*–Methode ist die Tatsache, daß die Daten in der URL übergeben werden, d.h. sie werden im HTTP–Header transportiert. Eine typische Anfrage sieht so aus:

`http://www.searchengine.com/cgi-bin/search?request=computer`.

Das Fragezeichen in der URL kennzeichnet den Anfang der Datenübergabe, d.h. die Daten können, wie oben beschrieben, im Format der Namens- und Wertepaare eingegeben werden. Das Namens- und Wertepaar kennzeichnet sich durch das unkodierte „=".

Sollte im gesamten Datenstring kein Gleichheitszeichen vorhanden sein, wird die Anfrage als *ISINDEX*–Abfrage interpretiert. Die Besonderheit der ISINDEX–Abfrage ist, daß die eingegebenen Daten zusätzlich als Komandozeilenparameter an das CGI–Programm übergeben werden. So muß zum Beispiel nur das `finger`–Komando in das `/cgi-bin`–Verzeichnis des Webservers kopiert werden, um fortan mit Hilfe einer ISINDEX–Abfrage aufgerufen werden zu können.

`http://www.foo.com/cgi-bin/finger?user`[7]

wird dann wie in einer Shell als `finger user` ausgeführt. Man muß hier aber unbedingt den Sicherheitsaspekt beachten[8].

Über eine URL können nicht beliebig viele Daten übermittelt werden, da die meisten Browser nur maximal 512 Byte lange URLs verkraften. Bei größeren Datenmengen stößt man schnell an diese Grenze. Besonders, wenn man bedenkt, daß die Daten durch das „URL Encoding" über die Namens- und Wertepaare übermittelt werden. Als Beispiel braucht man nur eine etwas detailliertere Anfrage an eine Suchmaschine abzusetzen.

Auf der anderen Seite bietet uns die Übertragung in der URL den Komfort, daß die URL als Bookmark abgespeichert werden kann. Es kann auch ein statischer Link auf das CGI–Programm mit entsprechenden Daten in einer Webpage erzeugt werden. Die Daten können also gespeichert werden.

Genauso kann aber auch ein Proxy diese URL „speichern". Die Folge ist, daß der Browser eine Seite aus dem Proxy anzeigt. CGI soll ja aber gerade Dynamik in die HTML–Seite bringen, die durch den Proxy schändlich zunichte gemacht wird. Besonders Suchmaschinen haben unter diesem Feature zu leiden, denn ihre neuen Informationen kommen nicht mehr zum Anwender durch. Dieser Aspekt sollte nicht unterschätzt werden, wenn man sich vor Auge führt, wie schnellebig das Internet ist.

Nachdem gezeigt wurde, wie die Daten vom Client zum Server gelangen, müssen sie nun auch dem CGI–Programm auf der Serverseite zur Verfügung gestellt werden. Bei der GET–Methode wird dazu die Umgebungsvariable $QUERY_STRING benutzt. Beim Beispiel `nachname=Mustermann&vorname=Max` wird genau dieser Text (mit allen Problemen des URL Encodings) in die Umgebungsvariable geschrieben. Auch hier kann es ein Problem mit der Länge des Datenstrings geben. Da CGI eine Schnittstelle ist und kein Programm, kann es vorkommen, daß es Plattformen gibt, die die Größe der Umgebungsvariablen beschränken.

3.4 Die POST–Methode

Die *POST*–Methode transportiert im Gegensatz zur GET–Methode die Daten im Body des Anfragepakets. Die URL der Anfrage sieht also unabhängig von den übermittelten Daten stets z.B. folgendermaßen aus:

`http://www.searchengine.com/cgi-bin/search`.

Die Daten `request=computer` stehen im Body, nicht in der URL. Dieses hat weitreichende Konsequenzen: Eine Kennzeichnung, ab wann die Daten folgen, ist nicht nötig, auch entledigt man sich dadurch des Problemes der Längenbeschränkung von URLs. Ebenso entfällt die Möglichkeit, einen statischen Link zu setzen oder Bookmarks mit „eingebauten" Daten zu speichern. Dafür läßt aber der Proxy auch die Anfrage durch, so daß den Client immer aktuelle Daten erreichen. Eine ISINDEX–Variante gibt es bei Verwendung der POST–Methode nicht.

[7] Mehrere Parameter werden durch ein „+" getrennt.
[8] s. dazu auch Kapitel 6

Der Server muß sich bei der POST–Methode auf eine vollkommen andere Arbeitsweise einstellen: Die Daten werden dem CGI–Programm nicht über die Umgebungsvariable $QUERY_STRING übergeben, sondern in seine Standardeingabe geschrieben. Dabei ist zu beachten, daß man sich nicht auf *End–of–File* (EOF) verlassen kann, man muß die Länge der einzulesenden Daten vielmehr der Umgebungsvariablen $CONTENT_LENGTH entnehmen. Diese Variable enthält die Anzahl von Zeichen, die in der Standardeingabe auf Abholung warten. In unserem Beispiel `nachname=Mustermann&vorname=Max` hat die Variable den Wert 31. Die Daten, die hier in der Standardeingabe stehen, sind die gleichen, die bei der GET–Methode hinter dem Fragezeichen folgten. Die potentielle Längenbeschränkung von Umgebungsvariablen spielt hier im Gegensatz zur GET–Methode natürlich keine Rolle.

3.5 Umgebungsvariablen

Wie man schon ein wenig bei den Methoden GET und POST gesehen hat, erfolgt die Kommunikation zwischen Webserver und CGI–Programm über Umgebungsvariablen. Hier soll eine Auswahl der wichtigsten[9] Variablen beschrieben werden.

Man kan die Variablen in zwei Gruppen unterteilen: Es gibt Variablen, deren Werte ausschließlich vom Server abhängen, an den die Anfrage gestellt wird. Auf der anderen Seite gibt es Variablen, deren Inhalte auch vom anfordernden Client abhängen. Einige haben wir schon bei der GET- und POST–Methode vorgestellt. Ihre Inhalte sind natürlich anforderungsspezifisch. Zuerst sollen aber die Variablen vorgestellt werden, die nur serverabhängig sind:

- $SERVER_SOFTWARE

  gibt an, welche Software in welcher Version auf diesem Server läuft.
  Z.B. `Roxen Challenger/1.3.53`

- $SERVER_NAME

  enthält den Namen des Hosts, auf dem der Webserver läuft.
  Z.B. `user.cs.tu-berlin.de`

- $DOCUMENT_ROOT

  absoluter Pfad des lokalen htdocs–Verzeichnisses

Wie schon angekündigt, gibt es auch Umgebungsvariablen, die anforderungsspezifisch sind. Eine kleine Schwierigkeit gibt es jedoch bei diesen Variablen. Es ist nicht immer sichergestellt, daß sie auch existieren. Sie hängen stark von der Anforderung ab. Ein Programm muß also robust gegen das Fehlen von Umgebungsvariablen sein.

- $SERVER_PROTOCOL

  speichert das Protokoll, mit dem die Anfrage geschickt wurde.
  Z.B. `HTTP/1.1`

- $SERVER_PORT

  gibt den TCP–Port an, auf dem der Server die Anfrage erhält.
  Z.B. `80`

[9] subjektive Meinung der Autoren

- $REQUEST_METHOD

  (`GET/POST`) Standard ist die GET–Methode.
  $REQUEST_METHOD wird in CGI–Bibliotheken zur automatischen Erkennung der benutzten Methode verwendet.

- $CONTENT_TYPE

  (z.B. `application/x-www-form-urlencoded`) gibt die Art der Kodierung an. Diese Variable wird allerdings nicht bei der GET–Methode gepflegt, da diese stets o.g. Kodierung benutzt.

- $CONTENT_LENGTH

  Diese Variable enthält die Länge der Daten.
  Sie wird aber auch nicht bei der GET–Methode gepflegt.

- $PATH_INFO

  zusätzliche Pfadinformationen

- $PATH_TRANSLATED

  Der Server stellt eine übersetzte Version von der virtuellen $PATH_INFO zum physikalischen Verzeichnis bereit.
  $PATH_TRANSLATED := lokales htdocs–Verzeichnis + $PATH_INFO

- $QUERY_STRING

  entält die Daten, die nach dem Fragezeichen folgen.
  Diese Variable sollte immer gesetzt sein.

- $REMOTE_HOST

  der Name des Hosts, der eine Anfrage stellt.
  Hat der Server diese Information nicht, so wird diese Variable nicht gesetzt und statt dessen $REMOTE_ADDRESS benutzt.

- $REMOTE_ADDRESS

  ist gesetzt, wenn $REMOTE_HOST nicht gesetzt ist und enthält die IP-Adresse des anfragenden Hosts.

Zusätzlich gibt es im HTTP–Header Einträge, die generisch nach $HTTP_XXX umgesetzt werden. Dabei werden all Mittestriche in Unterstriche umgesetzt. Aus dem Headereintrag „User–Agent“ wird also die Variable $HTTP_USER_AGENT mit entsprechendem Inhalt. Natürlich werden die Einträge, die schon unter anderen Namen eine Variable haben, übergangen. So wird $CONTENT_TYPE nicht auch als $HTTP_CONTENT_TYPE aufgenommen. Wenn nötig, schließt der Server Einträge aus, die die Umgebungsvariablengrenzen verletzen würden.

Beispiele für HTTP–Header sind:

- $HTTP_ACCEPT

  Die MIME–Types, die der Client akzeptiert. Die Einträge dieser Liste werden durch Kommata getrennt.

- $HTTP_USER_AGENT

  Browser, den der Client benutzt.
  Z.B. `Mozilla/4.07 [en] (X11; U; SunOS 5.6 sun4m)`

4 Realisierung/Implementierung

Zur Implementierung von CGI–Programmen oder -Skripten ist zu sagen, daß grundsätzlich jede Programmiersprache benutzt werden kann, für die auf dem Betriebssystem des Rechners, auf dem der Webserver läuft, Unterstützung in Form eines Compilers oder Interpreters vorhanden ist. Die Anforderungen an die Programmiersprache sind nämlich so gering, daß sie so gut wie jede Sprache erfüllt:

- Die Programmiersprache muß von der Standardeingabe lesen und auf die Standardausgabe schreiben können.
- Sie muß Zugriff auf Environment–Variablen ermöglichen.

4.1 Auswahlkriterien für die Programmiersprache

Wir wollen nun einige mögliche Auswahlkriterien betrachten, die eine Entscheidung für oder gegen eine Sprache beeinflussen können:

- Performance

  Die Performance hängt zwar nicht unmittelbar von der gewählten Sprache ab, es ist aber i.d.R. von einem Geschwindigkeitsvorteil von Programmen auszugehen, die einmalig von einem Compiler übersetzt wurden, gegenüber Programmen, die bei jedem Aufruf erneut von einem Interpreter abgearbeitet werden müssen. Zudem bietet ein Compiler evtl. auch eine Optimierungoption für die gewünschte Zielplattform an, was bei Interpretern regelmäßig nicht der Fall ist.

- Sicherheit

  Zur Sicherheit ist zunächst zu sagen, daß sich viele Probleme in diesem Bereich eher aus einem mangelnden Verständnis der CGI–Umgebung und daraus resultierenden Programmierfehlern ergeben, als aus der Sicherheitsproblematik der eingesetzten Programmiersprache an sich. Im Allgemeinen wird man auch hier „compilerbasierten“ Sprachen einen kleinen Bonus gegenüber „interpreterbasierten“ Sprachen einräumen können, da erstere zumindest den Sourcecode der Anwendung uneinsehbar abschotten, so daß Angriffe von Hackern sich nur auf das fertige Binary und nicht auf den evtl. einsehbaren Klartext des Programmcodes stützen können. Ein oftmals unterschätztes Problem, das unabhängig von der Art der eingesetzten Programmiersprache ist, sind die berühmt–berüchtigten *Pufferüberläufe*, die schon manches Programm zu offenen Türen für Eindringlinge gemacht haben[10].

- Zuverlässigkeit

  Die Zuverlässigkeit des Gesamtsystems hängt natürlich von jeder einzelnen, eingesetzten Komponente (z.B. Hardware, Betriebssystem, Compiler/Interpreter, ...) ab. Bezüglich der Wahl der Programmiersprache sollte man logischerweise darauf achten, daß man nicht gerade ein als unzuverlässig bekannten Compiler oder Interpreter einsetzt. Nicht nur bei diesem Punkt gilt der Rat, am besten ein System zu verwenden, mit dem man schon gute Erfahrungen (oder zumindest keine schlechten ...) gemacht hat.

[10] s. dazu auch Kapitel 6

- Pflegefreundlichkeit

  Spätestens bei der Pflegefreundlichkeit wird immer auch die subjektive Einschätzung der gesamten Entwicklungsumgebung eine gewisse Rolle spielen. Manchem mag dabei eine Skriptsprache pflegefreundlicher erscheinen, da man nur das Skript zu ändern braucht und ohne Übersetzerlauf oder dergleichen sofort ausprobieren kann, ob sich das gewünschte Resultat zeigt.

- Portabilität

  Auch den Aspekt der Portabilität sollte man nicht unterschätzen, weiß man doch nicht, auf welchem Betriebssystem und welcher Rechnerhardware das zu entwickelnde Programm in ein, zwei Jahren mal arbeiten können soll. I.d.R. wird man also mit weitverbreiteten, höheren Programmiersprachen arbeiten.

Zusätzlich zu all diesen Überlegungen sollte man sich aber auch nicht scheuen, der Progammiersprache, in der man sich „wohlfühlt“ und in der man schon öfters Programme geschrieben hat, einen gewissen Vorsprung einzuräumen.

Wir wollen hier noch zwei häufig auftretende Probleme im Zusammenhang mit der Programmierung von CGI–Programmen ansprechen: Das Problem der Nebenläufigkeit und der Zustandslosigkeit.

4.2 Problem der Nebenläufigkeit

Da Webserver i.d.R. auf Rechnern mit Multitasking–Betriebssystemen laufen, kann es passieren, daß ein CGI–Programm „gleichzeitig“ zweimal läuft, weil es von zwei unterschiedlichen Clients zur annähernd gleichen Zeit angefordert wurde. Dieses kann in dem Augenblick zu Problemen führen, wenn sich das Programm eines persistenten Datenträgers, also z.B. der Festplatte bedient. Es muß durch *File–Locking* oder ähnliche Maßnahmen, z.B. Verwendung einer Datenbank, sichergestellt werden, daß sich immer nur ein Prozeß im kritischen Abschnitt befindet, innerhalb dessen er auf die Datei zugreifen kann.

4.3 Problem der Zustandslosigkeit

Ein weiteres Problem bei der Implementierung von CGI–Programmen ist die Zustandslosigkeit, d.h. der Umstand, daß man zunächst mal keine Informationen von einem Programmaufruf zum nächsten Programmaufruf übertragen kann. Wenn man also z.B. ein Studentenanmeldesystem realisieren will, bei dem auf der ersten Formularseite die persönlichen Daten des Studenten erfaßt werden sollen und auf der zweiten Seite dann eine Tutoriumsterminwahl stattfinden soll, wird man dieses nur mit zwei voneinander unabhängigen CGI–Programmläufen implementieren können, wobei man dann beim zweiten Lauf nicht mehr weiß, für welchen Studenten die Terminwahl jetzt gespeichert werden soll. Zur Lösung dieses Problems gibt es zwei Ansätze: Man kann einerseits, wie schon im Kapitel 2 beschrieben, versteckte Formularfelder benutzen, um z.B. die Matrikelnummer des Studenten auch beim zweiten Programmlauf nochmal übermittelt zu bekommen oder man setzt die sog. *HTTP–Cookies* ein.

4.3.1 Setzen und Lesen von Cookies

HTTP hat ein zustandsloses Transaktionsmodell, d.h. es können keine Informationen von einer Übermittlung zur nächsten gespeichert werden. Eine Möglichkeit wäre, die Informationen in einer temporären Datei zu speichern, mit einer Id zu versehen und diese Id über eine verstecktes Formularfeld immer mitzunehmen. Diese Verwaltung kann schnell unübersichtlich werden. Um diesen Informationstransport zu ermöglichen, können *HTTP–Cookies* eingesetzt werden.

Ein Cookie ist ein Stück Information, das auf Clientseite gespeichert wird. Üblicherweise werden Cookies von dem Browser des Client verwaltet und in einer Datei gespeichert. Der Ablauf ist nun folgender: Beim Bestätigen „der ersten Seite“ kann das CGI–Programm dem Browser ein Cookie mitschicken, um z.B. die Matrikelnummer des Studenten zu speichern. Wird nun vom Browser eine weitere Seite aufgerufen, die bestimmten, innerhalb des Cookie festgelegten, Kriterien entspricht, schickt der Browser das Cookie als HTTP–Header wieder mit auf die Reise, so daß das CGI–Programm ihm z.B. die Matrikelnummer entnehmen kann.

Ein Beispiel–Header könnte so aussehen:

```
Set-cookie: name=wert; domain=tu-berlin.de; path=/;
expires=Sun, 31-Jan-99[11] 23:59:00 GMT
```

Der erste Teil ist die Rechtfertigung für die Cookies. Hier wird die Information im bekannten Namens- und Wertepaarformat gespeichert.

Als nächstes kommt die Domain, in der das Cookie gültig sein soll. Es wird also nur zurückgeschickt, wenn erneut eine Seite von einem Host dieser Domain angefordert wird. Diese Information ist aber optional. Wenn keine Domain angegeben wird, wird sie auf die Domain des erzeugenden Dokumentes gesetzt.

Als nächster Teil kommt die Pfadinformation. Hier kann angegeben werden, unter welchem Pfad der Domain das Cookie gültig sein soll. Es ist vielleicht nicht wünschenswert, sein Cookie der ganzen Domain zur Verfügung zu stellen, wenn man z.B. seinen Account /˜user besitzt. So wird das Cookie nicht verschickt, wenn Dokumente außerhalb dieses Pfades abgerufen werden. Auch die Pfadangabe ist nicht zwingend. Standard ist der Pfad des erzeugenden Dokumentes.

Expires notiert das Datum, ab dem das Cookie nicht mehr gültig sein soll. Das Datum und die Zeit werden in Standard–GMT angegeben. Auch expires ist nicht Pflicht. Ohne diese Angabe wird das Cookie beim Beenden des Browsers (d.h. der aktuellen Web–Sitzung) gelöscht.

Es gibt noch ein weiters Attribut: secure. Ist dieses Attribut gesetzt, wird das Cookie nur übertragen, wenn eine sichere Verbindung besteht, z.B. via SSL. Die Standardeinstellung ist false.

Das Empfangen von Cookies – man muß die gespeicherten Informationen ja wieder bekommen – ist auch einfach: Sind für einen Host in einer bestimmten Domain unter einem bestimmten Pfad gültige Cookies im Browser verfügbar, so steht im Header des empfangenen Datenpaketes der Eintrag „Cookie: ...“. Dieser Eintrag wird dann in die Umgebungsvariable $HTTP_COOKIE übernommen. Dort kann dann der Inhalt ausgelesen werden. Wenn mehrere Werte vorliegen, werden sie durch ein Semikolon getrennt. Auch hier liegen die Daten im Namens- und Wertepaarformat vor.

[11] Y2K – Y2Worry dieses Format könnte Probleme bereiten.

Zu beachten ist noch, daß Cookies nicht unbedingt von jedem Browser unterstützt werden und daß der Benutzer die Annahme und Speicherung von Cookies auch ablehnen kann, so daß nicht immer gewährleistet ist, daß die Informationen auch wirklich wieder zur Verfügung stehen werden.

5 Ausgabe von CGI–Programmen

Grundsätzlich sendet der Webserver alles, was das CGI–Programm auf die Standardausgabe schreibt, als Ergebnis zurück an den Client. Es sind dabei alle MIME–Typen[12] lieferbar, also z.B. reiner Text, HTML–Seiten, Grafiken in verschiedenen Bildformaten, selbst Videos oder Sound–Dateien. Der Browser kann auch mittels „Location“–Header einfach auf eine neue Webseite verwiesen werden.

Damit der Browser weiß, was für Daten jetzt auf ihn einströmen, muß ihm natürlich zu Beginn der Ausgabe mitgeteilt werden, welchem MIME–Typ die Daten entsprechen. Dazu wird der HTTP–Header `Content-type` benutzt, der dann also z.B. `text/html` oder `image/jpeg` beinhaltet. Diese Angabe ist obligatorisch, muß also in jedem Falle vom CGI–Programm als allererstes ausgegeben werden. Dabei ist auch zu beachten, daß alle HTTP–Header case–sensitiv sind, man also peinlichst auf die richtige Groß-/Kleinschreibung achten muß. Nach weiteren, optionalen Headern muß dann eine Leerzeile geschickt werden, selbst, wenn kein Body geliefert wird! Danach kann der eigentliche (wie gesagt optionale) Inhalt verschickt werden.

Bestimmte Standard–Header, z.B. `HTTP/1.1 200 OK`, `Date: Fri, 26 Feb 1999 17:20:32 GMT` und `Server: Apache/1.3.0 (Unix)`, werden noch vom Webserver vor dem „Content–type“ Header eingefügt. Wird dieses nicht gewünscht, weil man z.B. auch diese Header selbst ausgeben will, kann man dieses den meisten Webservern dadurch mitteilen, daß man seinen CGI–Programmnamen mit dem Kürzel „nph–“ beginnt. Das Skript ist dann selber dafür verantwortlich, einen kompletten und gültigen HTTP–Header zu erzeugen.

6 Sicherheit

Nach diesem chronologischen Durchlauf der Aspekte des Common Gateway Interfaces wollen wir uns nun einige Gedanken über die Sicherheit von Systemen machen, die das Ausführen von CGI–Programmen im Internet anbieten.

6.1 Motivation

Daß dieser Bereich durchaus Relevanz hat, kann man sich allein dadurch vor Augen führen, daß man durch das Anbieten von CGI–Programmen im Prinzip weltweit jedem, der Zugang zum Internet hat, das Ausführen von Programmen auf dem eigenen Rechner ermöglicht. Natürlich sollte man sich darüber im Klaren sein, daß es, wie auch sonst im Leben, keine absolute Sicherheit gibt! Jeder Rechner, der irgendeinen Anschluß an das Internet hat, läuft Gefahr, gehackt, sabotiert oder sonstwie geschädigt zu werden. Es bleibt einem also, wenn man sich entschieden hat, solcherlei Dienste anbieten zu wollen, nichts anderes übrig, als sich möglichst viele Gedanken über die Sicherheit zu machen und dementsprechend zu Handeln, sprich: Sicherungs- und Schutzmechanismen vorzusehen, und natürlich nicht zuletzt an ein regelmäßiges Backup aller „wichtigen“ Systeme und Daten zu denken! Doch

[12] Multipart Internet Mail Extension

nun genug der warnenden Vorrede, wir wollen jetzt einige grundsätzliche Prinzipien in Verbindung mit konkreten Maßnahmen besprechen, die man bei Implementierung und Zurverfügungstellung von CGI–Programmen beachten sollte.

6.2 Maßnahmen bei der Implementierung

Eines der wichtiges Prinzipien lautet: „Erwarte niemals eine bestimmte Eingabe oder Reaktion vom Benutzer, sondern rechne im Gegenteil mit dem potentiell Schlimmsten!“ – Das spielt besonders dann eine Rolle, wenn ein CGI–Programm Daten, die es von Clientseite bekommen hat, in irgendeiner Art und Weise aufbereitet, um daraus einen Befehls- oder Systemaufruf zusammenzubasteln. Berühmt–berüchtigt ist in diesem Zusammenhang das “finger–Beispiel“ aus Kapitel 3. Dieses sieht vor, einfach den Unix–Befehl „finger“ in das CGI–Verzeichnis eines Webservers zu kopieren. Alsdann kann jeder dieses Programm per ISINDEX–Abfrage (also ohne Namens-/Wertepaare) mittels `http://server/cgi-bin/finger?user` aufrufen, um einen Benutzer „anzufingern“. Durch den ISINDEX–Mechanismus wird dann einfach der Befehl `finger user` aufgerufen. Was passiert aber nun wenn jemand böswilligerweise den „Benutzer“ `dummy; rm -rf /` anfingert? Je nach Berechtigung, mit der das CGI–Programm gestartet wird, könnte man danach eine recht „übersichtliche“ Festplatte vorfinden! Auch die Benutzung der Systemaufrufe „exec“, „system“ und „popen“ sollte man in dieser Hinsicht nur mit äußerster Vorsicht benutzen. Auf jeden Fall sollte man, wenn es sich schon nicht vermeiden läßt, solche Befehle, die aus Daten gebildet wurden, abzusetzen, eine gründliche Filterung der Benutzereingaben vornehmen, um z.B. evtl. vorhandene Shell–Metazeichen zu entfernen. Dabei gibt es den Positiv- und den Negativansatz: Bei ersterem werden nur Zeichen durchgelassen, die als ungefährlich betrachtet werden, bei letzterem werden nur potentiell „böse“ Zeichen ausgefiltert. Es ist keine Frage, daß dabei dem Positivansatz der Vorzug zu geben ist.

Ein weiteres Prinzip sollte sein, daß man dem Benutzer so wenig Informationen wie möglich über Details der Implementierung gibt, damit eine Schwachstellenanalyse erschwert wird. In die gleiche Richtung zielt die Überlegung, möglichst viel „fest zu verdrahten“ – je weniger Parameter, Optionen oder dergleichen ein potentieller Hacker zum „Spielen“ findet, desto geringer sein Chance, eine Schwachstelle zu finden und ausnutzen zu können.

Eines der in der Realität am häufigsten anzutreffenden Probleme sind die unsäglichen *Pufferüberläufe*! Man sollte sich also stets genau überlegen, wie groß man einen Puffer dimensioniert und was passieren könnte, wenn er dann „trotzdem“ überläuft.

Man kann sich auch überlegen, ob man z.B. den HTTP–Header „Referer“ abfragen will. Dieser beinhaltet den Ausgangspunkt der Abfrage, d.h. man kann darüber z.B. feststellen, ob ein CGI–Programm wie vorgesehen über ein Formular aufgerufen wurde oder einfach so von einem „böswilligen Hacker“. Dabei sollte man jedoch beachten, daß auch dieser Header nur optional vom Browser gepflegt wird.

6.3 Webserver–Konfiguration

Auch die Webserver–Konfiguration sollte man kritisch überprüfen: Vernünftigerweise sollte man schlichtweg der Übersichtlichkeit halber sämtliche CGI–Programme in einem einzigen Verzeichnis halten und die Ausführung von CGI–Programmen außerhalb dieses Verzeichnisses explizit in der Konfiguration verbieten! Teilweise sind auch bei dem populären Webserver „Apache“ vermeintliche „Standardkonfigurationen“ im Umlauf, die CGI–Programme auch im Homeverzeichnis „ganz normaler“

Systembenutzer erlauben, die womöglich Tür und Tor öffnen, ohne sich dessen bewußt zu sein!

Auch die noch zu besprechenden „Server Side Includes“ sollte man tunlichst für das dedizierte CGI–Verzeichnis deaktivieren. Gibt das CGI–Programm Teile der Eingabedaten auf der Ergebnisseite zurück (z.B. Anmeldungs- oder Bestellbestätigungen), so könnte ein „Provokateur“ SSI–Befehle als Eingabedaten eingeben, die dann bei dieser Rücklieferung auf Serverseite ausgeführt würden!

Zu guter Letzt gehört aber natürlich auch eine regelmäßige Kontrolle der Serverlog- und Errorfiles genauso zu einer guten Sicherheitspolitik, wie die Überlegung, die Gefahren und Auswirkungen eines Angriffs durch eine Firewall oder dergleichen zu begrenzen.

7 Server Side Includes und PHP

Wir wollen nun zwei Alternativen zur Ausführung von CGI–Programmen darstellen. Die erste nennt sich *Server Side Includes*, kurz SSI, die zweite *Personal Home Page Tools*, kurz PHP.

7.1 Server Side Includes

Ein Nachteil von CGI–Programmen ist, daß man, wenn man sich einmal dafür entschieden hat, CGI einzusetzen, innerhalb des CGI–Programmes für die Erstellung einer jeweils kompletten HTML–Seite zuständig ist, auch wenn sich der „dynamische Anteil“ gerade mal auf ein, zwei Zeilen beschränkt. Ein möglicher Ausweg aus diesem Dilemma ist die Verwendung von Server Side Includes.

Das Konzept von SSI sieht vor, daß grundsätzlich eine statische Webseite angelegt wird, in die einzelne SSI–Statements eingebaut werden, die vor Auslieferung der Seite an den Client auf Serverseite abgearbeitet werden. Dadurch ist also eine einfache Trennung statischer und dynamischer Anteile einer Webseite möglich. Die allgemeine Syntax eines SSI–Statements lautet: <!--#*command tag*="*argument*" -->.

Hier nun kurz die wichtigsten Statements im Überblick, angegeben sind jeweils die command tags:

- `include file`

  fügt die angegebene Datei an der Stelle ein, wo dieser Befehl auftaucht.

- `echo var`

  erlaubt die Ausgabe von Environmentvariablen

- `fsize file`

  Dieser Befehl wird durch die Größe der angegebenen Datei ersetzt.

- `flastmod file`

  ergibt das Datum der letzten Änderung der genannten Datei

- `exec cgi/cmd`

  Exec ermöglicht das Ausführen von CGI- oder „normalen“ Programmen. Das gestartete Programm darf jedoch nur „text/plain“ oder „text/html“ liefern.

- `config`

  erlaubt das Konfigurieren der Ausgaben von Fehlermeldungen, Dateigrößenangaben und Zeitangaben

- `printenv`

  gibt alle Umgebungsvariablen aus

- `set`

  Damit kann der Inhalt von Umgebungsvariablen geändert werden.

- `if expr`, `elif expr`, `else` und `endif`

  Realisierung von Verzweigungen, um eine Flußkontrolle zu ermöglichen

Desweiteren ist zu beachten, daß man die Ausführung von SSI–Statements explizit im Webserver konfigurieren muß. Die Namenskonvention sieht für HTML–Seiten mit SSI–Statements `*.shtml`, `*.sht` oder `*.shtm` als Dateinamen vor.

7.2 PHP

Eine weitere Alternative zu CGI sind die sog. *Personal Home Page Tools*. Es handelt sich dabei um eine Sprache, die ebenfalls in HTML–Code eingebettet werden kann und sich syntaktisch an C, Java und Perl anlehnt. Auch hier findet die Abarbeitung auf Serverseite statt. Der wohl wichtigste Vorteil gegenüber SSI und normalen CGI–Programmen ist die „eingebaute“ Zugriffsmöglichkeit auf ein gutes Dutzend der populärsten Datenbanken, so daß Datenbankzugriffe äußerst bequem realisiert werden können. PHP–Programme haben auch vereinfachten Zugriff auf HTTP–Authentifizierungsverfahren und können neben Textausgaben auch z.B. Gif–Bilder zurückliefern.

Es gibt zwei Möglichkeiten, PHP zu installieren:

Findet der Webserver *Apache* Verwendung, kann PHP als Apache–Module installiert werden. Nur bei dieser Installationsart können PHP–Statements *in* HTML–Seiten, die auf den Namen `*.php` hören, ausgeführt werden. Dieser Modus wird auch als der performantere der beiden genannt.

Die zweite Möglichkeit besteht darin, PHP einfach als Interpreter zu installieren, so daß zumindest „normale“ CGI–Programme auch in PHP geschrieben werden können. Diese Installationsart sollte nur dann zum Einsatz kommen, wenn ein anderer Webserver als Apache benutzt werden soll.

PHP–Befehle können auf zwei verschiedene Weisen in HTML–Seiten eingefügt werden: <? *commands;* ?>, optional kann hinter dem einleitenden Fragezeichen auch noch `php` erscheinen, außerdem ist die Benutzung des <script language="php">–Tags analog zu JavaScript möglich. Eine detailliertere Vorstellung der Möglichkeiten von PHP verbietet sich hier leider ob des Umfanges, schließlich ist PHP eine vollständige Programmiersprache mit verschiedensten Variablentypen, Abfrage- und Verzweigungsmöglichkeiten, Schleifen, usw. Für weitergehende Informationen verweisen wir auf [2].

8 Zusammenfassung: Vor- und Nachteile von CGI

Nachdem wir nun alle relevanten Aspekte der CGI–Programmierung betrachtet haben, hier zusammenfassend die Vor- und Nachteile des Einsatzes von CGI:

Auf der Habenseite finden wir im wesentlichen die echte Interaktionsmöglichkeit. Nur dadurch sind moderne Anwendungen wie Auskunfts- und E–Commerce–Systeme im Web realisierbar.

Nachteilig ist dabei die etwas umständlich anmutende Art der eingesetzten Kodierung („URL–Encoding"), die zudem nicht weltweit eindeutig ist. Je nach Auslastung des Netzwerks, der beteiligten CPUs, Datenbanken usw. kann sich eine erhöhte Wartezeit bei der Übertragung von Webseiten ergeben, wobei der Besucher der Seite dann abwägen muß, ob es sich nun für ihn lohnt, auf den interaktiven Teil zu warten oder ob er lieber weitersurft. Gerade bei E–Commerce–Anwendungen wird er diese Wahl aber verständlicherweise nicht haben, da ist die Interaktion ja wirklich ein „Muß". Zu guter Letzt dürfen aber auch die besprochenen Sicherheitsrisiken nicht außer Acht gelassen werden. Schließlich besteht nicht nur die Gefahr, daß der eigene Rechner „geknackt" und manipuliert wird, auch potentiell sehr kostbare Kunden- und weitere Daten schweben in Gefahr, ausspioniert, verfälscht oder gelöscht zu werden.

Literatur

[1] Stephan Münz. HTML Seiten selbst erstellen.
http://www.netzwelt.com/selfhtml.

[2] PHP Documentation Group. *PHP3 Manual*, Dezember 1998.
http://www.php.net/docs.php3.

[3] Jay Eckles. An Introduction to CGI – The Common Gateway Interface.
http://www.ecklesweb.com/training/cgi.

[4] The Common Gateway Interface.
http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[5] Susanne Schmidt. Interaktiv im Web.
c't – magazin für computer technik, 24:226–235, 1998.

[6] Einführung in die CGI–Programmierung.
http://www.xwolf.com.

[7] Jason Nugent. The Common Gateway Interface.
http://www.irt.org/articles/js078/index.html.

[8] Was ist eigentlich CGI?
http://www.bingo.baynet.de/~ub304/cgisampl.html.

[9] Jason Nugent. Server Side Includes and CGI Security.
http://www.irt.org/articles/js099/index.html.

[10] Mark West. Server Side Includes.
http://www.carleton.ca/~dmcfet/html/ssi1.html.