FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Implementierung eines Analysewerkzeuges für Logdateien von WWW-Servern

Peter Kai Wimmer

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Alexander Keller
Abgabedatum:	März 1996

Inhaltsverzeichnis

Kapitel 1

Aufgabenstellung

1.1 Motivation

Die intensive und weiterhin zunehmende Nutzung des World Wide Web führt bei den Anbietern von WWW-Dokumenten zu einem hohen Bedarf an Unterstützungswerkzeugen für die Administration ihrer WWW-Server. Mit der zunehmenden Kommerzialisierung des Internet ist es ferner erforderlich, den Betreibern von WWW-Servern Informationen zur Verfügung zu stellen, die es ihnen erlauben, präzise Aussagen über die Nutzung dieser Server zu erhalten. Die daraus gewonnenen Erkenntnisse bilden eine wichtige Grundlage zur Kapazitätsplanung sowie zur Optimierung der angebotenen HTML-Dokumente. Ein für Unternehmen sehr wichtiger Aspekt ist die Möglichkeit, auf der Basis der erhobenen Daten Wirtschaftlichkeitsbetrachtungen bezüglich der angebotenen Dienste durchzuführen.

Im Rahmen dieses Fortgeschrittenenpraktikums soll untersucht werden, nach welchen Kriterien Logdateien, die von WWW-Servern angelegt werden, maschinell auszuwerten sind. Hieraus können unter anderem automatische Zugriffs- bzw. Fehlerstatistiken generiert werden, die Aussagen über die Nutzungshäufigkeit bzw. Verfügbarkeit der HTML-Dokumente zulassen. Ein typisches Szenario ist, festzustellen, welche HTML-Dokumente besonders häufig betrachtet werden, um diese anschließend auf Server zu verlagern, die über Kommunikationsschnittstellen mit hoher Bandbreite verfügen.

Aufgrund der großen Menge anfallender Daten (mehrere Megabytes pro Monat) ist es unmittelbar einleuchtend, daß manuelle Auswertungen aus Komplexitätsgrunden ausscheiden und daher dringender Bedarf an einem Werkzeug besteht, das die oben beschriebene Funktionalität bietet.

1.2 Anforderungen von BMW

Dieses Analysewerkzeug wird u.a. bei der BMW-AG verwendet, die über ihren WWW-Server Informationen zu ihren Produkten anbietet.

Neben der reinen Statistik des über den Server abgewickelten Verkehrs soll dort auch eine Analyse nach Marketing-Gesichtspunkten erfolgen. Von Interesse sind hierbei die Wiederkehr-Häufigkeit der Abrufer, die Anzahl der Zugriffe auf unterhaltende Angebote (Sounds, Videos) gegenüber denen auf reine Information, das Verhältnis von abgerufenen englischen Dokumenten im Vergleich zur jeweiligen deutschen Version, sowie der Typ des verwendeten Browsers.

Ziel ist die Identifikation des Abrufers, um ein persönliches Kundenprofil bzgl. der Interessen und Vorlieben anhand der betrachteten Werbeseiten auf dem BMW-Server zu erstellen. Potentielle Kunden sollten evtl. mit auf diese Erkenntnisse abgestimmten Informationsmaterial versorgt und zum Produkt hingeführt werden.

Kapitel 2

Technische Grundlagen

Das World Wide Web (WWW) basiert auf dem Client-Server Modell, d.h. Informationen werden von Servern zur Verfügung gestellt und von Clients abgerufen.

2.1 WWW Clients

Der Abruf von Informationen über das World Wide Web geschieht durch sogenannte Browser, deren bekannteste Vertreter *Mosaic* und *Netscape* sind.

Diese Clients holen die Informationen von WWW-Servern mittels des *HyperText Transfer Protocols*, kurz *HTTP*.

2.2 WWW-Server

Zwei Aufgaben fallen einem WWW-Server zu:

2.2.1 Server

Eigene Dokumente werden anderen zugänglich gemacht, indem lokale Dateien, die zumeist in bestimmten Verzeichnissen liegen, übertragen werden.

Ausserdem besteht die Möglichkeit, sog. CGI-BINs, das sind normale Skripten oder Binaries, vom Server ausführen zu lassen. Diese werden üblicherweise aus Formularen (*Forms*) heraus aufgerufen und erzeugen eine jeweils neue, der Anfrage entsprechende Seite.

2.2.2 Proxy

Zum anderen dient der WWW-Server als Proxy, der Hosts innerhalb der eigenen Domäne den Zugang nach Außen ermöglicht.

Diese Proxy-Funktionalität ist nötig, wenn beispielsweise ein Netzwerk durch einen sog. Firewall geschützt wird und nur bestimmte Rechner einen Zugriff nach Außen erlauben, oder wenn nur eine begrenzte Anzahl von IP-Adressen zur Verfügung steht.

2.3 Logging

Die Anfragen an einen WWW-Server werden in Logdateien mitprotokolliert, um dem Betreiber eine Möglichkeit zu geben, mit entsprechenden Werkzeugen Statistiken über die Zugriffe zu erstellen und eventuell auftretende Probleme zu erkennen.

Dies umfaßt sowohl Zugriffe von fremden Hosts auf lokale Dokumente als auch Verbindungen nach außen, die über den Server als Proxy für lokale Browser abgewickelt werden.

Verschiedene WWW-Server haben oft ihr eigenes Log-Format, wodurch sich die Notwendigkeit ergab, ein gemeinsames, einheitliches Format einzuführen, das *Common Log Format*[1].

Im Allgemeinen, d.h. beim Common Log Format, das von allen WWW-Servern unterstützt wird, werden der Name des zugreifenden Hosts, Datum und Uhrzeit, der genaue „Wortlaut“ der Anfrage, ein Statuscode sowie die Anzahl der übertragenen Bytes aufgezeichnet. Viele Server bieten darüber hinaus ein eigenes Format an, bei dem noch zusätzliche Informationen, wie beispielsweise der Name des verwendeten Browsers und das Betriebssystem, unter dem dieser läuft, hinzugefügt werden.

2.3.1 Common Log Format

Folgende Informationen werden beim Common Log Format in der Logdatei protokolliert:

- Name des zugreifenden Hosts
- evtl. Login-Name des Abrufers

  Diese Information wird nur auf Anfrage seitens des WWW-Servers übermittelt, und dies auch nur dann, wenn sie auf der Client-Seite zur Verfügung steht und freigegeben ist. In der Praxis taucht der Logname in so gut wie keiner Logdatei auf.
- Authentisierung

  Bestimmte Dateien bzw. sogar ganze Verzeichnisse können geschützt werden. Dieses Sperren kann auf bestimmte IP-Adressen beschränkt und der Zugriff nur für bestimmte Benutzer erlaubt sein, die sich mit einem Passwort authentifizieren müssen.
- Datum und Uhrzeit
- der genaue „Wortlaut“ der Anfrage

  Die einzelnen Bestandteile werden im Kapitel 3.1.4 ausführlich behandelt. Siehe außerdem Anhang C.
- Statuscode
- Anzahl der übertragenen Bytes

2.3.2 Netscape Log Format

Bei Netscape werden zusätzlich folgende Informationen eingetragen:

- Name des Browsers

[1] Siehe Anhang B

- Betriebssystem

Das beim Common Log enthaltene Übertragungsprotokoll (*http*, *ftp*, etc.) fehlt hier hingegen.

Kapitel 3

Bedienungsanleitung

3.1 `wwwlog`

Sämtliche Einträge in der Logdatei werden vom Perl-Skript abgedeckt. Ihre ausführliche Beschreibung ist deswegen in der Aufstellung der Parameter für die Filterung (s. 3.1.4) enthalten.

Nachfolgend werden mit *Optionen* die Argumente bezeichnet, die dem Skript mit vorangestelltem '`-`' übergeben werden, und mit *Parameter* die Argumente, die wiederum zur jeweiligen Option gehören. *Argument* bezieht sich sowohl auf Parameter als auch auf Optionen.

3.1.1 Logdatei

Standardmäßig wird die Datei `httpd-log.all` ausgewertet; dies ist der Logdateiname des CERN-httpd Servers. Mit der Option `-l` wird ein anderer Dateiname übergeben.

3.1.2 Verschiedene Log-Formate

Dieses Programm ist für die Auswertung von Common-Log Dateien konzipiert. Bei Logdateien des Netscape-Servers muß die Option `-netscape` angegeben werden, da dieser ein abweichendes Format erzeugt.

3.1.3 Kommandodatei

In sog. Kommandodateien werden die Argumente für häufig benötigte Abfragen gespeichert. Ein Klammeraffe (`@`) vor dem wahlfreien Dateinamen weist das Skript an, die Argumente aus dieser Datei zu lesen.

Erlaubte Trennzeichen zwischen Argumenten sind neben dem Leerzeichen auch der Zeilenvorschub, sodaß ein übersichtlicher Aufbau ermöglicht wird.

Kommentare werden mit einem Doppelkreuz `#` am Zeilenanfang gekennzeichnet.

3.1.4 Filter

In den meisten Fällen ist es sinnvoll, die Einträge in der Logdatei, die ausgewertet werden sollen, mittels bestimmter Kriterien auszuwählen. Dies hat einerseits mit der jeweiligen Fragestellung und andererseits mit der Menge der zu verarbeitenden Daten zu tun.

Die nachfolgenden Optionen erlauben eine Filterung nach *jedem* Eintrag in der Logdatei.

- `-host`

  Der zugreifende Host.

- `-logname`

  Loginname des Benutzers, der über den o.a. Host zugreift.

- `-authuser`

  Authentifizierung des Benutzers für geschützte Seiten.

- `-date` *dd/mm/yyyy-dd/mm/yyyy*

  Ein bestimmter Tag bzw. ein Datumsbereich wird ausgewählt.

- `-time` *hh:mm:ss-hh:mm:ss*

  Wie `-date`, nur für die Uhrzeit.

- `-method`

  Die Methode, mit der auf die Seite zugegriffen wird (*GET* oder *POST*).

- `-protocol`

  Das Übertragungsprotokoll[1]:

  - *http* - Hyper Text Transfer Protocol
  - *ftp* - File Transfer Protocol
  - *gopher*
  - *wais* - Wide Area Information Server

  Aufrufe von *news*, *mailto* und *file* werden nicht in die Logdatei eingetragen.

- `-domain`

  Der gesamte Domainname, evtl. mit der Portnummer (getrennt durch Doppelpunkt). Domainnamen beginnen bei HTTP üblicherweise mit *www.*, bei FTP mit *ftp.*.

- `-topdomain`

  Der Teil hinter dem letzten Punkt im Domainnamen, der die Art der Organisation angibt, der der Host angehört. Die Bekanntesten sind

  - *de* - Deutschland
  - *edu* - wissenschaftliche Einrichtungen, vor allem in den USA
  - *com* - kommerzielle Anbieter
  - *org* - Organisationen

[1] Siehe Anhang C, *predefined schemes*

- `-dir`

  Die interne Organisation der über das World Wide Web angebotenen Seiten sollte sich an der üblichen hierarchischen Ordnung mit Unterverzeichnissen, die man beim Verwalten von Dateien anwendet, orientieren.

  Dateien zu einem gemeinsamen Themengebiet werden also jeweils zu Verzeichnissen, die einen aussagekräftigen Namen erhalten, zusammengefaßt. Bekanntestes Beispiel hierfür ist das Verzeichnis `cgi-bin`, in dem die CGI-Skripten abgelegt werden.

- `-file`

  Die Datei `index.html` wird, soweit vorhanden, üblicherweise von WWW-Servern übertragen, wenn in der Anfrage nur ein Verzeichnis, aber kein Dateiname enthalten ist.

  Die Endungen vieler Dateien lassen Rückschlüsse auf ihren Verwendungszweck zu. Auf Rechnern mit veralteten Dateisystemen (DOS, Windows) beträgt die Länge von Endungen höchstens drei Buchstaben, aus *HTML* wird also beispielsweise *HTM*. Daher ist es zweckmäßig, bei längeren Endungen nach dem dritten Zeichen eine beliebige Anzahl weiterer Zeichen zuzulassen (mit '`*`', bzw. '`.*`' bei regulären Ausdrücken).

  - HTML-Dokumente: `*.html *.htm`
  - Postscript `*.ps`
  - Bilder: `*.gif *.jpg`
  - CGI-Skripten: `*.cgi`

- `-param`

  Formulare, die mit der Methode *GET* übertragen werden, geben den Inhalt ihrer einzelnen Eingabefelder nach dem Abschicken öffentlich sichtbar zurück.

  Die einzelnen Variablen werden, durch ein Fragezeichen (`?`) getrennt, an die URL angehängt. Untereinander sind die Variablen durch ein Kaufmannsund (`&`) voneinander abgegrenzt.

- `-status`

  Die Anfrage an einen WWW-Server wird mit einem Rückgabewert quittiert, der den erfolgreichen Abschluß oder einen Fehler anzeigt. Die Parameter `ok` und `error` bilden diese beiden Konditionen ab.

  Die genaue Bedeutung der einzelnen Werte ist im Anhang D zusammengefaßt.

- `-bytes`

  Die Anzahl der übertragenen Bytes, also die Größe des jeweiligen Dokumentes.

- `-browser`

  Der Name des verwendeten Browsers sowie seine Versionsnummer werden beim Netscape Server in einer zweiten Zeile in der Logdatei eingetragen.

3.1.5 Reguläre Ausdrücke

Als Argumente für die Filter-Parameter werden Shell-übliche Ausdrücke verwendet. Benutzer, die reguläre Ausdrücke bevorzugen, können diese verwenden, indem sie den Parameter `-regexp` angeben.

3.1.6 Ausgabe

Die Art der Ausgabe des Skripts wird anhand der jeweiligen Aufgabenstellung gewählt.

Logzeilen

Die einfachste Möglichkeit ist die unveränderte Ausgabe derjenigen Logzeilen, die die gewählten Filter passiert haben. Dies geschieht *während des Programmablaufes.*

Um eine ausführliche Aufspaltung der einzelnen Einträge zu erhalten, wird hier die Option `-verbose` verwendet. Pro Zeile wird dann jedes Element einzeln mit zugehöriger Bezeichnung ausgegeben

Zähler: `-count`

- `bytes`

  Gesamtzahl der übertragenen Bytes.

- `requests`

  Anzahl der Anfragen.

Listen: `-list`

Für den Parameter `-list` sind die nachfolgenden Parameter möglich, die zu den o.a. Filter-Optionen korrespondieren.

- `hosts`
- `lognames`
- `authusers`
- `methods`
- `protocols`
- `domains`
- `topdomains`
- `dirs`
- `files`
- `params`
- `browsers`

Die aufzulistenden Elemente und die jeweilige Anzahl, wie oft sie aufgetreten sind, werden, durch ein Leerzeichen getrennt, untereinander ausgegeben.

Sortierung: `-sort`

Die Ausgabe der `-list` Option ist normalerweise alphabetisch sortiert. Mit `-sort` werden die Einträge absteigend nach Anzahl der Elemente geordnet.

3.2 Typische Einsatzszenarien

3.2.1 Fragestellungen

Die vielfältigen Möglichkeiten dieses Analysewerkzeuges sollen nun an häufig anzutreffenden Fragestellungen verdeutlicht werden.

Eine der grundlegenden Informationen beim Betreiben eines WWW-Servers ist sicherlich die Anzahl der Zugriffe (`-count requests`) und die Größe der insgesamt übertragenen Dokumente (`-count bytes`).

Weiterhin ist die Kenntnis der am häufigsten bzw. am seltensten abgerufenen Dokumente (`-list files` bzw. `-list dirs`) ein guter Wegweiser bei der Gestaltung sowohl des Informationsangebotes wie auch der Information selber.

Auch die Herkunft der Abrufer, nach Ländern (`-list topdomains`) oder sogar nach einzelnen Institutionen (`-list domains`) geordnet, ergibt wichtige Hinweise für die Wahl von Sprache und Inhalt der eigenen Dokumente.

3.2.2 Anwendungsbeispiele

Anzahl Zugriffe und übertragene Bytes im Oktober 1995

```
> wwwlog -date 01/10/1995-31/10/1995 -count requests bytes
bytes 47781011
requests 4522
```

Alle Top-Domänen, sortiert nach Häufigkeit

```
> wwwlog -list topdomains -sort
de 1650
com 1531
edu 291
org 164
```

FTP-Zugriffe über 10000 Bytes, sortierte Liste der Domänen

```
> wwwlog -protocol ftp -bytes 10000-999999999 -list domains -sort
ftp.uni-mainz.de 17
server.berkeley.edu 12
ftp.seagate.com 3
```

Ausführliches Listing einer Logdatei

Hier ist nur ein einzelner Eintrag angeführt.

```
> wwwlog -l u-log.all -verbose
Host:      katmandu
Logname:   -
Authuser:  -
Date:      29/Sep/1995:15:47:32 -0100
Method:    GET
Protocol:  http
Domain:    www.pdb.sni.de
topDom:    de
Dir:       /sni/ccp/FF/
File:      welcome.htm
HTTP-Ver:  1.0
Status:    500
Bytes:     482
```

Kapitel 4

Implementierung

Zur Implementierung wurde die Skriptsprache *Perl* verwendet. Einige Routinen wurden aus dem Buch *Programming Perl* [1] entnommen.

4.1 Allgemein

4.1.1 Reguläre Ausdrücke vs. Shell-ähnliche Ausdrücke

Intern arbeitet *Perl* mit regulären Ausdrücken. Wenn anstelle dieser Shell-ähnliche Ausdrücke verwendet werden, findet eine Umwandlung der folgenden Sonderzeichen statt:

```
*   ->  .*
.   ->  \.
?   ->  .
[^  ->  [\^
```

4.1.2 Programmcode in Strings

Perl bietet die Möglichkeit, Programmcode in Strings abzulegen und mit dem Befehl `eval` ausführen zu lassen.

Dies wird für die Filterung der Logzeilen ausgenutzt, denn die entsprechenden Bearbeitungsschritte stehen durch die beim Programmstart angegebenen Parameter schon von vornherein fest. Siehe auch *Performance*, S. 19.

4.2 Hauptprogramm

4.2.1 Lesen der Parameter in Batch-Dateien

Die Argumente werden um den Inhalt der Batch-Dateien erweitert, die mittels `@` an beliebiger Stelle eingefügt werden können. Diese sind entweder durch Leerzeichen oder Zeilenumbrüche voneinander getrennt.

4.2.2 Parameter einlesen

In einer Schleife (gekennzeichnet mit der Marke `MAIN:`) werden die Argumente eingelesen. Falls eine unbekannte Option verwendet wird, bricht das Skript mit einer Fehlermeldung ab.

4.2.3 Hauptschleife: Parse, Filter, Listen und Zähler, Ausgabe

Die einzelnen Zeilen der Logdatei werden in dieser Schleife zuerst einmal vom Unterprogramm `&parse` in ihre Bestandteile zerlegt. `&filter` überprüft anschließend, ob diese Einträge den vom Benutzer gewählten Filter entsprechen. Wenn ja, bringt `&account` bei Verwendung von `-list` oder `-count` die entsprechenden Listen bzw. Zähler auf den neuen Stand, andernfalls wird die Logzeile mittels `&out` ausgegeben.

4.2.4 Ausgabe der Listen und Zähler

Ist einer der Optionen `-list` oder `-count` dem Skript übergeben worden, werden nach Durchlaufen der Logdatei die Listen, deren Elemente mit `&countItems` (s. 4.3.5) gezählt werden, und die Zähler durch die Routine `&printArray` (s. 4.3.6) ausgegeben.

4.3 Unterprogramme

4.3.1 parse

Die Einträge in die Logdatei werden aufgespalten und die zugehörigen Variablen entsprechend gesetzt. Hierbei wird zwischen dem Common-Log Format und dem des Netscape Servers unterschieden.

4.3.2 filter

Gemäß den gewählten Filtern wird bei jedem Aufruf der nächste passende Eintrag zurückgegeben.

Ein Großteil der Filterung wird durch einen `eval`-Aufruf des Programmcodes, der bereits beim Start erstellt wird (s.o.), bewerkstelligt. Somit läßt sich eine große Schleife einsparen, die für jede einzelne Zeile überprüfen müßte, welche Filter überhaupt angewendet werden. Ausserdem ist die Implementierung von regulären Ausdrücken auf diese Weise ohne jeglichen Aufwand möglich.

Bei `-bytes`, `-date` und `-time` können anstelle eines einzelnen Wertes auch Bereiche (*von-bis*) angegeben werden. Der entsprechende Programmteil ist sehr umfangreich und wurde deswegen nicht als Programmstring implementiert.

4.3.3 account

Der Aufbau von Listen (`-list`) und die Aktualisierung von Zählern (`-count`) wird in diesem Unterprogramm erledigt. Berücksichtigt werden alle Einträge in die Logdatei, die nicht durch die Filterung herausgefallen sind.

Von fast allen Informationen, die in einer Logzeile enthalten sind, werden ebenfalls von Programmcode in einem String die gewünschten Listen angefertigt.

Zähler sind für die Anzahl der Requests sowie die Summe der übertragenen Bytes implementiert.

4.3.4 out

Wenn keine Listen oder Zähler angefordert wurden, wird in diesem Unterprogramm die Ausgabe der Logzeilen, die den Filter passiert haben, durchgeführt. Falls die globale Variable

`$verbose` ungleich Null ist, wird eine ausführliche Unterteilung jedes Logeintrages vorgenommen, ansonsten einfach die Original-Logzeile ausgegeben.

4.3.5 countItems

Die identischen Elemente in den Listen werden gezählt und anschließend alphabetisch sortiert. Falls die Option `-sort` angegeben wurde, findet zusätzlich eine absteigende Sortierung anhand der Anzahl statt. Die Zähl- und Sortierroutinen wurden aus dem hervorragendem Buch *Programming Perl*[1] des Perl-Autors Larry Wall übernommen.

4.3.6 printArray

Um ein Array auszudrucken, wird eine kurze Routine[2] verwendet, die für große Arrays effizienter ist und vor allem weniger Speicherplatz verbraucht.

4.4 Wichtige Variablen

4.4.1 True und False

Da in *Perl* keine Konstanten für True und False existieren, diese aber die Lesbarkeit des Skriptes wesentlich erhöhen, werden die Variablen `$true` (1) und `$false` (0) definiert.

4.4.2 `@filterTypes`

In dieser Liste sind die möglichen Filter eingetragen. Um das Skript zu erweitern, genügt es normalerweise, hier einen zusätzlichen Eintrag vorzunehmen. Zu diesem muß auf jeden Fall eine globale Variable mit exakt demselben Namen existieren!

Auf der anderen Seite ist eine Beschränkung der Möglichkeiten (etwa aus Sicherheitsgründen oder im Hinblick auf den Datenschutz) durch einfaches Löschen von Einträgen in dieser Liste durchführbar.

4.4.3 `@listTypes`, `@countTypes`

Analog zu den Filter-Typen enthalten diese Listen die möglichen Typen, die in Listen ausgegeben oder gezählt werden sollen.

Hier wurden der Status sowie die Datums- und Uhrzeitangabe nicht mit aufgenommen, da es sicherlich keinen Sinn macht, von diesen Einträgen Listen zu erstellen.

4.4.4 `$list{}`, `$count{}`

In diesen assoziativen Arrays sind die Listen (Namen gemäß `@listTypes`) bzw. Zähler (gemäß `@countTypes`) abgelegt.

[1] [1], S.254, S.239, S.245
[2] [1], S.230

4.4.5 Bestandteile eines Eintrags in der Logdatei

Die folgenden globalen Variablen enthalten die Informationen aus jeweils einem Eintrag der Logdatei.

- `$host`
- `$logname`
- `$authuser`
- `$date`
- `$method`
- `$protocol`
- `$domain`
- `$topDomain`
- `$port`
- `$dir`
- `$file`
- `$param`
- `$httpVersion`
- `$status`
- `$bytes`
- `$browser`
- `$os`

4.4.6 Flags

- `$regexp`

  `$true`: reguläre Ausdrücke

  `$false`: Shell-ähnliche Ausdrücke werden verwendet

- `$verbose`

  `$true`: ausführliche Ausgabe der einzelnen Einträge

  `$false`: Logzeile im Originalformat

- `$printExtended`

  `$true`: Ausgabe während des Ablaufs

  `$false`: Liste oder Zähler am Ende

- `$sortNum`

  `$true`: sortiere Listen absteigend nach Anzahl

- `$netscape`

  `$true`: Netscape Log Format

  `$false`: Common-Log Format

4.4.7 Standard-Logdatei

Der Name der bereits erwähnten Standard-Logdatei läßt sich mittels der Variable `$logfile` auf den Defaultnamen des verwendeten WWW-Servers festlegen, also beispielsweise `httpd-log.all` für den CERN-httpd.

Somit kann in den meisten Fällen die explizite Angabe der Logdatei mittels der `-l` Option entfallen.

4.5 Besonderheiten der Skriptsprache Perl

4.5.1 Listen, aber keine Records

In Perl gibt es keine Datenstruktur des *Records*, wie sie von Pascal oder C bekannt sind.

Es ist jedoch möglich, einen Record mit bis zu zwei Einträgen, mit gewissen Grenzen bzgl. der Funktionalität, mittels eines *assoziativen Arrays* nachzubilden.

Konkret wurde ein Record der folgenden Struktur benötigt:

```
struct node
{
  char   *str;
  long   count;
};
```

In *Perl* kann dies so realisiert werden:

```
$node{$str} .= $entry . ´ ´;
```

Die einzelnen Einträge werden also einfach aneinandergehängt, wobei das Leerzeichen als Trennmarkierung verwendet wird.

Um die Anzahl der gleichen Einträge festzustellen, wird der String in ein Array umgewandelt:

```
@array = sort(split(´ ´, $node{$str}));
```

danach alphabetisch sortiert und die Anzahl der jeweils gleichen Einträge gezählt (siehe Routine `&countItems`, 4.3.5).

4.5.2 Bug in Perl Version 5

Die Funktion `eval`, mit der Programmcode zur Ausführung kommt, welcher in einem String abgelegt wurde, zeigt in Version 5 von Perl ein anderes Verhalten als in Version 4.

Ein `return` im Code wurde korrekt ausgeführt und der Rückgabewert konnte zur Programmsteuerung verwendet werden:

```
eval $prg_filter;
```

Anders in Version 5: Der Rückgabewert muss explizit ausgewertet werden (und der skalare Kontext läßt sich nicht einmal mit `scalar()` erzwingen):

```
((eval $prg_filter) eq ´0´) && return $false;
```

4.6 Performance

Zur Optimierung wurden einige Schleifen entfernt und stattdessen werden bestimmte Abfragen gleich beim Parsen der übergebenen Argumente in Perl-Code umgewandelt. Dieser wird in Variablen gespeichert, die später mittels `eval` ausgeführt werden.

Im Vergleich zu einer ähnlichen Implementierung in C++ ist das Perl-Skript etwa fünf- bis zehnmal, in Extremfällen (wenn keine Bildschirmausgabe erfolgt) bis zu 30 Mal langsamer. Bei der Menge der anfallenden Daten ist somit kaum an eine Online-Auswertung zu denken.

Üblicherweise werden die Auswertungen wohl als Cron-Job über Nacht durchgeführt werden, sodaß die Ergebnisse am nächsten Tag vorliegen. Daher ist die Frage der Geschwindigkeit eher zweitrangig.

Kapitel 5

Zusammenfassung

5.1 Möglichkeiten und Grenzen

Bei den umfangreichen Möglichkeiten zur Filterung und Auswertung der Daten in der Logdatei darf die Frage nach der Verwendung der erhobenen Ergebnisse nicht unbeachtet bleiben.

5.1.1 Statistik

Zur Darstellung allgemeiner Statistiken von WWW-Servern sind bereits eine große Anzahl graphisch aufwendiger Programme verfügbar. Die bekanntesten seien vollständigkeitshalber hier aufgeführt:

- wwwstat [1]
- wusage [2]
- getstats [3]

Ziel dieses Fortgeschrittenenpraktikums konnte nicht sein, diesen ein weiteres hinzuzufügen. Vielmehr ist die Ausgabe so einfach wie möglich gehalten, um eine Weiterverarbeitung zu vereinfachen.

Herausragendes Merkmal dieses Perl-Skriptes ist die Möglichkeit, gemäß *beliebigen* Kombinationen der Bestandteile von Logeinträgen die Filterung durchzuführen und die Ausgabe zu steuern. Diese Universalität wird von keinem der anderen Werkzeuge erreicht.

5.1.2 Accounting

Ein Accounting im Sinne von einer benutzerspezifischen Abrechnung der Übertragungsmenge wäre eigentlich kein Problem, ist aufgrund der fehlenden Protokollierung des Login eines Benutzers jedoch üblicherweise nicht möglich.

Die Aufstellung der von anderen Domänen abgerufenen Daten ist dagegen problemlos realisierbar.

[1] `http://www.ics.uci.edu/WebSoft/wwwstat/`
[2] `http://www.boutell.com/wusage/`
[3] `http://www.eit.com/software/getstats/getstats.html`

5.1.3 Marketing

Am Beispiel von BMW wurden schon die Marketing-Gesichtspunkte der Log-Analyse angesprochen. Ausgangsüberlegung war hier wohl, mit den gewonnenen Daten ein individuelles Kundenprofil zu erstellen.

Hier war die Ernüchterung groß, als man feststellen mußte, daß das Login des Benutzers so gut wie nie mitgeliefert wird. Man erhält zwar den Namen des zugreifenden Hosts, dieser ist aber in den meisten Fällen ein Proxy, der nur Rückschlüsse auf die Organisation zuläßt, von der die Anfrage kam.

5.1.4 Datenschutz

Die angesprochene kommerzielle Nutzbarkeit der Ergebnisse ist als eher eingeschränkt zu bewerten.

Bedenklicher ist die Möglichkeit bei Einsatz des WWW-Servers als Proxy, die Requests der Nutzer innerhalb der eigenen Domäne zurückzuverfolgen. Zwar mag auch hier die Protokollierung des Logins unterbunden sein, aber aufgrund des jeweils angegebenen *lokalen* Hosts und der Lastlogin-Liste sind der einzelne Benutzer und die von ihm abgerufenen Dokumente problemlos verknüpfbar.

5.2 Erweiterungen

5.2.1 SNMP-Schnittstelle

Die Ergebnisse der Analyse von Logdateien sind auch für das Management von Netzwerken relevant. Eine Schnittstelle zu solchen Network-Management Systemen ließe sich über SNMP realisieren. Das Analysewerkzeug würde hier als Subagent eines Hauptagenten in einem solchen System agieren.

Da die Ergebnisse des Analysevorgangs erst nach Beendigung des Skripts zur Verfügung stehen, ist es nur sinnvoll, die Anfragen über SNMP aufgrund bereits zur Verfügung stehender Daten zu beantworten. Diese Daten, die für diesen Zweck üblicherweise in einer Datei gespeichert werden, haben allerdings kein einheitliches Format und keine festgelegte Reihenfolge. Das SNMP-Protokoll ist andererseits so unflexibel, daß eine Implementierung wesentlich aufwendiger als zunächst angenommen ausfallen wird und deshalb hier unterblieb.

Es existiert bereits eine entsprechende MIB (`wwwlog.mib`), die als Grundlage verwendet werden kann. Ebenso ist eine rudimentäre Implementierung zumindest der Set-Funktionen vorhanden. Grundlage hierfür war die Portierung einer C-Schnittstelle auf Perl 5, die im Rahmen eines anderen Fortgeschrittenenpraktikums [4] durchgeführt wurde. Eine ausführliche Dokumentation ist ebenda zu entnehmen.

5.2.2 DNS-Lookup

Domainnamen werden bei der Auswertung nur berücksichtigt, wenn sie in den Logdateien bereits vorhanden sind, ansonsten wird die IP-Adresse angegeben.

Bei einer Erweiterung des Skriptes um diese Funktionalität ist folgendes zu beachten: Aufgrund der entstehenden Last ist es nicht sinnvoll, jedesmal DNS-Lookup aufzurufen. Vielmehr

[4] Schütz, Frank; *Fortgeschrittenenpraktikum*, TU München, 1996 (Kapitel 4)

sollte bei Bedarf eine lokale Sammlung der im Regelfall immer wieder auftretenden IP-Adressen angelegt werden, welche bei Bedarf den gewünschten Namen liefert. Abgesehen vom so erzielten Geschwindigkeitsvorteil wird darüberhinaus eine zu starke Belastung des Nameservers vermieden.

Eine weitere Möglichkeit wäre die Verwendung des Programmes httpd-analyse[5], das die vorgenannten Punkte berücksichtigt.

[5] `http://www.w3.org/hypertext/WWW/Tools/LogAnalysis/UserGuide.html`

Anhang A

Parameter des Perl-Skripts

Die nachfolgende Aufstellung der Parameter entspricht der Ausgabe des Perl-Skripts nach einem Aufruf mit der `-help` Option.

```
 -l         logfile
 -netscape  logfile is in NetScape format (default: common log)
 -regexp    use regular expressions instead of shell-style expr.
 -sort      sort -list output by number

Filter:
 -host
 -logname                          wimmer
 -authuser
 -date      dd/mm/yyyy-dd/mm/yyyy  01/10/1995-30/10/1995
 -time      hh:mm:ss-hh:mm:ss      06:00:00-07:30:00
 -method                           GET, POST, ...
 -protocol                         http, ftp, ...
 -domain                           www.ibm.com, ...
 -topdomain                        de, com, edu, ...
 -dir
 -file
 -param
 -status                           303, 20*, ok, error
 -bytes     xxxxx-yyyyy
 -browser

Output:
 -count     bytes | requests
 -list      hosts | lognames | authusers | methods | protocols |
            domains | topdomains | dirs | files | params | browsers
without ´-count´ or ´-list´: logline itself
 -verbose   print verbosely

´@´ preceding a file name reads parameters from a file  (e.g. ´@args´)
```

Anhang B

Common Log Format

Das Common-Log Format[1] ist wie folgt definiert:

`remotehost rfc931 authuser [date] "request" status bytes`

`remotehost`
Hostname des entfernten Hosts (oder die IP-Nummer, falls der DNS hostname nicht verfügbar ist, oder wenn DNS-Lookup ausgeschaltet wurde)

`rfc931`
Der Logname des entfernten Benutzers

`authuser`
Der Benutzername, mit dem der Benutzer sich authentifiziert hat

`[date]`
Datum und Uhrzeit der Anfrage

`"request"`
Die Anfragezeile, genau so, wie sie vom Client gesendet wurde

`status`
Der HTTP Status Code, der vom Client zurückgegeben wurde

`bytes`
Die Länge des übertragenen Dokuments in Bytes

[1] http://www.w3.org/hypertext/WWW/Daemon/User/Config/Logging.html #common-logfile-format

Anhang C

URL Format

Dies ist ein Auszug aus RFC 1738, Kapitel 5, das eine Definition des URL-Formats in BNF-Notation[5] enthält.

```
url =  httpurl | ftpurl | newsurl |
       nntpurl | telneturl | gopherurl |
       waisurl | mailtourl | fileurl |
       prosperourl
```

; URL schemeparts for ip based protocols:

```
ip-schemepart  = "/" login [ "/" urlpath ]
login          = [ user [ ":" password ] "@" ] hostport
hostport       = host [ ":" port ]
host           = hostname | hostnumber
hostname       = *[ domainlabel "." ] toplabel
domainlabel    = alphadigit | alphadigit *[ alphadigit | "-" ] alphadigit
toplabel       = alpha | alpha *[ alphadigit | "-" ] alphadigit
alphadigit     = alpha | digit
hostnumber     = digits "." digits "." digits "." digits
port           = digits
user           = *[ uchar | ";" | "?" | "&" | "=" ]
password       = *[ uchar | ";" | "?" | "&" | "=" ]
urlpath        = *xchar ; depends on protocol
```

; The predefined schemes:
; FTP (see also RFC959)

```
ftpurl    = "ftp://" login [ "/" fpath [ ";type=" ftptype ]]
fpath     = fsegment *[ "/" fsegment ]
fsegment  = *[ uchar | "?" | ":" | "@" | "&" | "=" ]
ftptype   = "A" | "I" | "D" | "a" | "i" | "d"
```

; FILE

```
fileurl = "file://" [ host | "localhost" ] "/" fpath
```

; HTTP

```
httpurl   = "http://" hostport [ "/" hpath [ "?" search ]]
hpath     = hsegment *[ "/" hsegment ]
hsegment  = *[ uchar | ";" | ":" | "@" | "&" | "=" ]
search    = *[ uchar | ";" | ":" | "@" | "&" | "=" ]
```

; GOPHER (see also RFC1436)

```
gopherurl       = "gopher://" hostport [ / [ gtype [ selector
                  [ "%09" search [ "%09" gopher+_string ] ] ] ] ]
gtype           = xchar
selector        = *xchar
gopher+_string  = *xchar
```

; MAILTO (see also RFC822)

```
mailtourl       = "mailto:" encoded822addr
encoded822addr  = 1*xchar ; further defined in RFC822
```

; NEWS (see also RFC1036)

```
newsurl    = "news:" grouppart
grouppart  = "*" | group | article
group      = alpha *[ alpha | digit | "-" | "." | "+" | "_" ]
article    = 1*[ uchar | ";" | "/" | "?" | ":" | "&" | "=" ] "@" host
```

; NNTP (see also RFC977)

```
nntpurl = "nntp://" hostport "/" group [ "/" digits ]
```

; TELNET

```
telneturl = "telnet://" login [ "/" ]
```

; WAIS (see also RFC1625)

```
waisurl       = waisdatabase | waisindex | waisdoc
waisdatabase  = "wais://" hostport "/" database
waisindex     = "wais://" hostport "/" database "?" search
waisdoc       = "wais://" hostport "/" database "/" wtype "/" wpath
database      = *uchar
wtype         = *uchar
wpath         = *uchar
```

; PROSPERO

```
prosperourl  = "prospero://" hostport "/" ppath *[ fieldspec ]
ppath        = psegment *[ "/" psegment ]
psegment     = *[ uchar | "?" | ":" | "@" | "&" | "=" ]
fieldspec    = ";" fieldname "=" fieldvalue
fieldname    = *[ uchar | "?" | ":" | "@" | "&" ]
fieldvalue   = *[ uchar | "?" | ":" | "@" | "&" ]
```

Anhang D

Statuscodes

Eine ausführliche Beschreibung der Bedeutung der einzelnen Werte ist in [7] zu finden.

Codes im 200er Bereich zeigen erfolgreichen Abschluß an, solche von 400 bis 599 sind das Ergebnis einer fehlerhaften Ausführung.
Die `-status` Option von `wwwlog` kennt diese Konditionen als `ok` (20x) und `error` (40x, 50x).

```
Status-Code  = 200 OK
             | 201 Created
             | 202 Accepted
             | 204 No Content
             | 301 Moved Permanently
             | 302 Moved Temporarily
             | 304 Not Modified
             | 400 Bad Request
             | 401 Unauthorized
             | 403 Forbidden
             | 404 Not Found
             | 500 Internal Server Error
             | 501 Not Implemented
             | 502 Bad Gateway
             | 503 Service Unavailable
```

Anhang E

Abkürzungen, Begriffe

CGI - Common Gateway Interface

DPI - Distributed Protocol Interface

FTP - File Transfer Protocl

HTTP - Hyper Text Transfer Protocol

MIB - Management Information Base

PERL - Practical Extraction and Report Language

SNMP - Simple Network Management Protocol

RFC - Request for Comments

URL - Uniform Resource Locator

WAIS - Wide Area Information Service

WWW - World Wide Web

Anhang F

Listing des Perl-Skriptes

```
#!/usr/bin/perl
#

#          INSTITUT FUER INFORMATIK
# der Ludwig-Maximilians Universitaet Muenchen
#
#          FAKULTAET FUER INFORMATIK
#    der Technischen Universitaet Muenchen
#
#          Fortgeschrittenenpraktikum
#          --------------------------
# Implementierung eines Analysewerkzeuges fuer
#  Logdateien von WWW-Servern bei der BMW-AG
#
# Peter Kai Wimmer
# Gabelsbergerstr. 28/IV
# 80333 Muenchen
# Tel./Fax: (089) 523 72 65
# E-Mail: peter@leo.org
#         wimmer@informatik.tu-muenchen.de
#

# Urwald is Bloedsinn, Ernie, ich geh´ nach Hause (Bert, Sesamstrasse)
#


$false = 0;
$true  = 1;


#
# standard log file name
```

```
#
$logfile = "httpd-log.all";

#
# filter types
#
# global variables with corresponding name and meaning
# must exist
#
@filterTypes = ('host',  'logname',  'authuser',  'method',  'protocol',
                'domain',  'topdomain',  'dir',  'file', 'param', 'browser',
                'status', 'bytes', 'date', 'time');

#
# list types
#
@listTypes = ('hosts', 'lognames', 'authusers', 'methods', 'protocols',
              'domains', 'topdomains', 'dirs', 'files', 'params', 'browsers');

#
# count types
#
@countTypes =  ('bytes', 'requests');

#
# Flags
#

#
# use regular expressions?
#
$regexp = $false;

#
# print lines verbosely?
#
$verbose = $false;

#
# print logline (false) or list/count (true) ?
#
$printExtended = $false;

#
# sort alphabetically (default; false) or numerically (true)?
#
$sortNum = $false;
```

```
#
# Log file format: Common (false) or Netscape (true)?
#
$netscape = $false;

#
# Map months to numbers
#
%monthName = (
  ´Jan´, 1,
  ´Feb´, 2,
  ´Mar´, 3,
  ´Apr´, 4,
  ´May´, 5,
  ´Jun´, 6,
  ´Jul´, 7,
  ´Aug´, 8,
  ´Sep´, 9,
  ´Oct´, 10,
  ´Nov´, 11,
  ´Dec´, 12,
);



#
# check if a batch file is used
# read parameters from that file
#
foreach $arg (@ARGV)
{
  local ($batchfile, @batchline);

  if ($arg =~ /^@.*/)
  {
    $batchfile = substr ($arg,1);
    open(BATCHFILE, $batchfile) || die "$0: Can´t open $batchfile\n";
    -T $batchfile               || die "$0: $batchfile: not a batchfile\n";

    while($batchline = <BATCHFILE>)
    {
      ($batchline =~ /^#/) && next;
      @batchline = split (´ ´, $batchline);
      foreach $arg (@batchline)
      {
        push(@argv, $arg);
```

```
        print "$arg\n";
      }
    }
  }
  else
  {
    push(@argv, $arg);
  }
}



#
# Main Loop
#
MAIN: while(@argv)
{
  local ($arg, $type, $outputType);

  $arg = shift (@argv);

  ($arg eq '-help')     && (&help)                       && exit;
  ($arg eq '-l')        && ($logfile = shift (@argv)) && next;
  ($arg eq '-verbose')  && ($verbose  = $true)        && next;
  ($arg eq '-sort')     && ($sortNum  = $true)        && next;
  ($arg eq '-netscape') && ($netscape = $true)        && next;
  ($arg eq '-regexp')   && ($regexp   = $true)        && next;

  #
  #  Filter
  #
  foreach (@filterTypes)
  {
    if ($arg =~ $_)
    {
      $type = substr($arg,1);
      $arg = shift (@argv);
      while (($arg !~ /^-.*/) && ($arg ne ""))
      {
        # use shell-style expressions
        # instead of regular expressions?
        #
        if (!$regexp)
        {
          #   . -> \.
          $arg =~ s|\.|\\\.|g;
          #   * -> .*
```

```
          $arg =~ s|\*|\.\*|g;
          #   ? -> .
          $arg =~ s|\?|\.|g;
          #  [^ -> [\^
          $arg =~ s|\[\^|\[\\\^|g;
        }

        if ($type eq 'status')
        {
  ($arg eq 'ok')    && ($arg = '20.');
          ($arg eq 'error') && ($arg = '40. 50.');
        }

        $filter{$type} .= $arg . ' ';
        $arg = shift (@argv);
      }

      if ($type !~ /bytes|date|time/)
      {
        $prg_filter .= <<'EOF';
          ($TYPE !~ /^FILTER$/) && return scalar($false);
EOF
        $filtertype = $filter{$type};
        chop $filtertype;
        $filtertype =~ s/ /\$|^/g;
        # escape '/' for regular expr.
        $filtertype =~ s|/|\\/|g;

        $prg_filter =~ s/TYPE/$type/;
        $prg_filter =~ s/FILTER/$filtertype/;
      }

      unshift(@argv, $arg);
      next MAIN;
    }
  }

  #
  #  Output
  #
  if ($arg =~ /-count|-list/)
  {
    # disable line-by-line listing
    $printExtended = $true;

    # 'list' or 'count'
    $outputType = substr($arg,1);
```

```
    $arg = shift (@argv);

    while (($arg !~ /^-.*/) && ($arg ne ""))
    {
      $output{$outputType} .= $arg . ' ';

      #
      # list
      #
      if ($outputType eq "list")
      {
        if (!grep (/^$arg$/, @listTypes))
        {
          print "$0: Unknown list type -- $arg\n";
          die "$0: Try '$0 -help' for more information.\n";
        }

        # cut off trailing 's'
        chop $arg;

        # the following $prg_filter results in (e.g.)
        #   $list{'hosts'} .= $host . ' ';
        $prg_account{'list'} .= <<'EOF';
$list{'TYPEs'} .= $TYPE . ' ';
EOF
        $prg_account{'list'} =~ s/TYPE/$arg/g;
      }

      #
      # count
      #
      if ($outputType eq "count")
      {
        if (!grep (/^$arg$/, @countTypes))
        {
          print "$0: Unknown count type -- $arg\n";
          die "$0: Try '$0 -help' for more information.\n";
        }
      }

      $arg = shift (@argv);
    }
    unshift(@argv, $arg);
    next MAIN;
  }
```

```
  #
  # Error
  #
  if ($arg ne "")
  {
    print "$0: Illegal option -- $arg\n";
    die "$0: Try ´$0 -help´ for more information.\n";
  }
}

# for debugging: code strings
#
# print"Filter: $prg_filter";
# print"List:   $prg_account{´list´}";



open(LOGFILE, $logfile) || die "$0: Can´t open $logfile\n";
-T $logfile             || die "$0: $logfile: not a logfile\n";

while($logline = <LOGFILE>)
{
  # two lines from Netscape server
  ($netscape) && (chop $logline) && ($logline .= ´ ´ . <LOGFILE>);

  &parse;
  if (&filter)
  {
    &account;
    &out;
  }
}

close LOGFILE;


foreach (@listTypes)
{
  ($list{$_}) && (&printArray (&countItems($list{$_})));
}

foreach (@countTypes)
{
  ($count{$_}) && (print "$_ $count{$_}\n");
}
```

```
#
#  Parse
#

sub parse
{
  #  if you use ´/´ as delimiter for RegExp, you need to
  #  escape it, since ´/´ also occurs in $path

  #
  # Parse $logline
  # parse $request: $file, $params
  #
  if ($netscape)
  {
    ($host, $logname, $authuser, $date, $request, $status, $bytes, $domain, $browser, $os)
      $logline =~ /^(.*) (.*) (.*) \[(.*)\] \"(.*)\" (.*) (.*) (.*) (.*) \((.*)\)$/;

    ($method, $path, $httpVersion) =
      $request =~ m|^(\w+) /(.*) HTTP/(.*)$|;
  }
  else
  {
    ($host, $logname, $authuser, $date, $request, $status, $bytes) =
      $logline =~ /^(.*) (.*) (.*) \[(.*)\] \"(.*)\" (.*) (.*)$/;

    ($method, $protocol, $path, $httpVersion) =
      $request =~ m|^(\w+) (\w+)://(.*) HTTP/(.*)$|;
  }

  #
  #  Date, Time  (e.g. 29/Sep/1995:13:13:11)
  #
  ($day, $month, $year, $hour, $minute, $second) =
    $date =~ m|^(\d+)/(\w+)/(\d+):(\d+):(\d+):(\d+)|;

  #
  # exchange name of month with number
  #
  $month = $monthName{$month};

  #
  # parse $path: ($domain,) $dir, $file, $param
  #
  ($path, $param) =
    $path =~ /([^\?]*)\??([^\?]*)$/;
```

```
  if  ($netscape)
  {
    ($dir, $file) =
      $path =~ m|^(.*/)?([^/]*)$|;
  }
  else
  {
    ($domain, $dir, $file) =
      $path =~ m|^([^/]+)/(.*/)?([^/]*)$|;
  }
  $dir = '/' . $dir;


  #
  # parse $domain: $topdomain, $port
  #
  ($topdomain, $port) =
    $domain =~ /.*\.([^:]*)?:?(\d+)?$/;
}



#
#  Filter
#

sub filter
{
  local ($filter, $type);
  local ($min, $max);
  local (@filterRange, $range);

  # perl versions 4 and 5 seem to interpret
  # 'eval' differently;
  # eval $prg_filter; # works for version 4, but f**k version 5
  ((eval $prg_filter) eq '0') && return $false;


  #
  #  Bytes
  #
  if ($filter{'bytes'})
  {
    @filterRange = split(' ', $filter{'bytes'});
    foreach $range (@filterRange)
    {
```

```
      ($min, $max) =
        $range =~ /(\d*)-(\d*)/;
      ($bytes < $min) && return $false;
      ($max) && ($bytes > $max) && return $false;
    }
  }

  #
  # date
  #
  if ($filter{'date'})
  {
    @filterRange = split(' ', $filter{'date'});
    foreach $range (@filterRange)
    {
      ($min, $max) =
        $range =~ m|([\d/]+)-?([\d/]*)|;

      ($minDay, $minMonth, $minYear) =
      ($maxDay, $maxMonth, $maxYear) =
        $min =~ m|^(\d+)/(\d+)/(\d+)$|;

      if ($max)
      {
        ($maxDay, $maxMonth, $maxYear) =
          $max =~ m|(\d+)/(\d+)/(\d+)|;
      }

      (($year < $minYear)
      || (($year == $minYear) && ($month < $minMonth))
      || (($year == $minYear) && ($month == $minMonth) && ($day < $minDay)))
      && return $false;

      (($year > $maxYear)
      || (($year == $maxYear) && ($month > $maxMonth))
      || (($year == $maxYear) && ($month == $maxMonth) && ($day > $maxDay)))
      && return $false;
    }
  }

  #
  # time
  #
  if ($filter{'time'})
  {
    @filterRange = split(' ', $filter{'time'});
    foreach $range (@filterRange)
```

```
    {
      ($min, $max) =
        $range =~ m|([\d:]+)-?([\d:]*)|;

      ($minHour, $minMin, $minSec) =
      ($maxHour, $maxMin, $maxSec) =
        $min =~ m|^(\d+):(\d+):(\d+)$|;

      if ($max)
      {
        ($maxHour, $maxMin, $maxSec) =
          $max =~ m|(\d+):(\d+):(\d+)|;
      }

      (($hour < $minHour)
      || (($hour == $minHour) && ($minute < $minMin))
      || (($hour == $minHour) && ($minute == $minMin) && ($second < $minSecond)))
      && return $false;

      (($hour > $maxHour)
      || (($hour == $maxHour) && ($minute > $maxMin))
      || (($hour == $maxHour) && ($minute == $maxMin) && ($second > $maxSecond)))
      && return $false;
    }
  }

  return $true;
}



#
#  Accounting
#

sub account
{
  local (@cntTypes);

  #
  # List
  #
  eval $prg_account{'list'};

  #
  #  Count
  #
```

```
  @cntTypes = split (´ ´, $output{´count´});
  (grep (/^bytes$/,    @cntTypes)) && ($count{´bytes´} += $bytes);
  (grep (/^requests$/, @cntTypes)) && ($count{´requests´}++);

}



#
#  Count items in an array
#

sub countItems
{
  local ($list) = @_;
  local (@array, @keys, @resultArray);
  local (%count);

  @array = sort(split(´ ´, $list));

  # count occurences
  # Programming Perl, p.254
  for (@array)
  {
    $count{$_}++;
  }

  # sort alphabetically
  # Programming Perl, p. 235
  for (reverse sort keys %count)
  {
    push (@resultArray, ($_ . ´ ´ . $count{$_}));
  }

  # sort numerically
  # Programming Perl, p. 249
  if ($sortNum)
  {
    foreach (@resultArray)
    {
      push (@keys, (split(/ /))[1] );
    }
  }

  # subroutine for sorting numerically
  sub byNumber { $keys[$a] <=> $keys[$b]; }
  @sortArray = @resultArray[reverse sort byNumber $[..$#resultArray];
```

```
}



#
# Print an array
#
# Programming Perl, p. 230
# this is preferred to  print join("\n", @array)
# since the following code is more efficient for
# a very long array and more space-conservative
#

sub printArray
{
  local (@array) = @_;
  local ($,, $\) = ("\n", "\n");
  print @array;
}



#
#  Output a logline
#

sub out
{
  ($printExtended) && return;

    if ($verbose)
    {
      print "\nHost:     $host\n";
      print "Logname:  $logname\n";
      print "Authuser: $authuser\n";
      print "Date:     $date\n";

      # request
      print "Method:   $method\n";
      (!$netscape) && print "Protocol: $protocol\n";

      # path
      print "Domain:   $domain\n";
      print "topDom:   $topdomain\n";
      ($port)  && print "Port:     $port\n";
      print "Dir:      $dir\n";
      print "File:     $file\n";
```

```
      ($param) && print "Param:    $param\n";

      print "HTTP-Ver: $httpVersion\n";

      print "Status:   $status\n";
      print "Bytes:    $bytes\n";

      if ($netscape)
      {
        print "Browser:          $browser\n";
        print "Operating System: $os\n";
      }
    }
    else
    {
      print $logline;
    }
}



#
# Help
#

sub help
{
  print <<EOF;

WWW log file analyzer 1.0
Peter Kai Wimmer (peter\@leo.org)

 -l         logfile
 -netscape  logfile is in NetScape format (default: common log)
 -regexp    use regular expressions instead of shell-style expr.
 -sort      sort -list output by number

Filter:
 -host
 -logname                         wimmer
 -authuser
 -date      dd/mm/yyyy-dd/mm/yyyy   01/10/1995-30/10/1995
 -time      hh:mm:ss-hh:mm:ss       06:00:00-07:30:00
 -method                          GET, POST, ...
 -protocol                        http, ftp, ...
 -domain                          www.ibm.com, ...
 -topdomain                       de, com, edu, ...
```

```
 -dir
 -file
 -param
 -status                          303, 20*, ok, error
 -bytes     xxxxx-yyyyy
 -browser

Output:
 -count     bytes | requests
 -list      hosts | lognames | authusers | methods | protocols |
            domains | topdomains | dirs | files | params | browsers
without ´-count´ or ´-list´: logline itself
 -verbose   print verbosely

´\@´ preceding a file name reads parameters from a file  (e.g. ´\@args´)

EOF
}
```

Index

Literaturverzeichnis

[1] Larry Wall and Randal L. Schwartz. *Programming Perl.* O'Reilly & Associates, Inc. 1992

[2] T. Berners-Lee, L. Masinter, M. McCahill, *RFC 1738: Uniform Resource Locators (URL)*

[3] *Basic HTTP.*
`http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTTP2.html`.
The World Wide Web Consortium

[4] *Names and Addresses, URIs, URLs, URNs, URCs.*
`http://www.w3.org/pub/WWW/Addressing/Addressing.html`.
a.a.O.

[5] *BNF for specific URL schemes.*
`http://www.w3.org/hypertext/WWW/Addressing/URL/5_BNF.html`.
a.a.O.

[6] *The Common Logfile Format,*
`http://www.w3.org/hypertext/WWW/Daemon/User/Config/Logging.html`
`#common-logfile-format`.
a.a.O.

[7] *Status codes.*
`http://www.w3.org/pub/WWW/Protocols/HTTP/HTRESP.html`.
a.a.O.