Studie: Datenbank-Webapplikationen und ihre Technologien

Informatikprojekt

vorgelegt von

David Vogel

2002

Leiter der Arbeit:
Dr. Igor Metz
Prof. Dr. Oscar Nierstrasz

Institut für Informatik und angewandte Mathematik

Abstract

Es gibt viele Möglichkeiten eine Webapplikation zu realisieren. Man hat die Wahl zwischen diversen Webservern. Zudem stehen viele Verabeitungstechnologien auf der Serverseite zur Verfügung, welche auf andere Applikationsserver zugreifen können.

Die Kommunikation vom Client zum Server wird durch das Protokoll HTTP geregelt. Dies ist ein primitives und zustandloses Protokoll. Die Anforderungen wie Usertracking und Zugriffskontrolle müssen daher von der Applikation selbst erfüllt werden.

Anhand von Applikationen mit Anbindung an eine Datenbank werden die verschiedenen Technologien erläutert. Zu jedem Lösungsweg wird ein Prototyp implementiert, welchen man als Ausgangslage für eine eigene Webapplikation anpassen kann.

Diese Studie eignet sich gut als Kursunterlagen zu einer Einführung in verschiedene Webtechnologien. Sie gibt einen Einblick in folgende Themen: Internet (Technologie, Protokolle und Dienste), Apache Web Server (Konfiguration, Zugriffskontrolle, Usertracking,https), Kryptographie, HTML, Javascript (Syntax, Objekte, Eventhandler), CGI, Perl (Filehandling, Module, Komplexe Datenstrukturen, Manipulation von Datenbanken), Datenbanken (SQL, Module), PHP (Grundlagen, Manipulation von Datenbanken).

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

Das Thema dieser Arbeit ist die Untersuchung einiger Aspekte des WWW (World Wide Web), insbesondere des Datenaustausches zwischen Webserver und Client. Die Kommunikation im WWW wird durch das Protokoll HTTP geregelt. Dies ist ein zustandloses Protokoll und speichert keine Informationen von einer Transaktion zur nächsten. Dadurch weiss ein einfacher Webserver nicht, ob er mit einem Client schon kommuniziert hat oder nicht. Ein solcher einfacher Webserver liefert nur statische Seiten, und kann dadurch nicht flexibel reagieren. Auf folgende Fragen wird hier näher eingegangen:

- Wie kann man eine interaktive Verbindung (Dialog) zum Client implementieren (->Sessiontracking, interaktiver Webserver)?
- Wie kann man flexibel und dynamische auf Interaktionen des Users reagieren (->dynamische HTML-Generierung)?
- Wie (und wo) kann man die Daten des Benützers ablegen, um sie später wieder zur Verfügung stellen (Clientseitig: Javascript und Cookie, CGI und Cookie. Serverseitig: CGI und Flatfile, MySQL)?
- Wie kann man serverseitig auf eine MySql-Datenbank zugreifen?
- Wie kann man Datensicherheit (*Gewährleistung ihrer Vertraulichkeit, Verfügbarkeit und Unversehrbarkeit*) und Datenschutz *(Wer hat Zugriff auf meine Daten)* gewährleisten?

Zudem wird hier ein Einblick in die folgenden Gebiete gegeben: Internet (Technologie, Protokolle und Dienste), Apache Web Server (Konfiguration, Zugriffskontrolle, Usertracking,https), Kryptographie, HTML, Javascript (Syntax, Objekte, Eventhandler), CGI, Perl (Filehandling, Module, Komplexe Datenstrukturen, Manipulation von Datenbanken), Datenbanken (SQL, Module), PHP (Grundlagen, Manipulation von Datenbanken). Anhand dieser Einführung kann man leicht eine eigene flexible Webapplikation mit Anbindung an eine Datenbank implementieren, welche Datensicherheit und Datenschutz gewährleistet.

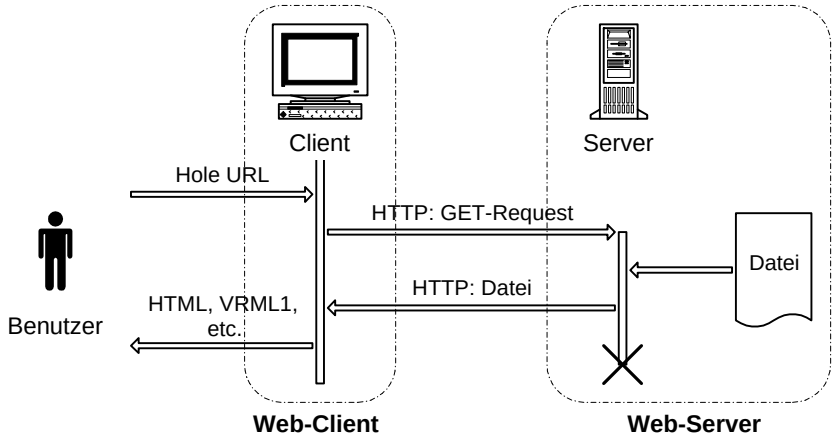

Abbildung 1: Das einfache Web-Modell [Met99]

2 Einführung

2.1 Problemstellung

Die Einführung des World Wide Web *(WWW)* brachte die erste grosse Popularitätswelle des Internets. Das *WWW*, das nur einen Teil des Internets darstellt, ermöglichte es plötzlich fast jedem, Dokumente zu beliebigen Themen auf Knopfdruck mehreren Millionen Internetnutzern weltweit zugänglich zu machen.

Das Internet beruht auf drei Basismechanismen [Met99]:

- ein eindeutiges Namensschema, das den Zugang zu allen Ressourcen im Internet in uniformer Art erlaubt (*URL*'s).
- Protokolle, die den Austausch benannter Ressourcen über das Internet erlauben (*HTTP, FTP, GOPHER, TCP/IP,* etc.).
- Hypertext für die einfache Navigation zwischen Ressourcen (*HTML*)

Im einfachsten Fall (siehe Abb. 1) fordert ein Benutzer im *WWW* über eine *URL* eine Ressource (z.B. HTML-Dokument) an. Die Anfrage geht über das Netz zum Web-Server, und dieser liefert die gewünschte Datei. HTTP (*Hypertext Transfer Protocol*) regelt den Ablauf dieser Kommunikation zwischen *HTTP-Server* (WWW-Server) und *HTTP-Client* (WWW-Browser). Hierbei schickt immer der Client eine Anforderung (*Request*) an den Server, der mit einer Antwort (*Response*) reagiert. Eine Anforderung kann z.B. die Bedeutung haben "Schicke mir das Dokument *sms.html*". Die Antwort besteht normalerweise aus dem angeforderten Dokument und zusätzlichen Informationen wie z.B. dem Datum der letzten Änderung (siehe Abb. 2).

HTTP ist ein *zustandloses* Protokoll. Es speichert keine Informationen von einer Transaktion zur nächsten und kennt den Client nur für die Dauer einer einzigen Transaktion. Jeder Request beginnt dadurch wieder von vorne. Bei einfachen Connections ist dies von Vorteil, da kein Overhead mit *Session-Tracking* von einer Verbindung zur nächsten entsteht. Dadurch kann ein HTTP-Server in einer bestimmten Zeit viele Clients bedienen.

Bei grösseren Businessanwendungen (z.B. Telebanking) sieht es wieder anders aus: hier ist *Session-Tracking* nötig, um *interaktive* Verbindungen zu ermöglichen. HTTP kann dieses Problem leider nicht lösen; aber schliesslich war HTTP für den Zugriff auf HTML-Seiten gedacht und nicht für Client/Server-Anwendungen im Internet. In diesem Fall muss man Session-Tracking mit anderen Technologien gewährleisten (z.B. *hidden input fields, cookies, user authorization oder URL-rewriting*) (mehr dazu später). Der Nachteil dieses Ansatzes liegt darin, dass die Daten bei einer Interaktion, die über mehrere Schritte geht, stets neu übertragen werden müssen. Bliebe die Verbindung zwischen den Anfragen bestehen, könnte der

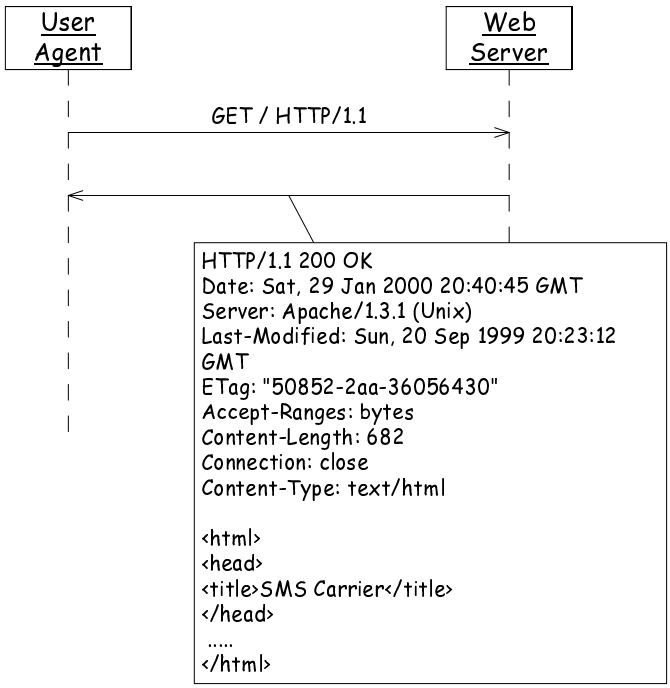

Abbildung 2: Interaktion über HTTP [Met99]

Server übermittelte Daten speichern, und es würde kein Overhead enstehen (...bei einem Telefongespräch stellen wir uns auch nicht bei jedem Satz mit Namen vor) [MP98].

Mit einem Web-Server, der nur statische Seiten liefert, hat man nicht viele Möglichkeiten. Idealer wäre ein Server, der individuell auf die Benutzereingaben reagiert und dann dynamisch HTML-Seiten generiert. In diesem interaktiven Web-Modell musste man die Funktionalität des Web-Servers erweitern: z.B. die *CGI*-Schnittstelle (Common Gateway Interface) erlaubt es einem WWW-Browser, über einen WWW-Server Programme auszuführen. *HTML* stellt die Möglichkeit zur Verfügung, mit Hilfe spezieller Befehle *Formulare* zu erstellen. In Formularen kann der Anwender Eingabefelder ausfüllen, in mehrzeiligen Textfeldern Text eingeben, aus Listen Einträge auswählen und Buttons anklicken. Wenn das Formular fertig ausgefüllt ist, kann der Anwender auf einen Button klicken, um das Formular abzusenden. Der Server erhält dadurch einen Request mit Benutzerdaten. Damit diese Daten weiterverarbeitet werden, wird eine Aktion im Web-Server ausgelöst. Dazu gibt man beim Erstellen eines Formulars an, was mit den Daten des ausgefüllten Formulars passieren soll. Man kann sich die ausgefüllten Daten beispielsweise per *E-Mail* zuschicken lassen oder von einem *CGI-Programm* auf dem *Server-Rechner* weiterverarbeiten lassen (siehe Abb. 3).

Formulare können sehr unterschiedliche Aufgaben haben. So werden sie zum Beispiel eingesetzt [Mün98]:

- um bestimmte, gleichartig strukturierte Auskünfte von Anwendern einzuholen,
- um Anwendern das Suchen in Datenbanken zu ermöglichen,
- um Anwendern die Möglichkeit zu geben, selbst Daten für eine Datenbank beizusteuern,
- um dem Anwender die Möglichkeit individueller Interaktion zu bieten, etwa um aus einer Produktpalette etwas Bestimmtes zu bestellen.

Man geht nun von einer einfachen HTML-Eingabemaske (siehe Abb. 4) zum Versand von *SMS-Nachrichten* (einen Text von max. 160 Zeichen an einen Mobilabonementen) aus. Der Benutzer kann in diesem Formular die Adresse (Mobile Nr.), den Absender und die Nachricht eingeben. Durch abschicken

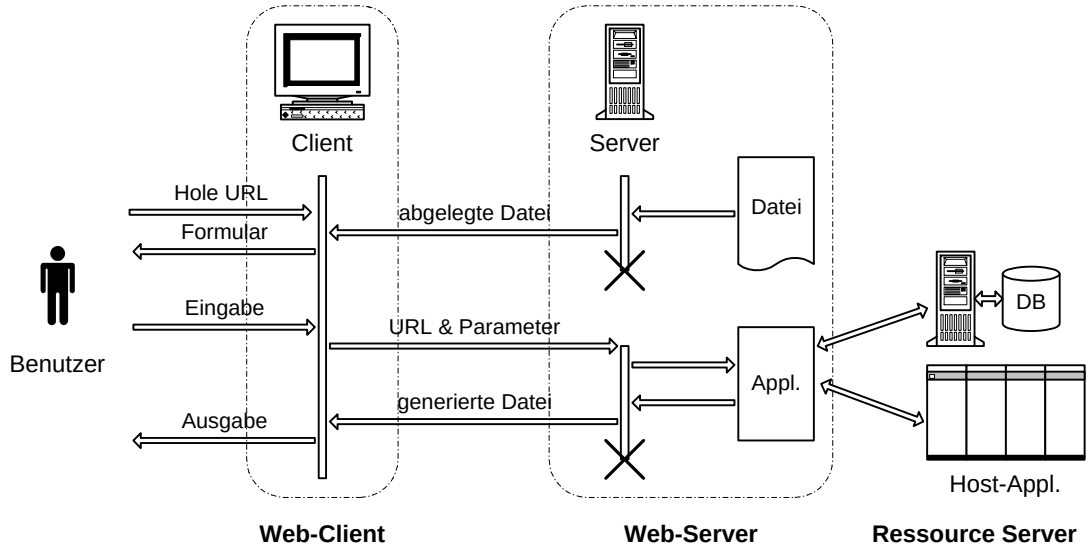

Abbildung 3: Das interaktive Web-Modell [Met99]

dieses Formulares werden die Benutzerdaten auf der Serverseite verarbeitet (CGI-Programm) und dieNachricht an das angegebene Mobiltelefon gesendet.

Um den Service zu verbessern, will man dem Benutzer ein *Adressbuch* zur Verfügung stellen. Nun kann der User seine Daten in einer Datenbank ablegen und jederzeit darauf zugreifen. Das Thema dieser Arbeit ist die Untersuchung folgender Probleme, die es nun zu lösen gilt:

- Generierung von HTML-Code (*auf Client- und/oder auf Serverseite; mit verschiedenen Programmiersprachen)*
- Session-Tracking *(mit wem spreche ich? -> Speichern von Informationen von einer Transaktion zur nächsten).*
- User-Authorization *(Identifizierung gegenüber dem Dienstanbieter)*
- Datenschutz *(Wer hat Zugriff auf meine Daten)*
- Datensicherheit (*Gewährleistung ihrer Vertraulichkeit, Verfügbarkeit und Unversehrbarkeit*)
- Datenhaltung *(auf Client- und/oder auf Serverseite).*

2.2 Grundlagen

Dieser Abschnitt beschreibt kurz, wie das Internet entstand, seine Architektur und einige *Applications Protocols*. Zudem wird auf einige Dienste des Internets, auf die *Markup-Language HTML* und auf *Cookies* eingegangen.

2.2.1 Entstehung des Internet

Das ARPA-Net (siehe: [ARPA1][ARPA2]). Die Ursprünge des heutigen Internet reichen in die 60er Jahre zurück. Es war die Zeit des Kalten Krieges zwischen den beiden Weltmächten USA und UdSSR. Neue Impulse in der Elektronischen Datenverarbeitung (*EDV*) kamen in jener Zeit hauptsächlich durch militärische Initiativen zustande.

Im Departement of Defense, dem amerikanischen Verteidigungsministerium, wurde überlegt, wie man wichtige militärische Daten besser schützen könnte. Selbst bei einem atomaren Angriff des Gegners sollten die Daten nicht zerstört werden können. Als Lösung kam nur ein elektronisches Datennetz in Frage. Die

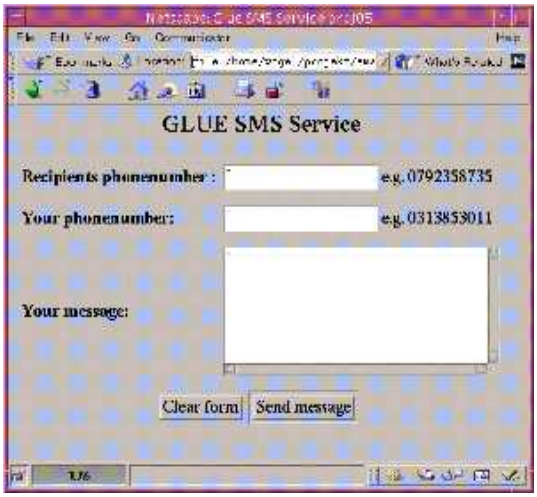

Abbildung 4: Eine einfache HTML-Eingabemaske

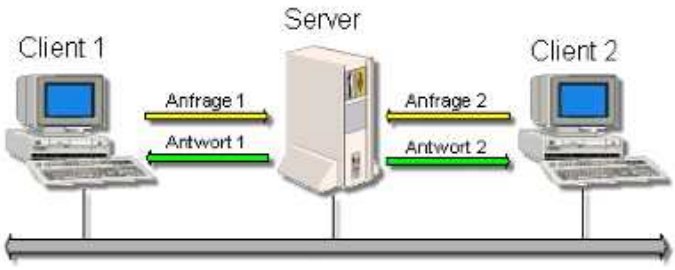

Abbildung 5: Das Client/Server-Prinzip [DO97]

gleichen Daten sollten dabei auf mehreren, weit entfernten Rechnern abgelegt werden. Bei neuen oder geänderten Daten sollten sich alle angeschlossenen Rechner binnen kürzester Zeit den aktuellen Datenstand zusenden. Jeder Rechner sollte dabei über mehrere Wege mit jedem anderen Rechner kommunizieren können. So würde das Netz auch dann funktionieren, wenn ein einzelner Rechner oder eine bestimmte Leitung durch einen Angriff zerstört würde. Die *Advanced Research Projects Agency (ARPA)*, Teil der US-Militärs, realisierte das geplante Projekt [Mün98].

2.2.2 Client-Server-Technologie

Für die einzelnen Internet-Dienste wie *World Wide Web*, *Gopher*, *E-Mail*, *FTP* usw. muss auf einem Hostrechner, der anderen Rechnern diese Dienste anbieten will, eine entsprechende Server-Software laufen. Ein Hostrechner kann einen Internet-Dienst nur anbieten, wenn eine entsprechende *Server*-Software auf dem Rechner aktiv ist, und wenn der Rechner "online" ist.

Server sind Programme, die permanent darauf warten, dass eine Anfrage eintrifft, die ihren Dienst betreffen. So wartet etwa ein WWW-Server darauf, dass Anfragen eintreffen, die WWW-Seiten auf dem Server-Rechner abrufen wollen.

Clients sind dagegen Software-Programme, die typischerweise Daten von Servern anfordern. Der WWW-Browser ist beispielsweise ein Client. Wenn man etwa auf einen Verweis klickt, der zu einer HTTP-Adresse führt, startet der Browser, also der WWW-Client, eine Anfrage an den entsprechenden Server auf dem entfernten Hostrechner. Der Server wertet die Anfrage aus und sendet die gewünschten Daten (siehe Abb. 5). Um die Kommunikation zwischen Clients und Servern zu regeln, gibt es entsprechende Protokolle. Client-Server-Kommunikation im WWW etwa regelt das *HTTP-Protokoll*. Ein solches Protokoll läuft oberhalb des TCP/IP-Protokolls ab.

Dass ein Client Daten anfordert und ein Server die Daten sendet, ist der Normalfall. Es gibt jedoch auch "Ausnahmen". So kann ein Client nicht nur Daten anfordern, sondern auch Daten an einen Server schicken: zum Beispiel, wenn man per FTP eine Datei auf den Server-Rechner hochlädt, wenn man eine E-Mail versendet oder im WWW ein Formular ausfüllt und abschickt. Bei diesen Fällen redet man auch von *Client-Push* ("Client drängt dem Server Daten auf").

Ein anderer Ausnahmefall ist es, wenn der Server zuerst aktiv wird und dem Client etwas ohne dessen Anforderung zuschickt. Das nennt man *Server-Push* ("Server drängt dem Client Daten auf"). Neue Technologien wollen diesen Ausnahmefall zu einer Regel erheben: die sogenannten *Push-Technologien*. Diese Technologien sollen ermöglichen, dass ein Client regelmässig Daten empfangen kann, ohne diese eigens anzufordern. Dadurch sind Broadcasting-Dienste wie aktuelle Nachrichten usw. realisierbar. Netscape und Microsoft Internet Explorer (beide ab Version 4.0) haben entsprechende Schnittstellen, um solche Dienste in Anspruch zu nehmen [Mün98].

2.2.3 Das ISO/OSI-Referenzmodell

Die Kommunikation zwischen Rechnern in offenen, heterogenen Systemen wird in der Regel anhand des sog. *ISO/OSI-Referenzmodells* beschrieben. Dies teilt die Aufgaben, die bei der Datenkommunikation zwischen zwei Anwendungen auf unterschiedlichen Rechnern anfallen, sieben aufeinander aufbauenden Schichten zu. Dabei bietet jede Schicht ihre Dienste der nächst höheren Schicht an und kann ihrerseits die Dienste der direkt unter ihr liegenden Schicht in Anspruch nehmen (siehe [Schn94]).

Die Schichten werden dabei in Anbetracht der angebotenen Dienste in 7 Kategorien eingeteilt:

- Die *Physical Layer* überträgt über das physikalische Medium, das die Computer verbindet, Einsen und Nullen.
- Die *Data Link Layer* überträgt sogennannte Frames. Sie stellt eine fehlerfreie Verbindung her und verhindert eine „Überschwemmung" eines langsamen Empfängers mit Daten.
- Die *Network Layer* verschickt Pakete an einen angegebenen Emfpänger. Sie übernimmt dabei die korrekte Weiterleitung der Pakete, das sogenannte „Routing".
- Die Funktion der *Transport Layer* ist es, die Daten der Session Layer in kleinere Teile aufzuteilen und den Empfang aller dieser Pakete in der richtigen Reihenfolge sicherzustellen. Ausserdem stellt sie einen Mechanismus bereit, um andere Computer bzw. die darauf laufenden Prozessse zu benennen, um den Empfänger der Daten angeben zu können. Die Verbindungsdienste, die die Layer bereitstellt, können sehr verschieden sein.
- Die *Session Layer* ermöglicht es sogenannte Sessions zwischen Computern herzustellen. Sie ist ausserdem für das sogenannte Token Management und für bestimmte Synchronisationsaufgaben verantwortlich.
- Währen alle bisherigen Schichten nur Bits übertragen, bietet die *Presentation Layer* die Übertragung von strukturierten Daten, d.h. Zeichen, Zahlen usw. an.
- Die *Application Layer* ist die Schnittstelle zum Benutzer und bietet Dienste wie Terminaldienste oder die Übertragung von Dateien oder eMails.

2.2.4 Protokolle im Internet: Übersicht

Das ISO/OSI-Modell entstand erst, als das Arpanet bereits einsatzfähig war. Die Erfahrungen, die bei der Entwicklung des Arpanet gemacht wurden, gingen jedoch in die Modellierung mit ein. Die Internetprotokolle können ebenfalls in ein Schichtenmodell - jedoch mit nur 4 Ebenen - eingeordnet werden. In Abbildung 6 und Abbildung 7 werden die Modelle einander gegenübergestellt [Sche93]. Gleichzeitig enthalten die Abbildungen die wichtigsten Protokolle und ihre Zuordnung zu den jeweiligen Ebenen.

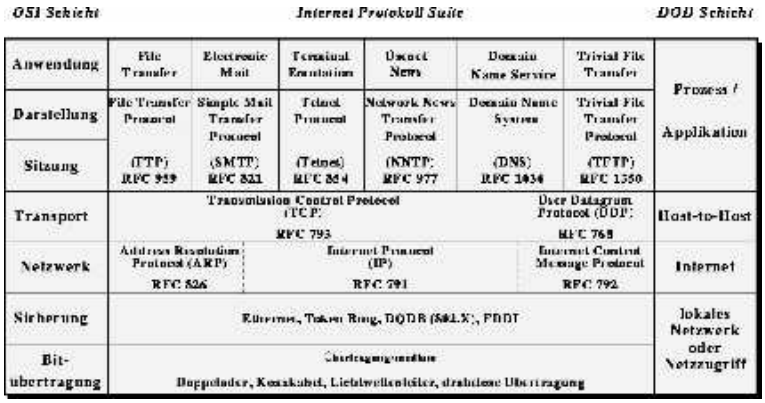

Abbildung 6: Protokollübersicht (Teil 1) [Schn94]

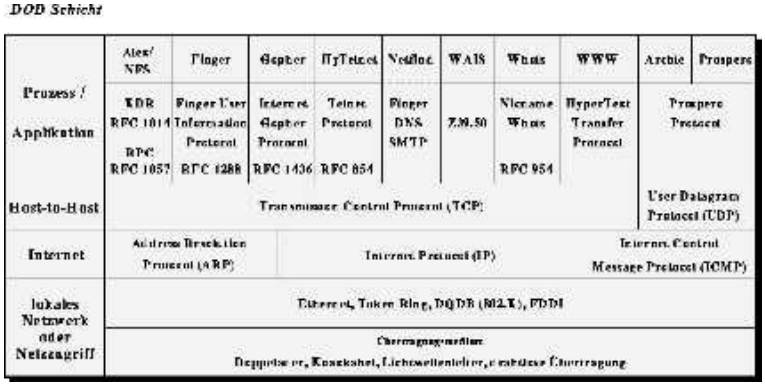

Abbildung 7: Protokollübersicht (Teil 2) [Schn94]

2.2.5 Das Transmission Control Protocol (TCP)

TCP (*Transmission Control Protocol*) ist auf Ebene 4 der Protokollhierarchie angesiedelt. Das verbindungsorientierte Transportprotokoll im Internet Es dient als Basis für Anwendungen wie *telnet* oder *ftp*, bei denen eine zuverlässige Übertragung der Daten gefordert wird, oder in ISO/OSI-Sprechweise: TCP erbringt der Anwendungsschicht einen zuverlässigen, verbindungsorientierten Dienst. Es stellt eine bidirektionale Verbindung zwischen den Partnern her. Zuverlässig bedeutet dabei, dass die Datenübertragung gesichert erfolgt und die gängigen Sicherungsverfahren wie Sequenznummernvergabe, Prüfsummenbildung mit Empfangsquittungen, Quittungen mit Zeitüberwachung und Sliding-Window-Verfahren angewendet werden.

Der Verbindungsaufbau erfolgt in Analogie zum herkömmlichen Telefonsystem. Client-Server-Architektur Ein Anrufer (Client) sendet einen sog. request (Anforderung) an einen Teilnehmer (Server). Dieser „hebt ab“, indem er ein reply (Antwort) an den Client zurücksendet. Anschliessend findet der Datenaustausch statt. Danach wird die Verbindung wieder abgebrochen [Schn94][Mün98].

2.2.6 Das Internet Protocol (IP)

Das Internet Protocol (IP), auf der Netzwerkschicht (Ebene 3) angesiedelt, bildet zusammen mit dem Transmission Control Protocol (*TCP*) (Transportschicht) das zentrale Protokollpaar der Internet-Architektur. Die Hauptaufgabe des IP adressiert Rechner Internet Protokolls ist das Adressieren von Rechnern sowie das Fragmentieren von Paketen der darüberliegenden Schicht. IP stellt also die Endsystemverbindung der Partnerrechner her. Der darüberliegenden Ebene (Transportschicht) bietet IP einen sog. unzuverlässigen und verbindungslosen Dienst an. Wenn also, wie z.B. beim Dateitransfer, eine zuverlässige Übertragung gefordert wird, dann ist es Aufgabe eines der übergeordneten Protokolle (z.B. des Transportprotokolls), die Zuverlässigkeit zu gewährleisten [Schn94][Mün98].

2.2.7 File Transfer (FTP)

FTP ist ein Internet-Dienst, der speziell dazu dient, sich auf einem bestimmten Server-Rechner im Internet einzuwählen und von dort Dateien auf den eigenen Rechner zu übertragen *(Download)* oder eigene Dateien an den Server-Rechner zu übertragen *(Upload)*. Ferner bietet das FTP-Protokoll Befehle an, um auf dem entfernten Rechner Operationen durchzuführen wie Verzeichnisinhalte anzeigen, Verzeichnisse wechseln, Verzeichnisse anlegen oder Dateien löschen [Mün98].

2.2.8 Domain Name Service (DNS)

Das Domain Name System (DNS) [Moc87a][Moc87b] wurde geschaffen, um Rechnern statt den IP-Adressen auch logische Namen zuordnen zu können. Dies erfolgte in den Anfängen des Internet über eine zentral gehaltene Datei (*/etc/hosts* auf UNIX Systemen) /etc/hosts des Network Information Centers (NIC) die an alle Rechner jeder Domain regelmässig mittels ftp verschickt wurde und die jeder IP-Adresse eindeutig einen Namen zuordnete. Damals war der Adressraum noch flach , jedoch wurde im Laufe der Zeit eine grössere Hierarchie nötig.

Als das Internet wuchs, war eine mittels ftp verschickte hosts-Datei für jeden Rechner nicht mehr möglich. Das Aktualisieren einer solchen Datei wäre zu aufwendig und würde das Netzwerk zu sehr belasten. */etc/hosts* ist mittlerweile zu einer Datenbank angewachsen und wird in Zonen aufgeteilt und von dedizierten Rechnern, den Domain Name Servern, innerhalb der Zone verwaltet. Jeder Domain Name Server sieht also nur einen Teil des gesamten Domain Name Space. Insgesamt existieren drei Hauptkomponenten, aus denen sich das DNS zusammensetzt: Der Domain Name Space, Name Server und Resolver (siehe Abb. 9).

2.2.9 Routing und Gateways

Im Internet als dem Netz der Netze ist es zunächst nur innerhalb des eigenen Sub-Netzes möglich, Daten direkt von einer IP-Adresse zu einer anderen zu schicken. In allen anderen Fällen, wenn die Daten an eine andere Netzwerknummer geschickt werden sollen, treten Rechner auf den Plan, die den Verkehr zwischen den Netzen regeln. Solche Rechner werden als *Gateways* bezeichnet. Diese Rechner leiten Daten von

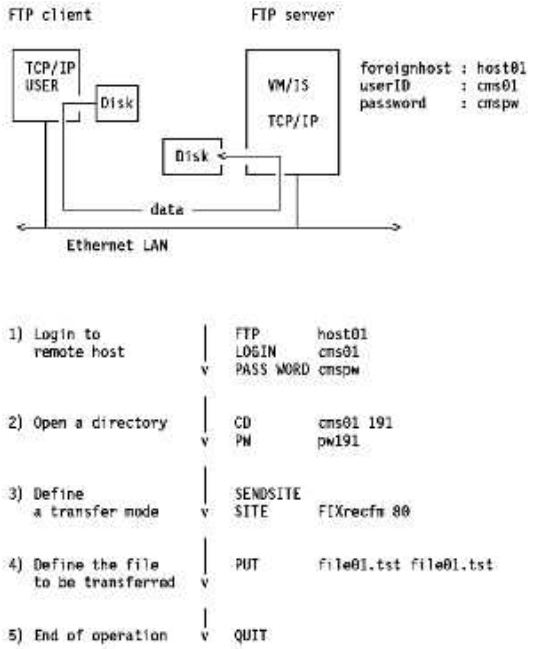

Abbildung 8: FTP-Szenario [IBM95]

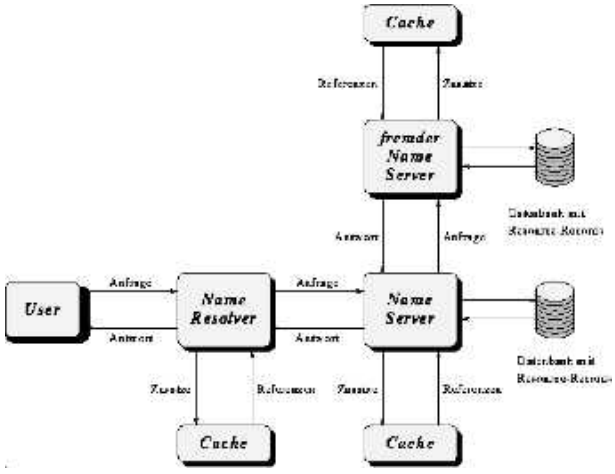

Abbildung 9: Beispiel einer Domain Name Anfrage [Schn94]

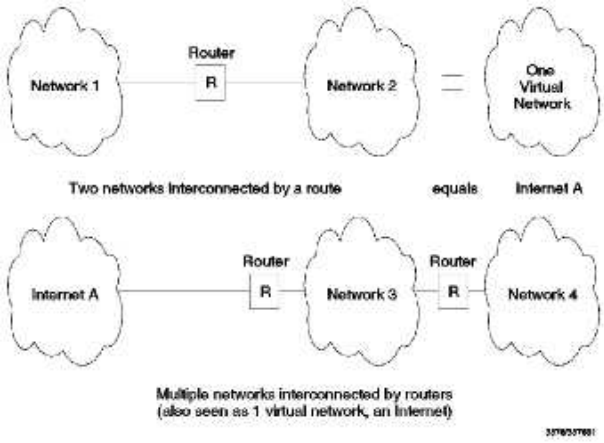

Abbildung 10: Internet-Architektur mit Router [IBM95]

Hostrechnern aus dem eigenen Sub-Netz an Gateways in anderen Sub-Netzen weiter und ankommende Daten von Gateways anderer Sub-Netze an die darin adressierten Host-Rechner im eigenen Sub-Netz. Ohne Gateways gäbe es gar kein Internet.

Das Weiterleiten der Daten zwischen Sub-Netzen wird als *Routing* bezeichnet. Die Beschreibung der möglichen Routen vom eigenen Netzwerk zu anderen Netzwerken sind in *Routing-Tabellen* auf den Gateway-Rechnern festgehalten.

Zu den Aufgaben eines Gateways gehört auch, eine Alternativ-Route zu finden, wenn die übliche Route nicht funktioniert, etwa, weil bei der entsprechenden Leitung eine Störung oder ein Datenstau aufgetreten ist. Gateways senden sich ständig Testpakete zu, um das Funktionieren der Verbindung zu testen und für Datentransfers "verkehrsarme" Wege zu finden.

Wenn also im Internet ein Datentransfer stattfindet, ist keinesfalls von vorneherein klar, welchen Weg die Daten nehmen. Sogar einzelne Pakete einer einzigen Sendung können völlig unterschiedliche Wege nehmen. Wenn Sie beispielsweise von der Schweiz aus eine WWW-Seite aufrufen, die auf einem Rechner in den USA liegt, kann es sein, dass die Hälfte der Seite über den Atlantik kommt und die andere über den Pazifik, bevor Ihr WWW-Browser sie anzeigen kann. Weder Sie noch Ihr Browser bekommen davon etwas mit [Mün98].

2.2.10 Uniform Resource Locator (URL)

Uniform Resource Locator, abgekürzt *URL*, ist ein weiterer Begriff der eng mit der Verbreitung des World Wide Web verknüpft ist. Zur Zeit existieren viele unterschiedliche Systeme zur Recherche im Internet. Bei der Recherche nach Dokumenten im Internet tritt das Problem auf, Ressourcen auf eine bestimmte Art zu identifizieren. Identifizierung von Dokumenten Ähnlich wie Büchern in einer Bibliothek eine eindeutige Nummer zugewiesen wird, damit sie für jedermann auffindbar sind, sollte es im Internet eine Möglichkeit geben, Ressourcen eindeutig zu benennen. Diese Aufgabe übernimmt der URL.

Da URLs nicht nur von Rechnern sondern auch von Menschen URL = Zugriffsart, Rechnername, Dateiname interpretiert werden, waren die Voraussetzungen bei deren Einführung, kurz und verständlich zu sein. Dazu sollen sie aus druckbaren Zeichen, ohne Leerzeichen bestehen. Man benötigt im wesentlichen drei Informationen, um eine Ressource im Netz zu beschreiben. Dies sind eine *Zugriffsmethode*, ein *Rechnernam*e und ein *Verzeichnis- oder Dateiname*. So liefert der String

```
ftp://ftp.glue.ch/pub/info/sms.ps
```

genügend Information, um das Postscriptfile sms.ps im Internet zu finden. Man baue eine FTP-Verbindung zum Rechner

```
ftp.glue.ch
```

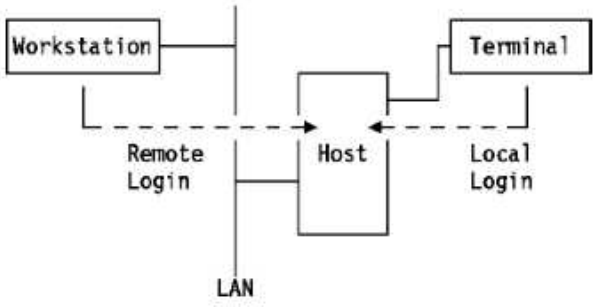

Abbildung 11: Remote login mit telnet [IBM95]

auf und hole sich die Datei

```
sms.ps
```

die im Verzeichnis

```
/pub/info/
```

liegt[Schn94].

2.2.11 E-Mail

E-Mail (elektronische Post) ist wohl der am meisten genutzte Internet-Dienst. E-Mail erlaubt die persönliche Übermittlung von Nachrichten und Dateien von einem Sender an einen Empfänger.

Das heutige E-Mail-System hat aber auch noch mit einigen Problemen zu kämpfen. Eine normale E-Mail ist auf dem Weg vom Sender zum Empfänger etwa so geheim wie eine Ansichtskarte. Für vertrauliche Mitteilungen oder sensible Daten ist sie ungeeignet. Mittlerweile gibt es Verschlüsselungsverfahren wie *PGP (Pretty Good Privacy)***,** die das individuelle Kodieren und Dekodieren von E-Mails und angehängten Dateien erlauben. Voraussetzung ist dazu jedoch, dass sowohl Sender als auch Empfänger über eine entsprechende Zusatzsoftware verfügen und zuvor ihre öffentlichen Kodierschlüssel austauschen [Mün98][Schn94].

2.2.12 Telnet

Telnet [PR83] ist der erste Dienst, der im Internet implementiert wurde [LR93]. Mit Telnet, dem Standard-Remote-Login-Dienst auf dem Internet, kann man auf Rechnern im Netz so arbeiten, als ob die eigene Tastatur und das eigene Terminal direkt am entfernten Rechner angeschlossen wären; abgesehen von stellenweise längeren Antwortzeiten bei langsamen Netzverbindungen.

Zwischen UNIX-Rechnern wird vor allem das Kommando *rlogin* eingesetzt, das etwa die gleichen Funktionen wie Telnet realisiert. Telnet und rlogin Es unterstützt speziell die UNIX-Umgebung und kann so konfiguriert werden, dass zum Aufbau von Verbindungen zwischen *Trusted Hosts* kein expliziter Login-Vorgang erforderlich ist [Kan91]. Für die Arbeit in heterogenen Umgebungen ist jedoch Telnet vorzuziehen, da es u.a. Eigenschaften besitzt, die die Zusammenarbeit mit Grossrechnersystemen erleichtert [San90].

2.2.13 World Wide Web (WWW)

Das World Wide Web (WWW) ist der jüngste Dienst innerhalb des Internet. Das Web zeichnet sich dadurch aus, dass es auch ungeübteren Anwendern erlaubt, sich im Informationsangebot zu bewegen. Wer etwa mit einem FTP-Programm einen FTP-Server aufruft, muss sich in komplexen, unbekannten Verzeichnisstrukturen zurechtfinden und sich an Dateinamen orientieren. Interessante Dateien kann er auf seinen Rechner downloaden, um sie später zu öffnen. Im WWW dagegen erscheinen Informationen gleich beim

Aufruf am Bildschirm. Wenn man mit einem WWW-Browser im Web unterwegs ist, braucht man sich nicht um Dateinamen oder um komplizierte Eingabebefehle zu kümmern [Mün98].

2.2.14 Eigenschaften von HTML

Allgemein *HTML* bedeutet *HyperText Markup Language*. Es handelt sich dabei um eine Sprache, die mit Hilfe von *SGML* (Standard Generalized Markup Language) definiert wird. SGML ist als ISO-Norm 8879 festgeschrieben.

HTML ist eine sogenannte Auszeichnungssprache (Markup Language). Sie hat die Aufgabe, die logischen Bestandteile eines Dokuments zu beschreiben. Als Auszeichnungssprache enthält HTML daher Befehle zum Markieren typischer Elemente eines Dokuments, wie Überschriften, Textabsätze, Listen, Tabellen oder Grafikreferenzen.

Das Beschreibungsschema von HTML geht von einer hierarchischen Gliederung aus. HTML beschreibt Dokumente. Dokumente haben globale Eigenschaften wie zum Beispiel einen Titel oder eine Hintergrundfarbe. Der eigentliche Inhalt besteht aus Elementen, zum Beispiel einer Überschrift 1. Ordnung. Einige dieser Elemente haben wiederum Unterelemente. So enthält ein Textabsatz zum Beispiel eine als fett markierte Textstelle, eine Aufzählungsliste besteht aus einzelnen Listenpunkten, und eine Tabelle gliedert sich in einzelne Tabellenzellen.

Die meisten dieser Elemente haben einen fest definierbaren Erstreckungsraum. So geht eine Überschrift vom ersten bis zum letzten Zeichen, eine Aufzählungsliste vom ersten bis zum letzten Listenpunkt, oder eine Tabelle von der ersten bis zur letzten Zelle. Auszeichnungen markieren Anfang und Ende von Elementen. Um etwa eine Überschrift auszuzeichnen, lautet das Schema:

```
[Überschrift] Text der Überschrift [Ende Überschrift]
```

Bei einem Element, das wiederum Unterelemente besitzt, etwa einer Aufzählungsliste, lässt sich das gleiche Schema anwenden:

```
[Liste]
[Listenpunkt] Text des Listenpunkts [Ende Listenpunkt]
[Listenpunkt] Text des Listenpunkts [Ende Listenpunkt]
[Ende Liste]
```

Grundgerüst einer HTML-Datei Eine gewöhnliche HTML-Datei besteht grundsätzlich aus folgenden zwei Teilen:

- Header (Kopf) (enthält Angaben zu Titel u.ä.)
- Body (Körper) (enthält den eigentlichen Text mit Überschriften, Verweisen, Grafikreferenzen usw.)

Beispiel:

```
<html>
<head>
<title>
  Text des Titels
</title>
</head>
<body>
  Text, Verweise, Grafikreferenzen usw.
</body>
</html>
```

Alle anderen HTML-Elemente werden in dieses Gerüst eingfügt.

Formulare definieren Wichtig für diese Untersuchung sind HTML-Formulare. Sie werden verwendet. um Dateneingabe zu ermöglichen. Man kann an einer beliebigen Stelle innerhalb des Dateikörpers einer HTML-Datei ein Formular definieren:

```
<form action="mailto:vogel@glue.ch" method=post enctype="text/plain">
... Elemente des Formulars wie Eingabefelder, Auswahllisten, Buttons usw. ...
</form>
```

Mit <*form ...*> definiert man ein Formular (form = Formular). Alles, was zwischen diesem einleitenden Tag und dem abschliessenden Tag <*/form*> steht, gehört zum Formular. Das sind hauptsächlich Elemente des Formulars wie Eingabefelder, Auswahllisten oder Buttons.

Im einleitenden <*form*>-Tag gibt man mit *action*= an, was mit den ausgefüllten Formulardaten passieren soll, wenn der Anwender das Formular abschickt (*action = Aktion*). Die Angabe bei *action*= ist entweder eine *E-Mail-Adresse* mit vorangestelltem *mailto:*. Dann werden die ausgefüllten Formulardaten an diese E-Mail-Adresse geschickt. Oder man ruft ein Programm auf dem Server-Rechner - meistens ein *CGI-Programm* - auf, das die Daten weiterverarbeitet.

Bei der Formulardefinition muss man als nächstes die Übertragungsmethode angeben. Dabei gibt es zwei Möglichkeiten:

Wenn man *method=get* wählt, werden die Daten des ausgefüllten Formulars auf WWW-Servern mit installiertem HTTP-Protokoll in der CGI-Umgebungsvariablen QUERY_STRING gespeichert (*method = Methode, get = bekommen*). Das CGI-Programm muss den Inhalt dieser Umgebungsvariablen auslesen und verarbeiten.

Wenn man *method=post* wählt, werden die Daten des augefüllten Formulars auf dem Server-Rechner von "stdin" zur Verfügung gestellt, und das *CGI-Programm* muss sie behandeln wie eine Benutzereingabe, die auf der Kommandozeile gemacht wurde (*post = verschicken*). Da in diesem Fall kein EndOfFile-Signal (EOF) gesendet wird, muss das CGI-Programm die CGI-Umgebungsvariable CONTENT_LENGTH auslesen, um die Länge der übermittelten Daten und damit deren Ende zu ermitteln [Mün98].

Weitere Formularelemente

- <INPUT TYPE="text" NAME="adr"> *definiert ein einzeiliges Eingabefeld mit Name* adr.
- <TEXTAREA NAME="Message" ROWS="7" COLS="30" ></TEXTAREA> *definiert ein mehrzeiliges Eingabefeld.*
- <INPUT TYPE=reset VALUE="Clear form"> *und*
- <INPUT TYPE=submit VALUE="Send message"> *definieren Buttons zum Abbrechen und zum Absenden.*

Ein Beispiel: HTML-Dokument zum Versenden von SMS-Nachrichten

```
<HTML><HEAD><TITLE>Message the world!</TITLE></HEAD>
<BODY>
<FORM NAME="SMS" action="mailto:vogel@glue.ch" method=post
      enctype="text/plain">
Recipients Phonenumber:
<INPUT TYPE="text" NAME="adr"> e.g. 0792358735 <BR>
Your Phonenumber:
<INPUT TYPE="text" NAME="abs"> e.g. 0313853011 <BR>
Your message:
<TEXTAREA NAME="Message" ROWS="7" COLS="30" ></TEXTAREA>
<INPUT TYPE=reset VALUE="Clear form">
<INPUT TYPE=submit VALUE="Send message">
</FORM>
</BODY>
</HTML>
```

(siehe auch [HTML-RFC] für aktuelle Spezifikationen vom W3C).

2.2.15 Cookies [STKO99]

Cookies sind sehr kleine Daten, die durch eine WWW-Seite auf dem Client-Computer gespeichert werden. Sie sind für folgende Anwendungsgebiete interessant:

- Speichern von Daten, so dass auf diese von verschiedenen Seiten einer Homepage zugegriffen werden kann (-> User-Tracking)
- Anzeigen, was sich seit dem letzten Besuch auf einer Homepage verändert hat
- Feststellen, wie oft jemand zu einer bestimmten Seite zurückkehrt
- Anpassen einer Homepage an die speziellen Wünsche der Besucher

Cookies sind dafür vorgesehen, dass ein Server einzelne Daten auf dem Client-Computer speichern kann. Diese Daten können über längere Zeit erhalten bleiben, so dass auf die Daten zu einem späteren Zeitpunkt wieder zugegriffen werden kann. Wird eine WWW-Seite eines Servers angefordert, wird überprüft, ob ein Cookie auf dem Client-Computer existiert, das von diesem Server erzeugt wurde. Kann der Client-Computer ein Cookie von diesem Server finden, wird die darin enthaltene Information als Teil des HTTP-Request zum Server geschickt.

In einem Cookie können z.B. Name und E-Mail-Adresse gespeichert werden, wenn ein Benutzer diese zuvor in einem Formular eingegeben hat. Cookies werden vom Browser in einen speziell dafür reservierten Bereich abgelegt (bei Netscape ist das die Datei cookies, beim MS IE das Verzeichnis \cookies).

Format Die Anweisung zum Setzen eines Cookie kann z.B. über einen *Set-Cookie-Header* als Teil einer *HTTP-Response* erfolgen, meistens mittels eines *CGI-Scriptes* (siehe später). In diesem Fall wird dem Header ein Datenstück mit folgender Syntax angehängt: *Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure* (siehe auch [COO99]).

Es ist aber auch möglich Cookies von der Clientseite aus zu setzen, z.B. kann man mittels JavaScript auf Benutzeraktionen (Drücken eines Buttons, Bewegen der Maus etc.) reagieren und ohne Kommunikation mit dem HTTP-Server ein Cookie ablegen. Dies erfolgt mit folgendem Befehl: *document.cookie = NAME+ "=" + escape(VALUE) + "; expires=" + DATE+ "; path=PATH";* (siehe auch [STKO99]).

Beschränkungen Cookies können nicht beliebig gross sein. Ein einzelnes Cookie kann max. 4 KByte gross sein. Ist ein Cookie grösser als 4 KByte, wird es entsprechend gekürzt. Der Name des Cookies bleibt dabei erhalten. Jeder Server oder jede Domain kann nur 20 Cookies auf einem Client erzeugen. Wird versucht, mehr als 20 Cookies zu speichern, werden die zuerst erzeugten Cookies überschrieben. Der Client-Computer kann max. 300 Cookies entgegennehmen. Sind auf einem Computer alle 300 Cookies gesetzt und möchte man ein neues Cookie schreiben, dann wird ein altes Cookie überschrieben.

Sicherheit / Datenschutz [HAS99] Die Möglichkeiten mit Cookies muss man nicht als Bedrohung ansehen, denn Cookies sind in ihrer Anwendung sehr eingeschränkt. Einer der wichtigsten Punkte ist, dassein Cookie nur Daten in einer bestimmten Datei speichern kann. Ein Cookie kann kein ausführbares Programm auf den Computer des Benutzers schleusen.

Cookies stehen im Verdacht, den Datenschutz auszuhöhlen, da in ihnen benutzerspezifische Daten gespeichert werden können. Zu den geäusserten Befürchtungen gehört, dass somit Benutzerprofile erstellt werden können, die von jedem Server weltweit abrufbar sind. Dazu einige Informationen.

- Entgegen landläufiger Meinung ist die Cookie-Datei für Server überhaupt nicht lesbar. Wenn ein Server ein Dokument verschickt, kann er den Browser bitten, ein Cookie zu speichern. Wenn später ein Browser ein Dokument anfragt, sieht er in seiner Cookie-Datei nach, ob es ein zur URL passendes Cookie gibt. Wenn dem so ist werden alle passenden Cookies - und sonst *keines* - mitgeschickt. Der Browser kontrolliert also die Herausgabe der Cookies.

- Jedes Cookie besitzt ein *Verfallsdatum*, das vom ausgebenden Server definiert wird. Wenn dieses überschritten ist, sollte der Browser das Cookie löschen, auf jeden Fall sollte er es nicht mehr herausgeben. Wenn er dies dennoch tut, ist er fehlerhaft und genügt nicht der Spezifikation. Besitzt ein Cookie kein explizites Verfallsdatum, so wird das Cookie beim Beenden der Browser-Sitzung gelöscht.
- Ein Cookie ist per Voreinstellung nur für den Server lesbar, der es ausgegeben hat, sogar nur für das Verzeichnis, in dem das ausgebende Dokument lag, sowie dessen Unterverzeichnisse. Dieser Lesbarkeitsbereich lässt sich sowohl innerhalb eines Servers erweitern als auch über Servergrenzen hinweg. Dazu müssen jeweils Domains angegeben werden, für die das Cookie ebenfalls lesbar ist. Jede solche Domain muss mindestens zwei Punkte enthalten (also wäre ".com" nicht zulässig).
- Im Übrigen muss natürlich auch dafür gesorgt werden, dass andere die Cookie-Datei nicht direkt von der Festplatte lesen können.

Alles Gesagte gilt natürlich nur für Browser, die der Spezifikation genügen und - in diesem Aspekt - fehlerfrei implementiert sind. Was den Netscape Navigator 3.x sowie den Internet Explorer 3.0 angeht, sind keine solchen Fehler bekannt.

2.3 Lösungswege

Für die Benutzer des SMS-Service soll also ein Adressbuch geführt werden. Diese Daten können auf der Client- oder auf der Serverseite abgelegt werden.

Auf der Clientseite geschieht dies mit Hilfe von *Cookies*. Solche Cookie-Daten können mit Hilfe eines zusätzlichen Programmes in eine Datenbank (zB. mySQL, einfaches Textfile, PalmPilot DB, Netscape-Adressbook etc.) exportiert und von dort wieder importiert werden. Eine andere Möglichkeit besteht darin, einen eigenen HTTP-Client zu entwickeln (anstatt einen Web-Browser zu benutzen), damit man die Daten ohne Umweg über Cookies in einer Datenbank ablegen kann.

Bei der Datenhaltung auf der Serverseite ist der Aufwand um einiges grösser, da man sich Gedanken über *Datenschutz* (wer hat Zugriff auf meine Daten), *Datensicherheit* (können die Daten verloren gehen) und *Verfügbarkeit* (habe ich jederzeit Zugriff auf meine Daten) machen muss. In der Lösungszkizze (siehe Abb. 12) sind einige Lösungswege mit verschiedenen Technologien dargestellt, auf die in den nachfolgenden Kapiteln detailiert eingegangen wird.

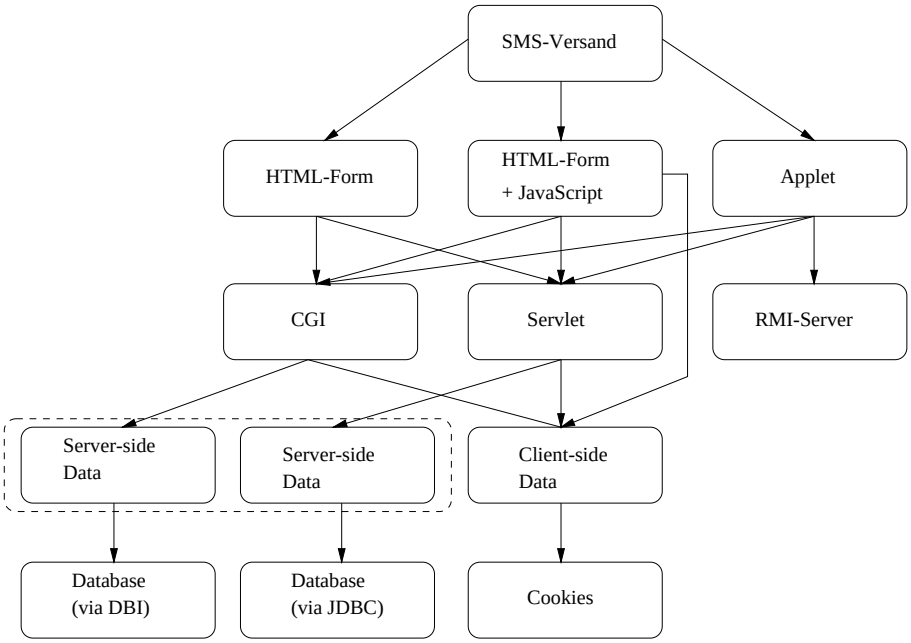

Abbildung 12: Lösungsszkizze

3 Der Apache Web Server[APA]

In diesem Kapitel wird der Apache Web-Server, der für diese Untersuchung benützt wird, kurz vorgestellt. Zudem wird gezeigt, wie man die Problematik des *Datenschutzes* (Zugriffskontrolle, sichere Datenübertragung) und des *Usertracking* mit Hilfe des Apache Web-Server lösen kann. Die Realisation von Usertracking mit *hidden fields* oder mit *url-rewriting* wird erklärt.

3.1 Allgemein

Der Name *Apache* stammt von *A PAtCHy server*, weil er ursprünglich aus existierendem Code und Patch-Files zusammengesetzt wurde. Der *Apache Web Server* wurde von einer Gruppe von Programmierern entwickelt, die zuvor bei der Entwicklung des NCSA-HTTPd-Servers mitgewirkt hatten. Das Apache-Projekt stand unter dem Vorzeichen, einen sicheren, effizienten und leicht zu erweiternden Server zu erhalten. Dadurch entstand ein Server, der sich einerseits stark an den NSCA-HTTPd-Server anlehnt aber andererseits durch die Behebung von Fehlern, einer strengen Anwendung des HTTP-Standards und einer besseren Perfomance eine wesentliche Verbesserung des NCSA-HTTPd-Servers darstellt. Aus diesen Gründen wurde aus dem Apache Web Server, laut Netcraft, einer der meistbenutzten im Internet mit einem Anteil von 43%.

3.2 Zugriffskontrolle

Es gibt prinzipiell zwei Möglichkeiten, um den Zugriff auf den Web-Server oder bestimmte Bereiche des Web-Servers zu kontrollieren. Zum einem kann ein *Authentikationsmechanismus* verwendet werden, bei dem der Client nur dann Zugriff erhält, wenn er eine *User-ID* und ein gültiges *Passwort* für den jeweiligen URL angegeben hat. Weiterhin besteht die Möglichkeit, einen Zugriff auf Basis der *Ip-Adresse* des zugreifenden Rechners zuzulassen oder abzuweisen [LAR98].

3.2.1 Passwortauthentikation

Bei diesem *Authentikationsmechanismen* unterscheidet man zwischen *Basic*- und *Digest-Authentikation*. Letztere ist die sicherere Variante, da das Passwort niemals über die Leitung geschickt wird. Jedoch wird

diese Variante bisher erst von wenigen Web-Clients unterstützt. Die Verwendung von *Basic-Authentikation* ist grundsätzlich als unsicher zu betrachten, da hierbei das Passwort unverschlüsselt vom Client an den Web-Server übertragen wird und es für einen Angreifer ein Leichtes ist, dieses mitzuschneiden. Aus diesem Grund sollte man auch niemals bei dieser Art der Authentikation die Systempasswörter verwenden.

Die *Basic-Authentikation* eignet sich in keiner Weise, um sicherheitsrelevante Daten auf dem Web-Server zu schützen. Allenfalls in Verbindung mit *SSL (Secure Sockets Layer, mehr dazu später)* ist ein Schutz vor dem Mithören des Passwortes gegeben.

Wo immer es geht, sollte man die *Digest-Authentikation* verwenden. Auch wenn bei diesem Mechanismus die Passwörter nicht übertragen werden, so werden doch die Daten selbst unverschlüsselt übertragen und können von einem Angreifer mitgehört (*Network Sniffing*) oder, was eventuell noch fataler sein kann, gegen andere Daten ausgetauscht werden (*Data Spoofing*) [LAR98].

3.2.2 IP-basierte Zugriffskontrolle

Über die Konfigurationsanweisungen *order, deny* und *allow* lässt sich auf Basis eines Hostname bzw. einer IP-Adresse der Zugriff auf bestimmte Verzeichnisse und Dateien kontrollieren, z.B. kann man ein Verzeichnis erstellen, das nur für Rechner aus der lokalen Domain zugänglich ist.

Doch auch hier können Angreifer mittels *UDP-Flooding* (Überfluten des Web-Servers mit tausenden von gefälschten Nameserver-Antworten), *Cache Loading* (Laden von Falschinformationen in den Nameserver) oder *IP-Spoofing* (Vorgaukeln einer falschen IP-Adresse) unberechtigten Zugang zu Ressourcen erhalten [LAR98].

3.2.3 Ein Blick in die Konfiguration für die benutzerdefinierte Zugriffskontrolle (*Basic Authentikation*)

Jeder Webserver besitzt eine zentrale Konfiguration, in welcher der Webmaster die Eigenschaften, Dienste, Zugriffsrechte usw. seines Servers zentral beschreiben kann. Hat er eine grosse Anzahl von Benutzern mit unterschiedlichen Anforderungen für die von ihnen auf dem Server abgelegten Dokumente, dann kann eine solche Konfiguration jedoch schnell unübersichtlich und schwer wartbar werden.

Sehr hilfreich ist hierbei das Konzept der *dezentralen Konfiguration* mit Hilfe von *benutzereigenen Konfigurationsdateien* des Webservers: Zusätzlich zur zentralen Konfiguration kann jeder Benutzer das Verhalten bei Zugriffen auf seine eigenen Seiten durch das Anlegen einer Textdatei mit dem Namen *.htaccess* selbst beeinflussen.

In einer solchen Konfigurationsdatei kann man z.B.

- die Verwendbarkeit von *Server Side Includes* und *CGI-Programmen* regulieren,
- die Art der *Anzeige bei directory browsing* in vielfältiger Weise beeinflussen,
- automatisch eines von mehreren alternativen Dokumenten in der *Landessprache des Besuchers* (welche dieser in seinem Browser eingestellt hat) zurückliefern (sehr praktisch bei mehrsprachigen Präsentationen z. B. einer international tätigen Firma),
- eigene *MIME-Typen* und zugehörige *Handler* definieren,
- eigene Dokumente zur *Behandlung von HTTP-Fehlern* zuweisen, und vor allem
- *Berechtigungsmechanismen* für den Zugriff aller Besucher auf den Inhalt dieses Verzeichnisses und aller seiner Unterverzeichnisse festlegen.

Beispiel - Datei .htaccess:

```
authType basic
       # Schutzverfahren
authName Web-Bereich_der_Abteilung_'Entwicklung'
       # Name des Berechtigungsbereichs)
```

```
authUserFile  /home/ms/httpzugriff/benutzer.txt
       # Datei für Benutzerbeschreibung (nicht im Dokumentbaum!)
authGroupFile /home/ms/httpzugriff/gruppen.txt
       # Datei für Gruppenbeschreibung (nicht im Dokumentbaum!)
order deny,allow
       # Genehmigungen überdecken Verbote ...
deny from all
       # ... aber erst einmal alles verbieten
satisfy all
       # nur Rechneradresse PLUS Benutzerkennung berechtigen zum Zugriff!
allow from 153.46.90.
       # nur Rechner dieses IP-Adressenbereichs dürfen zugreifen
require group entwickler
       # alle Entwickler dürfen zugreifen
require user chef sekretae
       # Firmenchef und Sekretärin dürfen auch zugreifen
```

Beispiel - Datei benutzer.txt:

```
# (Demo: jeweils Passwort = Benutzerkennung, crypt()-salt = "IN")
chef:IN3WY1lATStaY
sekretae:INz8B568yvs7Y
schmidt:INQaGJBu4yljQ
meier:INqq3xgT4zpp6
huber:INT.EAmojNwN6
test:INnA1BztLIaac
```

Beispiel - Datei gruppen.txt:

```
# Liste aller Mitglieder der definierten Gruppen
entwickler: meier huber schmidt
```

Erläuterung: Den Zugriff auf Dokumente eines Verzeichnisses kann dessen Besitzer zunächst einmal auf *Hostnamen*, *IP-Adressen* und *Präfixe* dieser beiden (also IP-Netzmasken und Domains!) beschränken. Das ist ganz praktisch in einem grossen Firmennetz mit mehreren Teilnetzen bzw. Subdomains, wo man Filialen, Abteilungen usw. einfach dadurch voneinander abgrenzen kann, indem man sich auf eine bestehende Netzstruktur bezieht. Insbesondere müssen sich Besucher, die auf bestimmte Seiten zugreifen dürfen, dazu nicht explizit ausweisen - und merken ggf. gar nicht, dass diese Seiten für andere Besucher gesperrt sind.

Will man weitergehende Kontrollen verwenden, bei denen der Besucher sich beim ersten Zugriff seiner Browser-Sitzung (danach merkt sich der Webserver das Ergebnis) explizit ausweisen muss, definiert man zunächst einmal einen *Berechtigungsbereich*, indem man für sein Verzeichnis einen logischen Namen vergibt. Betritt ein Besucher einen solchen Bereich, indem er eine URL innerhalb des geschützten Verzeichnisses anzusprechen versucht, dann schickt der Webserver zunächst eine *Aufforderung zur Authentifizierung* an den Client. Der Browser zeigt dann dem Besucher z. B. eine Dialogbox mit Eingabefeldern für *Benutzerkennung* und *Kennwort* an, in der auch der Name des Bereichs angezeigt wird, damit der Besucher erkennen kann, wofür er sich gerade ausweisen soll. Weist sich der Besucher (gemäss der definierten Berechtigungen) korrekt aus, dann darf er auf das Dokument zugreifen, andernfalls löst der Webserver einen *HTTP-Fehler 403* aus.

Innerhalb seines Berechtigungsbereichs kann der Besitzer eines Verzeichnisses Genehmigungen bzw. Verbote auf Benutzer- bzw. Gruppenkennungen beziehen. Die Definition dieser Kennungen erfolgt in einer entsprechenden *Benutzer-* bzw. *Gruppendatei*: In der *Benutzerdatei* stehen Zeilen mit Benutzerkennung und verschlüsseltem (!) Passwort (mit der UNIX-Systemroutine crypt()), in der *Gruppendatei* stehen Zeilen mit Listen von Benutzern der einzelnen Gruppenkennungen. Ausserdem kann man angeben, ob

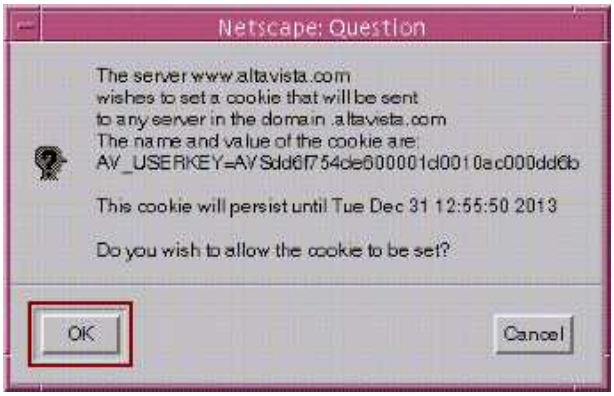

Abbildung 13: Beispiel einer Cookie-Anfrage

der Besucher sowohl von einem berechtigten Host aus als auch mit einer gültigen Benutzerkennung auf die Seite zugreifen kann oder ob die Kombination dieser beiden Rechte für einen erfolgreichen Zugriff erforderlich ist [Mün98].

3.3 User-Tracking

3.3.1 Das Usertrack-Modul *mod_usertrack*

Das Modulkonzept des Apache ist etwas, das ihn von anderen Web-Servern unterscheidet. Einerseits kann man durch die Einbindung bzw. Weglassen von Modulen kontrollieren, welche Funktionalität der Apache besitzen soll, und andererseits ist es möglich, eigene Module zu erstellen und einzubinden.

Das Usertrack-Modul *mod_usertrack* benutzt sogenannte *HTTP-Cookies*, um das Benutzerverhalten festhalten, bzw. protokollieren zu können. Das Prinzip ist recht simpel. Für einen neuen Benutzer wird ein *Cookie* gesetzt, das eine eindeutige Zahl enthält (siehe Abb. 13). Mit Hilfe der Logfiles lässt sich dann überprüfen, welche Dateien ein Benutzer in welcher Reihenfolge vom Web-Server geladen hat, und wie oft er den Web-Server besucht hat. Ein viel besuchter Web-Server verbraucht hier viel Rechenzeit mit dem Führen von Logfiles und Cookie-Handling. Die Aussagekraft ist jedoch zweifelhaft, denn nicht alle Browser untersützen Cookies, und wenn sie es tun, bieten sie dem Nutzer die Möglichkeit, das Setzen eines Cookies abzulehnen.

Um bei anderen Anwendungen, z.B. *CGI-Scripten*, die Cookies zu benutzen, wird mod_usertrack *nicht* benötigt [LAR98]. Will man mit einem Benutzer einen Dialog führen, so muss bei jedem request zuerst das Cookie aus den Headerdaten analysiert werden. Erst dann weiss man, mit welchem User man *spricht*. Das Cookie dient in diesem Fall als UserID. Falls der Browser des Users die Cookies ablehnt, so funktioniert der Dialog nicht. Der Webserver würde den User immer als Neu-Benutzer ansehen. Bessere Methoden für User-Tracking, welche unabhängig von den Einstellungen des Browsers sind, werden in den folgenden zwei Abschnitten beschrieben.

3.3.2 Hidden Fields

Für jeden neuen Benutzer legt man (serverseitig) eine eindeutige Sessionnummer an. Die HTML-Seite, welche für den User generiert wird, enthält diese Nummer als *Hidden Field* in einem Formular:

```
<form name="Feedback" action="/cgi-bin/order.pl" method=post enctype="text/plain">
 Ihr Name:
 <input name="UserName>"
 <input type=hidden name="sessionID" value="976523765">
 <input type=submit value="Absenden">
</form>
```

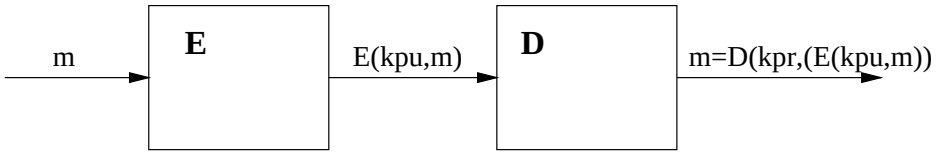

Abbildung 14: asymmetrisch verschlüsselte Nachrichtenübermittlung

Beim nächsten Request des Users kann der Webserver den Benutzer anhand der Sessionnummer identifizieren/wiedererkennen..

3.3.3 URL-Rewriting

Anstatt die Nummer in ein Formular zu integrieren, kann man sie direk an einen cgi-Aufruf anhängen (*URL-Rewriting*):

```
<a href="/cgi-bin/showinfo.pl?sessionID=8745723645836">weiter</a>
```

Der Effekt ist in beiden Fällen der gleiche: Die Sessionnummer wird als Parameter an das cgi-script übergeben. Somit kann man einen Dialog mit dem Benutzer gewährleisten. Dies erfordert das Führen von Logfiles auf der Serverseite.

3.4 Sicherer HTTP-Server ([GNUPRIV])

Damit die Daten währen der Übertragung vom Webserver and den Client von Dritten nicht gelesen werden können, muss man die Daten verschlüsseln. Dazu ist es notwendig, dass zwischen dem Webserver und dem Client ein geheimer Schlüssel vereinbart wird, so dass alle Daten mit einem festen Algorithmus ver- und entschlüsselt werden können. Hierzu sind einige kryptographische Grundbegriffe nötig.

3.4.1 Das asymmetrische Kryptographiesystem RSA

Das RSA Kryptographiesystem wurde nach seinen Erfindern R. Rivest, A. Shamir und L. Adelman benannt. Es gehört zu den asymmetrischen oder public-key Kryptographiesystemen . Das bedeutet, dass jeder Benutzer einen geheimen Schlüssel *kpr* und einen öffentlichen Schlüssel *kpu* generiert. *kpr* kann nicht aus *kpu* ohne Primfaktorzerlegung berechnet werden. Somit kann kein Angreifer auf den Inhalt der Nachricht schließen und es ist möglich, *kpu* mittels Webseite, E-Mail o.ä. ohne Bedenken zu veröffentlichen.

Unter der Annahme, dass Alice der öffentliche Schlüssel von Bob bekannt ist, gestaltet sich die Übermittlung einer chiffrierten Nachricht von Alice zu Bob folgendermassen:

Beispiel: Bob hat Alice seinen öffentlichen Schlüssel *kpu* mitgeteilt. Alice möchte eine Nachricht *m* an Bob senden. Alice verschlüsselt *m* mit Hilfe der RSA-Funktion *E* und dem öffentlichen Schlüssel *kpu* von Bob und sendet *E(kpu,m)* an diesen. Bob entschlüsselt *E(kpu,m)* mit Hilfe seines privaten Schlüssels *kpr* sowie der RSA-Funktion *D* und erhält die Nachricht *m*.

3.4.2 Digitale Unterschriften

Digitale Unterschriften werden durch die Entschlüsselung vom Hashwert einer Nachricht mit dem privaten Schlüssel des Absenders erzeugt. Dieser Hashwert wird zusammen mit der Nachricht an den Empfänger versendet.

Der Empfänger einer Nachricht mit digitaler Unterschrift verschlüsselt dann mit Hilfe des öffentlichen Schlüssels des Absenders diesen Hashwert. Dann erzeugt er mit dem gleichen Hashverfahren einen eigenen Hashwert und vergleicht ihn mit dem ihm gesandten. Der Empfänger kann sich, sofern der öffentliche Schlüssel wirklich von der Person stammt, die sich als Absender ausgibt, bei Übereinstimmung sicher

sein, dass die Nachricht nur vom Absender stammen kann, da nur dieser den Hashwert mit seinem privaten Schlüssel verschlüsseln kann, und dass die Nachricht von keinem Dritten verändert worden ist.

3.4.3 Zertifikate

Die bisherige Kommunikation verlief unter der Annahme, dass sowohl Sender als auch Empfänger sicher sind, dass der Partner wirklich derjenige ist, für den er sich ausgibt. Um Sicherheit darüber zu erlangen, dass die Gegenseite kein Angreifer ist, wurden autorisierte Zertifizierungsstellen eingerichtet, die Zertifikate von eingetragenen Stellen auf Anfrage verteilen.

Ein Zertifikat ist ein überprüfbares, öffentliches Dokument, das Informationen über seinen Besitzer enthält und von einer Zertifizierungsstelle unterschrieben ist. Zum Betrieb dieser Zertifizierungsstellen ist es notwendig, ein asymmetrisches Kryptographiesystem wie das hier vorgestellte RSA zu benutzen. Bei dem SSL-Protokoll , welches HTTP-Server mit verschlüsselter Datenübertragung benutzen, z.B. Apache-SSL, wird davon ausgegangen, dass der Client (hier der Browser des Benutzers) eine Auflistung von Zertifizierungsstellen sowie deren öffentliche Schlüssel kennt, wobei in der Regel auch Zertikfikate vom Server direkt akzeptiert werden können.

3.4.4 Verschlüsselung und Zertifikation

SSL steht für *Secure Socket Layer* und basiert auf asymmetrischer Verschlüsselung. In dem Falle, dass ein Browser einen Webserver per *https* anspricht, übergibt der Web-Server dem Browser den öffentlichen Schlüssel. Mit diesem Schlüssel können nun alle Daten, die der Client zum Server schickt, codiert werden. Somit können wir zwar gewährleisten das keiner in der Lage ist "mitzuhören", jedoch könnten die Daten an einen fremden Server geschickt werden. Um so etwas zu vermeiden muss der Schlüssel vom Server von einer *Certification-Authority(CA)* anerkannt und bestätigt werden. Unter *OpenSSL* kann sich der User seine eigene *CA* generieren lassen, oder der User wendet sich an eine kommerzielle CA, deren Signaturen Netscape schon bekannt sind. Der User kann sowohl für den Server als auch für die Clients CAs ausstellen. Falls der Server nur von einem kleinen Kreis von Clients angesprochen werden soll, wird jedem dieser Clients eine CA ausgestellt. Mit Hilfe dieser CAs kann sich der Client dem Server gegenüber als vertrauenswürdig erweisen. Die 2. Möglichkeit wäre für den Server ein Zertifikat auszustellen oder 3. für den Client und den Server. Das wechselseitige Vertrauen von Server und Client wird durch die Zertifikate gewährleistet. Diese Zertifikate enthalten Informationen wie Herausgeber des Zertifikats, Public Key, Gültigkeit/Verfallsdatum, Version/Seriennummer, sowie ein paar Felder, die die Identität des Herausgebers beschreiben: Common name(CN), Organization or Company(O), Organizational Unit(OU), City/Locality(L), State/Province(ST) und Country(C).

3.5 Rechnersicherheit

Die Bemühungen um die Sicherheit des Web-Servers selbst nützen wenig, wenn der Server-Rechner nicht sicher ist. Ein idealer Server-Rechner wäre ein Rechner, der einzig als Web-Server dient und sonst keinerlei andere Netzdienste wie z.B. E-Mail bereitstellt, der physikalisch abgekoppelt vom lokalem Netz agiert, von ausserhalb nur einen Zugriff auf den Port des Web-Servers erlaubt und auf dem nur der Administrator einen *Account* hat. In diesem Fall stellt sich aber die Frage, wie man die Web-Seiten auf den Rechner bekommt. In den meisten Fällen wird daher ein Zugriff für mehrere Personen von verschiedenen Rechnern aus auf den Web-Server erlaubt werden müssen, so dass ein Kompromiss zwischen Sicherheit und Nutzbarkeit gefunden werden muss.

Ein Betreiber eines Web-Servers muss sich auch Gedanken über *Betriebssystem-Updates* machen, wie man *Einbrüche* erkennnen kann und die Art und Weise wie er seine *Firewall* aufsetzt. Ein Augenmerk muss man auch den *CGI-Scripten* (mehr dazu später) schenken, die ein gewisses Sicherheitsrisiko darstellen [LAR98].

4 HTML-Formular mit JavaScript (Clientseitige Datenhaltung mit Cookies)

Nach einer kurzen Einführung in Javascript wird in diesem Kapitel der erste Prototyp implementiert. Bis jetzt waren die HTML-Seiten immer statisch, d.h. sie sahen bei jedem Request identisch aus. Nun will man aber die Seite anhand von persönlichen Adressdaten (->SMS-Telfonbuch) dynamisch aufbauen. Dazu wird HTML mit Javascript kombiniert/erweitert. Die Daten werden auf der Clientseite in Cookies gespeichert. Sessiontracking ist nicht notwendig, da die Adressdaten auf der Clientseite gehalten werden und mit Javascript clientseitig manipuliert werden. Der Webserver liefert also nur statische Seiten, die erst auf der Clientseite durch Javascript *dynamisch* werden.

4.1 Übersicht

JavaScript ist kein direkter Bestandteil von HTML, sondern eine eigene Programmiersprache. Diese Sprache wurde jedoch eigens zu dem Zweck geschaffen, HTML-Autoren ein Werkzeug in die Hand zu geben, mit dessen Hilfe sich WWW-Seiten optimieren lassen.

JavaScript-Programme werden wahlweise direkt in der HTML-Datei oder in separaten Dateien notiert. Sie werden nicht - wie etwa Java-Programme - compiliert (abgesehen von Server-side JavaScript), sondern als Quelltext zur Laufzeit interpretiert, also ähnlich wie Batchdateien bzw. Shellscripts. Dazu besitzen moderne WWW-Browser wie Netscape oder Microsoft Internet Explorer entsprechende Interpreter-Software. JavaScript kann auf der Client (CSJS) [GOO-CS][CSJS-REF][CSJS-GUI]- und auf der Servserseite (SSJS) [SSJS-REF][SSJS-GUI][GOO-SS] eingesetzt werden (siehe [HUS98]).

4.1.1 Client-side JavaScript

Wird von JavaScript gesprochen, ist im allgemeinen client-side JavaScript gemeint, dh. ein Web-Browser interpretiert den JavaScriptcode innerhalb einer HTML-Seite. Fordert ein Browser (Client) ein HTML-Dokument an, so schickt der Web-Server das Dokument - mit allen HTML und JavaScript Anweisungen - über das Netz zum Client. Damit der Browser die HTML-Seite anzeigen kann, muss er die JavaScript Anweisung ausführen (siehe Abb. 15).

4.1.2 Server-side JavaScript

Mit server-side JavaScript kann man JavaScript auf einem Internet-Server ähnlich wie CGI-Skripte verwenden. Dies erfordert aber einen passenden Web-Server (z.B. Netscape Web-Server) und wird hier nicht untersucht. Mit Server-side JavaScript kann man zB. auf Datenbanken oder auf andere Ressourcen zugreifen. Server-side JavaScript-Programme werden direkt auf den Server ausgeführt. Diese Programme generieren meist HTML-Code, der an den Browser des Anwenders geschickt und dort angezeigt wird.

Dies geschieht in zwei Schritten:

1. Der HTML-Code (mit seinen client- und server-side JavaScript Anweisungen) wird compiliert (siehe Abb. 16). Als Resultat erhält man ein ausführbares Programm.

2. Bei einem Request wird nun dieses Programm ausgeführt, d.h. HTML-Code und event. client-side JavaScript-Code wird generiert und an den Client geschickt (siehe Abb. 17).

4.2 Eine kleine Einführung in JavaScript

4.2.1 Allgemein

Für den allgemeinen Syntax von JavaScript (Variablen und Werte, Funktionen, Steuerzeichen und besondere Notationen, Operatoren, Bedingte Anweisungen (if-else/switch), Schleifen (while/for/do-while), Reservierte Wörter) wird auf die online-Referenzen verwiesen: [Mün98],[CSJS-REF] und [CSJS-GUI]. In den folgenden Abschnitten wird nur auf die für das Adressbuch relevanten JavaScript-Notationen eingegangen.

Für JavaScript Programmabschnitte kann man in HTML Bereiche definieren, zB.

```
<HTML>
<HEAD><TITLE>Glue SMS Service</TITLE></HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
function validateAbs(element) {
  if (element.abs.value.length == 0) {
    alert("Please fill in your phone number!");
    element.abs.focus();
    return false;
  }
  return true;
}
//-->
</SCRIPT>
<BODY>
<CENTER>
<H1>GLUE SMS Service</H1>
<FORM Name="SMS" ACTION="cgi-bin/sendSMS.pl"  METHOD="POST">
<TABLE CELLPADDING=2 NOSAVE NOBORDER>
.........
<TD NOSAVE><B>Your phonenumber:</B></TD>
<TD VALIGN=TOP NOSAVE ><INPUT TYPE="text" NAME="abs" SIZE="17"
    onblur="validateAbs(document.SMS)"> e.g. 0313853011</TD>
.........
</TABLE>
<INPUT  TYPE=reset  VALUE="Clear form">
<INPUT  TYPE=submit VALUE="Send message">
</CENTER>
</FORM></BODY></HTML>
```

sms.html

Internet

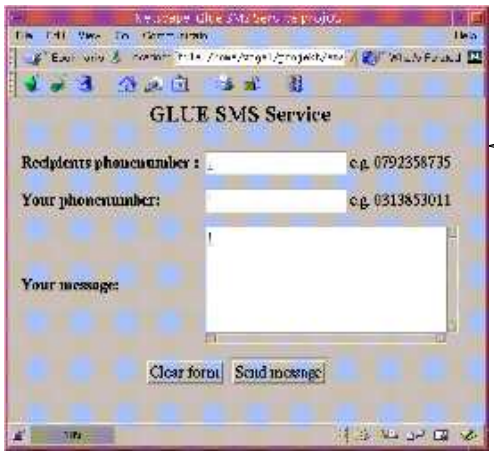

Abbildung 15: Client-side JavaScript

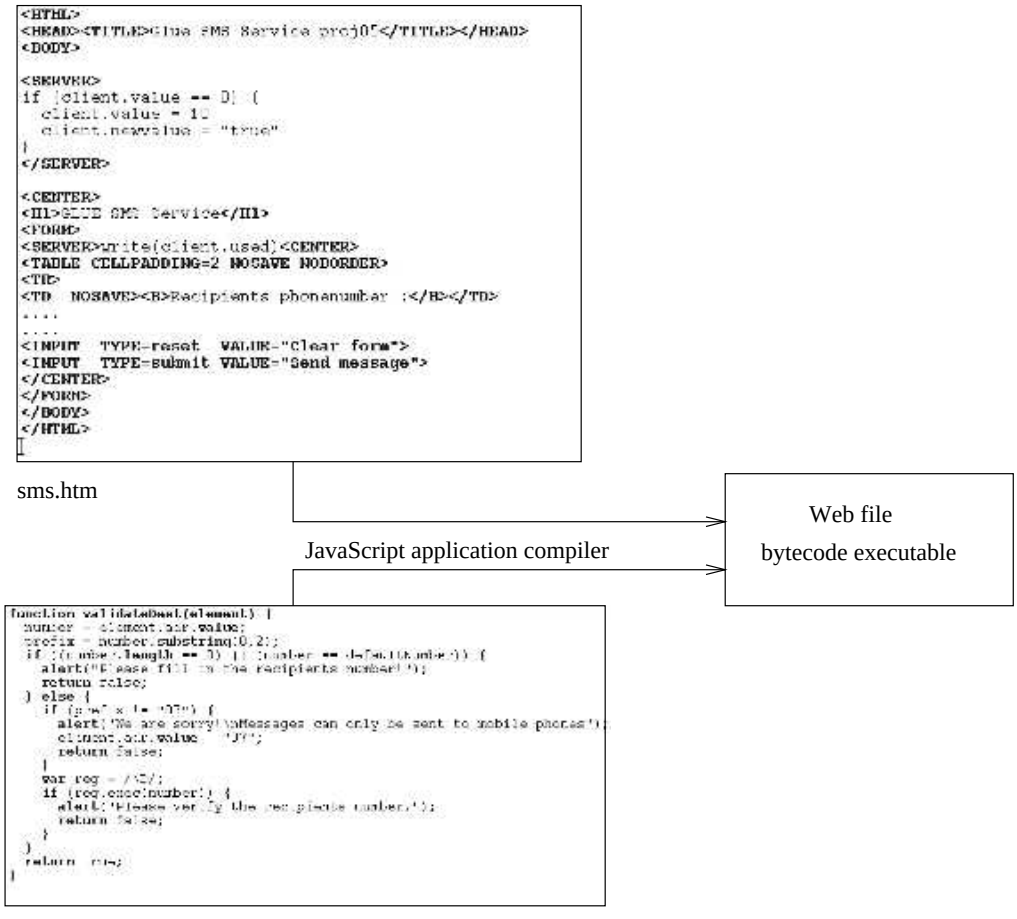

Abbildung 16: Server-side JavaScript während der Entwicklung

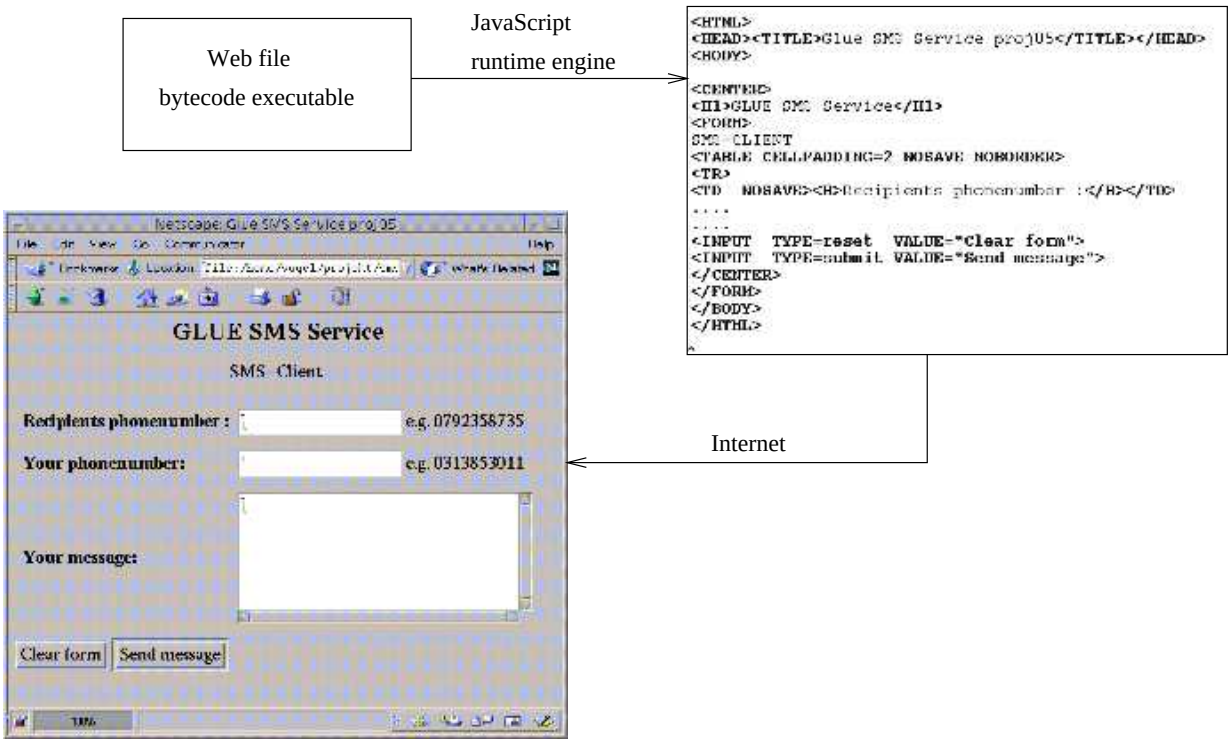

Abbildung 17: Server-side JavaScript während der Ausführung

```
<html><head><title>Test</title>
<script language="JavaScript">
<!--                // Code vor älteren Browsern verstecken
  alert("Hello world!");
//-->             // Code vor älteren Browsern verstecken
</script>
</head><body>
</body></html>
```

Beim Laden dieser HTML-Seite wird eine leere Seite angezeigt, aber zusätzlich wird eine alert-Box (ein kleines Fenster) mit dem Inhalt *"Hello world"* geöffnet.

4.2.2 JavaScript-Versionen

JavaScript hat sich in der letzten Zeit sehr schnell entwickelt. Dies hat zur Folge, dass ältere Browser nicht alle Fähigkeiten der neuen JavaScript-Browser haben. Aus diesem Grund kann es passieren, dass Programme nicht auf allen Browsern lauffähig sind. Beispielsweise wurde JavaScript mit dem Netscape Navigator 3.0 um wichtige Eigenschaften erweitert. Diese JavaScript-Version wird mit JavaScript 1.1 bezeichnet. Das Problem ist jedoch, dass JavaScript-Programme, die diese Fähigkeiten des Netscape Navigator 3.0 ausnutzen, nicht unter dem Netscape Navigator 2.0 oder dem Microsoft Internet Explorer 3.0 lauffähig sind. Diese beiden Browser unterstützen lediglich JavaScript 1.0. Um ein Script auf JavaScript 1.1 zu beschränken, verwendet man die Eigenschaft

```
language="JavaScript1.1"
```

im <*script*>-Tag. Diese Methode funktioniert leider nicht einwandfrei, wenn man die JavaScript-Funktionen vom einem File (sogenannte js-Bibliotheksdatei) *importieren* will.

```
<SCRIPT LANGUAGE="JavaScript1.1" src="scripts/adressbook1_1.js"
                                 type="text/javascript">
</SCRIPT>
```

Manche Browser importieren diese JavaScript-Bibliothek, obwohl sie nur JavaScript 1.0 interpretieren können. Das hat zur Folge, dass beim Ausführen dieser Funktionen Fehler auftreten können, da sie die importierten Funktionselemente nicht ausführen können. Will man auf Nummer sicher gehen, so hat man zwei Möglichkeiten:

- Man definiert in der eigentlichen HTML-Seite für jede JavaScript-Version einen eigenen Block: z.B.

```
<SCRIPT LANGUAGE="JavaScript1.0">
<!--
// regexp kann nicht verwendet werden, also return immer true
function dateIsValid(field) {
  return true;
}
//-->
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
<!--
// JavaScript 1.2 spezifische Funktionen, die Regexp verwenden duerfen
// Liefert true zurueck, falls das Versanddatum korrekt ist, false sonst
function dateIsValid(field) {
  var regexp = /^\s*(\d{1,2})(\W{1})(\d{1,2})(\W{1})(\d{4})\s*$/;
  if (regexp.exec(field.value)) {
    return true;
  } else {
      return false;
  }
}
//-->
</SCRIPT>
```

- oder man checkt per JavaScript den Browsertyp, -plattform und -version ab, und verwendet dann nur die erlaubten JavaScript Elemente für den jeweiligen Browser: z.B.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
function dateIsValid(field) {
  if (BrowserIs1.2_compatible) {
    var regexp = /^\s*(\d{1,2})(\W{1})(\d{1,2})(\W{1})(\d{4})\s*$/;
    if (regexp.exec(field.value)) {
      return true;
    } else {
        return false;
  }
  // for all other browsers
  else {
    return true;
}
//-->
</SCRIPT>
```

4.2.3 Mit JavaScript auf Formularobjekte zugreifen

Objekte sind fest umgrenzte Datenelemente mit Eigenschaften und oft auch mit objektgebundenen Funktionen (Methoden).

Dass JavaScript eine Erweiterung von HTML darstellt, liegt vor allem an den vordefinierten Objekten, die in JavaScript zur Verfügung stehen (siehe Abb. 18). Über diese vordefinierten Objekte kann man beispielsweise einzelne Elemente eines HTML-Formulars abfragen oder ändern .

Ein Objekt kann eine Teilmenge eines übergeordneten Objekts sein. Die JavaScript-Objekte sind deshalb in einer *Hierarchie* geordnet. Das hierarchiehöchste Objekt ist in JavaScript das *Fenster-Objekt* (window). Fenster haben Eigenschaften wie einen Titel, eine Grösse usw. Der Inhalt eines Fensters ist das nächstniedrigere Objekt, nämlich das im Fenster angezeigte *Dokument* (in JavaScript das Objekt *document*). In der Regel ist der Fensterinhalt eine HTML-Datei. Eine solche Datei kann bestimmte, durch HTML-Tags definierte Elemente enthalten, wie zum Beispiel *Formulare, Verweise, Grafikreferenzen* usw. Für solche untergeordneten Elemente gibt es wieder eigene JavaScript-Objekte, zum Beispiel das Objekt *forms* für Formulare. Ein Formular besteht seinerseits aus verschiedenen Elementen, zum Beispiel aus *Eingabefeldern, Auswahllisten* oder *Buttons* zum Absenden bzw. Abbrechen. In JavaScript gibt es dafür ein Objekt *elements*, mit dem man einzelne Felder und andere Elemente innerhalb eines Formulars ansprechen kann.

Beispiel: ein SMS-Versand Formular

```
<HTML><HEAD><TITLE>Message the world!</TITLE></HEAD>
<BODY>
<FORM NAME="SMS" action="mailto:vogel@glue.ch"
                 method=post enctype="text/plain">
Recipients Phonenumber:
<INPUT TYPE="text" NAME="adr"> e.g. 0792358735 <BR>
Your Phonenumber:
<INPUT TYPE="text" NAME="abs"> e.g. 0313853011 <BR>
Your message:
<TEXTAREA NAME="Message" ROWS="7" COLS="30" ></TEXTAREA>
<INPUT TYPE=reset VALUE="Clear form">
<INPUT TYPE=submit VALUE="Send message">
</FORM>
</BODY>
</HTML>
```

Formularobjekte haben Eigenschaften (z.B. dafaultValue, form, name, type, value ...) und Methoden (z.B. handleEvent()), mit denen man Zugriff (per JavaScript) auf die Objekte hat. Möchte man nun beispielsweise auf das abs-Textarea-Objekt zugreifen, schreibt man

```
document.forms[0].elements[1]
```

oder man spricht das Objekt direkt mit dem Namen an:

```
document.SMS.abs
```

Auf diese Weise kann man den aktuellen Wert des Formularfeldes abfragen:

```
var absender = document.SMS.abs.value;
```

oder ihm direkt einen neuen Wert zuweisen:

```
document.SMS.abs.value = 0794443322;
```

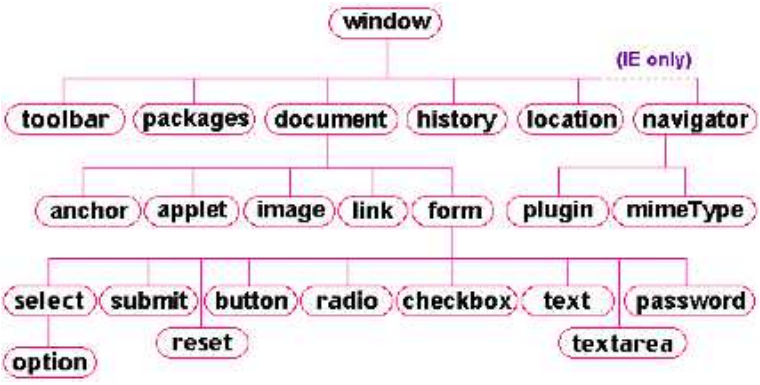

Abbildung 18: Netscape Navigator 4 Document Object Model Containment Hierarchy [GOO-CS]

4.2.4 Objekte selbst definieren

Mit JavaScript ist man nicht nur auf die vordefinierten Objekte des Browsers beschränkt. Man kann auch eigene Objekte definieren. Für das Adessbuch generiert man nun ein Objekt. Dieses Objekt hat keine Methoden, sondern nur Eigenschaften. Das Objekt *adresse* soll nur den Namen, den Spitznamen und die Mobilnummer einer Person speichern. Möchte man mehrere *adresse* speichern, so muss man entsprechend viele Insatanzen des Objektes *adresse* erzeugen.

Um ein Objekt zu erzeugen, definiert man zunächst einen **Konstruktor**:

```
function adresse(name, nickname, mobileNr) {
  //Elemente
  this.name = name;
  this.nickname = nickname;
  this.mobileNr = mobileNr;
}
```

Um eine Instanz eines Objektes zu erzeugen, verwendet man den **new-Operator**:

```
person1 = new adresse("hugentobler", "tobi", "0791237634");
person2 = new adresse("calogero", "gero", "0766489843");
```

4.2.5 Event-Handler

JavaScript kann auch innerhalb herkömmlicher HTML-Tags vorkommen. Das ist dann kein komplexer Programmcode, sondern in der Regel nur der Aufruf bestimmter Methoden, Funktionen, Objekte, Eigenschaften. Für den Aufruf gibt es sogenannte *Event-Handler*. Bei jedem eingetretenem Event wird automatisch ein event-Objekt erzeugt und an den Event-Handler übergeben. Mit den Informationen, die im Objekt gespeichert werden, kann man einiges über den Event erfahren.

4.3 Implementierung eines einfachen Adressbuches

Hier werden schrittweise die einzelnen Bestandteile des ersten Prototypes erklärt. Die Summe dieser Teile ergibt ein funktiontüchtiges Adressbuch.

4.3.1 Ausgangslage: reine HTML-Anweisungen

Mittels HTML wird ein Formular definiert, das der Benutzer ausfüllen und abschicken kann.

```
<HTML><HEAD><TITLE>Message the world!</TITLE></HEAD>
<BODY>
<FORM NAME="SMS" action="/cgi-bin/send-sms.pl"
```

```
                method=post enctype="text/plain">
Recipients Phonenumber:
<INPUT TYPE="text" NAME="adr"> e.g. 0792358735 <BR>
Your Phonenumber:
<INPUT TYPE="text" NAME="abs"> e.g. 0313853011 <BR>
Your message:
<TEXTAREA NAME="Message" ROWS="7" COLS="30" ></TEXTAREA>
<INPUT TYPE=reset VALUE="Clear form">
<INPUT TYPE=submit VALUE="Send message">
</FORM>
</BODY></HTML>
```

4.3.2 Setzen und überprüfen von Eingabefelder

In Kombination mit JavaScript kann man mittels den Namen *adr*, *abs* und *Message* auf die entsprechenden Objekten (und ihren Attributen und Methoden) zugreifen, zB.

```
// weist dem Objekt einen Wert zu
document.SMS.adr.value = defaultNumber
// selektiert/markiert das Objekt
document.SMS.adr.select()
```

Bevor der Benutzer die SMS-Nachricht sendet, werden seine Eingaben kontrolliert. Dazu benützt man den Event-Handler `onSubmit` (siehe Abb. **??**) .

```
<HTML><HEAD><TITLE>Message the world!</TITLE>
<script language="JavaScript">
<!--
function validateDest(element) {
  number = element.adr.value;
  prefix = number.substring(0,2);
  if ((number.length == 0) {
    alert("Please fill in the recipients number!");
    return false;
  } else {
      if (prefix != "07") {
        alert("We are sorry!\nMessages can only be sent to mobile phones");
        element.adr.value = "07";
        return false;
      }
   }
}
// -->
</script>
</HEAD>
<BODY>
<FORM onSubmit = "return validateDest(this.form);" name="SMS"
      action="/cgi-bin/send-sms.pl"
      method=post enctype="text/plain">
Recipients Phonenumber:
<INPUT TYPE="text" NAME="adr"> e.g. 0792358735
Your Phonenumber:
<INPUT TYPE="text" NAME="abs" > e.g. 0313853011
Your message:
<TEXTAREA NAME="Message" ROWS="7" COLS="30" ></TEXTAREA>
<INPUT TYPE=reset VALUE="Clear form">
<INPUT TYPE=submit VALUE="Send message">
</FORM> </BODY> </HTML>
```

Hier wird nur geprüft, ob das adr-Feld nicht oder mit einem falschen Präfix ausgefüllt wurde. JavaScript stellt aber auch *reguläre Ausdrücke* zur Verfügung, mit denen man flexiblere Tests durchführen kann [Mün98]:

```
function validateDest(element) {
  number = element.adr.value;
  prefix = number.substring(0,2);
  if ((number.length == 0) || (number == defaultNumber)) {
    alert("Please fill in the recipients number!");
    return false;
  } else {
      if (prefix != "07") {
        alert("We are sorry!\nMessages can only be sent to mobile phones");
        element.adr.value = "07";
        return false;
      }
   var reg = /\D/;
   if (reg.exec(number)) {
      alert("Please verify the recipients number.");
      return false;
   }
  }
return true;
}
```

Das Beispiel demonstriert den Zusammenhang: ein regulärer Ausdruck wird definiert (*var reg = /\D/;*), in diesem Fall ein regulärer Ausdruck mit dem Namen *reg*, welcher alle Zeichen ausser den Zahlen representiert. Mit einer Anweisung wie *reg.exec()* können Sie dann die Suche starten. Als Parameter wird der Methode in der Regel der zu durchsuchende Ausdruck übergeben. Die Methode gibt die gefundenen Suchtreffer zurück.

4.3.3 Speichern der Daten als Cookies

Die Adressdaten müssen nun in Form von Cookies auf der Clienseite abgelegt werden können. Man definiert hierfür Funktionen zum Setzen, Löschen und Lesen von Cookies:

```
function getCookie(name) {
  var cname = name + "=";
  var dc = document.cookie;
  if (dc.length > 0) {
    begin = dc.indexOf(cname);
    if (begin != -1) {
      begin += cname.length;
      end = dc.indexOf(";", begin);
      if (end == -1)
        end = dc.length;
      return unescape(dc.substring(begin, end));
    }
  }
  return null;
}
------------------------------------------------------------
function setCookie(name, value) {
  var now = new Date();
  var then = new Date(now.getTime() + 31536000000);
```

```
  document.cookie = name + "=" + escape(value) +
                    "; expires=" + then.toGMTString() + "; path=/";
}
----------------------------------------------------------
function deleteCookie (name) {
  var exp = new Date();
  exp.setTime (exp.getTime() - 1); // This cookie is history
  var cval = getCookie (name);
  document.cookie = name + "=" + cval +
                    "; expires=" + exp.toGMTString(); }
```

Nun ist es einfach ein Cookie zu speichern und wieder auszulesen, zB. mit

```
setCookie("Cookie-name",element.abs.value)
```

wird der Wert der Variable *abs* als Cookie mit Namen *Cookie-name* gespeichert. Mit

```
absender = getCookie("Cookie-name")
```

wird der Wert von *Cookie-name* gelesen und der Variable *absender* zugewiesen.

In dieser Implementation des Adressbuches wird beim Absenden des Formulares die Funktion *update_adressbook(element)* aufgerufen. Hier erfolgt - falls einige Voraussetzungen erfüllt sind - das Ablegen der neuen Daten als Cookie:

```
// checks if the number is already in the cookie's adress-array
// if not: store it at the end of the array
// sort the array and save the array as a cookie
// finally:send the sms message
function update_adressbook(element) {
  // check if book is activated
  adressbook_activated = getCookie("adressbook_activated");
  if ((adressbook_activated=="true") && (check_input(element.AdC.value))) {
    var newAdress = true;
    // get adressList from Cookie
    adress =   getCookie("adress");
    // check for an adress-cookie
    if (adress) {
      // create array
      adressArray = adress.split("!");
      // check if the number(!) is already stored in the cookie
      for (i=1; i < adressArray.length; i=i+1) {
        var tmp_array = new Array;
        tmp_array = adressArray[i].split("_TOKEN_");
          if (tmp_array[1] == element.AdC.value)
            newAdress = false;
      }
    }
    // add new number (if new, valid and wanted by user) into array
    // and also append it to the cookie
    if ( (newAdress) && (add_newNumber(element.AdC.value)) ) {
      adressArray[adressArray.length] = "noName_TOKEN_" + element.AdC.value;
     // first sort the new Array
      adressArray.sort();
      setCookie("adress", adressArray.join('!'));
```

```
      if (adressbook_activated == "true")
        location.reload();
    }
  }
}
```

4.3.4 Dynamisches Generieren von HTML-Code

Beim Aufbau der HTML-Seite wird dynamisch der Code für das Adressbuch eingefügt. Dies geschieht mittels *JavaScript-Code*, der an beliebiger Stelle einer HTML-Seite ausgeführt werden kann.In diesem Fall mitten in einer Tabelle:

```
</tr>
  <SCRIPT LANGUAGE="JavaScript1.1">;
  <!--
  // get adressList from Cookie
  var adress = getCookie("adress");
  // check if book is activated
  var adressbook_activated = getCookie("adressbook_activated");
  if (adress) {
    adressArray = adress.split("!");
    if ((adressArray.length > 1) && (adressbook_activated == "true")) {
      document.write("<TR>");  document.write("<TD VALIGN=TOP NOSAVE>");
      document.write("<SELECT NAME=\"AdressList\" onChange=\"update(this)\" >");
      document.write(" <option > ----- Adressbuch -----");
        for (i = 1; i < adressArray.length; i=i+1) {
          // split name and number var tmp_array = new Array;
          tmp_array[0] = "name";
          tmp_array[1] = "<number>";
          // only set tmp_array, if we have a real adresscookie
          if (adressArray[i].indexOf("_TOKEN_") != -1)
            tmp_array = adressArray[i].split("_TOKEN_");
          // shortens the values if necessary
          var _name = tmp_array[0];
          var _number = tmp_array[1];
          if (_name.length > 15)
            _name = _name.substring(0,15) + "...";
          if (_number.length > 20)
            _number = _number.substring(0,20) + "...";
          document.write(" <option >" + _name + " = " + _number);
        } // for
        document.write(" </select>");
        document.write("</TD>"); document.write("</TR>");
     } // if
  } // if
//-->
</SCRIPT>
<tr>
```

4.3.5 Resultat

Mit Hilfe dieser JavaScript-Funktionen wird nun für den Benutzer auf der Clientseite ein Adressbuch geführt. Aus einem Scrolldown-Menu kann er nun eine Nummer wählen, die dann automatisch in das Adressfeld eingetragen wird (siehe Abb. 19).

Abbildung 19: Eingabemaske mit Adressmenu

Man kann aber auch die Cookie-Datenbank direkt bearbeiten. Über eine andere HTML-Seite lassen sich Nummern und Namen eintragen und verändern, und man kann den gesammten Datenbestand auch löschen (siehe Abb. 20).

4.4 Zusatzprogramme zum Exportieren/Importieren des Adressbuches

Für den Datenschutz ist der Benützer selber verantwortlich. Die Daten werden in Form eines Cookies auf die Festplatte abgelegt. Je nach Betriebssystem können diese Daten von anderen Usern gelesen, manipuliert oder sogar gelöscht werden. Daher ist es sinnvoll, dass der Benutzer seine Daten selbst verwaltet: ideal wäre eine Export/Import-Funktion (diese Funktion ist an dieser Stelle nocht nicht implementiert). Es gibt bereits diverse Adressbücher, mit denen man seine Kontaktdaten wie Namen und Mobilnummern bequem verwalten kann (z.B. Netscape's Adressbuch, Outlook etc.). Nun geht es darum, die Daten dieser Adressbücher mit *diesem Adresstool* zu synchronisieren. Nach dem Exportieren der Adressdaten kann der User sein Adress-Cookie löschen, damit niemand diese Daten lesen kann (->Internetcafe).

4.4.1 Import:

Zuerst muss man die Daten aus dem Netscape- bzw. Outlook Applikation exportieren (mittels eigener Exportfunktion der Anwendung). Um diese Daten ins Web-Adressbuch zu übernehmen, muss man die exportierte Datei mit einem Editor öffnen, den Inhalt selektieren und in ein Textfield eines entsprechenden HTML-Formulars kopieren. Nun kann mittels Javascript diese Daten in das Sms-Adressbuch abgelegt werden.

4.4.2 Export:

Die Exportfunktion funktioniert analog: Mit Hilfe von JavaScript werden die SMS-Adressdaten im gewünschten Format (z.B. für Netscape) in einem Textfeld eines Html-Formulares angezeigt. Mittels Copy/Paste kann man diese Adressdaten in einem File abspeichern uns ins eigene (z.B.Netscape) Adressbuch importieren.

Abbildung 20: Maske zur Bearbeitung des Adressbuches

5 HTML und CGI-Scripts[Mün98]

Bis jetzt wurden die Adressdaten auf der Clientseite gehalten. Serverseitig hatte man noch keinen grossen Aufwand: nur HTML-Seiten mit Javascript, die für jeden Client identisch waren. Nun will man aber die Daten auf der Serverseite ablegen. Dazu muss man den Apache Webserver konfigurieren (CGI-Schnittstelle vorbereiten), Perl-module installieren, sowie eine MySql-Datenbank installieren und konfigurieren.

In diesem Kapitel wird die Technologie der CGI-Schnittstelle erläutert. Im ersten Prototyp wird das Cookie vom CGI-Script gelesen, verarbeitet, manipuliert und wieder gesetzt. Die Daten werden also wieder clientseitig gehalten. Danach folgen drei weitere Protoypen, welche die Daten serverseitig speichern: Zuerst werden die Daten in einem einfachen Textfile gehalten. Die anderen zwei speichern die Daten mittels *sql-Befehlen* in einem Flatfile bzw. in einer *MySql-Datenbank* ab. Dazu wird zuerst eine Einführung in Perl, Datenbanken und MySql gegeben.

5.1 Grundlagen zur CGI-Schnittstelle (*Common Gateway Interface*)

5.1.1 Die CGI-Schnittstelle

Anfangs implementierten die Server-Entwickler die zum Aufruf externer Programme und zur Parameterübergabe nötigen Mechanismen nach eigenem Gutdünken. Das führte aber schon bald zu Problemen, da ein Skript, das zum Beispiel für den NCSA-httpd entwickelt wurde, mit dem CERN-httpd nicht lief, und vice versa.

Eine weitere Schwierigkeit, auf die Entwickler von Server-Skripts schnell stiessen: Es war nicht möglich, innerhalb eines Skripts festzustellen, von welchem Client-Host aus der Zugriff auf das dynamische Dokument erfolgte, welche Datentypen der Client akzeptierte und vieles andere mehr.

Daher einigten sich die Server-Entwickler auf ein Standardverfahren zum Aufruf und zur Parameterversorgung solcher Skripts oder Programme - das *Common Gateway Interface* (CGI) war geboren.

CGI erlaubt es einem WWW-Browser, über einen WWW-Server Programme auszuführen. Solche Programme (oder Scripts) können beispielsweise Formulareingaben aus HTML-Dateien verarbeiten, auf dem Server-Rechner Daten speichern und dort gespeicherte Daten auslesen. Auf diese Weise werden WWW-Seiten zu Oberflächen für Anwendungen, beispielsweise für elektronische Warenbestellung oder zum Abfragen von Datenbanken.

Die sogenannte CGI-Schnittstelle steht zur Verfügung, wenn ein WWW-Server installiert ist, der CGI unterstützt. Man kann sich selbst lokal auf dem PC einen WWW-Server einrichten, um dort eine CGI-Schnittstelle zur Verfügung zu haben. Wenn man bei einem Provider auf einem öffentlichen WWW-Server im Internet Speicherplatz für eigene WWW-Seiten erhält, kann man die CGI-Schnittstelle dieses WWW-Servers benutzen, um eigene CGI-Scripts im WWW einzusetzen. Der Provider muss allerdings den Zugriff auf die CGI-Schnittstelle ermöglichen.

Die CGI-Schnittstelle besteht aus:

- einem bestimmten Verzeichnis auf dem Server-Rechner, das CGI-Programme enthalten darf. Meist erhält dieses Verzeichnis den Namen *cgi-bin* oder *cgi-local*. CGI-Programme oder CGI-Scripts werden dann nur ausgefüht, wenn sie in diesem Verzeichnis liegen. Welches das CGI-Verzeichnis ist und wie es heisst, muss man beim Einrichten des WWW-Servers festlegen.
- einer Reihe von Daten, die der WWW-Server speichert, und die ein CGI-Script auslesen kann (und zum Teil auslesen muss), um Daten verarbeiten zu können. Diese Daten speichert der WWW-Server in sogenannten *CGI-Umgebungsvariablen*.

Perl (siehe:[WCS-PP, REI-PERL, PE-TU]) ist noch immer die meistbenutzte Programmiersprache für CGI-Scripts. Der Grund dafür ist, dass Perl sehr mächtige Funktionen besitzt, z.B. für Zeichenkettenoperationen oder für das Lesen und Schreiben von Daten. Der Perl-Interpreter, der zum Ausführen eines Perl-Scripts erforderlich ist, ist für fast alle Betriebssysteme als Freeware verfügbar und auf fast allen Server-Rechnern im WWW installiert.

Wenn man in einer anderen Programmiersprache, z.B. in C, Pascal oder Fortran, bereits viel Erfahrung hat, kann man auch diese Sprachen für CGI-Programme verwenden. Das Problem bei diesen Sprachen ist, dass der damit erstellte Code erst ausführbar ist, nachdem er für die Betriebssystemumgebung compiliert und gelinkt wurde, unter der er ausführbar sein soll. Wenn man beispielsweise ein C-Programm schreiben und dieses Programm als CGI-Programm im WWW auf einem Server-Rechner mit Unix-Betriebssystem zum Einsatz bringen will, muss das Programm unter Unix compiliert werden. Ein C-Compiler, den man vielleicht auf dem DOS/Windows-PC hat, nutzt in diesem Fall nichts.

5.1.2 CGI-Aufrufe aus HTML und HTML-Ausgabe über CGI

HTML und CGI kommunizieren in beide Richtungen: es ist einerseits möglich, aus einer HTML-Datei, die gerade am Bildschirm angezeigt wird, ein CGI-Script aufzurufen; anderseits kann ein CGI-Script HTML-Code an den WWW-Browser übertragen, den dieser dann am Bildschirm ausgibt.

Ein CGI-Script kann Daten verarbeiten, die von einer HTML-Datei aus beim Aufruf übergeben werden. Zum Beispiel kann ein CGI-Script eine Datenbank durchsuchen, wobei der Anwender den Begriff, nach dem gesucht werden soll, in einem Formular angegeben hat. Die Ergebnisse einer Datenverarbeitung kann ein CGI-Script an den WWW-Browser in Form von HTML-Code zurücksenden. So kann ein Script, das eine Datenbank nach Begriffen durchsucht, zum Beispiel die Suchtreffer eines Suchvorgangs in Form einer dynamisch generierten HTML-Datei an den WWW-Browser zurücksenden.

CGI-Scripts können auch Daten auf dem Server speichern und zu einem späteren Zeitpunkt auslesen. Auf diesem Prinzip basieren beispielsweise Gästebücher oder Nachrichtenforen. Ein Anwender kann in einer HTML-Datei in einem Formular einen Beitrag eingeben. Beim Absenden des Formulars wird ein CGI-Script aufgerufen, das den Beitrag in einer Datei speichert. Ein zweites CGI-Script oder ein anderer Aufruf des CGI-Scripts kann anschliessend HTML-Code mit allen gespeicherten Beiträgen an einen WWW-Browser übertragen().

Ein CGI-Script kann aus einer HTML-Datei heraus auf verschiedene Arten aufgerufen werden:

- über ein Formular. Dabei steht im einleitenden *<form>*-Tag der Aufruf des CGI-Scripts (Beispiel: *<form action="/cgi-bin/guestbook.pl" method="get">*). Der Aufruf erfolgt nach dem Absenden des Formulars. Die vom Anwender eingegebenen oder ausgewählten Daten stehen dem CGI-Script zur Verarbeitung zur Verfügung. Auf diese Weise funktionieren etwa Suchdienste, Gästebücher oder elektronische Einkaufskörbe.

- über Verweise. Es genügt, als URL-Adresse des Verweisziels das ausführbare CGI-Script anzugeben (Beispiel: *Tagesstatistik aufrufen*). Dies ist sinnvoll für CGI-Scripts, die keinen Input vom Anwender benötigen, sondern lediglich feste Datenausgaben erzeugen, zum Beispiel für ein CGI-Script, das aktuelle Zugriffsstatistiken für WWW-Seiten ausgibt.

- über eine Grafikreferenz. Auch dabei genügt es, als URL-Adresse in der src-Angabe des *<img>*-Tags das ausführbare CGI-Script anzugeben (Beispiel: *<img src="/cgi-bin/counter.pl">*). Dabei muss das CGI-Script allerdings eine Grafikdatei im GIF- oder JPEG-Format an den WWW-Browser zurücksenden. Die meisten grafischen Zugriffszähler basieren auf diesem Prinzip.

- über eine *Server Side Include* Anweisung in einer HTML-Datei, z.B. mit der Anweisung *<!– #exec cgi="/cgi-bin/counter.pl" –>*. Das ist sehr praktisch, um mit Hilfe eines CGI-Scripts dynamische Information in Textform in eine HTML-Datei einzubinden. Diese Form ist zum Beispiel interessant für textbasierte Zugriffszähler.

- über automatisches Laden des ausführbaren CGI-Scripts/CGI-Programms. Dazu gibt man in dem *<meta>*-Befehl einfach anstelle einer anderen HTML-Datei die URL-Adresse des ausführbaren CGI-Scripts an (Beispiel: *<meta http-equiv="refresh" content="0; URL=/cgi-bin/welcome.pl">*).

(siehe Abb. 21).

Übertragung an den Web-Server Beim Betätigen des Submit-Knopfes wird das Formular abgeschickt. Im *<form>*-Tag wird mit "action=*http://www.glue.ch/cgi-bin/send-sms.pl*" festgelegt, mit welchem Protokoll an welchen Server ("www.glue.ch") die Daten gesendet werden sollen und welches Programm ("*cgi-bin/send-sms.pl*") diese Daten verarbeiten soll.

Bevor die Daten an den Server gesendet werden, werden sie zunächst vom Browser zu einer einzigen Zeichenkette verpackt. Für das Adressbuch könnte die Zeichenkette z.B. so aussehen : *adr=0793857735&abs=0763428735&M*

Die Art und Weise der Datenübertragung wird durch den Parameter "method=" gesteuert. Dabei wird grundsätzlich zwischen den beiden Methoden *GET* und *POST* unterschieden:

In unserem Beispiel wurde GET angegeben : In diesem Fall wird aus der im *<form>*-Tag angegebenen URL des CGI-Programms und der aus den Eingabedaten erzeugten Zeichenkette, getrennt durch ein *?*, eine Pseudo-URL erzeugt. Für unser Beispiel könnte die also so aussehen: *http://www.glue.ch/cgi-bin/send-sms.pl?adr=0793857735&abs=0763428735&Message=hello+mark.%0D%0Aplease+call+me+tonight+at+8+pm.%0D%0Ac* Falls man das oben dargestellte Formular einmal abgeschickt hat, dann wird der Browser im Anzeigefeld *Adresse* eine entsprechende Pseudo-Adresse angeben.

Die Methode *GET* eignet sich nicht zur Übertragung grösserer Datenmengen !

Alternativ kann als Methode *POST* verwendet werden. Dazu muss der HTML-Code wie folgt geändert werden: *<FORM NAME="SMS" action="http://www.glue.ch/cgi-bin/send-sms.pl" method=post enctype="text/plain">*

Was passiert diesmal, wenn man den OK-Knopf betätigt ? Bei Verwendung von "method=POST" bleibt die angegebene URL unverändert. In diesem Fall wird die verpackte Zeichenkette nach dem Verbindungsaufbau für den Anwender unsichtbar (ausser Option *Sicherheitseinstellungen / Weiterreichen eines nicht geschützten Dokuments* bei Netscape) an den Server übertragen. Hier wird der Browser im Anzeigefeld "Adresse" nur die echte Adresse des CGI-Programms, also *http://www.glue.ch/cgi-bin/send-sms.pl* anzeigen (siehe Abb. 21).

Start des CGI-Programms Der Server erkennt an dem Pfadnamen der angefragten URL, (i.A. "cgi-bin"), dass nicht ein vorhandenes Dokument zurückgesendet sondern ein CGI-Programm gestartet werden soll. Der Server muss diesem Programm die Daten aus dem Formular bereitstellen. Hierzu werden *Environment-Variablen* verwendet. Die Variable *REQUEST_METHOD* hat entweder den Wert *GET* oder *POST*. Im Fall von *GET* wird die verpackte Zeichenkette in einer weiteren Variablen *QUERY_STRING* bereitgestellt; bei *POST* wird in einer Variablen *CONTENT_LENGTH* die Länge der Zeichenkette abgelegt, die eigentliche Zeichenkette wird dem Programm auf *stdin* (Standard-Input) zur Verfügung gestellt (siehe Abb. 21).

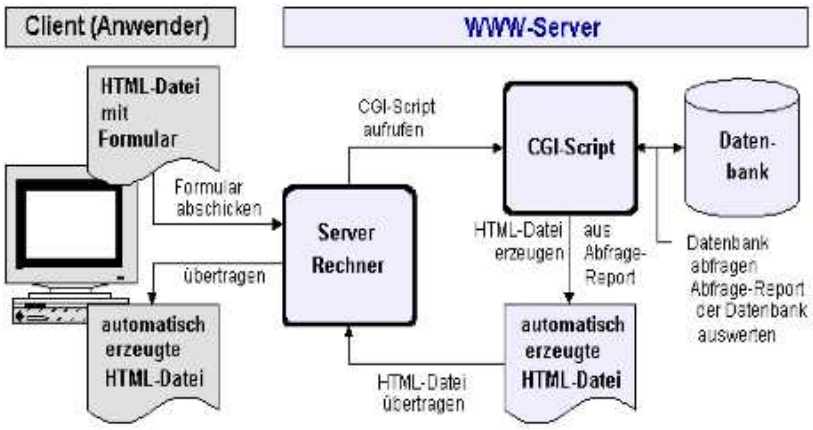

Abbildung 21: HTML-Generierung mittls CGI[Mün98]

Lauf des CGI-Programms Zur Erstellung eines CGI-Programms gibt es keine spezielle Programmiersprache. Im einfachsten Fall kann das Programm unter Unix ein *shell script* bzw. unter DOS eine *.bat-Datei* sein. Es können aber auch beliebige höhere Programmiersprachen wie *c, Pascal oder Fortran* benutzt werden. Sehr weit verbreitet ist auch die Script-Sprache Perl, die auch hier verwendet wird.

Das CGI-Programm packt zunächst die empfangene Zeichenkette wieder aus. Der weitere Programmablauf wird dann durch die so erhaltenen Parameter gesteuert.

Die Ausgabe des Programms erfolgt in einem Datenformat, dass vom Browser interpretiert werden kann; in den meisten Fällen ist das entweder *text/html* (also wieder ein HTML-Dokument), *image/gif* (also eine Grafik) oder andere (siehe Abb. 21).

5.2 Implementierung des Adressbuches: Speichern der Daten auf der Clientseite (Cookie)

Man geht nun wieder von der einfachen statischen HTML-Seite aus. Beim Abschicken des Formulares werden die Daten auf dem Server von einem Perl-Script (siehe:[REI-PERL, PE-TU, WCS-PP]) verarbeitet.

```
<HTML><HEAD><TITLE>Message the world!</TITLE></HEAD>
<BODY>
<FORM NAME="SMS" action="http://www.glue.ch/cgi-bin/send-sms.pl"
                 method=get enctype="text/plain">
Recipients Phonenumber:
<INPUT TYPE="text" NAME="adr"> e.g. 0792358735 <BR>
Your Phonenumber:
<INPUT TYPE="text" NAME="abs"> e.g. 0313853011 <BR>
Your message:
<TEXTAREA NAME="Message" ROWS="7" COLS="30" ></TEXTAREA>
<INPUT TYPE=reset VALUE="Clear form">
<INPUT TYPE=submit VALUE="Send message">
</FORM>
</BODY></HTML>
```

Auch hier werden die Daten mittels Cookies auf der Clientseite gespeichert. Der Aufwand ist hier um einiges grösser, da bei jeder Adresseingabe das CGI-Script ein neues Cookie setzen muss. JavaScript übernimmt hier nur die Aufgabe, die Adressdaten aus dem Select-Feld dynamisch (d.h. bei Auswahl durch den User) ins Adress-Feld einzutragen, sowie das Aufbereiten der neuen Daten für das Perl-Script.

Um dynamisch HTML-Code zu erzeugen benützt man am besten die Library *CGI.pm* (siehe:[REI-PERL]). CGI.pm arbeitet eigentlich mit einem CGI-Objekt:

```
use CGI;
$q = new CGI;
```

Über die Objektreferenz $q stehen anschliessend die Dienste als Methoden zur Verfügung. So gibt

```
print $q->header();
print $q->h1("Überschrift");
```

den Server-Header und eine H1-Überschrift etwa als

```
Content-type: text/html
<H1>Überschrift</H1>
```

aus. Mit einer Tag-Liste hinter dem use-Befehl werden die Methoden als Funktionen in den Namensraum des Skripts importiert:

```
use CGI qw/:standard/;
print header, h1("Überschrift");
```

Für CGI-Parameter, Formulare und Basis-HTML reicht :standard, kommen Tabellen mit ins Spiel, muss auch noch :html3 'ran. Doch noch einmal zurück zum Anfang:

```
print $query->header(-cookie=>$adressCookie);
```

senden wir den Headerteil und veranlassen den Browser ein Cookie zu setzen.

```
print $query->start_html(-title=>'SMS Versandformular',
-author=>'vogel@glue.ch',
-meta=>{'keywords'=>'sms adressbook',
        'copyright'=>'copyright 2001 vogel david'}, -script=>{-language=>'JavaScript1.1',
          -src=>'/cgi_client/javascript/adressbook.js',
          -type=>'text/javascript'},
-style=>{-code=>$newStyle,
         -type=>'text/css'},
-onLoad=>'loadHook()',
-BGCOLOR=>'white'),"\n";
```

Das setzt den Titel des zurückgelieferten HTML-Dokuments auf *SMS Versandformular* und die Hintergrundfarbe auf weiss. Zudem wird JavaScript- und StyleSheet Code included.

Auch für das Generieren von Formular-Tags stellt uns die Library Methoden zur Verfügung:

```
# Beginning of from
print $query->startform(-name=>'SMS',
-action=>'/cgi_client_cgi-bin/prototyp01.pl',
-method=>'POST'),"\n";
# Generate a textarea-field
print $query->textarea(-name=>'Message',
-rows=>5,
-onfocus=>'hitAndStart()',
-onblur=>'hitAndStop()',
-onchange=>'hitAndStop()',
```

Mit diesen Methoden wird das HTML-Dokument abgeschlossen:

```
# end of form
print $query->endform,"\n";
# end of html document
print $query->end_html();
```

Diese Version des Adressbuches ist weniger effizient, da das CGI das Cookie-Management übernimmt: Manipuliert der User seine Adressdaten, so werden jedesmal Daten übers Internet ans CGI geschickt, obwohl die Daten clientseitig abgelegt werden (siehe Abb. 22).

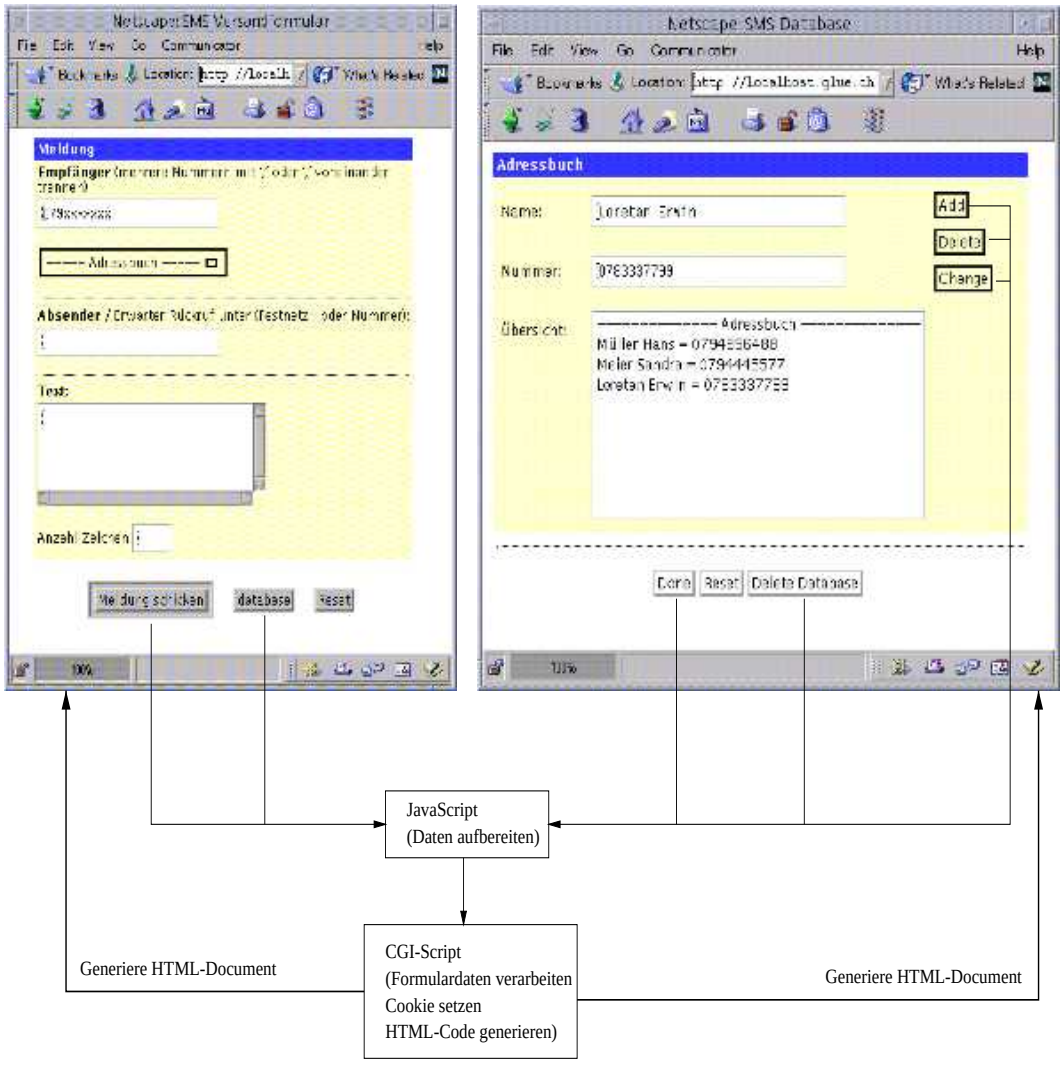

Abbildung 22: Interaktion: JavaScript - CGI

5.3 Implementierung des Adressbuches: Speichern der Daten auf der Serverseite (Flatfile, DB)

Hier werden die gesamten Daten serverseitig gespeichert. Um das ganze flexibler zu gestalten, wird zudem der ganze Adressverwaltungsteil separat (d.h. unabhängig vom sms-Modul) verwaltet. Zusätzliche Angaben zur Person wie E-Mail, Postadresse können auch angegeben werden.

Zuerst soll das ganze auf einem reinen *Textfile* als Datenbasis aufsetzen (->Adressbuch I), als nächster Schritt wird ein Austausch des Textfiles in eine *Flatfile DB* erfolgen (->Adressbuch II). Als letztes wird als Datengrundlage eine *echte Datenbank (mySQL)* verwendet (->Adressbuch III). Es wird also aufgezeigt, wie man mittels perl mit einer MySql-Datenbank kommuniziert.

5.3.1 Vorbereitung / Grundlagen / Perl

Schreiben und Lesen von Files mit Perl (siehe: [WCS-PP]) Ein File oder *Dateihandle* ist der Name für die Ein/Ausgabeverbindung des Perl Programmes mit der Aussenwelt. Standardfilehandles sind *STDIN* (normalerweise Tastatur), *STDOUT* (normalerweise Bildschirm das bedeutet im allgemeinen die aufrufende Shell) und *STDERR* (Standardfehler). Die Namen von Filehandles sind unabhängig vom Namensraum von Variablen, können also gleich sein. Üblicherweise verwendet man grosse Buchstaben für Filehandles.

Hier ein kleines Beispiel, das eine Datei öffnet, und die Zeilen zählt:

```
#!/home/vogel/usr/local/perl5.5/bin/perl -w
use strict;

# Name und Pfad des Datenfiles
my $file="/home/vogel/projekt/htdocs/cgi_server/doc/archiv/adress01.dat";
my $cnt = 0;

open(FILE, $file) || die "Kann $file nicht oeffnen: $!";
while (<FILE>) {
  $cnt++;
}

close(FILE) || die "Kann $file nicht schliesen $!";
print "$cnt Zeilen\n";
```

Das nächste Beispiel zeigt, wie man eine Datei liest und in eine andere Datei schreibt:

```
# !/home/vogel/usr/local/perl5.5/bin/perl -w
use strict;

my $infile ="/home/vogel/projekt/htdocs/cgi_server/doc/archiv/adress01.dat";
my $outfile="/home/vogel/projekt/htdocs/cgi_server/doc/archiv/adress02.dat";

open(LESEN, $infile) || die "Kann $infile nicht oeffnen: $!";
open(SCHREIBEN, ">$outfile") || die "Kann $outfile nicht oeffnen: $!";

while (<LESEN>) {
  print SCHREIBEN $_
}

close(LESEN) || die "Kann $infile nicht schliesen $!";
close(SCHREIBEN) || die "Kann $outfile nicht schliesen $!";
```

Grundsätzliches zu Modulen in Perl (siehe: [WCS-PP]) Perl bietet neben den Standard Funktionen die Möglichkeit Erweiterungen in Form von *Modulen* zu benutzen. Diese Module werden mit *use Modulname;* am Anfang des Perl Skriptes eingebunden. Im Standard Umfang von Perl sind bereits viele Module vorhanden. Hier nur zwei Beispiele: *strict* beschränkt unsichere Konstrukte, d.h. alle Variablen müssen mit *my* deklariert werden. *CGI* ermöglicht die einfache Nutzung des *Common Gateway Interface*. Es vereinfacht die HTML Ausgabe, die Übergabe von Formulardaten mit GET oder POST , das parsen der Daten (Umwandlung der mit hexadezimal Zahlen geschützten Sonderzeichen), Erstellen von HTML Tabellen, Verwendung und Erzeugung von Formularen und vieles mehr.

Pattern Matching (siehe: [WCS-PP]) Ein sehr mächtiges Werkzeug unter Perl ist das *Pattern Matching* (oder auch Reguläre Ausdrücke = Regex). Damit ist es möglich sehr komplexe *suchen* oder *suchen und ersetzten* Operationen durchzufüren. Das *Suchen* ermöglicht der *m//* Operator, das *Suchen und Ersetzten* der *s///* Operator. Die Slashes bezeichnet man hier als Quotingzeichen. Diese können auch durch andere Zeichen ersetzt werden.

Arrays (siehe: [WCS-PP]) Ein Array ist eine geordnete Liste von Skalaren, auf die über die Position des Skalars in der Liste zugegriffen wird. Diese Liste kann Zahlen, Strings oder beides enthalten (sie kann sogar Referenzen auf andere Arrays enthalten, womit man multidimensionale Arrays aufbauen kann). Um einem Array einen Listenwert zuzuweisen gruppiert man einfach die Variablen mit Klammern zusammen :

```
@heim = ( Sofa, Sessel, Tisch, Herd);
```

Andererseits, wenn man *@heim* in einem Listen-Kontext benutzt, wie z.B. auf der rechten Seite einer Listen-Zuweisung, dann erhält man dieselbe Liste, die man vorher hineingeschrieben hat. Auf diese Weise kann man z.B. vier Skalarvariablen aus einem Array mit Werten belegen :

```
($kissen, $lift, $schreib, $platte) = @home;
```

Das nennt sich Listen-Zuweisungen. Diese erfolgen natürlich parallel, so das man auch sagen kann :

```
($alpha, $beta) = ($gamma, $delta);
```

In Perl werden Arrays null-basiert indiziert. Der Zugriff auf einzelne Komponenten eines Arrays erfolgt durch in eckige Klammern eingeschlossene Subskripts. Man kann damit auch die Elemente eines Arrays separat zuweisen, zum Beispiel:

```
$heim[0] = Sofa; $heim[1] = Sessel; $heim[2] = Tisch; $heim[3] = Herd;
```

Da Arrays geordnet sind, kann man daüber auch Stackoperationen ausführen. Man kann Elemente am Ende der Liste (von rechts) hinzufügen: *push* oder entfernen: *pop*. Will man das mit dem Anfang (von links) des Arrays tun, so lauten die Befehle *unshift* und *shift*. Mit dem internen Kommando *sort* kann eine Liste sortiert werden.

Zerlegen von Datensätzen (siehe: [WCS-PP]) Wie man ein Array mit Trennzeichen versieht und in ein File schreibt zeigt das der folgende Code:

```
# !/home/vogel/usr/local/perl5.5/bin/perl -w
use strict;

# Name und Pfad des Datenfiles
my $file="/home/vogel/projekt/htdocs/cgi_server/doc/archiv/adress01.dat";
my ($eingabe, $string);
my @array =(); # leeres Array definieren

chomp ($eingabe = <STDIN>);
```

```
while ($eingabe ne "999") {
  push @array, $eingabe; #Eingabe an Array anhängen
  chomp ($eingabe = <STDIN>);
}

#String aus Liste mit | als Trennzeichen
$string = join '|', @array;

# File zum Schreiben öffnen
open (FILE, ">$file") || die "Kann File $file nicht öffnen: $!";

print FILE $string."\n";
close (FILE) || die "Kann File $file nicht schliessen: $!";
```

Aus den einzelnen Arrayelementen und einem Trennzeichen bastelt man einen String der Art: "Element1|Element2|Element3". Dies erledigt *join AUSDR, Liste* . Der *AUSDR* gibt das Trennzeichen an, die Elemente stehen in der *Liste*.

Der umgekehrte Befehl lautet *@array = split /Muster/, String* , Wobei das */Muster/* für einen regulären Ausdruck steht, der dem Trennzeichen entspricht. Wird kein String angegeben, verwendet split automatisch die Standardvariable $_ . Das folgende Beispiel liest Datensätze aus einem File ein und gibt sie sortiert aus:

```
#!/home/vogel/usr/local/perl5.5/bin/perl -w
use strict;

# Name und Pfad des Datenfiles
my $file="/home/vogel/projekt/htdocs/cgi_server/doc/archiv/adress01.dat";
my $file="daten.dat";
my (@array, @sortet, $string);

# File zum Lesen öffnen
open (FILE, "<$file") || die "Kann File $file nicht öffnen: $!";

while (<FILE>) {
  chomp; # einlesen und Zeilenvorschub entfernen
  @array = split /\|/; # an | auftrennen
}

close (FILE) || die "Kann File $file nicht schliessen: $!";

@sortet = sort { $a cmp $b } @array;

foreach (@sortet) {
  print $_."\n";
}
```

Komplexe Datenstrukturen (siehe: [WCS-PP]) Referenzen: Neben der normalen Möglichkeit Variablen mit ihrem Namen anzusprechen, gibt es in Perl *Referenzen auf Datentypen* d.h. intelligente Zeiger auf Datentypen. Damit lassen sich sehr einfach Arrays von Arrays (*List of Lists - LoL*), Arrays von Hashes (*LoH*), Hashes von Hashes (*HoH*) oder noch komplexere Datenstrukturen erzeugen. Ausserdem gibt es auch Referenzen auf Funktionen. Man beschränkt sich hier auf LoL, da diese Datenstruktur in der ersten Variante der Adressdatenbank angewendet wird.

Im Abschnitt *Zerlegen von Datensäzen* konnte man sehen, wie man einen Datensatz als String mit Trennzeichen erzeugt und in einer Datei abspeichert. Wie kann man nun eine solche Struktur aus dem File wieder einlesen und z.B. zuerst nach Nachname und bei Gleichheit noch nach Vorname sortieren, bevor man die Daten ausgibt? Am Besten funktioniert das mit *LoL:*

Zusammensetzen einer LoL

```
@LoL = ( [ "vogi", "barney" ],
         [ "igor", "susi", "steph" ],
         [ "jeeves", "armin", "archimedes" ], );
```

Man beachte die eckigen Klammern um die einzelnen Zeilenarrays. Dies erzeugt automatisch eine Arrayreferenz.

Generierung einer LoL

```
1) Aus einer Datei
while (<FILE>) {
  push @LoL, [ split ];
}

2) Zu einer existierenden Liste hinzufügen
push @{ $LoL[0] }, "wilma", "lorio";
```

Zugriff und Ausgabe von LoL

```
# Ein Element
$LoL[0][0] = "Fredy";
# Ein Anderes Element
$LoL[1][1] =~ s/(\w)/\u$1/;
# Ausgabe des gesamten Dings mit Referenzen
for $array_ref ( @LoL ) {
  print "\t @$array_ref \n";
}
# Ausgabe des gesamten Dings mit Indizes
for $i ( 0 .. $#LoL ) {
  print "\t @{$LoL[$i]} \n";
}
# Sortieren des Arrays
@sortetLoL = sort { $a->[0] cmp $b->[0] # Name
                                ||
                    $a->[1] cmp $b->[1] # Vorname
                  } @LoL;
# Alle Elemente nacheinander ausgeben
for $i ( 0 .. $#sortetLoL ) {
  for $j ( 0 .. $#{$sortetLoL[$i]} ) {
    print "element $i $j is $sortetLoL[$i][$j]\n";
  }
}
```

File Locking (siehe: [WCS-PP]) Es können mehrere Anfragen an einen *httpd* zur gleichen Zeit gestellt werden. Dies bedeutet für die Adressdatenbank, dass z.B. ein Benutzer aus dem Datenfile liest, während ein anderer schreibend darauf zugreift. Mit an Sicherheit grenzender Wahrscheinlichkeit wird dies schief gehen, am Schluss bleibt ein zerstörtes Datenfile zurück. Man muss also Massnahmen ergreifen um dieses

Abbildung 23: Suchformular für Adressbuch I

Verhalten zu unterbinden, das sogenannte *file locking* oder *flock*. Um ein *flock* erfolgreich durchzuführen, bedarf es allerdings der Ünterstützung des Betriebssystemes. Linux und Unix bieten diese Unterstützung, Windows nicht.

Locks gibt es in zwei Varianten, *shared* und *exclusiv*. Den *shared Lock* verwendet man normaler Weise zum *Lesen*. Ein so gelocktes File kann von weiteren Prozessen ebenso mit einem *shared Lock* versehen werden. Wenn man in eine Datei *schreiben* will, verwendet man den *exclusiv Lock*. Fordert ein weiterer Prozess einen *exclusiv*, oder *shared Lock* an, wird er kein Erfolg haben. D. h. der Prozess mit *exclusiv Lock* kann schreiben ohne das etws schief geht.

Allerdings funktioniert das Ganze nur, wenn *alle* Prozesse, die mit dem File arbeiten den Lock Mechanismus verwenden.

5.3.2 Resultat: Adressbuch I (*addressbook01.pl)* - benutzt einTextfile und File Locking

Auf eine detailierte Beschreibung des Skriptes wird hier verzichtet. Die relevanten Techniken für diesen Prototyp wurden oben beschrieben. Der Code ist gut dokumentiert und strukturiert.

Das Script kann verschiedene HTML-Seiten generieren:

1. Suchformular (siehe Abb. 23): Hier kann man sich entweder das ganze Adressbuch anzeigen lassen, oder man klickt auf einen Buchstaben und schon erscheint eine Liste der Einträge, welche mit diesem Buchstaben beginnen. Man kann aber auch einen neuen Eintrag vornehmen: dazu klickt man auf den Button '*Neuen Eintrag vornehmen*'.

2. Eingabeformular (siehe Abb. 24): Wenn alle Daten eingegeben sind, einfach den *Speichern-Button* drücken, und schon wird der eben erstellte Datensatz im File: *adress.dat* abgelegt, danach erhält man wieder die Startmaske.

3. Ausgabe der Suchfunktion (siehe Abb. 25): Hier werden die Datensätze der Suche als Tabelle dargestellt. Der Link unter dem Namen ruft das Skript erneut auf, übergibt aber die ID des Datensatzen. Damit erhält man wieder die Eingabemaske (siehe Abb. 24) mit den Daten des Datensatzes, den man jetzt löschen oder ändern kann.

Das Skript speichert seine Daten in einem File mit folgendem Format:

```
979906566681|Petra|Meier|87645647|p@sunrise.ch|Alle 3|8000|Zürich
979907849723|Dave|Vogel|3021632|v@glue.ch|baldweg 13|3012|Bern
979907895725|Susi|Vogel|5678843|s@wer.ch|Lasustr. 33|3000|Bern
979907950727|Erwin|Clinton|989475|c@gov.gov|whiteHouse|47|Oberfultigen
979908084741|Chris|Mey|745698|c@http.org|Rosenstr.5|8876|Wettingen
```

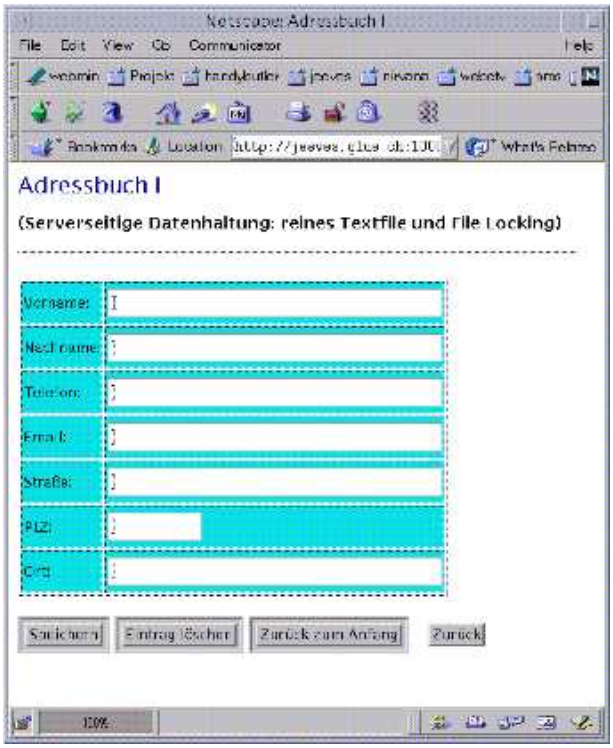

Abbildung 24: Eingabemaske für Adressbuch I

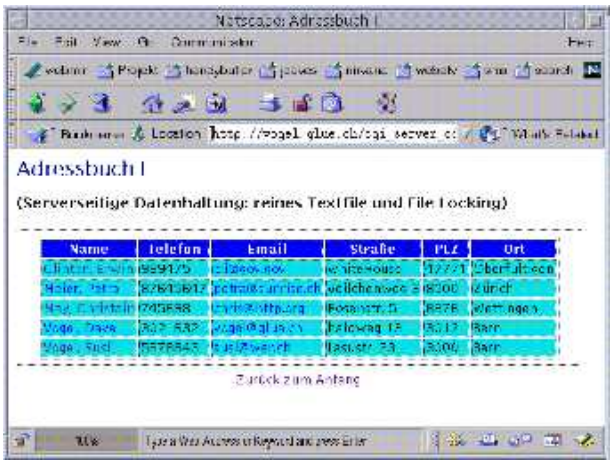

Abbildung 25: Übersichtsmaske für Adressbuch I

Die Zweite Version des Skriptes wird auch auf einem Textfile als Datenbasis beruhen. Die Manipulation der Daten wird aber einfacher, denn man nutzt SQL Befehle um Datensätze zu ändern, löschen oder hinzuzufügen. Für Ungeübte wird das Programm also besser leserlich, da man nicht mehr kryptische perl Befehle benutzt, um z.B. eine Zeile zu lesen uns zu spliten. Zuerst wird hier aber kurz auf Datenbanken, SQL und auf Module für den Datenbankzugriff unter Perl eingegangen.

5.3.3 Grundsätzliches zu Datenbanken (siehe: [REI-PERL] [MySQL] [REI-SQL] [MySQL-Dok])

Was ist eine Datenbank ? <Zitat: Duden Informatik>

System zur Beschreibung, Speicherung und Wiedergewinnung von umfangreichen Datenmengen, die von mehreren Anendungsprogrammen benutzt werden. <Zitat Ende>

Üblicherweise hat man eine Datenbasis (dort werden die eigentlichen Daten abgelegt) und Verwaltungsprogramme (*DBMS Datenbank Magement System*), mit denen man die Daten auslesen und verändern kann. Dabei dient das *DBMS* als Schnittstelle zwischen Daten und Anwendung. Das *DBMS* übernimmt dabei auch die Aufsicht, welche Datensätze gerade von einer Anwendung bearbeitet werden und deshalb für andere Anwendungen gesperrt sein müssen (entspricht dem *File Locking*). Dies lässt sich sogar auf eine ganze Gruppe von Datensätzen und auf eine Gruppe von Aktionen ausdehnen. Man spricht dann von *Transaktionen*. Die Datenbasis kann aus mehreren Tabellen bestehen, die über sogenante *Relationen* miteinader verknüpft sind. Dann spricht man von *relationalen Datenbanken*. Als Beispiel kann man die *Verwaltung der Daten einer Firma* nehmen: Diese Firma hat Kunden, welche von Zeit zu Zeit das eine oder andere Produkt bestellen. Dafür müssen Rechnungen erstellt werden. Sehr sinvoll ist es nun, nicht alle Daten in einer Tabelle abzuspeichern, sondern z.B. die Kundenanschriften mit ihrer Kundennummer in einer Tabelle, die Rechnungen in einer anderen Tabelle. Um jetzt die Zuoordnung (*Realation*) Kunde -> Rechnung zu erhalten speichert man in der Rechnungsdatei einfach noch die zugehörige Kundennummer ab. Damit sind keine Redundanzen (sich wiederholende Tabellenspalten) in der Datenbank vorhanden, man spricht dann von der *ersten Normalform*.

SQL[MySQL-Dok] Unter *SQL* (structured query language) versteht man eine weit verbreitete *Sprache zur Definition und Manipulation relationaler Datenbanken*. Diese Sprache lehnt sich stärker an die menschliche Sprache an, als an eine abstrakte Programmiersprache wie Perl. Hier wird auf die notwendigen Befehle für die Adressdatenbank eingegangen. Um eine Tabelle für die Adressdatenbank zu erzeugen, kann man den *CREATE* Befehl verwenden. Dazu muss man dem *DBMS* noch mitteilen, was man erzeugen will, nähmlich eine Tabelle, und die dazugehörigen Spalten:

```
CREATE TABLE adressbook (id INT NOT NULL,
                         name CHAR(40),
                         email CHAR(40))
```

Erzeugt eine Tabelle mit dem Namen *adressbook* und den Spalten *id* , welche eine Ganze Zahl (INT) sein muss, aber nicht leer sein darf (NOT NULL) sowie die Spalten *name* und *email* , welche beide einen String der Länge 40 aufnehmen können.

id	INT NOT NULL
name	char(40)
email	char(40)

Nun möchte man auch Daten in dieser Tabelle ablegen. Das wird mit dem Befehl *INSERT* erledigt:

```
INSERT INTO adressbook (id, name, email),
                        VALUES (1, 'Metz',  'igor@iam.unibe.ch')
INSERT INTO adressbook (id, name, email),
                        VALUES (2, 'Vogel', 'dave@iam.unibe.ch')
INSERT INTO adressbook (id, name, email),
```

```
VALUES (3, 'Meier', 'hans@iamexwi.unibe.ch')
```

erzeugt dann folgendes:

id	name	email
1	Meiermann	mm@iam.unibe.ch
2	Vogel	dave@iam.unibe.ch
3	Meier	hans@iamexwi.unibe.ch

Will man nun einen Eintrag ändern, so benutzt man den Befehl *UPDATE*

```
UPDATE addressbook SET id = 2, name='Vogel',
                       email='vogel@iam.unibe.ch' WHERE id = 2
```

Damit ändert man den Eintrag, bei dem die *id* gleich der Nummer 2 ist mit den obengenannten Daten und man erhält:

id	name	email
1	Meiermann	mm@iam.unibe.ch
2	Vogel	vogel@iam.unibe.ch
3	Meier	hans@iamexwi.unibe.ch

Um jetzt z.B. alle Einträge der Tabelle auszulesen, bei denen der Name "Meier" enthalten ist, wobei die id nicht relevant sein soll, kann man folgen SQL Befehl verwenden:

```
SELECT name, email FROM adressbook WEHRE name LIKE 'Meier%'
            ORDER by name
```

Damit wählt man die Spalten *name* und *email* aus der Tabelle *adressbook* aus, wobei der Name mit Meier beginnen soll (% dient hier als *Wildcard*). Gleichzeitig sortiert man aufsteigend nach dem Namen. Man erhält also:

name	email
Meier	hans@iamexwi.unibe.ch
Meiermann	mm@iam.unibe.ch

Man kann auch Datensätze löschen:

```
DELETE FROM addressbook WHERE id = 2
```

Löscht den Datensatz mit der *id* 2 aus der Tabelle:

id	name	email
1	Meier	hans@iamexwi.unibe.ch
3	Meiermann	mm@iam.unibe.ch

Leider sprechen nicht alle Datenbanken die gleiche Sprache. Es gibt unterschiedliche *SQL Dialekte*, näheres kann man aus der Dokumentation der benutzten Datenbank erfahren.

5.3.4 Module: DBI *(Data Base Interface)*, Text::CSV_XS, SQL::Statement und DBD::FileModule *(Data Base Driver)*[REI-PERL]

Funktion Das *DBI (Data Base Interface)* Modul stellt eine *API (Application Programming Interface)* für den Datenbankzugriff unter Perl dar. Dabei ist es von der verwendeten Datenbank unabhänging. Dafür sind dann die unterschiedlichsten *DBD (Data Base Driver)* Module zuständig. Damit stellt DBI nur eine Standardschnittstelle und den (Software-) Rahmen zur Verfügung, innerhalb dessen sich die Treiber bewegen. Das Perlskript greift über das *API* auf die Funktionen und Variablen des *DBI Modules* zu. Dadurch wird der benötigte *DBD* angesprochen, welcher dynamisch geladen wird (siehe:26).

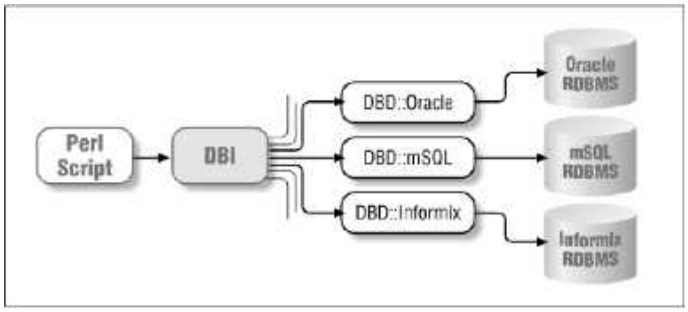

Abbildung 26: Datenfluss mit DBI[DES-BUN]

Verwendung des DBI Moduls[DES-BUN][REI-PERL] Zuerst definiert man die notwendigen Variablen. Im nachfolgenden Code wird das *DBD::CSV Modul* verwendet um auf eine Textdatei als Datenbank zuzugreifen, welche die Daten mit Komma getrennt enthält. Wechselt man die Datenbank, so sind in der Regel nur diese Zeilen an das entsprechende *DBD* Modul anzupassen. Gelegntlich ist auch eine Überarbeitung des *SQL* Codes notwendig, da nicht alle Datenbanken den gleichen *SQL Dialekt* sprechen.

```
use DBI;
use strict

my $DB_NAME = "addressbook02";
my $DB_DIR = "/home/vogel/projekt/htdocs/cgi_server/doc/archiv";
my $DB_DSN = "DBI:CSV:f_dir=$DB_DIR";
my $DB_USER = "";
my $DB_PASSWD = "";
```

Schreiben von Daten in die Datenbank[MySQL-Dok][REI-PERL] Angenommen man will aus einem File, mit Komma separierten Daten, die Daten in eine Datenbank übertragen, könnte das ganze so aussehen:

```
foreach $line (<FILE>) {
  chomp $line;
  ($eins, $zwei, $drei, $vier) = split(/,/,$line);
  $sql = qq[
      insert into $table (spalte_a,spalte_b,spalte_c,spalte_d)
      values($eins,$zwei,$drei,$vier)
  ];

  $dbh = DBI->connect($dsn,vogel,diesIstMeinPasswort)
  $sth = $dbh->prepare($sql);
  $sth->execute;
  $dbh->disconnect;
}
```

Man liest eine Zeile des Files ein, trennt die Daten mit *split*, definiert eine SQL Anweisung:

```
$sql=qq[..]
```

Verbindung mit der Datenbank:

```
$dbh=DBI->connect($dsn,vogel,diesIstMeinPasswort)
```

Vorbereiten der SQL Anweisung:

```
$sth=$dbh->prepare
```

Ausführung:

```
$sth=$dbh->execute
```

und schliesslich Trennung von der Datenbank:

```
$dbh->disconnect .
```

Das macht man in der Schleife für jede Zeile des einzulesenden Files. Diese Methode ist also nicht sehr performant. Man sollte sich also merken, alles was nicht unbedingt in die Schleife muss, sollte ausserhalb geschehen. Das Verbinden mit der Datenbank und das Trennen braucht man nur einmal:

```
$dbh = DBI->connect($dsn,vogel,diesIstMeinPasswort);

foreach $line (<FILE>) {
  chomp $line;
  ($eins, $zwei, $drei, $vier) = split(/,/,$line);
  $sql = qq]
    insert into $table (spalte_a,spalte_b,spalte_c,spalte_d)
    values($eins,$zwei,$drei,$vier)
  ];

  $sth = $dbh->prepare($sql);
  $sth->execute;
}
$dbh->disconnect;
```

Das reduziert die Ausführungszeit erheblich. Man kann noch mehr optimieren, wenn man die Definition des SQL Statements und die Anweisung

```
$sth=$dbh->prepare
```

aus der Schleife nimmt, und das

```
$sth=$dbh->execute
```

Komando modifiziert:

```
$dbh = DBI->connect($dsn,vogel,diesIstMeinPasswort);

$sql = qq[
  insert into $table (spalte_a,spalte_b,spalte_c,spalte_d)
  values(?, ?, ?, ?)
];

$sth = $dbh->prepare($sql);

foreach $line (<FILE>) {
  chomp $line; ($eins, $zwei, $drei, $vier) = split(/,/,$line);
  $sth->execute($eins, $zwei, $drei, $vier);
}

$dbh->disconnect;
```

Beim SQL Statement übergibt man jetzt nicht sofort die Werte, sondern verwendet Platzhalter. Erst beim Ausführen von

```
$sth=$dbh->execute
```

übergibt man die Werte. Dies reduziert die Ausführungszeit nochmals.

Lesen von Daten aus der Datenbank[MySQL-Dok][REI-PERL] Auch das Lesen aus der Datenbank ist recht einfach. Zuerst Verbinden, SQL Statement definieren, Ausführen und dann in einer Schleife solgange Daten lesen, bis keine mehr zurückgegeben werden; *fetchrow_...* erledigt das.

```
$dbh = DBI->connect($dsn, vogel, diesIstMeinPasswort);

$sql = qq[
  select * from $table
];

$sth = $dbh->prepare($sql);
$sth->execute;

while (@row = $sth->fetchrow_array) {
  print "@row\n";
}

$dbh->disconnect;
```

Man liest hier eine Datenbanktabellenzeile in ein Array ein. Effizienter ist es, wenn man eine Arrayreferenz verwendet:

```
$dbh = DBI->connect($dsn, vogel, diesIstMeinPasswort);

$sql = qq[
  select * from $table
];

$sth = $dbh->prepare($sql); $sth->execute;

while ($row = $sth->fetchrow_arrayref) {
  print "@$row\n";
}

$dbh->disconnect;
```

Man kann natürlich auch die Daten in eine *Hashreferenz* einlesen, welche die Spaltennamen als Schlüssel benutzt:

```
$dbh = DBI->connect($dsn, vogel, diesIstMeinPasswort);

$sql = qq[
  select * from $table
];

$sth = $dbh->prepare($sql);
$sth->execute;

while ($ref = $sth->fetchrow_hashref) {
  foreach $key (keys %{$ref}) {
    print "$ref->{$key}, ";
  }
  print "\n";
}

$dbh->disconnect;
```

Anstelle des Vorbereitens der SQL Anweisung und Ausführen der SQL Anweisung, kann man das Ganze auch in einem Befehl ausführen:

```
$dbh = DBI->connect($dsn, vogel, diesIstMeinPasswort);

$sql = qq[
  select * from $table
];

$dbh->do($sql);

while ($ref = $sth->fetchrow_hashref) {
  foreach $key (keys %{$ref}) {
    print "$ref->{$key}, ";
  }
  print "\n";
}

$dbh->disconnect;
```

Fehlerbehandlung[MySQL-Dok] Bei allen Codestücken wurde die Fehlerbehandlung nicht berücksichtigt, was in dem einen oder anderen Fall zu Problemen führen kann. Besser ist es, man fügt bei allen DBI Aufrufen ein *or die Konstrukt* hinzu:

```
$dbh = DBI->connect($dsn,vogel,diesIstMeinPasswort) || die $DBI::errstr;
$sth = $dbh->prepare($sql) || die $dbh->errstr;
$sth->execute || die $sth->errstr;
```

Es gibt aber eine schönere Lösung. Jedes DBI Handle hat ein boolsches Attribut: *RaiseError* . Wenn wir dieses auf wahr setzen, wird das Skript bei einem Fehler vom DBI Handle gestoppt und die entsprechende Fehlermeldung wird ausgegeben. Am besten man setzt dieses Attribut beim Verbinden mit der Datenbank:

```
$dbh = DBI->connect($dsn, vogel, diesIstMeinPasswort, {
                    RaiseError => 1 });
```

Man kann das natürlich auch nachholen, wenn die Verbindung zur Datenbank bereits hergestellt ist:

```
$dbh->{RaiseError} = 1;
```

5.3.5 Resultat: Adressbuch II (ad*dressbook02.pl)* - benutzt eine Flatfile Datenbank[REI-PERL]

Es haben sich nur dort Änderungen gegenüber dem *Adressbuch I* ergeben, wo auf die Daten zugegriffen wird. Zudem muss man beim ersten Mal mit dem Befehl *adress_2.pl?init=1* das Skript aufrufen, damit die Datenbankdatei *addressbook02* erstellt wird.

Man benutzt das DBI Modul und den DBD für eine Flatfile Datenbank. Die dazu notwendigen Variablen werden am Anfang definiert. Mit dem Command

```
my $dbh = DBI->connect($DB_DSN, $DB_USER, $DB_PASSWD, {
                          RaiseError => 1 });
```

wird die Verbindung zur Datenbank erstellt. Alle Unterprogramme, die auf die Datenbank zugreifen, bekommen beim Aufruf das Datenbankhandle *$dbh* übergeben. In den folgenden Zeilen wird - je nach gesetzten Parametern (vom Web-Formular) - in die entsprechenden Unterprogramme verzweigt:

```
if (param('insert')) {
   &insert($dbh, map { param($_) } @dbfields);
   &web_header;
   &search_mask;
} elsif (param('delete')) {
   &delete($dbh);
   &web_header;
   &search_mask;
} elsif (defined param('search')) {
   &web_header;
   &search($dbh);
} elsif (param('edit')) {
   &web_header;
   &get_data($dbh);
   &dataform;
} elsif(param('init')) {
   &web_header;
   &search_mask;
   &init_db($dbh);
} else {
   &web_header;
   &search_mask; }
```

Um Daten Einzufügen oder zu Ändern (&insert), verwendet man die *UPDATE* bzw *INSERT* Anweisung. Sollen Daten gelöscht werden (&delete) definiert man eine entsprechende SQL Anweisung:

```
my $sql = qq[ DELETE FROM addressbook02
              WHERE id = $id];
```

und führt sie aus:

```
$dbh->do($sql);
```

Das Suchen (&search) ist jetz wesentlich einfacher, man muss nicht mehr das ganze Textfile einlesen, sondern kann mit einer *SELECT* Anweisung auf die Daten zugreifen:

```
my $sql = qq[ SELECT id, $dbflist
              FROM addressbook02 $where_clause
              ORDER BY lname];
```

Vorher legt man noch die Bedingung zum Suchen in der *WHERE* Anweisung fest:

```
if (param('lname')) {
   my $keyword = $dbh->quote(param('lname').'%');
   $where_clause = qq[
        WHERE fname CLIKE $keyword OR
              lname CLIKE $keyword];
}
```

Auch die Prozedur zum Füllen der Eingabemaske mit Daten - wenn eine Änderung ansteht - lässt sich mit einer *SELECT* Anweisung durchführen:

```
my $sql = qq[ SELECT id, $dbflist
              FROM addressbook02
              WHERE id = '$id' ];
```

Die notwendigen Parameter füllt man aus einer Hashreferenz, die man mit Daten aus der Datenbank belegt:

```
while ($row = $sth->fetchrow_hashref) {
  foreach $key (keys %{$row}) {
    param(-name => $key, -value => $row->{$key});
  }
}
```

5.3.6 Installation/Konfiguration/Vorbereitung einer MySQL Datenbank

Zur Installation und Konfiguration werden hier keine Angaben gemacht. Zu diesen Themen gibt es unzählige Dokumentationen und Tutorials (siehe: http://www.mysql.com/documentation/index.html `http://www.mysql.com/documentation/index.html`). Nach der erfolgreichen Installation und Konfiguration der MySQL-Datenbank muss man noch eine Datenbasis anlegen: mit folgendem Befehl erzeugt man die Datenbasis *adressbuch02*:

```
mysqladmin create adressbuch02
```

Um die Datenbasis verwenden zu können, legt man einen Benutzer an und vergibt die notwendigen Schreibrechte. Dazu verbindet man sich mit der Datenbasis mysql (da werden die Berechtigungen verwaltet) mit dem Befehl:

```
mysql mysql
```

Danch können die Rechte vergeben werden (siehe:[REI-SQL]).

Als *DBD (Data Base Driver)* muss man noch das Modul *DBD::mysql* installieren (bei der Version I des Adressbuches hatte man das Modul *DBD::FileModule* verwendet).

Damit sind alle Voraussetzungen erfüllt, um diese Adressdatenbank mit MySQL nutzen zu können.

5.3.7 Resultat: Adressbuch III (ad*dressbook03)* - benutzt eine MySQL Datenbank

Wie beim Adressbuch II, muss man auch hier die Prozedur *init_db* (*adress_3.pl?init=1*) aufrufen. Dadurch wird die Datenbanktabelle *adressen* erstellt (Adressbuch II: eine Datenbankdatei ad*dressbook02*). Man kann die Tabelle auch manuel erstellen: entweder mit den mysql-Commands, oder mit dem Tool *phpMyAdmin*[PHP-ADM], das sich hervorragend für solche Administrationsarbeiten eignet. Das Skript funktioniert analog der zweiten Version. Die Änderungen gegenüber adress02.pl sind minimal. Man muss nur die Variablen, welche für das Ansprechen der MySQL Datenbank verwendet werden anpassen:

```
use CGI qw/:standard :netscape/;
use CGI::Carp qw/fatalsToBrowser/;
use strict; use DBI;
my ($dbh, $rv);
my $db_type = 'mysql';
my $port = 9306;
my $hostname = "localhost";
my $db_name = 'adressdatenbank';
my $DB_DSN = "DBI:$db_type:$db_name:$hostname:$port:";
my $DB_USER = "vogel";
my $DB_PASSWD = "diesIstMeinPasswort";
my $DB_Table = 'adressen';
```

Weiterhin kennt MySQL den Befehl CLIKE nicht, sondern verwendet den Befehl LIKE.

5.4 Zusatzprogramme zum Exportieren/Importieren des Adressbuches

Auch hier kann man wieder eine Applikation implementieren, mit der man die Sms-Adressdaten mit verschiedenen Adressbücher synchronisieren kann.

5.4.1 Import:

Zuerst muss man die Daten aus dem Netscape- bzw. Outlook Applikation exportieren (mittels eigener Exportfunktion der Anwendung). Um diese Datei auf den Server zu laden, muss man ein entsprechendes HTML-Formular erstellen. Damit kann man mittels der Upload-Funktion die eigene Festplatte durchsuchen, das Adressfile selektieren und mittels http dem Server schicken. Serverseitig wird die Datei z.B. von einem Perl-Script verarbeitet, und die Daten werden in der *Datenbank* gespeichert.

5.4.2 Export:

Die Exportfunktion stellt eine Datei auf dem Server zum Download bereit, die aus den Adressdaten der Datenbank dynamisch generiert wurde. Vorher muss der User angeben, für welches Adressbuch (z.B. Netscape Adressbuch) die Datei erstellt werden soll. Nach dem Download kann man dieses File mittels der eingebauten Importfunktion ins Adressbuch übernehmen.

5.5 Sicherheit[PE-TU]

Über das Common Gateway Interface lassen sich vielfältige Attacken gegen das System und die gespeicherten Daten machen. Skripte auf dem Server erlauben dem Besucher, interaktive Prozesse vom Browser auslösen zu lassen. Da es jedoch kaum im Interesse des Webmasters ist, sich von böswilligen Eindringlingen die Festplatte formatieren zu lassen gilt es, dem Inhalt der Skripte höchste Aufmerksamkeit zu schenken.

Die Gefahren, die hinter dem Common Gateway Interface auf den Server lauern, sind zahlreich: der Diebstahl vertraulicher Daten oder Systeminformationen, das Blockieren des Servers durch einen *Denial-of-Service-Angriff* (siehe:[DOS-RES]) oder gar das Ausführen von Systembefehlen auf dem Server und damit das Anrichten schwerster Schäden. Sicherheitslücken entstehen durch die Art und Fähigkeiten der Serversoftware, die Konfiguration der Programmfunktionen und durch den Code des Skripts. Dazu ist es nicht notwendig, dass das Skript von Haus aus dazu angelegt ist, Schaden anzurichten. Oft reicht schon eine fehlende Sicherheitsabfrage, auf die unabsichtlich oder aus Bequemlichkeit verzichtet wurde, um dem Eindringling Tür und Tor zu öffnen. Wird beispielsweise der User-Input über ein Formular nicht kontrolliert, lassen sich so auf einfache Weise Systemfunktionen zur Ausführung bringen.

Dabei ist es nahezu ausgeschlossen, jede Lücke zu stopfen. Denn nicht nur, dass CGI-Skripte schon von Natur aus ein erhebliches Risiko darstellen; zudem besitzt jedes Serversystem ähnlich den Browsern seine eigenen, programmspezifischen Sicherheitsmängel. Ein Skript, das auf einem Server zuverlässig und sicher seine Arbeit verrichtet, kann auf einem anderen Server zu einem unkalkulierbaren Risiko werden.

5.6 Der CGI Lebenszyklus[HUN-CRA]

Mit Hilfe des CGI ist es mit relativ geringem Aufwand möglich, auf die vom Client übergebenen Parameter zu reagieren und die Webinhalte, unter Berücksichtigung der übergebenen Parameter, dynamisch aufzubauen. Diese Tatsache führte innerhalb kürzester Zeit dazu, dass sich CGI als eine Art Pseudostandard bei einem Grossteil der heute verfügbaren Webservern etablieren konnte. Betrachtet man die Arbeitsweise der CGI-Programme einmal etwas genauer, so fallen einem allerdings schnell einige Nachteile dieser Technologie ins Auge. Erhält ein Webserver einen Request vom Client, der an ein CGI-Programm adressiert ist, so muss dieser einen neuen Prozess erzeugen und mit Hilfe von Umgebungsvariablen die zur Bearbeitung notwendigen Informationen und Daten an das CGI-Programm übergeben. Dies bedeutet für den Server einen erheblichen Zeit- und Ressourcenverbrauch, so dass die maximale Anzahl an parallel zu verarbeitenden Requests von vornherein stark eingeschränkt wird (siehe Abb.27).

Wenn die erste Anfrage eintrifft, starten Server, die die CGI-Spezifikation einhalten, die erste Instanz des designierten CGI-Skripts. Während der Abarbeitung der erforderlichen Prozesse (z. B. Datenbankzugriff) erreicht den Server eine weitere Anfrage an dasselbe CGI-Skript. Trotz der Existenz einer Instanz dieses Prozesses wird nun eine weitere Instanz erzeugt. Sobald die Skripten ihre Arbeit beendet und ihre Ergebnisse an den WWW-Server weitergegeben haben, werden sie beendet (siehe Abb.28) Die Abbildung 29 zeigt eine detailliertere Ansicht des Lebenszyklus der Datenbank-Gateways.

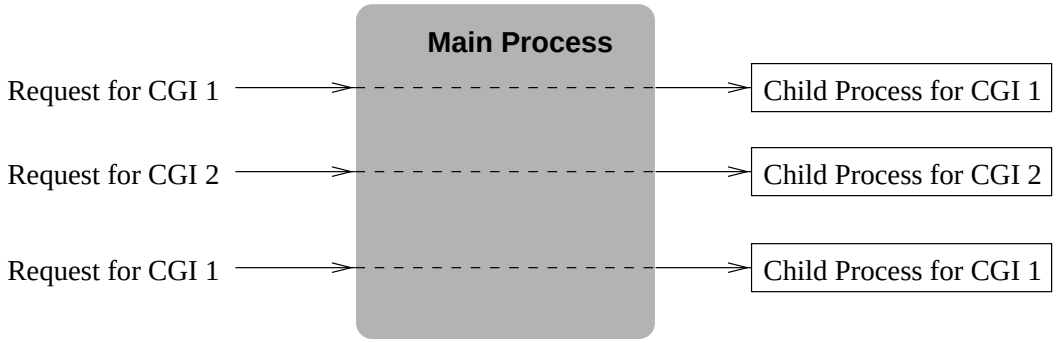

Abbildung 27: Der CGI-Lebenszyklus

Im Bereich der CGI Programmiersprachen konnte sich vor allem die Sprache Perl etablieren. Dies hat seinen Ursprung in den guten Textbearbeitungsmöglichkeiten, die Perl bietet und deren breite Verfügbarkeit auf allen gängigen Plattformen. Als Nachteil bringt ein Perl-Programm allerdings mit sich, dass zu seiner Ausführung jedesmal ein Interpreter gestartet werden muss, was wiederum den Verbrauch von Zeit und Ressourcen bedeutet.

5.7 Grenzen von Perl

Beim Einsatz von CGI muss das vom WWW-Server aufgerufene Programm den übergebenen Request analysieren. Da keine Schnittstellenvereinbarung vorliegt und keine Typüberprüfung der Parameter erfolgt, ist die Bearbeitung recht fehleranfällig und vor allem schwer wartbar. Für einfache Programme mag dieser Ansatz genügen, bei Businessanwendungen stösst man aber schnell an Grenzen und muss nach besser geeigneten Lösungen suchen.

CORBA ist eine Spezifikation der 1989 gegründeten *Object Management Group (OMG)* für verteilte, objektorientierte Softwaresysteme in heterogenen Umgebungen. Als Non-profit-Organisation ist die *OMG* mit über 750 Mitgliedern für die Prüfung eingereichter Vorschläge und Spezifikationen zuständig. Die Implementierung der im Zentrum von *CORBA* stehenden *Object Request Broker (ORBs)* bleibt den Herstellern am Markt überlassen und verspricht daher technische Innovation sowie die Verfügbarkeit der Produkte auf diversen Plattformen und Betriebssystemen. Im Gegensatz zu Middleware-Ansätzen wie *DCOM* oder *DCE* liegt die Implementierung nicht in der Hand einer einzelnen Firma.

5.8 FastCGI[FASTCGI]

Eine erste Alternative zu CGI entwickelte die Firma Open Market. Ihre *FastCGI-Technik* sollte die Problematik der multiplen Prozesserzeugung einschränken, indem lediglich für jedes CGI-Programm und nicht für jeden Client Request ein einzelner, persistenter Prozess erzeugt wird. Mehrere Requests an ein CGI-Programm können somit innerhalb eines Prozesses nacheinander abgehandelt werden (siehe Abb. 30).

Doch auch bei dieser Technik gibt es einige entscheidende Nachteile, die letztendlich dazu geführt haben, dass sie in keinen der grossen Webserver jemals implementiert worden ist. Zunächst einmal müssen weiterhin separate Prozesse erzeugt werden, die eine Interaktion mit dem Webserver erheblich erschweren. Darüber hinaus muss für die parallele Bearbeitung mehrerer Requests an dasselbe FastCGI-Programm ein Pool von Prozessen zur Verfügung gestellt werden. Und letztendlich gilt auch für FastCGI-Programme, dass bei der Verwendung von Perl durch den benötigten Interpreter zusätzliche Ressourcen benötigt werden. Es existierenaber für einige wenige Webserver, wie z.B. den Apache Webserver oder Microsofts Internet Information Server externe Module bzw. Programme von Drittanbietern am Markt, die einen Perl-Interpreter in den Webserver integrieren oder aber direkt mit der nativen API des Servers kommunizieren, was zu einer beschleunigten Ausführung der Perl-Programme führt.

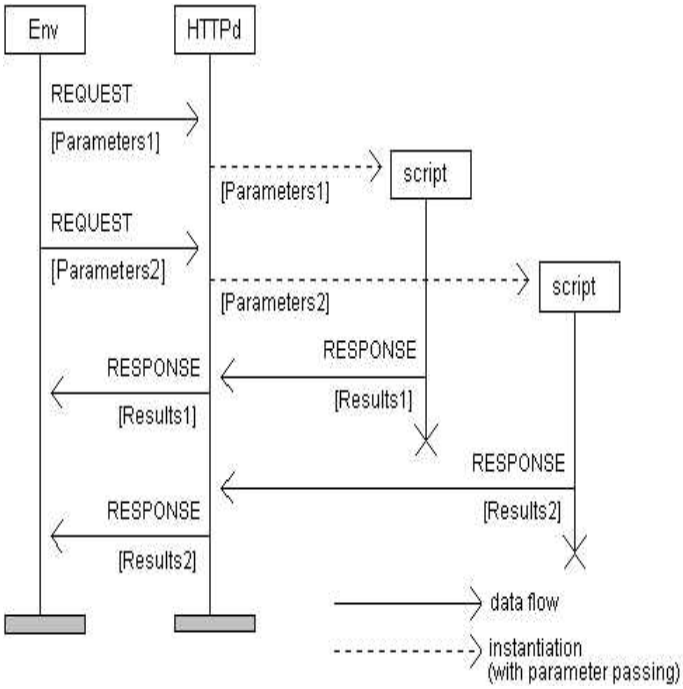

Abbildung 28: Aktivierung von CGI-Prozessen[GUT-97]

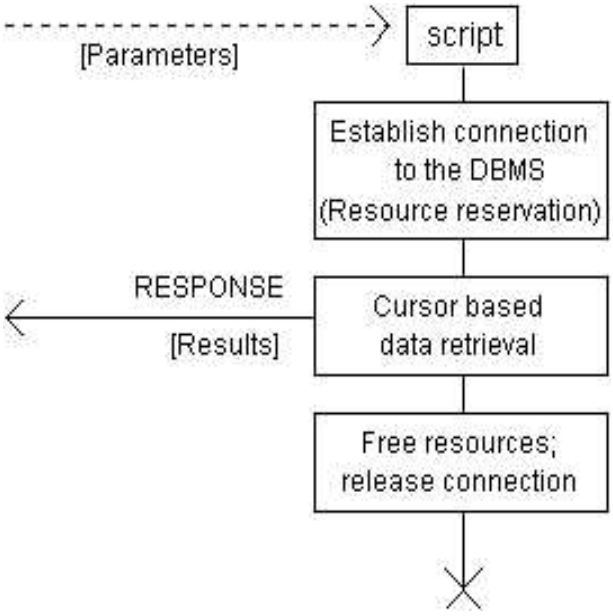

Abbildung 29: Lebenszyklus der Datenbank-Gateways[GUT-97]

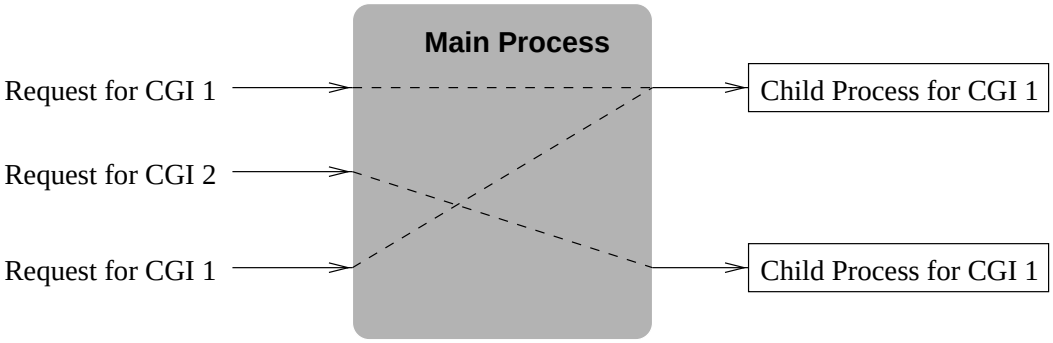

Abbildung 30: Der FastCGI-Lebenszyklus

5.9 mod_perl[APA]

Falls man den Apache Web Server einsetzt, so kann man die CGI Leistung mit *mod_perl* erhöhen. *mod_perl* ist ein Apache-Modul, welches eine Kopie des Perl Interpreter in httpd einfügt. Dadurch steht dem Apache Server die gesamte Funktionalität von Perl zur Verfügung. Die CGI-Skripte werden vor-kompiliert und werden dadurch viel schneller ausgeführt.

6 PHP

Kann man nur über die CGI-Schnittstelle dynamisch auf Interaktionen eines Clients reagieren, oder gibt es noch andere Möglichkeiten auf der Serverseite Applikationen zu starten? Kann man einem Client durch eine andere Technologie Daten aus einer MySql-Datenbank übers Web (http(s)) anbieten?

PHP ist eine von diesen Möglichkeiten. Dieses Kapitel ist eine Einführung in die Grundlagen der Programmiersprache PHP und zeigt wie man auf eine MySql-Datenbank zugreifen kann und dynamisch HTML-Code generiert.

6.1 PHP Grundlagen[REE-DSP]

6.1.1 Einleitung

PHP ist eine serverseitige, in HTML eingebettete Scriptsprache - oder mit anderen Worten: PHP-Scripte werden auf dem Server ausgeführt, im Gegensatz z.B. zu üblichem JavaScript und Java. Der Programmcode wird in die HTML-Quelldatei geschrieben und somit i.A. nicht in einer extra *PHP-Date*i abgelegt. Als Script werden Programme bezeichnet, die keine eigenständigen Programme sind, weil sie nicht kompliziert genug sind und andere Programme benötigen, um ausgeführt zu werden.

Im Gegensatz zu HTML und JavaScript erscheint der eigentliche PHP-Code i.A. nicht auf der Clientseite, d.h. der Quellcode, den man sich im Browser auch ansehen kann, enthält für gewöhnlich keinen PHP-Code. Der HTML-Code wird beim Abruf der Webseite, wie bei normalen Seiten auch, 1:1 an den Client geschickt; der PHP-Code wird durch den Server ausgeführt und dann die Ausgabe an den Client gesandt. Es ist auch möglich, PHP-Scripte ganz ohne (sichtbare) Ausgabe laufen zu lassen - auch das kann sinnvoll sein.

Immer wenn man sich fragt, ob etwas möglich ist, muss man überlegen, ob dazu eine Aktion auf dem Server (wo die Webseite liegt) oder auf dem Client (wo die Webseite angezeigt wird) notwendig ist. Z.B.: Ist es möglich, mit einem PHP-Befehl die aktuelle Webseite auszudrucken? Die Antwort ist ganz einfach: Damit auf dem Client die Seite ausgedruckt wird, muss dem Browser ein Befehl übermittelt werden. Da PHP aber auf dem Server ausgeführt wird, kann es diesen Befehl folglich nicht selbst in die Tat umsetzen. Auf dem Client wird aber z.B. JavaScript ausgeführt, das einen Befehl anbietet, der die Seite ausdruckt (sofern JavaScript aktiviert ist). Für viele andere Aktionen ist aber nicht einmal JavaScript nötig.

6.1.2 Grundbefehle

PHP wird einfach in den HTML-Quellcode geschrieben. Damit der Server weiss, in welcher Datei er nach PHP-Scripten suchen soll, müssen die Dateien die richtige Endung (Extension) haben. Bei PHP3 waren *.php3* und *.phtml* üblich, bei PHP4 ist dagegen *.php* gebräuchlicher. Man kann natürlich den Webserver so konfigurieren, dass er jede beliebige Endung als PHP-Script identifiziert. Damit der Server darüber hinaus noch weiss, welche Ausdrücke er in der Datei interpretieren soll, müssen jeweils der Anfang und das Ende des PHP-Teils gekennzeichnet werden. Dafür gibt es drei Möglichkeiten:

```
1. <? echo "Hello world!"; ?>
2. <?php echo "Hello world!"; ?>
3. <script language="php">
     echo "Hello world!";
   </script>
```

Die erste Möglichkeit ist die kürzeste und damit bei vielen die beliebteste. Sie ist allerdings nicht XML-konform, so dass später Probleme auf den Programmierer zukommen können, wenn man sie benutzt. Ausserdem gibt es Server, die diese Variante nicht erkennen. Besser ist die zweite Variante; sie ist kurz aber dennoch XML-Konform.

Die Sprache PHP ist hauptsächlich von C, aber auch von Java und Perl (die ihrerseits von C beeinflusst wurden) beeinflusst. Aber auch für Pascal/Delphi-Programmierer ist die Sprache nicht schwer zu erlernen. Eine Anweisung wird immer mit einem *;* abgeschlossen.

Der echo-Befehl Den wichtigsten Befehl wird oben schon verwendet: den *echo-Befehl*, der Strings ausgibt. Im obigen Beispiel wird jeweils *Hello world!* ausgegeben. Der Text, der ausgegeben werden soll, muss natürlich in Anführungsstrichen stehen, da der Server sonst versucht, ihn als PHP-Befehl zu interpretieren. Bei den Anführungsstrichen gibt es zwei verschiedene: einmal das einfache ' und das doppelte " . Es gibt auch einen Unterschied zwischen den beiden: Bei den doppelten Anführungsstrichen versucht der Server, den Text zu interpretieren, bei den einfachen hingegen behandelt er ihn nicht speziell, sondern gibt ihn z.B. direkt aus.

```
$var = 123;
echo 'Die Variable $var hat den Wert 123!\n';
echo "Die Variable $var hat den Wert 123!\n";
```

Das erste echo gibt *Die Variable $var hat den Wert 123!\n* aus, das zweite hingegen *Die Variable 123 hat den Wert 123!* mit folgendem Zeilenumbruch.

```
echo "Say \"Hello World\" my friend"; // Gibt aus: Say "Hello World!" my friend
```

Wie man sieht, müssen doppelte Anführungsstriche anders geschrieben werden. Dieses Vorgehen nennt man *Quoten* oder *Quoting*. Es ist insbesondere für das Ausgeben von HTML-Quelltext in Verbindung mit *echo* und *print* nötig und kann u.U. zu Problemen führen, wenn man vergisst, in allen Teilstrings zu quoten.

Der print-Befehl Neben dem *echo*- gibt es auch den *print*-Befehl. Im Endeffekt leisten beide dasselbe: Sie geben Text aus. *echo* ist ein internes Sprachkonstrukt, während hingegen *print* eine Ausdruck (Expression) ist. *echo* kann mehrere Argumente haben, die nicht in Klammern stehen dürfen. *print* kann nur genau ein Argument haben. Alle folgenden Anweisungen sind zulässig und geben dasselbe aus:

```
$var1 = "Hallo";
$var2 = "Welt!";
echo $var1," ",$var2;
echo $var1." ".$var2;
print ($var1." ".$var2);
$res = print ($var1." ".$var2);
```

Auf alle anderen Befehle wird hier nicht mehr eingegangen. Unter Php-Center ([PHP-CEN]) oder unter Php-Net ([PHP-NET]) findet man genaue Angaben zur Syntax und Befehlen.

6.2 PHP und HTML[REE-DSP]

6.2.1 Ein Beispiel: Formular

Mit PHP können Formulardaten relativ einfach eingelesen werden. Man muss im Formular als *action* den Dateinamen der PHTML-Datei angeben und als method empfiehlt sich *post*. Bei *get* kann man sonst die Werte der Eingabe in der URL wiederfinden.

Nun ein kleines Beispiel. Die Eingaben, die auf der Seite *eingabe.html* eingetragen wurden, werden durch OK an die Datei *ausgabe.phtm*l übermittelt. Durch diese werden sie dann ausgegeben. Quelltext der Datei *eingabe.html*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
  <title>Eingabe</title>
</head>
<body>
<form action="ausgabe.phtml" method="post">
```

```
Feld1: <input name="feld1" size="60" maxlength="60"><br>
Feld2: <input name="feld2" size="60" maxlength="60"><br>
<input type="submit" value="OK">
<input type="reset" value="Abbrechen">
</form>
</body>
</html>
```

Quelltext der Datei *ausgabe.phtml*:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
  <title>Ausgabe</title>
</head>
<body>
<?php
print "Feld 1:".$feld1."<br>Feld 2:".$feld2;
?>
</body>
</html>
```

6.2.2 Werte übergeben

Es gibt auch Situationen, da möchte man dem PHP-Skript Werte übergeben, ohne ein Formular zu verwenden. Dies ist natürlich auch möglich. Ein normaler HTML-Link sieht folgendermassen aus:

```
<a href="datei.phtml">Linktext</a>
```

Wenn man jetzt der Datei die Werte *Wert1* und *2* in den Variablen *VAR1* und *VAR2* übergeben will, sieht der Link folgendermassen aus:

```
<a href="datei.phtml?var1=Wert1&var2=2">Linktext</a>
```

Allgemeiner formuliert: An das Verweisziel (in diesem Fall *datei.phtml*) wird mit einem *?* beginnend der Variablenname und mit einem Gleichheitszeichen der Wert herangehängt. Weitere Werte werden mit einem *&* statt *?* angehängt. Es dürfen keine Leerzeichen dabei entstehen.

Wenn man die übergebenen Werte verwendet, darf man nicht vergessen, dass jeder die Werte beim Aufruf verändern kann. Deshalb sollten die Variablen vor der Weiterverarbeitung auf korrekte Werte hin überprüft werden.

6.3 PHP und MySQL[REE-DSP]

Hier werden jetzt Befehle von PHP und mySQL verwendet. Zwischen diesen beiden Sprachen ist streng zu unterscheiden.

6.3.1 Syntax

mysql_connect

Syntax:

```
int mysql_connect(string hostname ,string username ,string password);
```

Mit *mysql_connect()* wird eine Verbindung zum Server geöffnet.

Wenn der zurückgegebene Wert positiv ist, verlief die Verbindung erfolgreich; bei Null gab es einen Fehler. Bei einem zweiten Aufruf mit denselben Parametern wird keine neue Verbindung erstellt, aber es wird der *link_identifier* zurückgegeben. Der *link_identifier* wird benötigt, um bei mehreren Verbindungen eine eindeutig bestimmen zu können.

Die Verbindung wird automatisch bei Beenden des Scripts geschlossen; sie kann aber auch mit *mysql_close()* explizit geschlossen werden.

mysql_close

Syntax:

```
int mysql_close(int link_identifier);
```

mysql_close schliesst eine bestehende Verbindung zum Server.

Es wird entweder *true* bei Erfolg oder *false* bei einem Fehler zurückgegeben.

Mit *link_identifier* kann explizit angegeben werden, welche Verbindung geschlossen werden soll. Wenn nichts angegeben wurde, wird die zuletzt geöffnete Verbindung geschlossen.

mysql_select_db

Syntax:

```
int mysql_select_db(string database_name ,int link_identifier);
```

Mit *mysql_select_db* wird die Datenbank ausgewählt, auf die sich die Anfragen beziehen soll.

Es wird entweder *true* bei Erfolg oder *false* bei einem Fehler zurückgegeben.

Wenn kein *link_identifier* angegeben wurde, wird die zuletzt geöffnete Verbindung zum Server genommen.

mysql_query

Syntax:

```
int mysql_query(string query , int link_identifier);
```

mysql_query sendet die Befehlsfolge *query* an den Server mit der DB, die durch den *link_identifier* festgelegt ist. Wird kein *link_identifier* angegeben, wird die zuletzt geöffnete Verbindung genutzt.

Es wird ein sogenannter Zeiger auf das Ergebnis (*result* pointer) zurückgegeben. Wenn dieser (Result-)Zeiger den Wert 0 hat, gibt es einen Fehler. Mit dem Zeiger an sich kann man aber nicht viel anfangen; vielmehr benötig man ihn, um die folgenden Funktionen nutzen zu können. Dort wird er als Parameter *result* übergeben.

mysql_fetch_array

Syntax:

```
array mysql_fetch_array(int result [,int resulttype]);
```

Dies ist eine erweiterte Version von *mysql_fetch_row* (siehe nächste Funktion). Die Indizes des Arrays werden nicht von 0 ausgehend durchgezählt, sondern nach den Spaltennamen benannt.

Die Funktion ist in der Ausführung nur unwesentlich langsamer als *mysql_fetch_row*, obwohl es deutlich angenehmer zu programmieren ist.

Wenn beim Join zweier Tabellen zwei Spalten denselben Namen haben, müssen ihnen mit Hilfe der SELECT-Anweisung andere Namen gegeben werden. Beispiel:

```
SELECT t1.s1 AS foo, t2.s1 AS bar FROM t1, t2
```

Die Spalte *s1* der Tabelle *t1* hat nun den Namen *foo* und *s1* aus *t2* hat den Namen *bar*.

Das optionale zweite Argument *result_type* in *mysql_fetch_array* ist eine Konstante und kann die folgenden Werte annehmen: MYSQL_ASSOC, MYSQL_NUM und MYSQL_BOTH.

```
<?php
mysql_connect($host,$user,$password);
mysql_select_db("database");
$result = mysql_query("SELECT user_id, fullname FROM users");
while($row = mysql_fetch_array($result)) {
  echo $row["user_id"];
  echo $row["fullname"];
}
?>
```

mysql_fetch_row

Syntax:

```
array mysql_fetch_row(int result);
```

Holt eine Zeile aus der DB-Abfrage, gibt diese als Array zurück und setzt den Result-Zeiger auf die nächste Zeile. Für *result* muss der Rückgabewert (=Zeiger) der *mysql_query*-Funktion genommen werden.

Dasselbe Beispiel wie bei *mysql_fetch_array*:

```
<?php
mysql_connect($host,$user,$password);
mysql_select_db("database");
$result = mysql_query("SELECT user_id, fullname FROM users");
while($row = mysql_fetch_row($result)) {
  echo $row[0];
  echo $row[1];
}
?>
```

mysql_error

Syntax:

```
string mysql_error([int link_identifier]);
```

Gibt die Fehlermeldung des letzten SQL-Befehls zurück. Wenn es keinen Fehler gab, wird nichts zurück gegeben. Wird kein *link_identifie*r angegeben, wird die zuletzt geöffnete Verbindung genutzt.

mysql_errno Syntax:

```
int mysql_errno([int link_identifier]);
```

Gibt die Fehlernummer des letzten SQL-Befehls zurück. Wenn es keinen Fehler gab, wird 0 zurückgegeben. Wird kein *link_identifier* angegeben, wird die zuletzt geöffnete Verbindung genutzt.

mysql_insert_id

Syntax:

```
int mysql_insert_id(int link_identifier);
```

Gibt die Nummer zurück, die beim letzten INSERT dem Feld mit AUTO_INCREMENT zugewiesen wurde.

mysql_num_rows

Syntax:

```
int mysql_num_rows(int result);
```

Gibt die Anzahl der Zeilen im Ergebnis zurück.

6.3.2 Ein Beispiel: Adressbuch (PHP und MySQL)

Als Prototyp nimmt man hier das Beispiel von Ryan Powers (siehe: [POW-RAY]).
Zuerst definiert man die notwendigen Variablen:

```
$host = "jeeves.glue.ch:9306";
$user = "root";
$pass = "";
$database = "adressdatenbank";
```

Mit

```
require "template.inc";
```

wird ein template File eingelesen. Hier sind einige Funktionen definiert, die nur mit reiner HTML-Generierung zu tun haben:

```
function list_top(): list all table headers
function list_bottom(): generate and of table
function address_top(): header for all pages
function address_bottom(): footer for all pages
```

Mit dem Command

```
$db= mysql_connect("$host","$user","$pass");
mysql_select_db("$database", $db);
```

wird die Verbindung zur Datenbank erstellt. In den folgenden Zeilen wird - je nach gesetzten Parametern (vom Web-Formular) - in die entsprechenden Programmblöcke verzweigt. In der folgenden Aufzählung wird nur der Dialog mit der DB aufgelistet. Auf das Generieren von HTML-Code wird an dieser Stelle nicht eingegangen. Den vollständigen Code findet man im Anhang.

- **if ($submit):** Einen neuen Eintrag in der Tabelle vornehmen

```
mysql_query("INSERT INTO address (ID,lname,fname,address,
                                  phone, email, wphone,
                                  aim, icq)
                          VALUES (0, '$lname','$fname','$address',
                                  '$phone', '$email', '$wphone',
                                  '$aim', '$icq')");
```

- **if ($action == "search"):** Suchformular anzeigen
- **if ($action == "add"):** Eingabemaske anzeigen
- **if ($action == "delete1"):** Eintrag löschen (Aufruf vom ersten Web-Formular aus)
- **if ($action == "delete2"):** Eintrag löschen (Aufruf vom Confirm Web-Formular aus)

```
if ($action == "delete2") {
  $result = mysql_query("SELECT * FROM address where id = $id", $db);
  while($row = mysql_fetch_row($result)) {
    mysql_query("Delete FROM address where id = $id", $db);
  }
}
```

- **if ($search):** Suchen und Resultate anzeigen

```
if ($search) {
  $result = mysql_query("SELECT * FROM address
                         WHERE lname LIKE '%$query%'
                         OR fname LIKE '%$query%'", $db);
}
```

- **if ($action == "edit"):** Eintrag mittels ID aus DB lesen und in Editiermaske darstellen

```
if ($action == "edit") {
  $result = mysql_query("SELECT * FROM address
                         where id = $id", $db);
}
```

- **if ($edit):** Eintrag in DB schreiben

```
if ($edit) {
  mysql_query("update address set lname = '$lname' where id = '$id'");
  mysql_query("update address set email = '$email' where id = '$id'");
  mysql_query("update address set fname = '$fname' where id = '$id'");
  mysql_query("update address set phone = '$phone' where id = '$id'");
  mysql_query("update address set address = '$address' where id = '$id'");
  mysql_query("update address set aim = '$aim' where id = '$id'");
  mysql_query("update address set icq = '$icq' where id = '$id'");
  mysql_query("update address set wphone = '$wphone' where id = '$id'");
}
```

- **if ($action == "profile"):** Profil anzeigen

```
if ($action == "profile") {
  $result = mysql_query("SELECT * FROM address where id = $id",$db);
}
```

- **if ($action == "list"):** Alle Einträge anzeigen

```
if ($action == "list") {
  $result = mysql_query("SELECT * FROM address order by lname",$db);
}
```

- **if ($action == "printid"):** Eintrag mit bestimmter ID druckfertig anzeigen

```
if ($action == "printid") {
  $result = mysql_query("SELECT * FROM address where id = $id",$db);
}
```

- **if ($action == "printall"):** Alle Einträge druckfertig anzeigen

```
if ($action == "printall") {
  $result = mysql_query("SELECT * FROM address", $db);
}
```

6.4 PHP und HTTP[REE-DSP]

6.4.1 Header

Neben der eigentlichen Seite schickt der Server an den Client (Browser) noch einige Zusatzinformationen. Diese werden vor der eigentlichen Seite im sog. *Header* gesendet. Mit diesen Informationen sagt der Server z.B., ob die angezeigte Seite wirklich die gewünschte Seite ist (Status *200 Found*), oder ob die Seite nicht gefunden werden konnte und deshalb eine Fehlerseite angezeigt wird (Status *404 Not Found*). Auch kann der Server dem Client mitteilen, dass die Seite sich unter einer anderen Adresse befindet (Status *301 Moved Permanently* oder *302 Found*). Es kann auch die Aufforderung geschickt werden, sich zu authentifizieren (Status *401 Unauthorized*).

Zusätzlich zum Status einer Seite kann auch übermittelt werden, wann die Seite zum letzten Mal verändert wurde (Last-Modified), ob sie gecacht werden darf (Cache-Control) und wenn ja wie lange (Expires), oder welchen Typ ihr Inhalt hat (Content-Type).

Normalerweise sendet der Webserver (in der Regel Apache) automatisch den richtigen Header. Mit PHP kann man den gesendeten Header allerdings beeinflussen. Zu beachten ist, dass kein einziges Zeichen vor der header-Anweisung ausgegeben werden darf! Wenn PHP als CGI installiert ist, gibt es ausserdem einige Einschränkungen, z.B. kann keine Authentifizierung gemacht werden.

Wie der Header aussehen muss, ist in dem *RFC 2616* (siehe: [HTML-RFC]) festgelegt. Er spezifiziert das *HTTP/1.1 Protokoll*. Im Folgenden werden ein paar Möglichkeiten der Anwendung der header-Anweisung gezeigt.

6.4.2 Weiterleiten

Wie bereits oben erwähnt, kann man, neben JavaScript, auch mit PHP den Client auf eine andere Seite weiterleiten. Dies geschieht mit folgender Anweisung:

```
header('Location: absolute_URL');
exit;
```

absolute_URL muss natürlich durch die gewünschte URL ersetzt werden. Es muss nach RFC die absolute URL angegeben werden, auch wenn fast alle Browser eine relative verstehen!

Das *exit* ist nicht unbedingt notwendig, allerdings würde es nichts bringen, nach dem header noch etwas auszugeben, da es sowieso nicht angezeigt wird.

Bei dieser Anweisung sendet Apache automatisch den Statuscode 302.

6.4.3 Nicht gefunden

Wenn man Apache so konfiguriert hat, dass er als Fehlerseite eine PHP-Seite anzeigt, wird als Statuscode 200 (OK) gesendet. Da dies aber unpraktisch ist, weil so z.B. Suchmaschinen die Fehlerseite in ihren Index aufnehmen, sollte man den Statuscode 404 (Not Found) senden, wodurch diese Seite als Fehlerseite erkannt wird. Die Anweisung dazu lautet wie folgt:

```
header('HTTP/1.0 404 Not Found');
```

6.4.4 Authentifizierung

Mit PHP besteht die Möglichkeit, den Browser ein Fenster öffnen zu lassen, in dem Name und Passwort eingetragen werden müssen. Wenn PHP nicht als Modul, sondern als CGI läuft, funktioniert das allerdings nicht.

Es ist eigentlich ganz einfach, eine solche Datei muss vom Prinzip her so aussehen:

```
<?php
if($PHP_AUTH_USER!="Christoph" OR $PHP_AUTH_PW!="Reeg") {
  Header('HTTP/1.1 401 Unauthorized');
  Header('WWW-Authenticate: Basic realm="Top Secret"');
```

```
  echo "Mit Abbrechen kommst Du hier nicht rein. ;-) \n";
  exit;
}
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
  <title>Authentification</title>
</head>
<body>
<h1>Hier ist der Top-Secret Bereich</h1>
<h2>
<?php
  echo "Username: ".$PHP_AUTH_USER." Passwort: ".$PHP_AUTH_PW;
?>
</h2>
</body>
</html>
```

Das Funktionsprinzip ist ganz einfach: Beim ersten Aufruf sind die beiden Variablen ‘PHP_AUTH_USER und PHP_AUTH_PW nicht gesetzt. Dadurch wird der Bereich in der IF-Abfrage bearbeitet. Hier werden die beiden Header zurückgegeben, die den Browser veranlassen, nach Usernamen und Passwort zu fragen. Diese beiden Zeilen müssen fast genau so übernommen werden, damit es funktioniert! Das einzige, was geändert werden darf, ist das *Top Secret*. Der Text danach wird nur dann ausgegeben, wenn jemand bei der Passwortabfrage auf *Abbrechen* klickt (oder, im Falle des Internet Explorers, drei Versuche, sich zu authentifizieren, misslungen sind); dann springt der Webserver nach dem *echo* aus der Datei und der Rest wird nicht mehr ausgegeben. Wenn jedoch jemand das richtige Passwort mit dem richtigen Usernamen eingegeben hat, wird der Bereich in der IF-Abfrage nicht bearbeitet und der Rest der Datei wird abgearbeitet. In unserem Fall wird die Überschrift *Hier ist der Top-Secret Bereich* und die Zeile *Username: Christoph Passwort: Reeg* im HTML-Format ausgegeben.

Es gibt noch ein kleines Sicherheitsproblem bei der ganzen Sache - der Browser behält sich nämlich den Usernamen und das Passwort, so dass die Autoren derjenigen Seiten, die man nach der Passworteingabe abruft, theoretisch das Passwort abfragen könnten. Dies kann man jedoch ganz einfach verhindern, indem man den Browser komplett beendet.

Auf fast dieselbe Weise kann man sich natürlich auch direkt für den Zugriff auf eine Datenbank authentifizieren. Der folgende Quelltext zeigt, wie man dies erreicht:

```
<?php
if ($PHP_AUTH_USER == "" OR !@mysql_connect("localhost",$PHP_AUTH_USER,$PHP_AUTH_PW)) {
  Header('HTTP/1.0 401 Unauthorized');
  Header('WWW-Authenticate: Basic realm="Top Secret"');
  echo "Mit Abbrechen kommst Du hier nicht rein. ;-)\n";
  exit;
}
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
  <title>Authentification</title>
</head>
<body>
<h1>Hier ist der Top-Secret Bereich</h1>
<h2>
<?php
```

```
  echo "Username: ".$PHP_AUTH_USER." Passwort: ".$PHP_AUTH_PW;
?>
</h2>
</body>
</html>
```

Das @-Zeichen vor dem *mysql_connect* hat nichts mit der if-Abfrage zu tun. Es sorgt dafür, dass keine Fehlermeldung beim Aufrauf von *mysql_connect* ausgegeben wird. Die Fehlermeldung würde nicht nur stören, sondern sie würde auch die Passwortabfrage zuverlässig verhindern. Vor dem header-Aufruf darf nichts ausgegeben werden.

Der Bereich in der obigen IF-Abfrage wird genau dann nicht bearbeitet, wenn mittels Benutzername und Passwort eine Verbindung zur Datenbank aufgebaut werden konnte. In jedem anderen Fall wird, wie im ersten Beispiel, abgebrochen und (in diesem Fall) der Text *Mit Abbrechen...* ausgegeben. Um sich Probleme zu ersparen, sollte man obige Bedingung der IF-Anweisung einfach 1:1 ubernehmen, denn diese ist bestens erprobt! :-)

Noch eine Anmerkung zum Schluss: Anstatt der Zeichenkette *HTTP/1.0 401 Unauthorized* kann auch *Status: 401 Unauthorized* benutzt werden. Im Falle des o.g. PHP-CGI-Problems scheint es dann so, als ob die Authentification funktionieren würde (es tritt kein Fehler 500 mehr auf); dies ist jedoch ein Trugschluss, denn trotz allem werden die beiden benötigten Authentifizierungs-Variablen nicht mit den Werten gefüllt, die der Browser nach der Eingabe durch den Benutzer im entsprechenden Dialog zurückliefert.

7 Ausblick/Vorschlag für die Weiterführung

An dieser Stelle möchte ich aus Zeitgründen meinen Teil der Untersuchung abschliessen. Bis zu diesem Punkt wurden die *traditionellen* Technologien erkläutert. Für kleine, übersichtliche Anwendungen reichen diese Technologien völlig aus. Bei grösseren, stetig wachsenden oder komplizierten Applikationen sollte man aber weitere modernere Programmiersprachen/Technologien berücksichtigen, insbesondere Java-Technologien:

- Java: Eine objektorientierte Programmiersprache, stark an C und C++ angelehnt, aber von den Konzepten her abgespeckt zugunsten einfacher und sicherer Benutzung. Hier das für WWW-Anwendungen wichtigste Feature: Java-Compiler erzeugen keinen direkt ausführbaren Maschinencode, sondern Byte-code für die sogenannte *Java Virtual Machine*, einen abstrakten Rechner. Das heißt, Java-Programme und die Byte-codes sind plattformunabhängig. Zum Ausführen der Byte-codes unter einer bestimmten Systemarchitektur verwendet man dann einen systemspezifischen Interpreter. Ein- und dasselbe Programm *läuft* also im Prinzip überall (inklusive Benutzerschnittstellen, Grafikausgabe etc.), kann deshalb netzweit verfügbar sein und von jedem WWW-Browser, der einen entsprechenden Interpreter enthält, ausgeführt werden. Der Interpreter-Modus sorgt dabei dafür, daß die Anwendung nach dem Laden vom Netz ohne Verzögerung startet.

- Java Applets: Java-Programme, die aus Html-Seiten heraus aufgerufen werden können. Es gibt auch noch die sogenannten *stand-alone applications*. Man benötigt aber nur eine triviale Html-Seite und einen java-kompatiblen Browser (z.B. Netscape), um ein Applet auszuführen.

- Java Servlets: Ein Servlet ist eine Java-Klassendatei, die auf einem Web-Server läuft. Ein Servlet erweitert die Funktionalität des Servers ähnlich wie ein CGI-Programm in Perl oder PHP. Ein Servlet nimmt GET- oder POST-Anfragen eines Client entgegen. Es verarbeitet dann auf dem Server die im QueryString enthaltenen Daten. Mehrere Servlets können untereinander kommunizieren. Als Output erzeugt ein Servlet Zeichenketten, die an den Client zurückgeschickt werden.

- Java Server Pages: Eine Java Server Page bietet die Möglichkeit dynamischen Inhalt mit statischem Inhalt einer HTML Seite zu mischen. Hierbei werden die dynamischen Teile vom statischem Code durch gesonderte Tags getrennt. Die meisten dieser Tags beginnen mit <% und enden mit %>. Java Server Pages benötigen eine JSP Engine die eine Umsetzung der JSP-Seite in ein Servlet durchführt und die Ergebnisseite als HTML zurückliefert.

Abb.31 gibt eine Übersicht über die Zusammenhänge der einzelnen Java-Technologien.

Zudem wäre es interessant die verschiedenen Technologien bezüglich ihrer Performance zu vergleichen. Hierzu bietet sich das Grinder Tester-Framework (siehe: The Grinder `http://grinder.sourceforge.net/`) an.

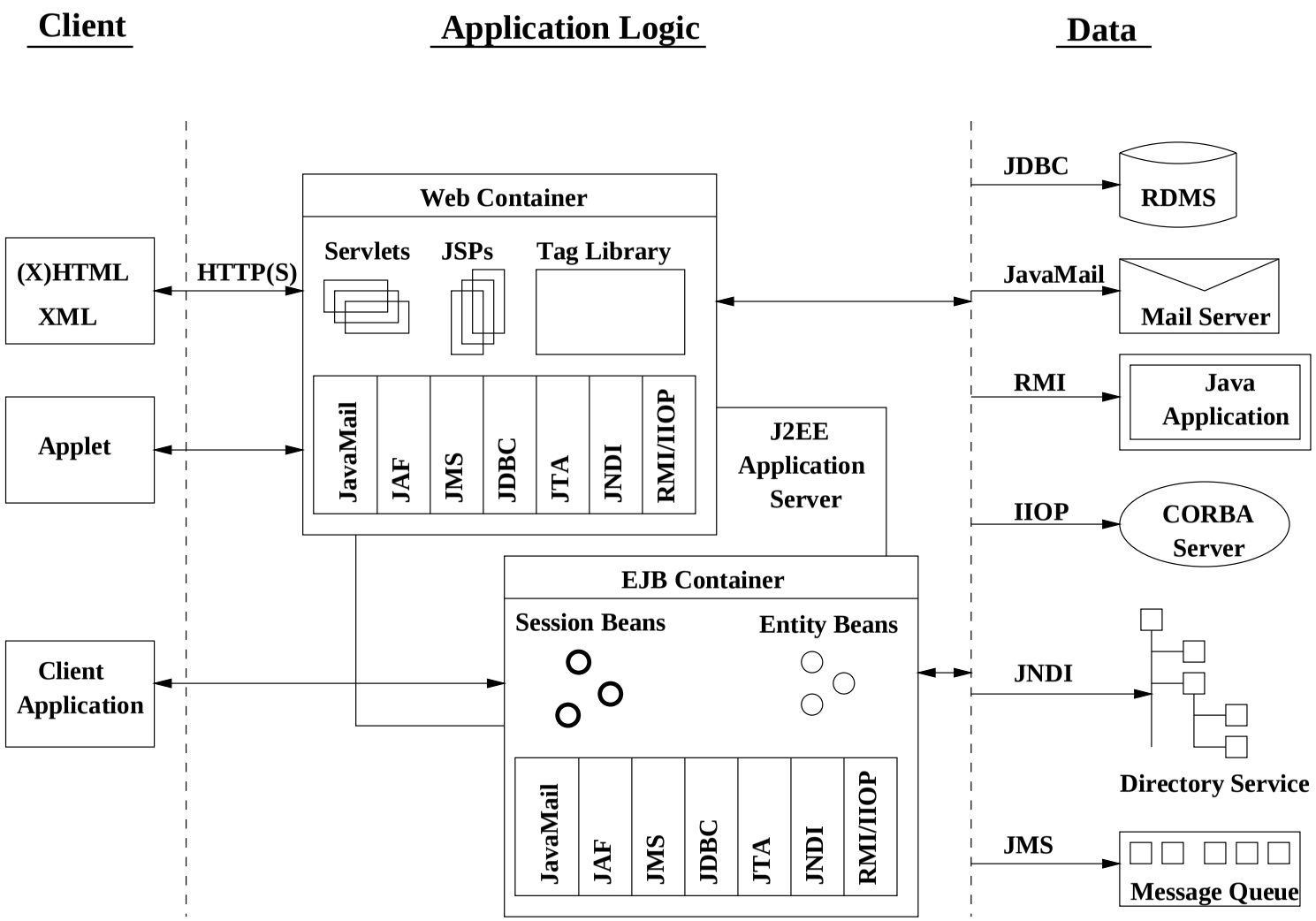

Abbildung 31: Java Overview

8 Schlusswort

8.1 Resultat

Die Problemstellungen, die ich am Anfang aufgeführt habe, wurden in dieser Untersuchung gelöst. Es gibt natürlich immer mehrere Wege eine Lösung zu implementieren, und ich habe hier nur einige von vielen aufgezeigt. Mit Hilfe dieser Dokumentation und den Prototypen kann man leicht eigene Web-Applikationen implementieren, z.B:

- Online Bibliotheksverwaltung.
- Unterstützung einer Vorlesung mit Webapplikationen (z.B. Anbieten von Prüfungen übers Web, Automatische Korrektur der Prüfungen, Feedback an Schüler).
- Erstellung eines Online-Rapportsystems (zur Erfassung der Arbeitsstunden).

Wenn ich diese Arbeit als Uni-Projekt betrachte, dann bin ich damit nicht ganz zufrieden (siehe: 8.3 Was würde ich anders machen). Persönlich habe ich aber enorm viel bei der Realisation der Untersuchnung profitiert. Durch die begangenen Fehler habe ich viel mehr gelernt, als wenn ich immer auf dem richtigen Weg geblieben wäre.

8.2 Was lernte ich

Dieses Projekt lief unbewusst unter dem Motto: der Weg ist das Ziel. D.h. ich hatte während der Realisation dieser Arbeit Einblick in verschiedene Themengebiete und konnte dabei viel lernen:

- Apache Web Server: Zuerst musste ich einen Apache Webserver aufsetzten (download, kompilieren) und konfigurieren (Zugriffskontrolle, htaccess, Virtueller Server, Einbinden von Modulen, etc.).
- HTML: Die Grundlage der Arbeit stellt HTML dar. Es gibt zwar verschiedene Applikationen (z.B. Frontpage, Dreamweaver), mit denen man schnell und einfach Webseiten erstellen kann. Aber zur Erstellung einer CGI-Applikation nützen solche Anwendungen nicht viel, da man dynamisch HTML-Code erzeugen muss. Daher sind gute HTML-Kenntnisse erforderlich. Eine grosse Hilfe zum Erlernen dieser Sparche war die Online-Dokumentation von Stefan Münz ([Mün98]).
- JavaScript: Auch JavaScript war etwas neues für mich. Nun konnte man mit Hilfe von JavaScript auf der Clientseite HTML-Seiten dynamisch gestalten. Mit dieser Programmiersprache habe ich mich am meisten geärgert, da man nicht von einer Basis ausgehen konnte: mancher JavaScript-Code funktioniert auf Netscape 4.0 gut, aber nicht auf Netscape 3.0, und Microsoft's Internet Explorer verhält sich wieder anders als Netscape's Communicator oder als Opera's Browser. Zudem war die Funktionsweise von JavaScript auf den Browser noch plattformabhängig (Unix und Windows). Man musste also immer sehr *defensiv* programmieren und den Code auf den verschieden Browsern und Betriebssystemen testen.
- Perl: Mit Perl erlernte ich eine populäre Skriptsprache. Es ist schon erstaunlich, dass man mit sehr wenigen Programmzeilen so viel erreichen kann. Wenn mich jemand fragte, ob man dies oder jenes programmieren kann, so antwortete ich: "Ja, das gibt einen Vier- oder Fünfzeiler in Perl". Aber bevor ich mein erstes CGI-Skript testen konnte, musste ich den Apache Web Server konfigurieren (cgi aliases). Zudem musste ich Perl installieren. Für die Adressapplikation musste ich noch einige Perl Module (DBI:Data Base Interface, Text::CSV_XS, SQL::Statement und DBD::FileModule) hinzufügen.
- MySql: Ich wollte ja eine Anbindung zu einer richtigen Datenbank übers Web implementieren. Ich entschied mich für MySql und installierte die Datenbank. Nach langem Studium verschiedener Referenzen hatte ich die DB richtig aufgesetzt und konfiguriert. Ich konnte endlich mit dem Perlskript auf die Elemente der MySql-datenbank zugreifen und die Daten beliebig manipulieren.

- PHP: Auch PHP war wieder etwas neues für mich. Nun wurde der PHP Code - ähnlich wie bei JavaScript - mit den entsprechenden PHP Tags direkt in die HTML-Seite eingefügt. Dieser Programmcode wird aber auf der Serverseite ausgeführt.
- Unix: Als Entwicklungs Plattform nutzte ich das Betriebssystem Unix (Solaris). Durch dieses Projekt habe ich dieses gute und stabile OS mit seinen Tools besser kennen gelernt.
- Kleine Helferlein, die ich benützt habe: Latex2Html (Generierung von Html-Code aus Latex-Dokumenten, damit ich die Arbeit auf dem Web bereitstellen konnte), Lyx (Tool zur Generierung von LatexCode), PhpMyAdmin (Mit diesem Tool kann man eine MySql Datenbank einfach und schnell manipulieren), Grinder Tester-Framework (Webapplikation-Performance Tester), NS JavaScript Debugger.

8.3 Was würde ich anders machen

Das Resultat dieser Arbeit könnte man gut als Kursunterlagen benützen, was eigentlich nicht die ursprüngliche Absicht war. Es gibt also einige Punkte, die ich im nachhinein anders angehen würde:

- Ziel definieren: Die Problemstellung muss man genauer beschreiben, andernfalls ist der Spielraum zu gross.
- Das Thema und das Ziel der Arbeit immer vor Augen halten: Damit man nicht auf Abwegen gerät und sich nicht über grosse Umwegen dem Ziel nähert. Während einer Arbeit hat man immer wieder gute Ideen, oder man stösst auf interessante Technologien, welche aber bei der unmittelbaren Realisation der Arbeit nicht sehr hilfreich sind.
- Feedback von anderen Personen (->Betreuer) einholen: Der Kommentar anderer hilft immer wieder sich über die Qualität der Arbeit ein Bild zu machen. Unklarheiten und Überflüssiges werden leichter aufgedeckt. Leider habe ich davon mit meinem Betreuer Markus Lumpe zuwenig Gebrauch gemacht.
- Zielpublikum definieren: Damit man bei der Einführung nicht bei Adam und Eva beginnen muss, sollte man sich ein Zielpublikum auswählen. Dadurch hätte ich mir wohl das Einführungskapitel sparen können.
- Technologien, welche man untersuchen will am Anfang bestimmen: Dies ist wichtig, damit man den Umfang der Arbeit besser abschätzen kann. Dadurch kann man jeder Technologie einen bestimmten Raum einräumen, und der Weg ist vorgezeichnet. Ich habe immer von Kapitel zu Kapitel gearbeitet. Am Ende wollte ich ja noch den Java-Teil in Angriff nehmen, was den Rahmen dieser Untersuchnung bei weitem gesprengt hätte.
- Zeitlimit bestimmen: Für eine Projektarbeit sollte man sich einen zeitlichen Rahmen stecken, damit man bald einmal zu einem Schluss kommt. Man verliert viel Zeit wenn man immer wieder Pausen einlegt uns sich wieder reinhängen muss.

Literatur

[Met99] Web-basierte Anwendungen: Architekturen und Technologien. Autor Dr. Igor Metz, 1999

[Mün98] SELFHTML. Autor Stefan Münz, 1998; (*http*://teamone.de/selfaktuell)

[ARPA1] Defense Advanced Research Projects Agency (DARPA); (*http://www.arpa.mil)*

[ARPA2] History of ARPANET; (*http://www.dei.isep.ipp.pt/docs/arpa.html)*

[Schn94] Internet: Werkzeuge und Dienste, Prof Dr. G. Scheider 1994; (http://www.ask.uni-karlsruhe.de/books/inetbuch/node32.html)

[DO97] Klaus Dopheide. Aufbau und die technische Nutzung eines Intranets. 01.07.1997

[Sche93] Martin Scheller. Internet Resource Guide. Diplomarbeit, Fakultät für Informatik, Universität Karlsruhe, 1993.

[Moc87a] Mockapetris. Domain Names - Concepts and Facilities. RFC 1034, 1987.

[Moc87b] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, 1987.

[IBM95] IBM Corporation:Transmission Control Protocol/Internet Protocol (TCP/IP) (http://www4.ulpgc.es/tutoriales/tcpip/pru/3376fm.htm)

[PR83] J. Postel und J. Reynolds. Telnet Protocol Specification. RFC 854, Mai 1983.

[LR93] Daniel C. Lynch und Marshall T. Rose. Internet System Handbook. Addison-Wesley Publishing Company, 1993.

[Kan91] B. Kantor. BSD Rlogin. RFC 1282, Univ. of Calif. San Diego, Dezember 1991.

[San90] M. Santifaller. TCP/IP und NFS in Theorie und Praxis, Unix in lokalen Netzen. Addison-Wesley, 1990. 2. Nachdruck 1992.

[APA] Apache Http Server Project (apache httpd server project http://www.apache.org)

[GNUPRIV] Das GNU-Handbuch zum Schutze der Privatsphäre (http://www.gnupg.org/gph/de/manual/?1006357932 http://www.gnupg.org/gph/de/manual/?1006357932)

[MP98] R. Maurer. O. Paukstadt. HTML und CGI-Programmierung. 1998.

[HTML-RFC] W3C: HTML 3.2 Reference Specification (http://www.w3.org/TR/REC-html32)

[COO99] HTTP COOKIES - Specification. (http://www.netscape.com/newsref/std/cookie_spec.html)

[STKO99] Stefan Koch. Javascript. 1999.

[HAS99] Hase. Cookies und Datenschutz. 1999.

[LAR98] Lars Eilebrecht. Apache Web-Server. 1998

[HUS98] Robert W. Husted. A HOLISTIC LOOK AT JavaScript, 1998 (http://developer.iplanet.com/viewsource/husted_js/husted_js.html)

[GOO-CS] Danny Goodmans JavaScript Object Roadmap and Compatibility Guide. (http://developer.iplanet.com/viewsource/goodman_roadmap/goodman_roadmap.html)

[GOO-SS] Server-Side JavaScript Object Roadmap and Compatibility Guide. (http://developer.iplanet.com/viewsource/goodman_ssjsrmap/goodman_ssjsrmap.html)

[CSJS-REF] Client-Side JavaScript Reference v1.3 (http://developer.netscape.com/docs/manuals/js/client/jsref)

[CSJS-GUI] Client-Side JavaScript Guide v1.3 (http://developer.netscape.com/docs/manuals/js/client/jsguide)

[SSJS-GUI] Server-Side JavaScript Guide (http://developer.iplanet.com/docs/manuals/js/server/jsguide)

[SSJS-REF] Server-Side JavaScript Reference (http://developer.iplanet.com/docs/manuals/js/server/jsref)

[WCS-PP] Larry Wall, Tom Christiansen, Randal L. Schwartz. Programmieren mit Perl.O'Reilly Verlag 1997.

[REI-PERL] O Reilly Perl Module Doku http://www.oreilly.de/catalog/perlmodger/manpage/index.html

[PE-TU] Perl Tutorial, Nik Silver http://www.comp.leeds.ac.uk/Perl/start.html http://www.comp.leeds.ac.uk/Perl/start.html

[DES-BUN] Programming the Perl DBI, Alligator Descartes & Tim Bunce. 1st Edition February 2000.

[MySQL] MySQL/PHP eine Deutsche Dokumentation zu MySQL und PHP http://ffm.junetz.de/privat/reeg/DSP/

[REI-SQL] MySQL & mSQL aus dem O'Reilly Verlag (englisch) http://www.ora.de/catalog/msql/

[MySQL-Dok] Deutsche Doku zu MySQL mit allen SQL Befehlen http://www.4t2.com/mysql/

[PHP-ADM] phpMyAdmin - Datenbanken mit phpMyAdmin erstellen-http://phpmyadmin.sourceforge.net/ http://phpmyadmin.sourceforge.net/

[HUN-CRA] Jason Hunter & William Crawfird. Java - Servlet Programming.

[GUT-97] Didier Gutacker. WWW Datenbankanbindung http://trumpf-3.rz.uni-mannheim.de/www/sem97s/gutacker/ http://trumpf-3.rz.uni-mannheim.de/www/sem97s/gutacker/

[DOS-RES] Denial of Service (DoS) Attack Resources http://www.denialinfo.com/ http://www.denialinfo.com/

[FASTCGI] FastCGIhttp://www.fastcgi.com http://www.fastcgi.com

[REE-DSP] Christoph Reeg (dsp@reeg.net).Datenbank, MySQL und PHP.22. Juni 2001.http://ffm.junetz.de/members/reeg/DSP/main.html http://ffm.junetz.de/members/reeg/DSP/main.html

[PHP-CEN] PHP CENTER. http://www.php-center.de/ http://www.php-center.de/

[PHP-NET] PHP-NET http://www.php.net http://www.php.net

[POW-RAY] Ryan Powers. Addressbook website. http://www.rpowers.f2s.com/ http://www.rpowers.f2s.com/

[GOM-AST] The Grinder. A pure Java load-testing framework (http://grinder.sourceforge.net/TheGrinder)