

DIPLOMARBEIT

Sessionhandling und Usertracking über HTTP am Beispiel eines Formulargenerators

durchgeführt am
Studiengang für Telekommunikationstechnik und -systeme
der
FH-Salzburg Fachhochschulgesellschaft mbH

vorgelegt von
Stefan Peer



FH SALZBURG

Leiter des Studienganges: Prof. Dipl. Ing. Dr. Gerhard Jöchtl
Betreuer: Kristijan Mihalić

Salzburg, Juni 2001

Vorbemerkungen

Ich habe das Thema „*Sessionhandling und Usertracking über HTTP am Beispiel eines Formulargenerators*“ einerseits aus persönlichem Interesse an der Entwicklung des Internet, im speziellen des Dienstes WWW (in weiterer Folge Web genannt), und andererseits auf Grund von Vorkenntnissen im Bereich der Programmierung von Benutzerumgebungen im Web gewählt.

Im letzten Jahrzehnt hat das Web, oft fälschlicherweise als *das* Internet bezeichnet, einen enormen Aufschwung erlebt. Das Web ist nach E-Mail der wohl wichtigste Dienst des Internet geworden. Mit der Erfindung von HTTP und HTML und den ersten Browsern für grafische Oberflächen ging die Ära von BTX und Gopher zu Ende.

Immer billigere Hard- und Software tat das ihrige dazu, dass heute ca. 75% der US-Amerikaner entweder einen privaten, oder am Arbeitsplatz einen Internetzugang haben. Nach Angaben des statistischen Bundesamtes in Deutschland sind 45% aller deutschen Haushalte mit mindestens einem PC ausgestattet, wobei 1999 bereits 11% der deutschen Haushalte einen privaten Internetanschluss besaßen.

Die Statistik der Internetzugänge pro Einwohner wird in Europa von Schweden (50%), dicht gefolgt von Norwegen und Finnland angeführt.

Neue, einfachere Techniken, wie etwa die Datenübertragung über Stromleitungen, werden in Zukunft dafür sorgen, dass immer mehr Haushalte einen Internetanschluss bekommen. Laut einem Artikel der Zeitschrift E-Media könnte der Startschuss für „*Internet aus der Steckdose*“ im Herbst 2001 fallen. Die ersten erschlossenen Gebiete werden Niederösterreich und Tirol sein [E-M01].

Angesichts der Tatsache, dass sich immer mehr Menschen im virtuellen Raum bewegen und auch ihre Geschäfte dort erledigen ist es klar, dass sich auch die Werbeindustrie an das Medium Internet anpasst. Um Kundenprofile zu gewinnen, werden Surfverhalten und Interessensgebiete der Benutzer erfasst und katalogisiert. Aus diesen Daten können gezielte Online-Bannerwerbungen plziert und somit das Internet als Werbepattform effizienter genutzt werden. Um das Surfverhalten und die Interessensgebiete der einzelnen Benutzer herausfinden

zu können, bedarf es verschiedener Mechanismen des *Usertrackings*¹ und auch des *Sessionhandlings*².

Diese Mechanismen werden aber nicht nur zu Werbezwecken verwendet, sondern auch in der Software-Entwicklung. Webapplikationen kommen ohne Sessions nicht mehr aus. Ein Online-Shop, der sich nicht merken kann, was der Kunde in seinem Einkaufswagen gelegt hat, wird wenig bis gar keine Zustimmung ernen.

Es gibt sehr viele unterschiedliche Techniken, Sessionhandling und Usertracking zu betreiben. Die gängigsten werden in dieser Arbeit behandelt.

¹Usertracking, engl. Benutzerverfolgung/-beobachtung

²Sessionhandlings, engl. Sitzungsverwaltung, Sitzungsmanagement

Details

| | |
|--------------------------|--|
| Name: | Stefan Peer |
| Universität: | FH-Salzburg Fachhochschulgesellschaft mbH |
| Studiengang: | Telekommunikationstechnik und -systeme |
| Titel der Diplomarbeit: | Sessionhandling und Usertracking über HTTP am Beispiel eines Formulargenerators |
| Erstbegutachter: | Kristijan Mihalić |
| Betreuer im Unternehmen: | Andreas Profunser |

Abstract

Since the beginning of the World Wide Web dynamic content has become more and more important. Serverside-Includes, CGI programs and -scripts, application servers and last but not least Java Server Pages and Servlets are platforms for dynamically generated web content. This technique bears a certain interest for business and economy. In order to provide web-based application it is necessary to use stateful protocols. HTTP, the internet's hyper text transfer protocol, however is a stateless protocol, so it requires the means of *Sessionhandling* and/or *Usertracking* to maintain a stateful network-connection. These means are session identification strings, which are transmitted between server and client in many different ways, or sending the whole information with each request and response. They could be stored in Cookies, be encoded into the URL or be send as an HTTP parameter.

Each way has it's own advantages and disadvantages, which are discussed within this work, concerning technical and security matters.

The FormGenerator is an example web-application, which generates databases for online forms. This application uses Sessionhandling, because it has to know different users with different permissions. A user has the possibility to log in. Any logged in user may upload a webform, which is then compiled into a Java DOM object, which is used to set up the database. The complete details of how a session is stored and why Usertracking is needed for the FormGenerator is described in chapter 4.

Inhaltsverzeichnis

| | |
|---|------------|
| Vorbemerkungen | i |
| Details | iii |
| Abstract | iii |
| Abbildungsverzeichnis | ix |
| 1 Grundlegendes | 1 |
| 1.1 Einleitung | 2 |
| 1.1.1 Von statischen Informationen zur Interaktivität | 3 |
| 1.2 HTML | 4 |
| 1.3 HTTP | 5 |
| 1.4 HTTP-Server | 6 |
| 1.5 Proxyserver | 7 |
| 1.6 URL - Uniform Resource Locator | 8 |
| 1.7 QUERY_STRING | 8 |
| 1.8 CGI | 9 |
| 1.9 Java Servlets | 9 |
| 1.10 Multi-Tier-Umgebungen | 11 |
| 1.11 Session | 12 |
| 2 Sessionhandling und Usertracking | 14 |
| 2.1 Sessionhandling | 15 |
| 2.1.1 Sessionhandling & HTTP | 15 |
| 2.1.2 Cookies | 16 |

| | | |
|----------|---|-----------|
| 2.1.3 | URL Kodierung | 20 |
| 2.2 | Usertracking | 23 |
| 2.2.1 | Was ist Usertracking? | 23 |
| 2.2.2 | Ziele | 24 |
| 2.2.3 | Methoden | 24 |
| 2.3 | Proxies und DNS-Caches | 28 |
| 2.4 | Privatsphäre und Anonymität im Web | 28 |
| 2.4.1 | Verschlüsselung | 29 |
| 2.4.2 | Anonymizer | 29 |
| 2.4.3 | Proxyserver und Firewalls | 30 |
| 2.5 | Zusammenfassung | 30 |
| 2.5.1 | Unterschied zwischen Sessionhandling und Usertracking | 30 |
| 2.5.2 | Verwendete Techniken | 31 |
| 2.5.3 | Anonymität und Privatsphäre im Web | 32 |
| 3 | Rechtliche Aspekte von Usertracking-Mechanismen | 33 |
| 3.1 | Datenschutz in Österreich | 34 |
| 3.1.1 | Einleitung | 34 |
| 3.1.2 | Schutzwürdige Daten | 35 |
| 3.2 | Datenschutz und Internet | 38 |
| 3.2.1 | Datenerhebung | 38 |
| 3.2.2 | Naivität der Benutzer | 38 |
| 3.3 | Kritik am DSG 2000 | 39 |
| 3.4 | Zusammenfassung | 40 |
| 4 | Der Formulargenerator | 41 |
| 4.1 | Projekt Formulargenerator | 42 |
| 4.1.1 | Die Idee | 42 |
| 4.2 | Funktionalität | 43 |
| 4.2.1 | Benutzeradministration | 44 |
| 4.2.2 | Formularadministration | 45 |
| 4.2.3 | Formularupload | 46 |

| | | |
|----------|---|-----------|
| 4.2.4 | Formularansicht | 47 |
| 4.3 | Sessionhandling | 47 |
| 4.3.1 | Datenbankverbindungen, Datenbankpools | 47 |
| 4.3.2 | Sessionvariablen | 49 |
| 4.4 | Usertracking | 50 |
| 4.4.1 | Logdatei | 50 |
| 4.5 | Eine Beispielsession | 51 |
| 4.5.1 | Anmeldung | 51 |
| 4.5.2 | Upload | 52 |
| 4.5.3 | Formularverwaltung | 53 |
| 4.5.4 | Abmelden | 53 |
| 4.6 | Programmierung | 54 |
| 4.6.1 | Die Klasse FormGenerator | 55 |
| 4.6.2 | Die Klasse ProcessForm | 58 |
| 4.6.3 | Die Klasse SendMail | 59 |
| 4.6.4 | Die Klasse Database | 60 |
| 4.6.5 | Die Klasse Compiler | 60 |
| 4.7 | Einsatz | 60 |
| 4.7.1 | Auswirkungen | 61 |
| 4.8 | Zusammenfassung | 61 |
| 5 | Schlussfolgerungen | 63 |
| 5.1 | Sessionhandling vs. Usertracking | 64 |
| 5.2 | Sessionhandling in Webanwendungen | 64 |
| 5.2.1 | Benutzerverwaltung | 64 |
| 5.2.2 | Online-Shops | 64 |
| 5.2.3 | Navigation | 65 |
| 5.3 | Methoden des Sessionhandlings | 65 |
| 5.3.1 | Fazit | 67 |
| 5.4 | Usertracking im Web | 67 |
| 5.4.1 | Der Handel mit Daten | 67 |

| | | |
|----------|--------------------------------------|------------|
| 5.4.2 | Methoden des Usertrackings | 67 |
| 5.4.3 | Schutz vor Datensammlern | 68 |
| 5.4.4 | Fazit | 68 |
| 5.5 | Formulargenerator | 69 |
| 5.5.1 | Sessionhandling | 69 |
| 5.5.2 | Usertracking | 69 |
| 5.5.3 | Geplante Auswirkungen | 69 |
| 5.5.4 | Eingetretene Auswirkungen | 70 |
| 5.5.5 | Ausblicke | 70 |
| 5.5.6 | Fazit | 70 |
| A | Java Klassengerüste | A-1 |
| A.1 | Admin | A-1 |
| A.1.1 | Constructor Detail | A-1 |
| A.1.2 | Method Detail | A-1 |
| A.2 | Compiler | A-5 |
| A.2.1 | Constructor Detail | A-5 |
| A.2.2 | Method Detail | A-5 |
| A.3 | Database | A-6 |
| A.3.1 | Constructor Detail | A-6 |
| A.3.2 | Method Detail | A-6 |
| A.4 | DebugInfo | A-8 |
| A.4.1 | Constructor Detail | A-8 |
| A.4.2 | Method Detail | A-8 |
| A.5 | FormClassLoader | A-10 |
| A.5.1 | Method Detail | A-10 |
| A.6 | FormError | A-10 |
| A.6.1 | Constructor Detail | A-10 |
| A.6.2 | Method Detail | A-11 |
| A.7 | FormGenerator | A-13 |
| A.7.1 | Field Detail | A-14 |

| | | |
|----------|--------------------------------------|------------|
| A.7.2 | Constructor Detail | A-15 |
| A.7.3 | Method Detail | A-15 |
| A.8 | Logging | A-17 |
| A.8.1 | Constructor Detail | A-17 |
| A.8.2 | Method Detail | A-17 |
| A.9 | MyDate | A-19 |
| A.9.1 | Constructor Detail | A-19 |
| A.9.2 | Method Detail | A-19 |
| A.10 | ProcessForm | A-21 |
| A.10.1 | Constructor Detail | A-22 |
| A.10.2 | Method Detail | A-22 |
| A.11 | SendMail | A-23 |
| A.11.1 | Constructor Detail | A-23 |
| A.11.2 | Method Detail | A-23 |
| A.12 | Show | A-23 |
| A.12.1 | Constructor Detail | A-23 |
| A.12.2 | Method Detail | A-24 |
| B | CREATE TABLE Definitionen | B-1 |
| B.1 | Tabelle forms | B-1 |
| B.2 | Tabelle logging | B-1 |
| B.3 | Tabelle permissions | B-1 |
| B.4 | Tabelle user | B-1 |
| | Literaturverzeichnis | I |
| | Stichwortverzeichnis | IV |

Abbildungsverzeichnis

| | | |
|-----|---|----|
| 1.1 | ISO-OSI Netzwerkschichtenmodell | 6 |
| 1.2 | Proxyserver als Schnittstelle zwischen LAN und Internet | 7 |
| 1.3 | CGI Aufbau, logisches Modell | 10 |
| 1.4 | Multi-Tier-Umgebung, logisches Modell | 12 |
| 2.1 | Ablauf einer Session, die mit dem HTTP-Statuscode 302 arbeitet. | 21 |
| 2.2 | Usertracking am Beispiel von DoubleClick. | 23 |
| 4.1 | Konzeptionelles Design der Datenbank | 44 |
| 4.2 | Benutzeradministration aus der Sicht eines Administrators | 45 |
| 4.3 | Formularverwaltung aus der Sicht eines Administrators | 46 |
| 4.4 | Datenfluss-Diagramm des Formulargenerators | 51 |
| 4.5 | Anmeldung am Formulargenerator als Administrator | 52 |
| 4.6 | UML-Diagramm des Formulargenerators | 54 |
| 4.7 | Anzeigen eines Formulars | 59 |

Kapitel 1

Grundlegendes

Dieses Kapitel widmet sich der Einführung in die Thematik WWW. Es werden Grundbegriffe wie HTTP, CGI, URL, Proxyserver und Session erklärt, die zum Verständnis der restlichen Diplomarbeit unbedingt notwendig sind.

1.1 Einleitung

Die Entwicklung des WWW begann 1989 in Genf, als ein Informatiker namens Tim Berners-Lee mit einigen Kollegen zusammen die Möglichkeit, wissenschaftliche Publikationen im Internet bereitzustellen, schaffen wollte. Diese Dokumente sollten simple Textformatierungen, Grafiken und, wenn möglich, Hypertextfunktionalität beinhalten, um auf Referenzen, ähnliche Dokumente, etc. verweisen zu können. Daraus entstand einerseits HTTP¹ als Übertragungs- und andererseits HTML² als Darstellungskomponente.

Das HTTP liegt heute in der Version 1.1 vor. HTML hat sich seit 1989 sehr stark verändert; die Version 4.01 ist die zur Zeit aktuelle. Im Laufe der Zeit kamen immer neue HTML-Tags hinzu, wie zum Beispiel das Tabellen-Element `<table> . . . </table>`, wodurch genauere Anordnungen verschiedener Elemente möglich wurden. Framesets sind erst seit der HTML Version 4.0 im Standard enthalten. [W3C01]

Erweiterungen wie JavaScript, das ursprünglich von Netscape Communications Corp. entwickelt wurde, und CSS³ ermöglichen aufwendige Web-Designs und Navigationselemente. Plugins der Browser für Audio, Video oder Flash bieten zusätzlich multimediale Unterstützung. Java Applets, eine Entwicklung von Sun Corp., wurden 1996 eingeführt und boten die Möglichkeit der Interaktivität über HTTP. Vor allem Animationen und kleine Spielereien fanden mit Applets ihren Weg auf viele private Homepages. Große Firmen wie zum Beispiel IBM verwendeten (und verwenden heute noch) Applets als Managementkonsole von verteilter Software. So lassen sich zum Beispiel Datenbank- und Applicationserver über Applets warten und konfigurieren.

Marc Andreessen entwickelte 1993 den ersten WWW-Browser für grafische Oberflächen: Mosaic [NCS97]. Später war er Mitbegründer der Firma Netscape

¹HTTP - HyperText Transfer Protocol

²HTML - HyperText Markup Language

³CSS - Cascaded Style Sheets

Communications Corp., dessen Browser NetscapeTM lange Zeit Marktführer blieb. Vor ca. zwei bis drei Jahren wurde diese Marktführerschaft von Microsofts Internet ExplorerTM abgelöst.

1.1.1 Von statischen Informationen zur Interaktivität

In den letzten Jahren entwickelte sich das Web weg von statischen Informationsseiten hin zu dynamische, auf Profile abgestimmte Benutzerumgebungen. In einer solchen Umgebung sind benutzerbezogene Daten auf einem Server gespeichert, die zur Informationsfilterung und zur optischen Darstellung dieser Informationen dienen. Deshalb wurden Mechanismen kreiert, die ein Sessionmanagement für Verbindungen über HTTP ermöglichen. Eine Session (siehe Abschnitt 1.11, Seite 12) wird mit der ersten Anfrage eines Benutzers initiiert. Durch eine Anmeldung am Server mit Benutzernamen und Passwort können am Server gespeicherte, benutzerbezogene Daten in die Session mit einbezogen werden.

Als Beispiel hierfür sei hier der bekannte Internetshop Amazon genannt, bei dem getätigte Käufe gespeichert und auf Basis der so erhobenen Informationen Vorschläge für Produkte gemacht werden. Amazon versucht anhand von Buchtiteln und Musikstilen den Geschmack der Kunden zu erraten. Dieses Feature funktioniert aber nur, wenn sich der Benutzer bei Amazon mit seiner Kennung und seinem Passwort angemeldet hat.

Es existieren noch weitere Beispiele für Benutzerumgebungen im Netz. Online-Communities wie z.B. GMX (www.gmx.net) oder SMS.AT (www.sms.at) speichern je ein frei konfigurierbares Profil für jeden Benutzer, das mit einem Login aktiv wird. Im Profil von SMS.AT sind unter anderem ein persönliches Telefonbuch mit Adressen und Informationen über sich selbst vorhanden, die jederzeit geändert werden können. Des Weiteren werden Informationen zur Darstellung

der Informationsleiste, Signaturen etc. gespeichert. Das Angebot von GMX umfasst in etwa denselben Umfang, Unterschiede bestehen in den Anwendungsgebieten. GMX bietet gratis E-Mail Adressen und speichert im Profil einige nützliche Daten, die in Zusammenhang mit E-Mails gebraucht werden können, wie ein Adressbuch, Farbeinstellungen bei HTML-Mails, Filterregeln und AntiSpam-Filter.

1.2 HTML

HTML ist eine Layout-Sprache mit Hypertextfunktionalität. Es ist ein Abkömmling des in den 60ern von IBM entwickelten SGML⁴, und arbeitet mit sogenannten Tags. Ein solches Tag wird mit einem `<TAGNAME ATTRIBUTE>` eingeleitet und hat einen Namen und verschiedene Attribute. Ein `</TAGNAME>` schließt es wieder. Dazwischen stehen Daten, die zu diesem Tag gehören. Bei HTML handelt es sich hierbei ausschließlich um Textdaten.

Ein Beispiel:

```
<a href="mylink.html">Mein Link</a>
```

Das Anchor-Tag `a` steht in HTML für einen Verweis auf eine andere HTML-Seite oder einen seiteninternen Anker (bzw. es definiert einen solchen). Das Attribut `href` enthält das Ziel des Verweises, in diesem Beispiel den URL `mylink.html` (siehe auch Abschnitt 1.6, Seite 8).

In einem Browser wird der Text "Mein Link" als Link dargestellt und lädt bei Aktivierung des Links (zum Beispiel durch einen Klick mit der Maus) den angegebenen URL `mylink.html`.

⁴SGML - Standard Generalized Markup Language

Ein Verweis kann auch auf eine Bilddatei, oder ausführbaren Code zeigen. Ein Bild wird mit dem Image-Tag `` geladen, wobei der Browser für jedes Bild eine eigene Anfrage an den Server stellt. Ausführbarer Code kann entweder eine serverseitige Anwendung, wie zum Beispiel ein CGI-Programm oder ein Java Servlet, oder ein herunterladbares Programm, das in einem Plugin gestartet wird, wie zum Beispiel ein Java Applet oder eine Flash-Animation.

Die Organisation W3C⁵ kümmert sich um die Pflege von HTML [W3C01]. Eine Anleitung und Erklärung aller HTML-Tags findet sich auch auf der Website Selfhtml [Mü98].

1.3 HTTP

Das HTTP⁶ ist ein Übertragungsprotokoll der OSI-Schicht 7 (Anwendungsschicht, Application Layer) und liegt zur Zeit in der Version 1.1 vor. Ein HTTP-Server liefert die Daten, die der Client in Form eines URL (siehe Abschnitt 1.6, Seite 8) angefordert hat. Nachdem der Server diese Daten geschickt hat, wird die Verbindung zum Client abgebrochen. Bei jedem Zugriff wird eine neue Verbindung auf- und gleich wieder abgebaut. Dadurch ist es mit HTTP alleine unmöglich, verbindungsorientiert zu arbeiten.

Das HTTP besteht aus zwei Teilen, einem HTTP-Header und einem HTTP-Body. Der HTTP-Header enthält Informationen wie URL, Größe, MIME⁷-Typ, Cookies und andere, protokollspezifische Informationen. Des weiteren gibt es verschiedene Statusmeldungen, die mit Nummern von 100 aufwärts (bis 505 bei HTTP 1.1) bezeichnet sind. Genaue Definitionen der Statusmeldungen sind in [FGM⁺99], Kapitel 10, nachzulesen.

⁵W3C - World Wide Web Consortium

⁶HTTP - HyperText Transfer Protocol

⁷MIME - Multipurpose Internet Mail Extensions

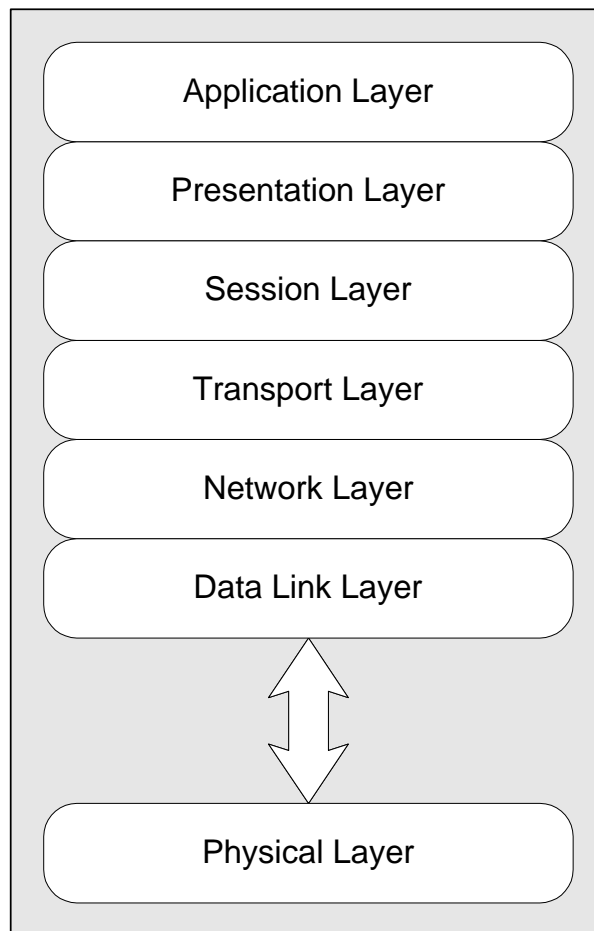


Abbildung 1.1: ISO-OSI Netzwerkschichtenmodell

1.4 HTTP-Server

Ein HTTP-Server (oder auch Web-Server) ist ein Programm auf einem Rechner bzw. Rechnerverbund, der HTTP-Anfragen entgegennehmen und beantworten kann. Auf einem solchen Server sind Sites⁸ gespeichert, die über URLs adressiert werden. Außerdem führt ein CGI-fähiger Web-Server CGI-Programme aus und liefert deren Ausgaben an den Client weiter.

⁸Site - Bezeichnung für eine Menge von HTML-Seiten, die einem Gesamtkonzept unterliegen.

Vielfach werden Web-Server in sogenannte *Multi-Tier-Umgebungen* (siehe auch Abschnitt 1.10, Seite 11) integriert und dienen dort als Schnittstelle zum World Wide Web.

1.5 Proxyserver

Ein Proxyserver ist ein Server, der als Vermittler und als Zwischenspeicher fungiert. Bei HTTP-Proxyservern werden Anfragen weitergeleitet, wobei der Zielserver nicht mit dem eigentlichen Client kommuniziert, sondern ausschließlich mit dem Proxyserver. Dieser leitet alle Antworten an den Client weiter.

Eine weitere Aufgabe von Proxyservern ist die Zwischenspeicherung von Daten, um den Netzwerkverkehr zu begrenzen. Wird dieselbe Anfrage innerhalb einer definierten Zeitspanne getätigt, so erfolgt kein neuerlicher Zugriff auf den Webserver, sondern der Proxyserver beliefert den Client mit den zwischengespeicherten Daten.

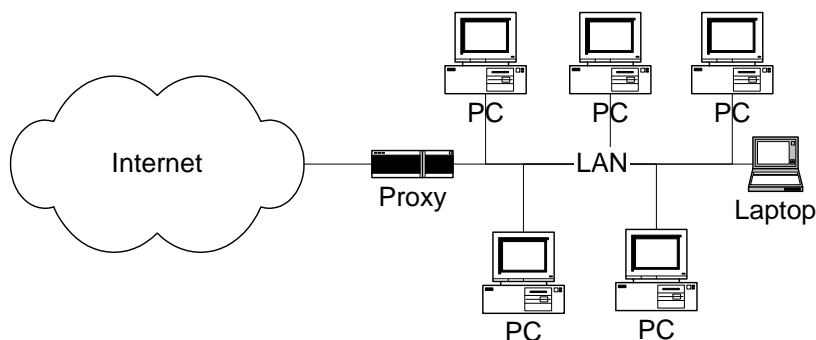


Abbildung 1.2: Proxyserver als Schnittstelle zwischen LAN und Internet

Ein Proxyserver hat auch den Vorteil, dass er als Schnittstelle zwischen LANs⁹ und dem Internet fungieren kann. So können öffentliche IP-Adressen gespart werden.

⁹LAN - Local Area Network, lokales Netzwerk

1.6 URL - Uniform Resource Locator

Ein URL teilt einem Serverprogramm mit, wo sich eine gewünschte Resource befindet. Für HTTP ist ein solcher URL wie folgt aufgebaut:

```
http://SERVERNAME[:PORT][ /PFAD ][ ?QUERY_STRING ]
```

`http://` gibt das Protokoll an, `SERVERNAME` ist der Bezeichner für den Rechner in der Form eines Domännennamens¹⁰, zum Beispiel `http://www.w3c.org`. Fehlt die Domäne, so wird der Rechner im LAN gesucht. Der optionale Parameter `PORT` steht für die Portnummer (Standard ist Port 80). `PFAD` gibt den Pfad zur gewünschten Datei an, wobei dieser nicht unbedingt mit dem Pfad im Dateisystem des Servers übereinstimmen muss. Meistens gibt es ein Serververzeichnis, das irgendwo im Dateisystem liegt, aber als Basisverzeichnis (root-directory) für den Server fungiert. Der letzte Parameter `QUERY_STRING` kann Informationen für serverseitige Scripts und Programme beinhalten (siehe auch Abschnitt 1.7). Die genaue Spezifikation eines URL findet man in [BMM94].

1.7 QUERY_STRING

Ein `QUERY_STRING` ist eine Zeichenkette, mit deren Hilfe Informationen von einer HTML-Seite zur nächsten weitergereicht werden können. Eine solche Zeichenkette ist Bestandteil des URL und wird durch ein Fragezeichen eingeleitet. Ein `QUERY_STRING` ist wie folgt aufgebaut:

```
VAR_1=WERT_1&VAR_2=WERT_2&...&VAR_N=WERT_N
```

Die so übermittelten Informationen werden hauptsächlich für CGI¹¹-Scripts (siehe auch Abschnitt 1.8) und andere serverseitige Programme verwendet, die

¹⁰fully qualified domain name: `host.domain.top-level-domain`

¹¹CGI - Common Gateway Interface

eine Benutzereingabe erfordern. Ein `QUERY_STRING` wird nur dann übermittelt, wenn die HTTP-Methode `GET` verwendet wird. Bei der Methode `POST` stehen diese Werte im HTTP-Header und sind unsichtbar für Benutzer. `POST` eignet sich daher für die Übertragung von Daten, die nicht sofort ersichtlich sein sollten, wie zum Beispiel Passwörter.

1.8 CGI

CGI ist ein Akronym für Common Gateway Interface. Dahinter verbirgt sich eine Schnittstelle zu einem Programm, das am Server ausgeführt wird und sämtliche Ausgaben über HTTP an den Benutzer zurückliefert (siehe Abbildung 1.3).

CGI war lange Zeit die einzige Möglichkeit Interaktivität in HTTP einzubinden. Über CGI-Scripts lassen sich Datenbankverbindungen oder Verbindungen zu anderen Rechnern herstellen (z.B. über TELNET), eingegebene Daten verarbeiten oder speichern (Gästebücher, Counter, ...), etc. Mit CGIs wurde der erste Schritt in Richtung Webapplikation unternommen.

1.9 Java Servlets

Sun Microsystems entwickelte 1998 die Java Servlet API, die als Ersatz für CGI gedacht war. Die API besteht aus einem Set von Java Klassen, die das Servlet-Modell abbilden. Das Modell umfasst ein `HttpServlet`, das die 3 Basismethoden `init()`, `doGet()` und `doPost()` neben vielen anderen enthält. Mit diesen Methoden lassen sich HTTP-Anfragen über `GET` bzw. `POST` verarbeiten, die Methode `init()` setzt die Startparameter des Servlets.

Der große Vorteil von Servlets gegenüber CGI-Programmen ist die Geschwindigkeit. CGI-Programme müssen bei jedem Zugriff gestartet werden. Das Starten

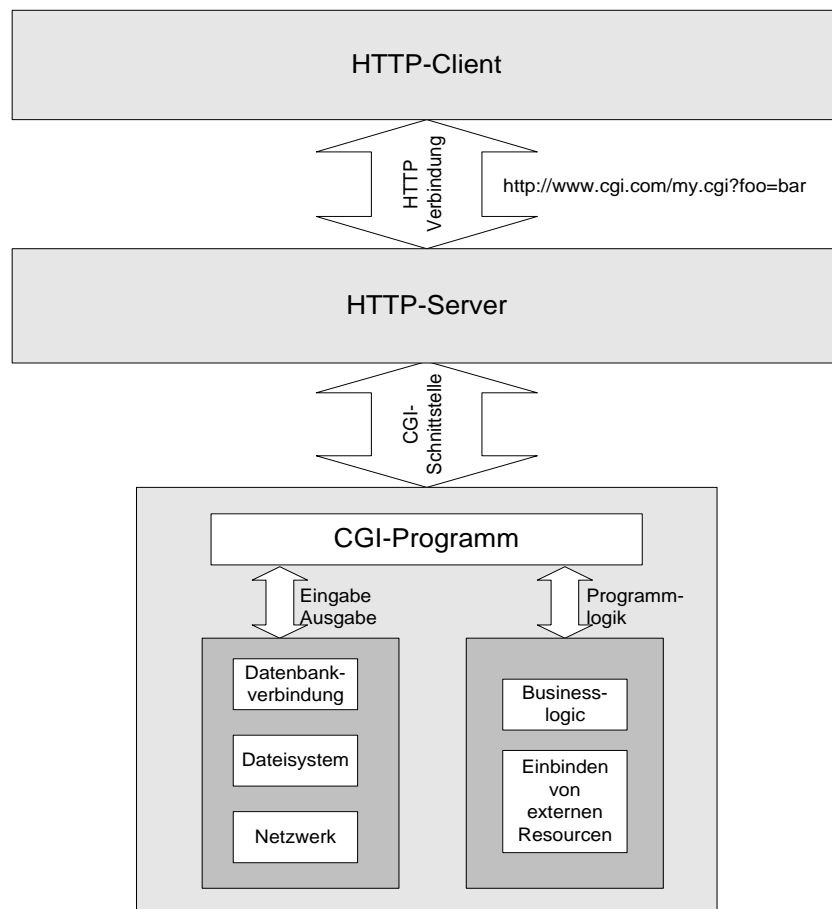


Abbildung 1.3: CGI Aufbau, logisches Modell

eines Prozesses benötigt viel Zeit. Servlets können mit dem Starten der Java Servlet Engine bereits instantiiert werden und laufen in einem Thread. Falls Servlets erst beim Aufruf über HTTP instantiiert werden, so wird kein eigener Prozess, sondern nur ein Thread gestartet, was weniger Prozessorzeit in Anspruch nimmt. Des weiteren hat Java gegenüber Skriptsprachen wie Perl den Vorteil, dass der Code nicht bei jedem Zugriff neu übersetzt bzw. interpretiert wird, was wiederum einiges an Zeit spart.

Des weiteren besitzen Servlets alle Vorteile, die die Programmiersprache Java mit sich bringt: Plattformunabhängigkeit, mächtige Entwicklungsumgebungen und gute Dokumentation bzw. Dokumentationswerkzeuge sind nur einige der vielen.

Der Einsatz von Java Servlets bringt natürlich auch Schwierigkeiten mit sich, bzw. sind Java Servlets für manche Aufgabenstellungen eine ungeeignete Lösung. Es macht wenig Sinn, Servletlösungen für kleine Programme zu verwenden, da auf Grund der API z.B. für Zugriffszähler oder Guestbooks sehr viel Overhead entsteht, der die Entwicklungszeit verlängert und die damit verbundenen Kosten in die Höhe treibt. Ausserdem werden sehr viele solcher Programme bereits gratis zum Download angeboten.

Für manche kleine Datenbankanwendungen ist die Skriptsprache PHP besser geeignet, da schneller Ergebnisse erzielt werden können.

Dafür bringt die objektorientierte Programmiersprache Java leichtere Wartbarkeit und Erweiterbarkeit mit sich, was die Kosten der Wartung verringert.

1.10 Multi-Tier-Umgebungen

Multi-Tier heißt vielschichtig und bedeutet, dass mehrere verschiedene Server eine Plattform bilden, die zusammen einen Zweck erfüllen. So werden häufig HTTP-Server zusammen mit Datenbanken und eventuell noch weiteren Servern zu einem Serververbund geschlossen. Ein solcher Verbund kann auch als Applikationsserver bezeichnet werden.

Die Abbildung 1.4 zeigt ein solches Modell. Ein Client steuert einen HTTP-Server, der sich die geforderten Ressourcen teilweise von anderen Servern beschafft. Online-Shops haben oft ihre Logistik-Systeme in solche Umgebungen eingebunden, um Lagerbestand und voraussichtliche Lieferzeiten im Web verfügbar zu machen.

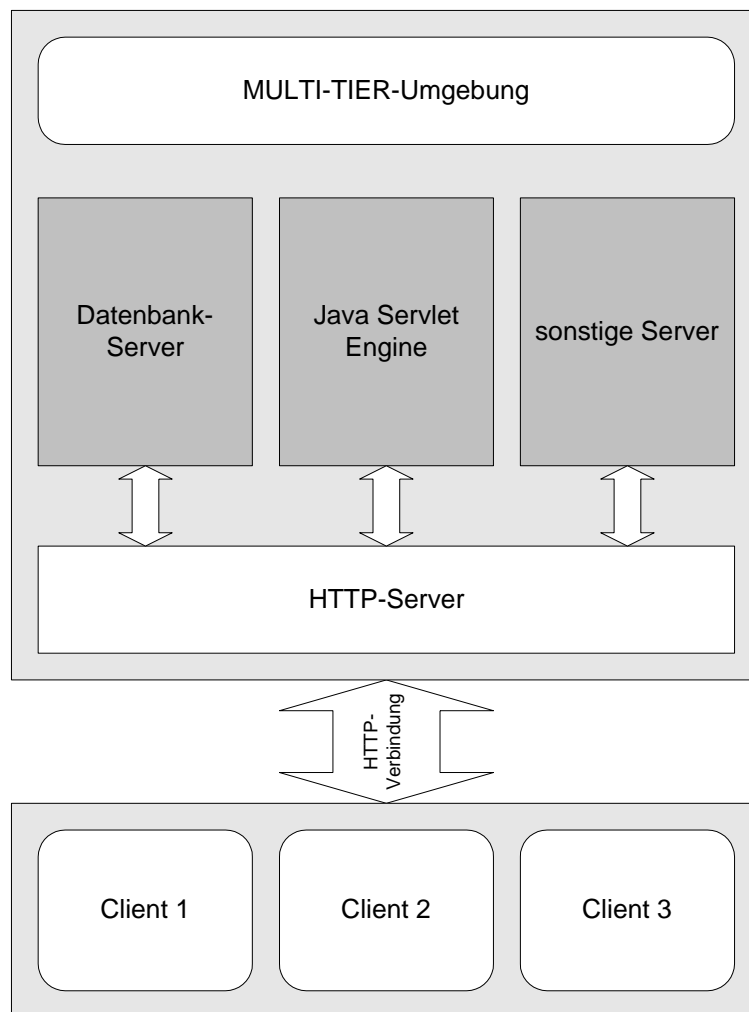


Abbildung 1.4: Multi-Tier-Umgebung, logisches Modell

1.11 Session

Eine Session¹² ist der über mehrere Anfragen verteilte Austausch von Daten zwischen Client und Server. Sie beginnt mit der ersten Anfrage und endet nach einem Timeout, während dem keine weiteren Forderungen vom Client gestellt werden oder nach einem Abbruch durch einen der Teilnehmer. Eine Session hat folgende Eigenschaften [KM00]:

1. Jede Session hat einen Anfang und ein Ende.

¹²Session, engl. Sitzung

2. Jede Session hat eine relativ kurze Lebenszeit.
3. Der Client und der Server können die Session beenden.
4. Eine Session impliziert den Austausch von Statusinformationen.

Zu einem Zustandekommen muss einer Session der Datenaustausch von Statusinformationen zwischen Server und Client möglich sein. Diese Informationen werden entweder client- oder serverseitig gespeichert und bei jedem Zugriff mitgesendet.

Wenn die Daten am Server gespeichert werden, muss der Server in der Lage sein, den Client wiederzuerkennen. Da HTTP ein verbindungsloses Übertragungsprotokoll ist, wurden zusätzlich Mechanismen kreiert, die eine durchgehende Verbindung über mehrere Anfragen ermöglichen (siehe Abschnitt 2.1, Seite 15).

Kapitel 2

Sessionhandling und Usertracking

Dieses Kapitel erläutert die Unterschiede zwischen Sessionhandling und Usertracking, sowie deren Aufgaben und Anwendungsgebiete.

Es werden mehrere Methoden des Sessionhandling und des Usertracking mit unterschiedlichen Vor- und Nachteilen vorgestellt, ohne konkret auf das Anwendungsbeispiel *Formulargenerator* einzugehen. Es dient mehr dem Überblick, um die derzeit vorhandenen Techniken des Sessionhandlings und Usertrackings kennenzulernen.

2.1 Sessionhandling

Sessionhandling befasst sich mit dem Management einer *Session*. Dabei werden Sessiondaten gespeichert, die Auskunft über den Status der Session geben. Diese Daten sind allen Teilnehmern bekannt. Anhand von eindeutigen Kennungen werden diese wiedererkannt. Die Verwaltung der Session erfolgt meistens am Server, der die notwendigen Ressourcen dafür haben muss. Solche Ressourcen sind sowohl hardware- als auch softwaretechnischer Natur. Die Hardware muss den Overhead, den eine Session mit sich bringt, verkraften können. In der Software muss ein Sessionmanagement (Wiedererkennungsmechanismen für die Teilnehmer, Zwischenspeicherung von Daten, Protokolle für Datenbanken, etc.) implementiert sein.

2.1.1 Sessionhandling & HTTP

Am Beginn einer Session wird eine eindeutige Kennung, eine sogenannte SID¹ erzeugt, die bei jeder Anfrage vom Client an den Server mitgeschickt wird. Anhand der SID werden dem Client zwischengespeicherte Daten zugeordnet - z.B. ein Warenkorb, in dem alle bisher ausgewählten Waren enthalten sind. Eine SID wird mit Hilfe von kryptologischen Mitteln erzeugt. Eine Zufallszahl ist Ausgangsbasis für die Erzeugung eines Schlüssels, der lang genug ist, um die Möglichkeit eines Duplikates desselben Schlüssels innerhalb der Dauer einer Session bzw. über einen längeren Zeitraum auszuschließen.

Um die SID zu übertragen, werden verschiedene Methoden verwendet:

¹SID - Session IDentification

2.1.2 Cookies

Eine Methode SIDs mitzuschicken sind Cookies, kleine Datenkekse, die nachträglich Bestandteil des HTTP-Headers wurden. Ein Cookie wird vom Server an den Client geschickt, in dem im HTTP Response-Header ein Set-Cookie Kommando geschickt wird, gefolgt von einer Reihe von Attributen. Diese Attribute sind Werte, die über den Status der Session Auskunft geben. Ein Cookie ist wie folgt aufgebaut:

Cookie Version 1:

```
set-cookie = "Set-Cookie:" cookies
  cookies = 1#cookie
    cookie = NAME "=" VALUE * (";" cookie-av)
      NAME = attr
      VALUE = value
  cookie-av = "Comment" "=" value
    "Domain" "=" value
    "Max-Age" "=" value
    "Path" "=" value
    "Secure"
    "Version" "=" 1*DIGIT
```

Normalerweise enthält ein Response-Header den Set-Cookie-Befehl nur einmal; alle Cookies werden, mit Semikolons getrennt, hintereinander geschickt. Jedes Cookie beginnt mit einem NAME=VALUE-Paar. Es können danach beliebig viele, durch ein Semikolon getrennte Attribut-Wert-Paare folgen. (vgl. [KM97, KM00]).

```
NAME=attr; VALUE=value
```

Dieses Element darf in keinem Cookie fehlen, da es den Namen des Cookies bestimmt. Der Inhalt dieses Cookies ist nur für den Herkunftsserver relevant, wobei der Inhalt unter Umständen für andere Server lesbar ist, die den Set-Cookie-Header auswerten.

`Comment=comment`

Dieses Feld ist optional. Weil ein Cookie private Informationen über den Benutzer, wie zum Beispiel Kreditkartennummern, enthalten kann, sollte dieses Feld eine kurze Beschreibung der im Cookie gespeicherten Daten enthalten.

`Domain=domain`

Dieses Feld ist optional und spezifiziert die Domäne, für die das Cookie gilt. Beginnt eine Domäne mit einem Punkt (zum Beispiel `.doubleclick.net`), so stellt sie ein Netzwerk dar. Ansonsten wird ein einzelner Rechner adressiert (zum Beispiel `www.doubleclick.net`, der Rechner `www` der Domäne `.doubleclick.net`).

`Max-Age= Δ -seconds`

Dieses Feld ist optional und definiert die maximale Gültigkeitsdauer des Cookies in Sekunden. Nach Ablauf dieser Zeit sollte das Cookie vom Client verworfen werden.

`Path=path`

Dieses Feld ist optional und definiert ein Subset von URLs, zu denen das Cookie gehört.

`Secure`

Dieses Feld ist optional und bedeutet, dass der Client eine sichere Verbindung, wie z.B. TLS² [AD99] oder S-HTTP³ [RS99] zum Herkunftsserver herstellen muss, um das Cookie wieder an den Server zu senden.

`Version=version`

Dieses Feld enthält das Versionsattribut, eine Integer-Zahl, die die verwendete Cookie-Version widerspiegelt.

²TLS - Transport Layer Security

³S-HTTP - Secure-HyperText Transfer Protocol

Cookie Version 2:

```

set-cookie = "Set-Cookie:2" Cookies
  cookies = 1#cookie
  cookie = NAME "=" VALUE * (";" cookie-av)
    NAME = attr
    VALUE = value
  cookie-av = "Comment" "=" value
    "CommentURL" "=" <"> HTTP_URL <">
    "Discard"
    "Domain" "=" value
    "Max-Age" "=" value
    "Path" "=" value
    "Port" [ "=" <"> portlist <"> ]
    "Secure"
    "Version" "=" 1*DIGIT
  portlist = 1#portnum
  portnum = 1*DIGIT

```

Die Cookie-Version 2 ist eine Weiterentwicklung der 1. Version. Es wurden fünf zusätzliche Felder definiert - CommentURL, Discard, Port, portlist und portnum.

CommentURL="HTTP_URL"

Dieses Feld ist optional und dient zur Beschreibung des Zwecks des Cookies für den angegebenen URL.

Discard

Dieses Feld ist optional und teilt dem Client mit, bei Beendigung des User-Agents das Cookie zu verwerfen.

Port[="portlist"]

Dieses Feld ist optional und schränkt die Gültigkeit des Cookies auf die angegebenen Ports ein.

portlist

Dieses optionale Feld enthält eine Liste aller erlaubten Ports.

portnum

Dieses Feld enthält die Portnummer des erlaubten Ports.

2.1.2.1 Netscape's Version der Cookies

Netscape hat eine Cookie-Spezifikationen herausgebracht, die sich nur marginal von den in den RFCs beschriebenen Cookies unterscheidet. Der HTTP-Header für ein Netscape-konformes Cookie sieht folgendermaßen aus:

```
set-cookie = "Set-Cookie:"  
    cookie = NAME "=" VALUE * (";" cookie-av)  
    cookie-av = "expires" "=" DATE  
               "Path" "=" value  
               "Domain" "=" DOMAIN_NAME ]  
               "Secure"
```

Auch Netscape bietet die Möglichkeit, mehrere Cookies in nur einem HTTP-Header zu schicken.

Das Attribut `expires` entspricht dem Attribut `Max-Age`, wobei hier nicht Dauer in Sekunden, sondern der Zeitpunkt in Millisekunden seit 1970 angegeben wird, zu dem das Cookie ungültig wird. (vgl. [Net95])

2.1.2.2 Vorteile

Die Vorteile von Cookies liegen auf der Hand: Es ist relativ einfach, eine verbindungsorientierte Kommunikation zwischen Server und Client aufzubauen, wobei Daten clientseitig gespeichert werden können.

Sie sind standardisiert und werden somit von den gängigsten Browsern unterstützt.

2.1.2.3 Nachteile

Ein großes Problem liegt in den Sicherheitsmängeln der Browser, die es ermöglichen, Cookies auch von fremden Domänen auszuspionieren. Bei vielen Online-Shops werden Kreditkartennummern und Passwörter in Cookies gespeichert, damit der Kunde diese Daten nicht jedesmal erneut eingeben muss. Doch diese Bequemlichkeit hat ihren Preis. Kreditkartennummern und Passwörter zählen zu den begehrtesten Informationen für Hacker und sogenannte Script-Kiddies⁴. Seit geraumer Zeit bieten Web-Browser Funktionen zum Deaktivieren von Cookies an, um die Entscheidung, ob Daten in Cookies oder auf dem Server gespeichert werden sollen, dem Benutzer zu überlassen.

2.1.3 URL Kodierung

Es gibt mehrere Ansätze, einen URL so zu kodieren, dass eine serverseitige Wiedererkennung möglich wird.

Die wohl einfachste Lösung ist, eine SID als `QUERY_STRING` (siehe Abschnitt 1.7, Seite 8) an den URL zu hängen und diese SID serverseitig auszuwerten. Viele Web-Programmiersprachen wie ASP, PHP und Perl bieten Session-Bibliotheken an, die genau nach diesem Muster verfahren. Für PHP [PHP00] und Perl [PER00] existieren jeweils Bibliotheksfunktionen, die die SID entweder als Cookie speichern oder, sofern keine Cookies akzeptiert werden, einen Fallback-Mechanismus bereitstellen, der die SID als Parameter übergibt.

Die Java Servlet API [SUN00b] bietet die Methode `encodeURL(String url)` in dem Interface `HttpServletResponse` an, die an eine URL den Parameter `jsessionId` anhängt. Ein so kodierter URL sieht wie folgt aus:

```
http://server.com/pfadangabe/Servlet;jsessionId=<SID>?a=b
```

⁴Script-Kiddies sind jener Personenkreis, die Programme von Hackern verwenden, um in fremde Rechner einzudringen. Kiddies deswegen, weil ihr Alter oft zwischen 14 und 20 Jahren liegt.

Diese Methode erfordert, dass der URL vor dem Bearbeiten durch den HTTP-Server zerlegt und danach weitergereicht wird. Diese Aufgabe übernimmt die Java Servlet Engine.

Ein weiterer Ansatz ist das Verwenden des HTTP Statuscodes 302. Dieser besagt, dass eine Resource temporär unter einem anderen URL zu finden ist. Der Trick hierbei besteht darin, dass der Webserver die Request entgegennimmt, eine SID generiert (sofern noch keine vorhanden ist) und diese in den Redirect-URL integriert.

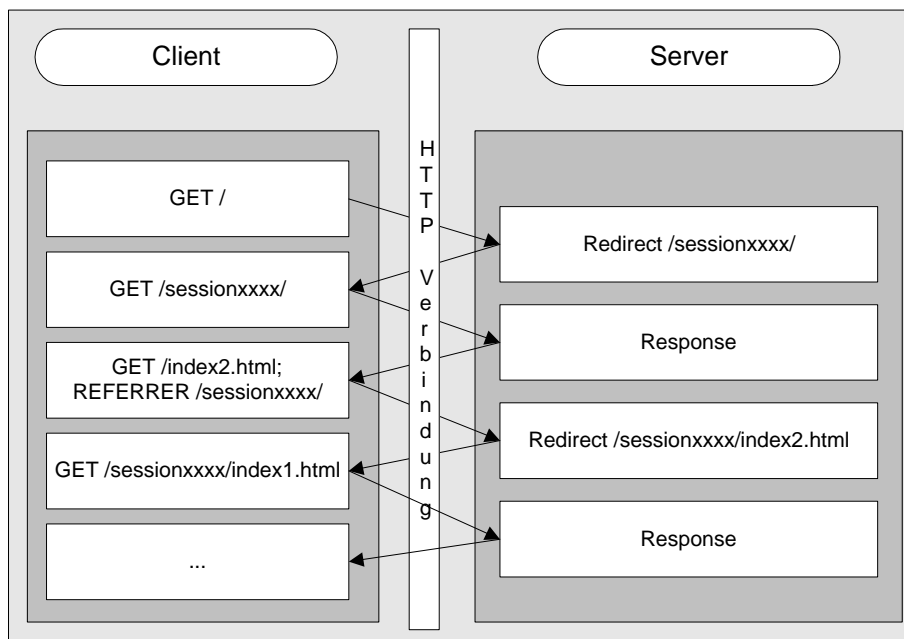


Abbildung 2.1: Ablauf einer Session, die mit dem HTTP-Statuscode 302 arbeitet.

Die Grafik 2.1 offenbart einen gravierenden Nachteil dieser Methode – um eine erfolgreiche Anfrage zu stellen sind mindestens zwei Anfragen notwendig. Laut [FGM⁺99] darf der Client nicht automatisch auf den neuen URL verwiesen werden, weil sich die Umstände der Anfrage und der gesendeten Daten verändert haben könnten.

In Abbildung 2.1 sieht man, dass der Pfad des URL verändert wird. Eine weitere Möglichkeit bietet sich mit dem Hostnamen des Servers. Wenn der Hostname zu einer ID mutiert⁵, so kann mit Hilfe von Wildcards in der DNS-Konfiguration und URL-Rewriting die ID herausgefiltert werden. Ein DNS-Eintrag dazu könnte folgendermaßen aussehen: * IN A 1.2.3.4. Dadurch wird gewährleistet, dass alle Anfragen an den Rechner 1.2.3.4 geleitet werden.

2.1.3.1 Vorteile

Die Vorteile der URL-Kodierung liegen vor allem darin, dass kein Zusatz zum Protokoll benötigt wird. Man erreicht clientseitige Unabhängigkeit, sofern der Client HTTP versteht. Ob Cookies erlaubt sind oder abgewiesen werden, ist bei dieser Methode egal.

Die Daten werden auf dem Server gespeichert. Somit sind alle Versuche, Cookies über Sicherheitslücken der Browser auszulesen sinnlos.

2.1.3.2 Nachteile

Diese Methode ist nicht standardisiert, sondern beruht auf Programmierbibliotheken, die, je nach Programmiersprache, unterschiedliche Methoden zur Realisierung der Übertragung der SID anbieten können. Es wird schwieriger, eine Session über mehrere, unterschiedliche Server zu realisieren. Java arbeitet mit URL-Rewriting, andere Bibliotheken übergeben die SID als GET- oder POST-Parameter des HTTP.

Da HTTP an sich ein unverschlüsseltes Protokoll ist, kann es vorkommen, dass eine SID von dritten ausspioniert und missbraucht wird. Um dieser Art von Attacken vorzubeugen, sollte man Sessions immer über verschlüsselte Kanäle, wie S-HTTP oder TLS, übertragen.

⁵Ein Beispiel der Firma Sevenval: <http://XB7458FE79B8DBCAB0A656BFA664483AD.sevenval.com>

2.2 Usertracking

2.2.1 Was ist Usertracking?

Im Unterschied zum Sessionhandling bezieht sich Usertracking nicht auf den Austausch von Statusinformationen, sondern vielmehr auf das Verfolgen und Beobachten des Benutzers. Die Firma DoubleClick [Dou01] rühmt sich damit, die größte Datenbank mit Benutzerprofilen, die mit Mitteln des Usertrackings erstellt wurden, zu besitzen. Um diese Daten zu sammeln ist DoubleClick mit vielen Firmen Kooperationen eingegangen, die jede Bannerwerbung über DoubleClick leiten. Ein Wiedererkennen der Benutzer ist mittels Cookies möglich. Somit können Interessensgebiete und Kaufverhalten eruiert und Zielgruppen ausgeforscht werden.

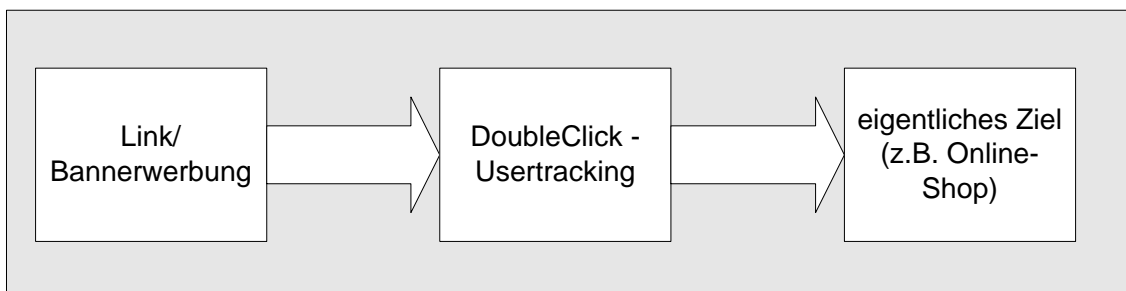


Abbildung 2.2: Usertracking am Beispiel von DoubleClick.

Auch große Suchmaschinen wie Altavista haben Verträge mit solchen *Datensammelstellen*.

„Dass Suchanfragen bei Altavista den direkten Weg zu DoubleClick finden, demonstriert der Quelltext-Auszug [...]: `<iframe src=http://ad.doubleclick.net/adi/avfilteredsearchresults.com/results;sz=468x80;kw=Schlafzimmergeheimnisse;lang=XX;cat=stext;tan=f000;ord=47101122?width=486height=60marginwidth=0marginheight=`

```
0frameborder=0scrolling=no><iframe>“
```

[BS01]

2.2.2 Ziele

Usertracking dient dazu, das Surfverhalten von Webusern zu protokollieren, damit Interessensgruppen herausgefunden werden können. Diese Interessensgruppen bilden die Grundlage für gezielte Informationsangebote und gezielte Bannerwerbung.

Usertracking kann jedoch auch anders genutzt werden, zum Beispiel um unüberlegte oder voreilige Schritte in einer Webapplikation rückgängig zu machen, oder um dem Benutzer selber Statistiken wie seine durchschnittliche Verweildauer, seine Anzahl der Besuche, etc. auf der Website zu zeigen.

2.2.3 Methoden

Im Allgemeinen gibt es drei verschiedene Methoden Usertracking zu betreiben. Diese Methoden lassen sich nach der Art der Übertragung der ID einteilen:

2.2.3.1 Logdateien

Eine der ersten Methoden, Surfverhalten zu untersuchen, war eine statistische Auswertungen von Logdateien. Jeder Zugriff auf einen Server hinterlässt einen Eintrag in einer Logdatei, wobei einzelne Clients anhand ihrer IP-Adresse wiedererkannt werden können.

Es gibt mittlerweile zahlreiche Werkzeuge, die eine solche Auswertung unterstützen. Ein weit verbreiteter Vertreter eines solchen Werkzeugs ist das Perl-Skript

*Webalizer*TM⁶, das Zugriffsstatistiken aus Logdateien erstellt und als HTML-Seite darstellt.

Proxyserver

Diese Methode hat allerdings einen gravierenden Nachteil: Proxyserver (siehe Abschnitt 1.5, Seite 7) hinterlassen nicht die IP-Adresse des Clients, sondern die eigene. Dadurch erscheinen ganze Firmennetze als ein einzelner Client. Dies macht eine statistische Auswertung nutzlos, weil das Surfverhalten der einzelnen Person nicht mehr eruierbar ist.

DHCP

Eine weitere Schwierigkeit sind Benutzer, deren IP-Adresse nicht statisch, sondern dynamisch ist. Da dies bei fast allen Modem-Benutzern der Fall ist, lassen sich Rückschlüsse nur über die Dauer einer Session ziehen, da der Benutzer beim nächsten Besuch mit sehr hoher Wahrscheinlichkeit eine andere IP-Adresse besitzt.

Aus den Logdateien des Proxyservers lassen sich allerdings genauso Onlineverhalten von Benutzern ablesen. Da ein Proxyserver zumeist für ein lokales Netzwerk existiert, lassen sich aus Zeitstempel und IP-Adresse sehr schnell der Rechner bzw. der Benutzer herausfinden. Auf diese Weise sammelt der Proxy genau die Daten, die für Usertracking relevant sind.

⁶*Webalizer*TM ist aber kein geeignetes Programm, um auf einzelne Benutzer Rückschlüsse ziehen zu können. Es bietet lediglich eine statistische Auswertung, wieviel Prozent der Besucher aus welchem Land kommen, welche Seiten die höchsten Zugriffszahlen besitzen, wieviel Fehlercodes zurückgegeben wurden, wieviel Kilobytes pro Stunde/Tag/Monat/Jahr transferiert wurden, etc. Aus diesen Daten kann man aber Benutzerfreundlichkeit und Attraktivität der Website erkennen. Ist die Zugriffszahl der Startseite sehr hoch und alle anderen Seiten haben eher gemäßigte Statistiken so ist die Site entweder uninteressant für die meisten Benutzer oder aber unübersichtlich und unattraktiv gestaltet, sodass die meisten Besucher diese Site sofort wieder verlassen.

2.2.3.2 Web-Bug

Eine Unterart der Logdateien ist der sogenannte Web-Bug, bei dem in einer HTML-Datei ein unsichtbares Bild versteckt ist, das von einem Usertracking-Server geladen wird. Jedes Bild in einer HTML-Seite erfordert eine neue Anfrage an einen Web-Server. Diese Anfrage hinterläßt Spuren in der Logdatei des Servers, die über IP-Adresse, Browsertyp, von welcher Seite aus der Zugriff erfolgt ist und ähnliche Informationen.

Eine weitere Möglichkeit ist, ein Skript als solches Bild zu tarnen. Ein Skript bietet wesentlich mehr Möglichkeiten Informationen über einen Benutzer zu sammeln und zu speichern bzw. in Zusammenhang mit anderen Daten zu bringen.

Ein solches Skript kann auch in einem HTML E-Mail verschickt werden. Falls ein HTML-fähiges E-Mail Programm verwendet wird und dieses auch noch Skripts ausführen darf, so können persönliche Daten vom Heimcomputer erhoben und an einen Server übermittelt werden. Dieser Methode bedienen sich viele Spammer, die Auskünfte über ihre Opfer einholen wollen.

2.2.3.3 Cookies

Eine Möglichkeit, Benutzer im Internet zu beobachten, sind Cookies. Dabei wird eine ID erzeugt, anhand der jedem Client ein am Server gespeicherter Datensatz zugeordnet werden kann. Dieses Cookie wird dann bei jedem Zugriff zusammen mit dem HTTP-Header ausgewertet. So kann ein Profil des Surfverhaltens einzelner Surfer erstellt werden. Cookies haben allerdings die Einschränkung, dass sie nur für eine bestimmte Domäne gültig sind. Firmen wie DoubleClick spielen eine Verteilerrolle für Bannerwerbung. Klickt man solch eine Bannerwerbung an, wird eine DoubleClick-Seite geöffnet, die die Auswertung und Speicherung der relevanten Daten vornimmt und der Client erst danach auf das eigentliche Ziel weiterleitet (siehe Abbildung 2.2).

2.2.3.4 Usertracking mit Hilfe des URL - Location-Poisoning

Falls ein Client Cookies ausgeschaltet hat, kann eine SID über einen HTTP-Header mitgegeben werden. Tom Vogt beschreibt in seiner Mail vom 8. Februar 2000, wie eine SID zwischen verschiedenen Domänen ausgetauscht werden kann, ohne dass der Benutzer etwas davon mitbekommt [Vog00]. Dabei wird beim Aufbau der Session eine SID erstellt und als Teil des URL verwendet. Dieser URL wird als 302 - Statusmeldung an den Client geschickt, der sich dann zu der neuen URL verbindet. Ein Verbindungsaufbau zu einem Partner dieser Website z.B. über einen Link enthält einen `Referer-Header`, der wiederum die SID an diese Seite weiterleitet. Auf diese Art kann dieselbe SID über Domänengrenzen hinweg verwendet werden. Die Server können Daten, die in Zusammenhang mit dieser SID stehen, an einen zentralen Rechner übermitteln. So kann ein Profil des Benutzers erstellt werden, zumindest solange die SID Teil des URL bleibt.

Anmerkung: Diese Methode wird *Location-Poisoning* genannt, da der HTTP-Header `Location` für Usertrackingzwecke verwendet wird. Diese Verwendung ist nicht im Standard vorgesehen und stellt somit einen Missbrauch dar.

2.2.3.5 E.T. - Programme

Einen anderen Weg der Datenbeschaffung gehen manche Firmen, in dem sie Freeware-Programme anbieten, die mit einem Zusatz-Programm ausgestattet sind. Diese Zusätze haben Zugriff auf den lokalen Rechner und können dort nach Lust und Laune Daten ausspionieren, Web-Caches durchforsten, Cookies lesen, Daten aus der Windows-Registrierung erheben oder lokal gespeicherte Passwörter, Kreditkartennummern und ähnliches sammeln. Solche Daten werden dann bei einer bestehenden Internetverbindung an einen Server übermittelt und dort ausgewertet.

Der Name *E.T.*-Programme rührt von der Figur *E.T.* her, die immer „nach Hause telefonieren“ will. (vgl. [BS01])

2.3 Proxies und DNS-Caches

Um den Netzwerkverkehr zu verringern, wurden Proxyserver eingeführt, die Webinhalte eine Zeit lang zwischenspeichern. Vor allem häufig besuchte Webserver werden mit dieser Methode entlastet. Dynamisch gestaltete Websites können aber nicht zwischengespeichert werden, weil jeder Benutzer andere Inhalte bekommt.

Proxyserver arbeiten mit URL-Vergleichen. Stimmt ein angeforderter URL mit einem gespeicherten überein, so wird, sofern ein Timeout nicht überschritten wurde, die zwischengespeicherte Seite geliefert. Genau hier liegt das Problem: Wenn Usertracking mit Hilfe des URL vorgenommen wird, bekommt jeder Client für dieselbe Internetseite einen anderen URL. Somit sind Proxyserver als Web-zwischenspeicher wirkungslos.

Ein ähnliches Problem haben DNS⁷-Caches. Ein DNS-Server ist für die Namensauflösung zuständig. Wenn sich der Name eines Webservers ändert, was bei Location-Poisoning der Fall ist, so werden auch diese Caches wirkungslos.

2.4 Privatsphäre und Anonymität im Web

Auf Grund der Möglichkeit Benutzer verfolgen zu können, ohne dass diese etwas davon mitbekommen, stellt sich die Frage nach Datenschutz, Privatsphäre und Anonymität im Web. Simplen Trackingmechanismen wie Logdateien und Cookies kann man ohne größeren Aufwand entkommen, aber um dem Usertracking mit Hilfe des URL auszuweichen benötigt es schon ein wenig mehr Aufwand.

⁷DNS - Domain Name Service

2.4.1 Verschlüsselung

HTTP ist ein Protokoll, das Daten im Klartext über das Internet transportiert. Somit ist es ein leichtes, Daten mitzulesen und ganze Online-Sessions nachzuvollziehen. Dieses Problem ist gerade in Zeiten von E-Commerce und E-Business nicht so einfach von der Hand zu weisen.

Es existieren zwei Verschlüsselungsstandards für das HTTP, einerseits SSL und andererseits TLS [AD99]. Diese zwei Protokolle unterscheiden sich nur in Details, wobei auf diese hier nicht näher eingegangen wird.

Sowohl SSL als auch TLS bieten ein eigenes Übertragungsprotokoll an, das den Übertragungskanal verschlüsselt. Am Anfang einer verschlüsselten Verbindung findet ein sogenanntes Handshake statt, wobei sich Client und Server die Art der Verschlüsselung und einen geheimen Schlüssel ausmachen. Haben sich die beteiligten Hosts geeinigt, so findet der verschlüsselte Datentransfer statt. Danach wird die Verbindung beendet.

Passwörter, Kreditkartennummern, persönliche Daten, etc. sollten immer verschlüsselt übertragen werden, da diese besonders gerne ausspioniert werden.

2.4.2 Anonymizer

Um simple Trackingmechanismen auszutricksen genügt es, sogenannte Anonymizer zu verwenden. Anonymizer arbeiten ähnlich wie Proxies, in dem sie Anfragen entgegennehmen und an den entsprechenden Server weiterleiten. Dadurch kommuniziert der Client ausschließlich mit dem Anonymizer, der für ihn durch das Web surft und anonyme Spuren hinterlässt. Cookies kann jeder Benutzer selber ausschalten und ist somit für Beobachtungen durch Cookies selbst verantwortlich.

Ein Beispiel für Anonymizer ist Anonymizer.com [Ano00], wobei Anonymizer.com

eine Logdatei führt, mit deren Hilfe der echte Benutzer jederzeit ermittelt werden kann. Anonymizer.com schützt somit nur vor Usertracking durch andere Firmen, pflegt aber eine eigene Logdatei, mit der sich das Surfverhalten eines Benutzers jederzeit rekonstruieren lässt.

2.4.3 Proxyserver und Firewalls

Mit Proxyservern und Firewalls lassen sich HTTP-Header austauschen bzw. umschreiben, sodass auch die Location-Poisoning Methode nicht mehr in der Lage ist, Sessions aufrecht zu erhalten. Ein kleines Perl-Programm auf der Lemuria-Homepage [Lem01], das als Plugin⁸ für den Proxyserver Squid [Nor01] erhältlich ist, verändert die SID im URL bei jedem Zugriff. Dadurch wird dem Webserver jedesmal eine neue Session vorgegaukelt und ein Usertracking unmöglich gemacht.

2.5 Zusammenfassung

2.5.1 Unterschied zwischen Sessionhandling und Usertracking

Der Unterschied zwischen Sessionhandling und Usertracking besteht darin, dass eine Session über einen kurzen Zeitraum besteht, während dem der Benutzer wiedererkannt wird. Außerdem ist der Benutzer nur einem Server bzw. einer Domäne bekannt.

Usertracking hingegen befasst sich mit der Verfolgung von Benutzern, um ein Profil erstellen zu können, das das Surfverhalten und die daraus resultierenden Interessensgebiete und Kaufverhalten einzelner Personen beinhaltet.

Die verwendeten Techniken überschneiden sich teilweise, da technisch gesehen das selbe Ziel verfolgt wird: Wie erkenne ich eine Person wieder?

⁸Plugin - Zusatzprogramm, das die Funktionalität eines Programms erweitert

2.5.2 Verwendete Techniken

2.5.2.1 Logdatei

Eine Logdatei wird vom Server angelegt und beinhaltet Informationen über Zugriffe auf den Server. Es werden u.a. IP-Adressen, Referrer und der angeforderte URL gespeichert. Mit Hilfe der Logdatei können Zugriffsstatistiken erstellt und ausgewertet werden. Diese Ergebnisse sollten in das Design mit einfließen, um die Benutzerführung zu optimieren.

Sofern Benutzer eigene, statische IP-Adressen verwendet werden, kann eine Benutzererkennung erfolgen und auf diese Weise Informationen gesammelt werden.

2.5.2.2 Cookies

Cookies kommen in beiden Fällen zum Einsatz, können aber vom Benutzer ausgeschaltet werden und funktionieren deswegen nicht immer. Neue Browser stellen dem Benutzer mehr oder weniger umfangreiche Cookie-Filter zur Verfügung, die zwischen dauerhaft gespeicherten und Session-Cookies unterscheiden können.

Cookies speichern Daten lokal beim Benutzer. Kreditkartennummern, Passwörter und andere schützenswerte Daten konnten in der Vergangenheit auf Grund unzureichender Sicherheitsvorkehrungen der Browserhersteller immer wieder ausgespielt werden.

2.5.2.3 URL-Kodierung

Die URL-Kodierung baut eine SID in den URL ein, um eine Benutzererkennung auf diese Weise zu ermöglichen. Sie funktioniert immer, sofern der Benutzer den URL nicht mit einer anderen SID versieht oder die SID herauslöscht.

Eine besondere Form der URL-Kodierung ist das Location-Poisoning, wo dem

Client vorgegaukelt wird, die Internetseite sei unter einem anderen URL zu finden und ein `Location`-Header des HTTP mitgesendet wird, der einen URL mit einer SID beinhaltet. Partnerfirmen, die über Links aufgerufen werden können, verwenden so die selbe SID weiter, um Informationen über den Benutzer zu sammeln.

2.5.3 Anonymität und Privatsphäre im Web

Den einzig sicheren Schutz gegen Usertracking bieten Anonymizer. Solche Server agieren als Proxyserver, allerdings tauschen sie gewisse Informationen im HTTP-Header aus, sodass eine Benutzerverfolgung unmöglich wird. Durch diese Server wird aber auch Sessionhandling sehr eingeschränkt bzw. teilweise unmöglich gemacht, da eben genau die Daten, die zur Wiedererkennung benötigt werden, weggelassen oder mit zufälligen Daten überschrieben werden.

Kapitel 3

Rechtliche Aspekte von Usertracking-Mechanismen

Dieses Kapitel soll die rechtlichen Aspekte wie Datenschutz, Privatsphäre und Handel mit persönlichen Daten abdecken.

In wie weit das Geschäft mit benutzerbezogenen Daten rechtlich abgedeckt ist, hängt auch vom jeweiligen Staat bzw. dessen Regierung ab. Dieses Kapitel bezieht sich hauptsächlich auf die Situation in Österreich.

3.1 Datenschutz in Österreich

3.1.1 Einleitung

Datenschutz ist gerade im heutigen Informationszeitalter ein wichtiges Thema. Da Daten generell computerunterstützt verarbeitet werden, müssen auch gewisse gesetzliche Regelungen auf die EDV¹ angepasst bzw. dafür entworfen werden.

Die Übertragung der Daten in Netzwerken erfordert einen besonders sensiblen Umgang, da Zwischenspeicher wie Proxyserver zumeist öffentlich zugänglich und daher auch ein beliebter Angriffspunkt für Datensammler sind. Das Datenschutzgesetz sah ursprünglich für das Ermitteln, Verarbeiten, Benützen, Übermitteln, Überlassen und Löschen von Daten unterschiedliche Rechte und Pflichten vor. Diese Begriffe sind nach heutigem Stand der Technik nicht mehr eindeutig voneinander trennbar.

Usertracking macht sich übermittelte Daten zu Nutze, um ein Profil eines Benutzers zu erstellen. Inwieweit schützenswerte Daten davon betroffen sind, hängt einerseits von der Art des Usertrackings und andererseits von der Freigiebigkeit der Benutzer ab. Wird ein verschlüsselter Kanal gewählt, so bleiben die Daten während der gesamten Übertragung für Dritte verborgen bzw. sind nicht ohne weiteres lesbar.

1995 wurde die Richtlinie 95/46/EG des Europäischen Rates und des Rates zum Schutz natürlicher Personen erlassen, die den Datenschutz bei Verarbeitung personenbezogener Daten regeln soll. Auf Grund dieser Richtlinie wurde in Österreich das DSG² von 1978 novelliert und am 13. Juli 1999 im Nationalrat beschlossen. Am 1. Jänner 2000 trat das DSG 2000 in Kraft.

¹EDV - Elektronische Datenverarbeitung

²DSG - Datenschutzgesetz

3.1.2 Schutzwürdige Daten

Es gibt zwei Arten von personenbezogene Daten. Schutzwürdige und solche, die jederzeit für jeden zugänglich sind. Zu den nicht schutzwürdigen Daten gehören zum Beispiel Name, Geburtsdatum, Wohnort, Telefonnummer, etc. Ausnahmeregelungen können aber auch solche Daten als schutzwürdig deklarieren. Dies erfordert aber eine Begründung der Betroffenen.

Schutzwürdige personenbezogene Daten sind jene Daten, die die Privatsphäre einer Person betreffen, wie zum Beispiel Religionsbekenntnis, politische und philosophische Überzeugung, rassische und ethnische Herkunft, Gewerkschaftszugehörigkeit, Gesundheit oder Sexualeben [Kna00, Seite 16 ff.].

Persönliche Interessen, die Rückschlüsse auf Kaufverhalten zulassen sind keine schutzwürdigen Daten, es sei denn, sie betreffen einen von den oben genannten Punkten.

Im DSG 2000 werden die alten Begriffe *Ermitteln*, *Verarbeiten*, *Benützen*, *Übermitteln*, *Überlassen* und *Löschen von Daten* durch den Begriff *Verwenden von Daten* ersetzt, da im Zeitalter der Vernetzung keine klaren Grenzen zwischen ihnen mehr gezogen werden können.

3.1.2.1 Das Verwenden von Daten

Für das *Verwenden von Daten* müssen bestimmte Voraussetzungen erfüllt sein: Daten dürfen nur nach *Treu und Glauben* und auf rechtmäßige Weise verwendet werden. Die Begriffe *Treu und Glauben* werden in den einzelnen Bereichen festgelegt - im privaten Bereich können dies gesetzliche Interessensvertretungen, sonstige Berufsverbände und vergleichbare Einrichtungen ausarbeiten.

Bestimmte *Qualitätskriterien* der Daten müssen erfüllt sein. Zum Beispiel müssen Daten für einen eindeutig bestimmt Zweck erhoben und dürfen nur für diesen Zweck verwendet werden. Eine spätere Änderung des Zwecks ist nicht zulässig. Die Zulässigkeit des Zwecks muss ebenfalls gegeben sein. Diese Zulässigkeit hat

zwei Voraussetzungen: die Berechtigung des Auftraggebers und die Berücksichtigung der schutzwürdigen Interessen der Betroffenen. (vgl. [MSB99, DSG 2000 §§ 6 und 7]).

Jede natürliche Person hat das Recht auf Löschen bzw. Richtigstellen seiner Daten. Doch dazu muss dieser natürlichen Person erst einmal klar sein, welche Daten über sie gespeichert sind.

„Über eines muss sich der Surfer allerdings klar sein: Zwar hat er die rechtliche Möglichkeit, der Nutzung seiner Daten nachträglich zu widersprechen, aber es ist unglaublich schwierig, hinterher nachzuvollziehen, wer denn nun welche Daten weitergegeben hat. Wenn Daten einmal draußen sind, kann man sie kaum wieder einfangen.“

[BS01, Kasten auf Seite 209]

Weiters ist es sehr schwer nachzuvollziehen, wer aller eine Kopie dieser Daten besitzt.

Ein Problem bilden manche Softwareprodukte, wie zum Beispiel SAP R/3, das ein Löschen von Datensätzen durch ein Deaktivieren der Daten realisiert. Diese Daten sind zumindest theoretisch jederzeit wieder aktivierbar.

3.1.2.2 Handel mit personenbezogenen Daten

Laut dem DSG ist der gewerbliche Handel mit schutzwürdigen, personenbezogenen Daten untersagt. Allerdings können Adressen, Telefonnummern und andere personenbezogene Daten zwischen Firmen ausgetauscht werden, sofern der Betroffene die ausdrückliche Genehmigung dazu erteilt hat. So bieten Firmen wie zum Beispiel Schober [Sch01] über ihre Internetseite Adressen aus ihrer Datenbank nach Zielgruppen geordnet zum Bestellen an. Die Preise für Postadressen liegen zwischen 3 Pfennigen und 20 Mark, je nach Stückzahl und Filtergrad³.

³Gefiltert wird zum Beispiel nach Geschlecht, Wohnort, Staatsbürgerschaft, Haushaltseinkommen, Interessensgebiete, etc.

Sind auch Kaufverhalten der Kunden bekannt, kann der Wert einer Adresse auf mehrere 1000 Mark steigen. [BS01, Seite 201]

3.1.2.3 Probleme und rechtliche Grauzonen

Das Problem mit Online-Verhalten besteht darin, dass sich die meisten Benutzer nicht bewusst sind, welche Daten über sie gespeichert werden bzw. sind. Über den ISP⁴ lässt sich zum Beispiel jederzeit herausfinden, welcher Benutzer welche Anfragen an welche Server gesendet hat. Zusätzlich besteht die Möglichkeit, die Antworten der Server festzuhalten. Dadurch kann ohne größere technische Probleme ein spezifischer Benutzer beobachtet und ein persönliches Profil erstellt werden, das eindeutig Interessensgebiete und auch das daraus resultierende Kaufverhalten beinhaltet. Mit Verschlüsselungstechniken wie SSL⁵ oder TLS⁶ werden die Inhalte der Übertragung zwar verschlüsselt, aber Benutzer und Zielsever sind immer noch erkennbar und Rückschlüsse auf Interessensgebiete möglich.

Ein rechtlicher Graubereich ist auch die Überwachungen von Mitarbeitern. Sind Video-Überwachung untersagt und leichter zu kontrollieren, so können Telefongespräche und -dauer, E-Mails und Surfverhalten der Mitarbeiter ohne großen technischen Aufwand überwacht und für Kontrollzwecke verwendet werden. Jeder Server besitzt eine Logdatei, die Aufschluss darüber gibt, welcher Rechner zu welcher Uhrzeit welche Resource verlangt hat. In einem Firmennetzwerk sind anhand dieser Informationen sofort die entsprechenden Mitarbeiter gefunden.

Eines der bislang ungelösten Probleme ist die rasante Entwicklung von neuen Technologien, die ein ständiges Anpassen von Gesetzen nach sich zieht. Bis ein

⁴ISP - Internet Service Provider

⁵SSL - Secure Socket Layer

⁶TLS - Transport LayerSecurity

Gesetzesentwurf bzw. eine Novelle an eine Technologie angepasst wurde, diese vom Nationalrat beschlossen wurde und in Kraft tritt, ist dies ursprüngliche Technologie schon längst weiterentwickelt oder durch eine andere abgelöst worden.

3.2 Datenschutz und Internet

3.2.1 Datenerhebung

Die Marktforschung möchte möglichst komplette Profile von Benutzern anlegen, weshalb online erhobene Daten nicht immer ausreichen. Laut einer Schätzung werden derzeit ca. 5% der Daten online, der Rest wird mit herkömmlichen Methoden wie Umfragen, Telefonbefragungen, Gewinnspielbeteiligungen etc. erhoben [BS01].

Um Usertracking sinnvoll zu betreiben, werden personenbezogene Daten wie Benutzername, Name, Telefonnummer oder IP-Adresse, etc. in Zusammenhang mit dem Online-Verhalten des Benutzers gebracht und auch so gespeichert. Gewinnspiele, Fragebögen, E-Mail Kettenbriefe etc. dienen oftmals der Datenerfassung, die dann mit dem Surfverhalten des einzelnen in Zusammenhang gebracht ein fast komplettes Profil des Benutzers ergeben.

3.2.2 Naivität der Benutzer

Auch wird der Umgang mit personenbezogenen Daten von den Benutzern selber nicht immer als kritisch eingestuft. Viele naive Benutzer vertrauen auf die Geheimhaltung ihrer Daten, lesen sich aber selten die Informationen über den Umgang mit personenbezogenen Daten der einzelnen Firmen durch. So kann es vorkommen, dass ihre Daten, wie zum Beispiel Adressen, zwischen Partnerfirmen ausgetauscht werden und der Betroffene persönlich adressierte Werbepost

dieser Partnerfirma erhält. Ob diese Post in digitaler Form oder über herkömmliche Briefe verschickt wird, spielt hier eine keine Rolle. Ein solcher Austausch ist sogar gesetzlich erlaubt, wenn die Daten gemäß dem DSG 2000 erhoben und der Benutzer die Weitergabe ausdrücklich erlaubt. Diese Erlaubnis wird online meist in Form eines Kontrollkästchens erfragt. Die Standardeinstellung des Kontrollkästchens ist in vielen Fällen so gesetzt, dass der Benutzer sich mit dem Datenaustausch einverstanden erklärt.

3.3 Kritik am DSG 2000

Der Beschluss des DSG 2000 stieß nicht nur auf Übereinstimmung, sondern auf sehr viel Kritik. Einer der schwersten Kritikpunkte war und ist die Definition von *Verwenden von Daten*. Die EU-Richtlinie sieht vor, dass alle Daten, unabhängig von ihrer Verarbeitungsweise unter diesen Begriff fallen. Das DSG sieht nur Datenverarbeitung, die „zur Gänze oder auch nur teilweise automationsunterstützt erfolgt“ als ein *Verwenden von Daten* vor.

Des weiteren wird daran Kritik geübt, dass die Verpflichtung zum Löschen bzw. Richtigstellen von Daten fehlt, sofern die Daten nicht dem vorgesehenen Zweck dienen bzw. der Datensatz unvollständig ist.

Auch politische Kritik musste das DSG 2000 über sich ergehen lassen. Bemängelt wird, dass eine Verankerung des „*Grundrechtes auf informationelle Selbstbestimmung*“ fehlt. Damit ist gemeint, dass einzelne grundsätzlich selbst über die Preisgabe und Verwendung ihrer persönlichen Daten entscheiden können.

3.4 Zusammenfassung

Das Datenschutzgesetz wurde 1999 novelliert, um der EU-Richtlinie 95/46/EG zu entsprechen und sich an die neuen Gegebenheiten im Informationszeitalter anzupassen.

Die kommerzielle Nutzung von schützenswerten Daten, wie rassische oder ethnische Herkunft, Religionsbekenntnis, politische und philosophische Überzeugung, Gewerkschaftszugehörigkeit, Gesundheit und Sexualleben, etc. ist in Österreich sowie den meisten anderen Staaten der Welt untersagt.

Firmen dürfen personenbezogene Daten nur auf ausdrückliche Zustimmung des Betroffenen hin untereinander austauschen. Diese Zustimmung kann jederzeit widerrufen werden.

Jeder hat das Recht auf Löschen bzw. Richtigstellen seiner Daten. Dieser Punkt wird in der Praxis nicht immer durchführbar sein, da das Löschen von Daten zu Inkonsistenzen von Datenbeständen und somit zum Fehlverhalten großer Datenbanken führen kann. Außerdem ist ein Löschen in einem System wie zum Beispiel SAP R/3 nicht möglich. Dort werden die Daten eingefroren, könnten aber theoretisch jederzeit weiter verwendet werden.

Kapitel 4

Der Formulargenerator

Dieses Kapitel behandelt eine Webanwendung, die Methoden des Sessionhandlings und des Usertrackings verwendet: ein Formulargenerator.

Der hier beschriebene Formulargenerator ist in Java programmiert worden und ermöglicht es, HTML-Formulare auf einen Webserver zu laden. Er stellt eine Datenbankverbindung für das jeweilige Formular her, um eingegebene Daten für spätere Auswertungen zu speichern.

4.1 Projekt Formulargenerator

4.1.1 Die Idee

Die Firma TIWAG Tiroler Wasserkraftwerke AG betreibt eine Firmenwebsite [TIW01], auf der WebFormulare für Kundenanfragen bereitgestellt sind. Diese WebFormulare haben weder eine Datenbankanbindung, noch sind sie in der Lage, eingegebene Daten irgendwie zu speichern. Die Daten werden sofort in ein E-Mail verpackt und dieses an eine zentrale Stelle geschickt, die es je nach Zuständigkeitsbereich an die verantwortliche Abteilung weiterleitet. Dieser nicht vorhandene Workflow erfordert massiven händischen Eingriff und kann zu Inkonsistenzen und Fehlern führen.

Da es in der Vergangenheit immer wieder zu Verlusten solcher E-Mails kam und kein Backup der eingegebenen Datensätze aufbewahrt wurde, entstand die Idee, die Formulare an eine Webdatenbank anzubinden. Die Frage, die sich nun stellte, war, ob eine Neuprogrammierung der bestehenden Formulare ausreicht, oder ob ein Werkzeug zur Erstellung von Formularen notwendig werden würde.

- Für die Nachprogrammierung aller bestehenden Webformulare wäre ein erhöhter Schulungsaufwand notwendig, da sich bis dato nicht alle Webmaster der verschiedenen Abteilungen mit Programmiersprachen oder Datenbanken befasst haben.
- Des weiteren sind auch alle zukünftigen Formulare von dieser Maßnahme betroffen. Mit dem Formulargenerator entfällt die Aufgabe, Webformulare zu programmieren. Dadurch werden Entwicklungszeiten verkürzt bzw. überhaupt eingespart.
- Jeder Webmaster verwendet seinen Lieblingseditor zum Erstellen der Websites. Viele verwenden WYSIWYG¹-Editoren, die kaum die Möglichkeit der

¹WYSIWYG - What You See Is What You Get

Programmierung eines Skripts lassen.

- Es soll nicht jeder Webmaster CGIs oder Java Servlets programmieren dürfen.
- Statistische Auswertungen über Aufträge sollen in Zukunft möglich sein. Unter anderem sollen die durchschnittliche Bearbeitungsdauer eines Datensatzes, sowie die Anzahl neuer Datensätze pro Zeiteinheit erfasst und eventuelle Schwachpunkte beseitigt werden.

Aus diesen Überlegungen kristallisierte sich der Bedarf eines Formulargenerators, der existierende HTML-Formulare an eine Datenbank anbinden kann.

4.2 Funktionalität

Die Funktionalität des Formulargenerators besteht darin, HTML-Dateien mit eingebetteten Formularen an ein Java Servlet anzuknüpfen, das die über die Formulare eingegebenen Daten in eine Datenbank speichert und ein E-Mail an eine in der Datenbank hinterlegte Adresse mit dem Link auf den eingegebenen Datensatz schickt.

Dabei gibt es zwei Benutzergruppen: *normale Benutzer*, die Formulare in den Formulargenerator laden und diese dann verwalten dürfen und *Administratoren*, die sowohl sämtliche Formulare, als auch alle Benutzer administrieren.

Die Benutzerkonten und Metadaten der Formulare werden in vier Tabellen einer Datenbank gespeichert: *user*, *forms*, *permissions* und *logging*.

Für jedes Formular wird eine weitere Tabelle angelegt, in Abbildung 4.1 als *Beispiel-Formular* dargestellt. Diese Tabellen werden mit den Daten aus den entsprechenden Formularen gefüllt.

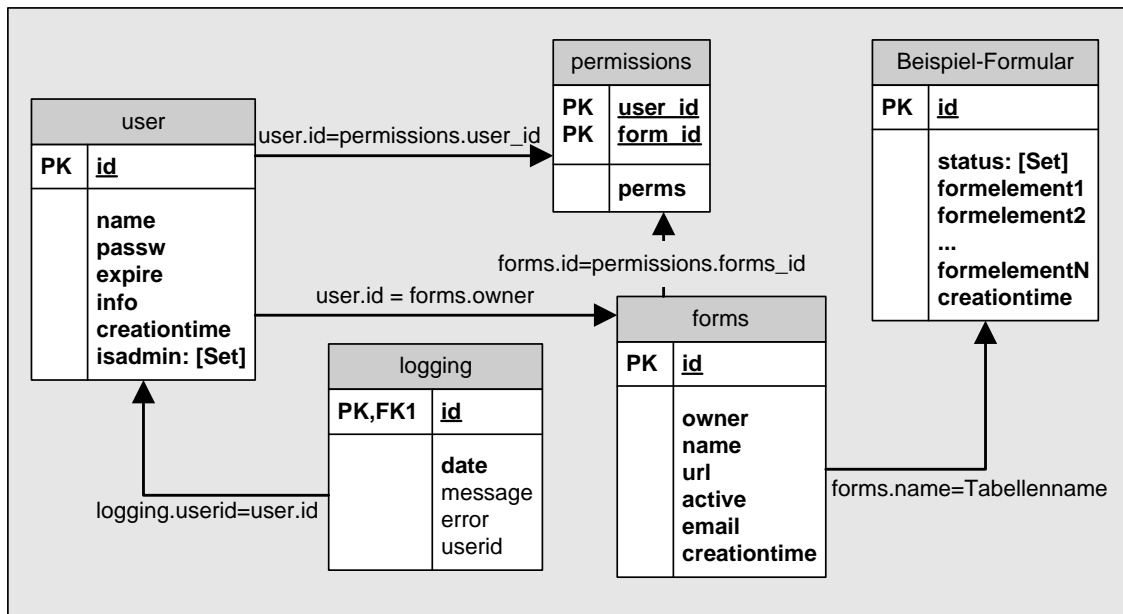


Abbildung 4.1: Konzeptionelles Design der Datenbank

4.2.1 Benutzeradministration

Die Benutzeradministration beinhaltet das Anlegen und Entfernen von Benutzern, sowie das Ändern von bestehenden Datensätzen, wie Benutzername, Benutzerpasswort, Benutzerinformation und das Ablaufdatum des Benutzerkontos.

Zusätzlich hat jeder Benutzer die Möglichkeit, seine eigenen Daten (mit Ausnahme des Ablaufdatums des Benutzerkontos) zu ändern.

In Abbildung 4.2 sieht man fünf bereits bestehenden Benutzerkonten mit ihren Eigenschaften. Die Passwörter werden nicht angezeigt; sie sind aber von Administratoren jederzeit veränderbar. Die letzte Zeile der Abbildung ist für die Erstellung eines neuen Benutzerkontos. Die Daten werden einfach in die entsprechenden Felder eingetragen und nach einem Klick auf das Feld „Benutzer anlegen“ wird der neue Benutzer angelegt. Er ist sofort aktiv.

Jede Änderung wird unmittelbar in die Datenbank übertragen und gilt sofort. Da

| Benutzername | Passwort | Ablaufdatum | Administrator | Information | | |
|--------------|----------|-------------|---------------|----------------------|------------------|---------|
| admin | ***** | 3001-01-01 | Ja | Ein Admin-Account | Update | Löschen |
| georg | ***** | 2001-05-30 | Nein | | Update | Löschen |
| hari | **** | 2001-05-30 | Nein | | Update | Löschen |
| Mike | **** | 2001-05-30 | Nein | | Update | Löschen |
| user | ***** | 2001-05-30 | Nein | Ein normaler Benutze | Update | Löschen |
| | | YYYY-MM-DD | Nein | | Benutzer anlegen | |

Abbildung 4.2: Benutzeradministration aus der Sicht eines Administrators

Benutzer anhand einer nicht editierbaren Kennung identifiziert werden, sind sogar die Benutzernamen veränderbar. Allerdings darf ein Benutzername nur einmal vergeben werden, d.h. es kann keine zwei Benutzerkonten mit dem selben Benutzernamen geben, da dieser für das Login verwendet wird.

4.2.2 Formularadministration

Die Formularverwaltung ermöglicht es, bestehende Formulare zu aktivieren bzw. zu deaktivieren, eine E-Mail Adresse, die bei Eingabe von Daten in das Formular angeschrieben wird, zu hinterlegen, oder das Formular zu löschen. Mit dem Browse-Knopf werden alle bisher eingegebenen Daten des Formulars angezeigt.

Jeder Benutzer hat die Möglichkeit, die von ihm erstellten Formulare zu verwalten. Administratoren können alle Formulare verwalten, unabhängig vom Eigentümer. In Abbildung 4.3 sieht man zwei Formulare, die unterschiedlichen Benutzern gehören. Der Benutzer georg besitzt das Formular feedback, der Benutzer Mike das Formular test. Beide Formulare sind aktiv.

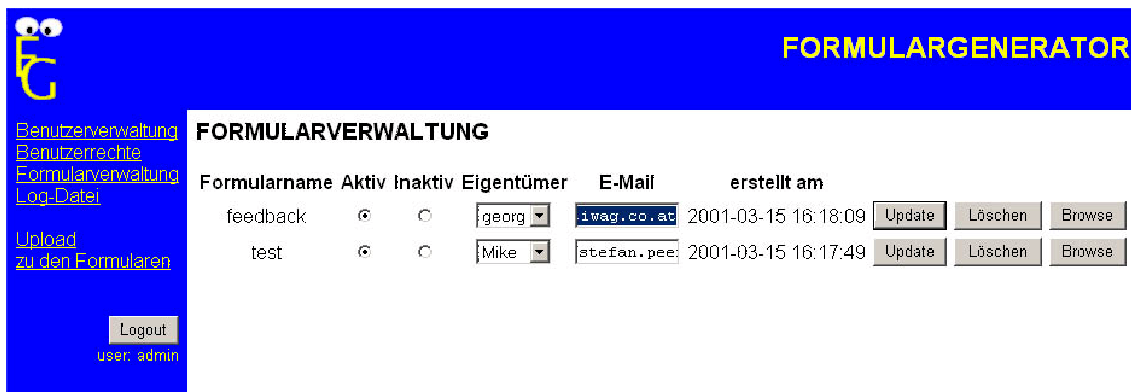


Abbildung 4.3: Formularverwaltung aus der Sicht eines Administrators

4.2.3 Formularupload

Der Formularupload bietet die Möglichkeit, HTML- und XML-Dateien über HTTP auf den Server zu laden, das Formular zu benennen und eine E-Mail Adresse, die bei einer Eintragung eines neuen Datensatzes angeschrieben wird, zu hinterlegen. Die HTML-Datei wird auf dem Server zwischengespeichert, in ein DOM²-Objekt umgewandelt und danach eine entsprechende Tabelle in der Datenbank angelegt. Die einzelnen Felder der Tabelle bestehen aus den Formularelementen und haben den Namen des über die Eigenschaft NAME im HTML-Tag des Formularelements bestimmten Wertes. Des weiteren werden noch zusätzliche Felder angelegt, `id`, `status` und `creationtime`, das den Zeitpunkt des Anlegens eines Datensatzes speichert (siehe Abbildung 4.1).

Das Feld `status` bedarf genauerer Erklärung:

Auf Grund des Systems über E-Mail konnte zu keinem Zeitpunkt festgestellt werden, ob der Datensatz bereits bearbeitet oder ob dieser noch überhaupt nicht beachtet wurde. Durch das Status-Feld in der Datenbank ist es möglich, jederzeit nachzuvollziehen, was mit dem Datensatz geschehen ist. In Verbindung mit einer Logdatei (siehe Abschnitt 4.4.1, Seite 50) läßt sich herausfinden, wer den Status

²DOM - Dynamic Object Model

wann verändert hat.

4.2.4 Formularansicht

Die Formularansicht liefert eine Auswahl aller aktiven Formulare. Der Benutzer hat die Möglichkeit ein Formular zu wählen und bekommt dieses dann über den Browser zu sehen. Wenn Daten gesendet werden, so wird ein Servlet aufgerufen, das diese Daten in die entsprechende Tabelle der Datenbank schreibt (siehe auch Abbildung 4.1).

In der Datenbank in der Tabelle `forms` gibt es das Feld `url`, in dem der Name der Java-DOM-Klasse enthalten ist, die das Formular representiert. Dieser Name wird der Klasse `Show` übergeben, die die Klasse mit Hilfe eines `FormClassLoader`-Objektes instantiiert und als HTML-Code, den die Methode `toDocument` liefert, an den Client leitet. (siehe Abbildung 4.7, Seite 59)

4.3 Sessionhandling

Das Sessionhandling wird von der Java Servlet Engine vorgenommen, die Sessionvariablen entweder in Dateien, in eine Datenbank oder in ein shared-memory Segment zwischenspeichert. Für den Formulargenerator wird die Java Servlet Engine `ResinTM` verwendet. Die Entscheidung für `ResinTM` und gegen die Referenzimplementierung `TomcatTM` fiel auf Grund der in `TomcatTM` mit der zum Zeitpunkt der Planung aktuellen Versionsnummer 3.1 nicht vorhandenen Möglichkeit, Sessionhandling ohne Cookies zu betreiben.

4.3.1 Datenbankverbindungen, Datenbankpools

`Resin` bietet die Möglichkeit, einen Pool von Datenbankverbindungen zu verwalten. Ein Pool hat den Vorteil, dass Verbindungen zur Datenbank mehrfach

genutzt werden können und spart somit einen Auf- und Abbau der Verbindung für Datenbanktransaktionen. Des weiteren wird eine Datenbankunabhängigkeit der Applikation erreicht, da der Treiber von der Java Servlet Engine geladen wird. Hier sieht man den Teil der Konfiguration, der einen Datenbankpool für die Sessiondatenbank bereitstellt:

```
<resource-ref>
  <res-ref-name>jdbc/sessions</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <init-param driver-name="org.gjt.mm.mysql.Driver" />
  <init-param url="jdbc:mysql://localhost:3306/resin\_session"/>
  <init-param user="dbuser"/>
  <init-param password="dbpasswd"/>
  <init-param max-connections="5"/>
  <init-param enable-transaction="false"/>
</resource-ref>
```

Um die Sessionvariablen (siehe auch Abschnitt 4.3.2) in dieser Datenbank zu speichern, muss der Datenbankpool in der Konfiguration der Sessions bekannt gemacht werden³:

```
<session-config>
  <!-- Session definitionen hier -->
  <jdbc-store>
    <data-source>jdbc/sessions</data-source>
  </jdbc-store>
</session-config>
```

Somit wird eine Datenbankverbindung auf die Datenbank `resin_session` geöffnet, die den JDBC-Treiber `org.gjt.mm.mysql.Driver` verwendet. Die Aufgabe des Programmierers ist es, Variablen, die in die Session gespeichert werden sollen, der Session bekannt zu machen. Die hierfür verwendete Methode ist `setAttribute(String name, Object obj)` des Interface `HTTPSession`. Diese Datenbankpools können auch die herkömmliche Initialisierung von JDBC-Datenbanktreibern ablösen, in dem die Verbindung über eine definierte `<resource-ref>` der Konfiguration hergestellt wird:

```
javax.naming.Context env = (javax.naming.Context)
    new javax.naming.InitialContext().lookup("java:comp/env");
javax.sql.DataSource source = (javax.sql.DataSource)
    env.lookup("jdbc/formgenerator");
java.sql.Connection con = source.getConnection();
```

³**Anmerkung:** Sessionvariablen können auf unterschiedliche Weise persistent gemacht werden. Sie können in Dateien, im Hauptspeicher oder in Datenbanken gehalten werden. Die Auswahl wird bei der Konfiguration der Java Servlet Engine getroffen, tangiert aber Programmierer in keiner Weise.

Bei diesem Codefragment wird ein `Context`-Objekt instantiiert, das mit den Werten aus der Java Servlet Engine initialisiert wird. Ein spezieller `Context` ist die `<resource-ref>`, hier mit dem Namen `jdbc/formgenerator`. Diese Referenz wird als Datenquelle betrachtet und erlaubt eine `java.sql.Connection` zu öffnen. Auf diese Verbindung kann wie auf eine direkte Datenbankverbindung über `JDBC`⁴ zugegriffen werden.

4.3.2 Sessionvariablen

Es stellt sich immer die Frage, bei welchen Variablen sich der Overhead einer Session auszahlt. Es gibt keine generelle Antworten darauf. Jede Variable, die mehr als nur einmal gebraucht wird, kann in einer Session gespeichert, oder aber als HTTP-Parameter mitgegeben werden. Sessionvariablen sind beim Formulargenerator die Benutzerkennung, das Ablaufdatum des Benutzerkontos, der Authentifizierungsstatus, Rechte des Benutzers (Administrator oder normaler Benutzer) und ob Debugging-Meldungen angezeigt werden sollen oder nicht. Da diese Variablen einmal gesetzt und danach nicht mehr verändert werden, sind diese in der Session gespeichert.

Diese Variablen werden in der Methode `setupValues()`, die bei jedem Aufruf des Formulargenerators ausgeführt wird, aus der Session gelesen und stehen danach dem Programm zur Verfügung.

Variablen, die zum Navigieren gebraucht werden, sind hier nicht in der Session gespeichert, weil sich die Werte der Variablen mit jedem Zugriff sehr wahrscheinlich ändern. Diese Variablen werden über HTTP-Parameter übergeben und bei jedem Zugriff aktualisiert.

Zur Navigation stehen zwei Variablen zur Verfügung: `link` und `action`. `link` gibt an, um welchen Zweig des Formulargenerators es sich handelt – Benutzeradministration, Formularverwaltung, Rechte, Die Variable `action` beinhaltet, ob ein Datensatz neu angelegt, verändert oder gelöscht wird.

Neben diesen Hauptvariablen werden noch weitere, den Datensatz spezifizierende Variablen mitgegeben. Diese können zu Debug-Zwecken mit dem Parameter `debug=true` unter der Rubrik `Request Parameters` eingesehen werden.

⁴JDBC - Java DataBaseConnection

4.4 Usertracking

Das gesamte Sessionhandling reicht leider nicht aus, um Aktionen vollständig nachzuvollziehen. Vor allem Administratoren sollten die Möglichkeit haben das System zu beobachten, damit sie gegebenenfalls Aktionen rückgängig machen können. Aus diesem Grund werden alle Aktionen, die ein Benutzer tätigt, mitprotokolliert und in der Datenbank in der Tabelle `logging` gespeichert.

4.4.1 Logdatei

Die Logdatei ist nur für Administratoren sichtbar und beinhaltet alle Aktionen mit einem Datumstempel, Benutzerkennung und die Beschreibung der Aktion.

Hier ein kleiner Auszug aus der Logdatei:

```
[2001-03-22 13:34:52; UID: <1>; Username: <admin>]: Form "test" (36) updated.  
[2001-03-22 13:34:45; UID: <1>; Username: <admin>]: Form "feedback" (35) updated.  
[2001-03-22 13:34:20; UID: <1>; Username: <admin>]: Form "test2" (37) created.  
[2001-03-22 13:30:54; UID: <1>; Username: <admin>]: Form "test" (36) created.
```

Anhand dieses Beispiels läßt sich feststellen, dass der Benutzer `admin` mit der `id` 1 am 22. März um 13:30 Uhr das Formular `test` und vier Minuten später das Formular `test2` erstellt und danach das Formular `feedback` und das Formular `test` verändert hat.

Die Logdatei wird zwar Datei genannt, ist aber in Wahrheit eine Tabelle in der Datenbank (siehe Abbildung 4.1, Seite 44) mit dem Namen `logging`. In dieser Tabelle werden das Datum, die Benutzerkennung und eine Meldung gespeichert. Die Ansicht ist nach dem Datum in absteigender Reihenfolge (letzter Eintrag an erster Stelle) geordnet und kann auf Tage eingeschränkt werden.

Diese Tabelle enthält ein Feld mit dem Namen `error`, welches zum Speichern von Fehlermeldungen des Programms verwendet wird. Tritt ein Systemfehler auf, so wird eine Meldung in diesem Feld gespeichert. Administratoren haben dann die Möglichkeit, die Fehlermeldungen auszuwerten und entsprechende Maßnahmen zu ergreifen.

4.5 Eine Beispielsession

Dieses Beispiel soll demonstrieren, wie das Zusammenspiel von Sessionhandling und Usertracking im Formulargenerator funktioniert.

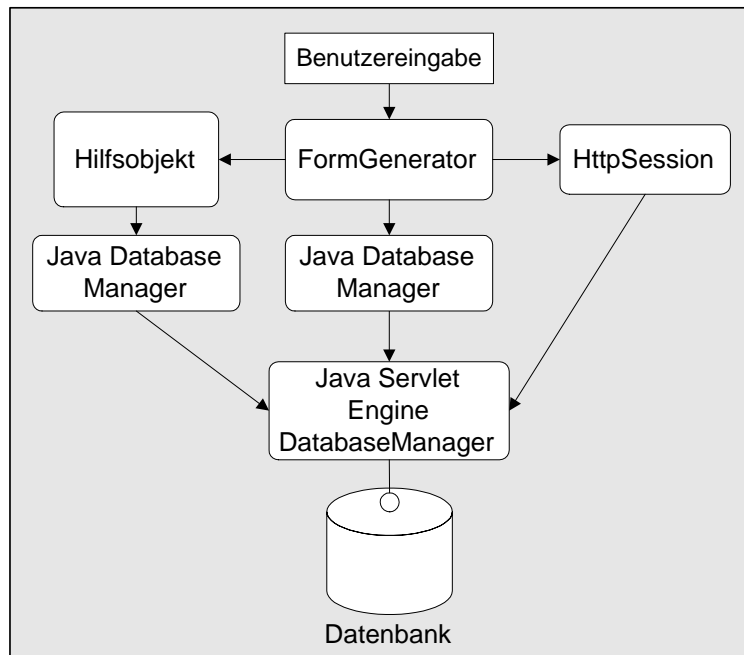
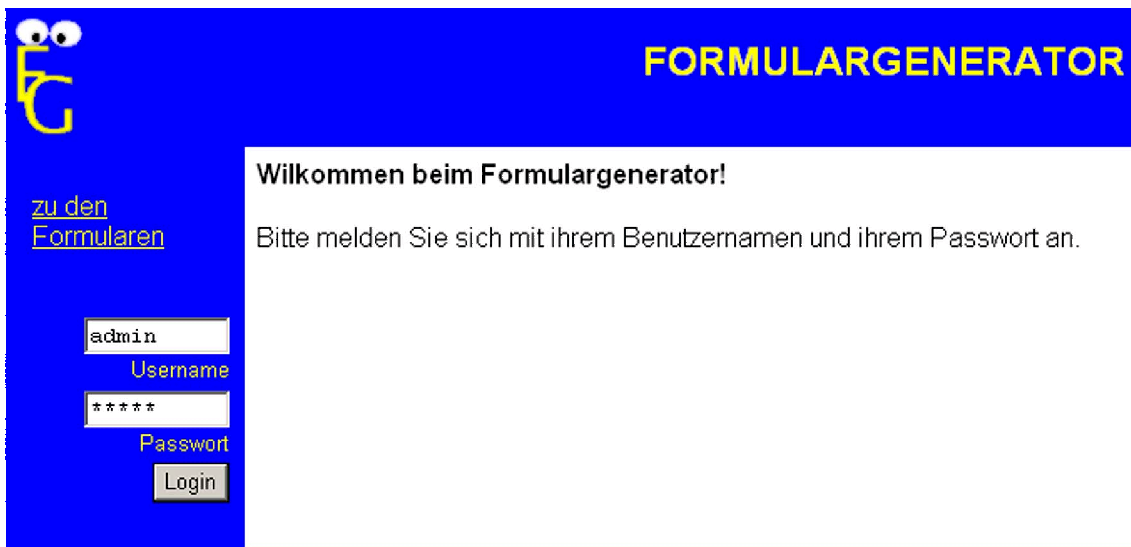


Abbildung 4.4: Datenfluss-Diagramm des Formulargenerators

4.5.1 Anmeldung

Beim Start des Formulargenerators wird ein Begrüßungsbildschirm gezeigt, wobei auf der linken Seite ein Benutzername - Passwort Eingabefeld zu sehen ist (siehe Abbildung 4.5). Der Benutzer authentifiziert sich über die Eingabe seines Benutzernamens und des dazugehörigen Passworts gegenüber der Datenbank. Wird ein Datensatz mit dem eingegebenen Benutzernamen in der Tabelle `user` gefunden und stimmt das Passwort mit dem im gefundenen Datensatz gespeichertem überein, so ist der Benutzer authentifiziert. Ist das Feld `isadmin` auf `yes` gesetzt, erhält der Benutzer Administratorenrechte, ansonsten gehört er der Gruppe der normalen Benutzer an.



zu den Formularen

admin
Username

Passwort

Login

FORMULARGENERATOR

Willkommen beim Formulargenerator!

Bitte melden Sie sich mit ihrem Benutzernamen und ihrem Passwort an.

Abbildung 4.5: Anmeldung am Formulargenerator als Administrator

Die Variablen Benutzername, Benutzerkennung (eine Integerzahl in der Datenbank), ist der Benutzer Administrator oder nicht und das Ablaufdatum des Benutzerkontos werden in die Session gespeichert. Damit braucht sich der Benutzer kein weiteres mal anzumelden.

Des weiteren wird ein Eintrag "User logged in." mit der entsprechenden Benutzerkennung in der Tabelle `logging` gespeichert.

4.5.2 Upload

Unter der Annahme, dass sich ein normaler Benutzer angemeldet hat, darf dieser Benutzer ein Formular auf den Server laden. Falls die Session keine Authentifizierungsvariablen enthält, wird die Fehlermeldung "Sie müssen sich einloggen, um diese Funktion benutzen zu können!" gezeigt.

Der Benutzer wird aufgefordert, eine HTML-Datei, einen Formularnamen und eine E-Mail-Adresse anzugeben. Bei Bestätigung wird die HTML-Datei auf den Server geladen.

Jetzt wird der Benutzer gefragt, ob eine Tabelle für das Formular angelegt werden soll. Bei Bestätigung wird die HTML-Datei in ein Java-DOM-Objekt gewandelt, der Name der resultierenden Klasse in der Tabelle `forms` im Feld `url` und der

Name des Formulars im Feld `name` gespeichert und eine Tabelle mit dem Namen des Formulars erzeugt.

Diese Aktion wird im Log mit der Meldung "Form <NAME> (ID) created." und der dazugehörigen Benutzerkennung vermerkt.

4.5.3 Formularverwaltung

Die Formularverwaltung ermöglicht es ein eben generiertes Formular aktiv zu setzen. Standardmäßig sind kreierte Formulare nicht aktiv. Der Benutzer muss den Auswahlknopf `Aktiv` markieren und mit einem Klick auf `Update` bestätigen.

Sollte der Benutzer sich nicht angemeldet und über einen `QUERY_STRING` die Navigationsvariablen `link` und `action` so gesetzt haben, dass er die Formularverwaltung zu sehen bekäme, wird wiederum die Fehlermeldung "Sie müssen sich einloggen, um diese Funktion benutzen zu können!" gezeigt.

Auch diese Aktion wird im Log mit der entsprechenden Benutzerkennung vermerkt.

4.5.4 Abmelden

Während der gesamten Session bleiben die Sessionvariablen unberührt, sie werden nur ausgelesen, um den Status der Authentifizierung zu bekommen bzw. um den Benutzer zu erkennen. Beim Logout werden alle Authentifizierungsvariablen gelöscht, die Session bleibt jedoch weiterhin bestehen. Somit kann sich der Benutzer unter einem anderen Namen anmelden (zum Beispiel als Administrator) und andere Aufgaben erfüllen.

Eine andere Möglichkeit die Session zu beenden ist den Browser zu schließen. Dadurch geht die SID verloren und die serverseitige Wiedererkennung des Clients ist nicht mehr gegeben. Nach einem Timeout von 30 Minuten⁵ wird die Session auch serverseitig beendet und die Sessionvariablen gelöscht. Sollte es gelingen,

⁵Das Session-Timeout ist ein Parameter, der über die Konfigurationsdatei der Servlet-Engine gesetzt wird.

innerhalb dieser Zeitspanne die SID zu rekonstruieren, so bekommt der Benutzer seine alte Session mit allen gespeicherten Variablen wieder.

4.6 Programmierung

Der Formulargenerator ist eine 100% Pure JavaTM Anwendung auf der Basis von Java Servlets. Die gesamte Anwendung besteht aus 12 Klassen, wobei es zwei Servlets mit unterschiedlichen Aufgaben gibt. Das Servlet `FormGenerator` ist für die Administration des Formulargenerators zuständig und bietet die oben beschriebene Funktionalität. Das Servlet `ProcessForm` ist zur Darstellung der gespeicherten Formulare und zum Befüllen der Datenbank mit Benutzerdaten programmiert worden.

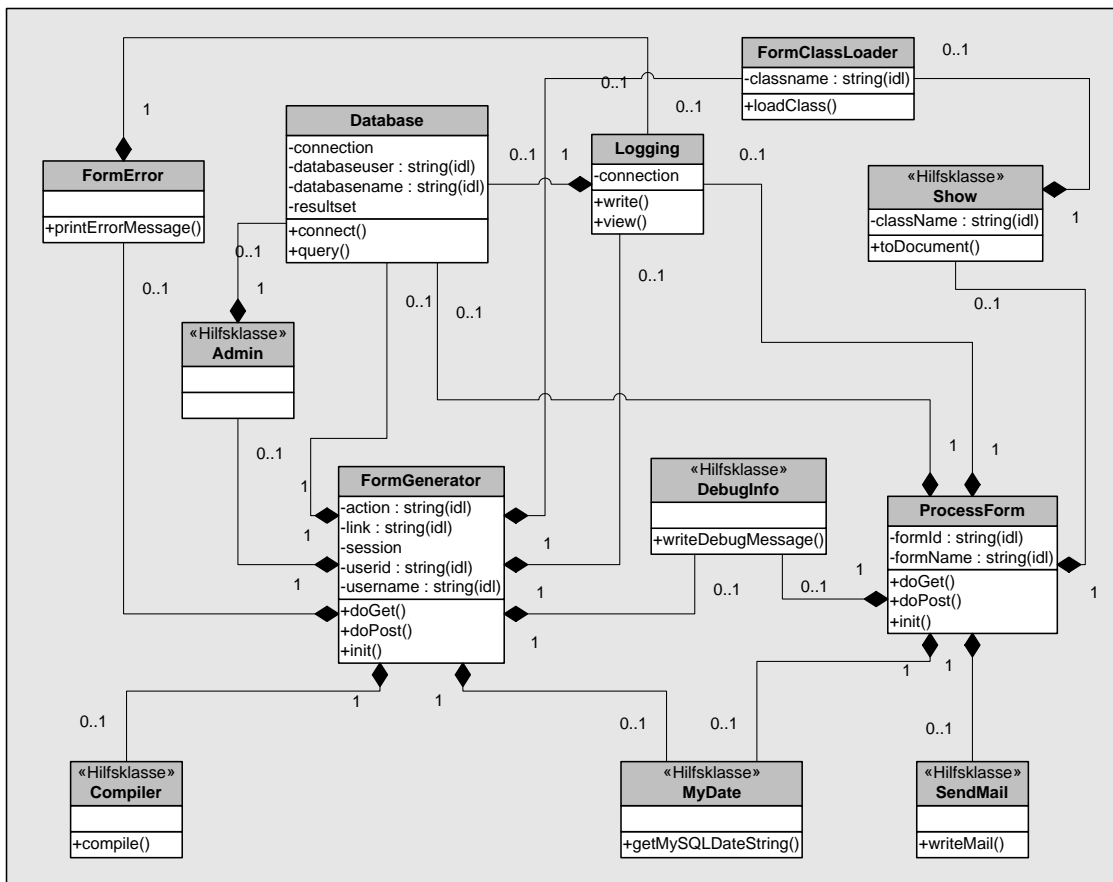


Abbildung 4.6: UML-Diagramm des Formulargenerators

In Abbildung 4.6, dem UML-Diagramm des Formulargenerators, erkennt man die Beziehungen der einzelnen Klassen zueinander. Die zwei Hauptklassen, die eben erwähnten Java Servlets bilden die Schnittstelle zum Benutzer, die Klasse Database ist die Schnittstelle zur Datenbank bzw. zum Datenbankverbindungs-pool der Java Servlet Engine.

Die Entscheidung, ob der Formulargenerator als Java Servlet, als PHP-Anwendung oder als CGI-Programm realisiert werden sollte, fiel sehr schnell auf die Servlet-Variante. Eine Probeversion wurde als PHP-Anwendung geschrieben. Probleme mit der Wartbarkeit von PHP-Code in diesem Ausmaß und die bessere Performance von Java Servlets waren die ausschlaggebenden Argumente für Java.

4.6.1 Die Klasse `FormGenerator`

Die Klasse `FormGenerator` stellt das Herzstück der Administration des Formulargenerators dar. Sie ist die Schnittstelle zum Benutzer und liefert alle HTML-Formulare, die zum Anlegen, Löschen, Ändern von Formularen und Benutzern benötigt werden. `FormGenerator` erbt von `HttpServlet`, das Bestandteil der Java Servlet API [SUN00b] ist. Ein Servlet nimmt eine `HttpServletRequest` entgegen und sendet eine `HttpServletResponse` an den Client zurück. Die Anfragen werden über HTTP gestellt; dem entsprechend besitzt ein Servlet die Zugriffsmethoden von HTTP: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `OPTIONS` und `TRACE`, wobei `GET` und `POST` wohl die wichtigsten HTTP-Methoden darstellen. Die Klasse `FormGenerator` hat die Methoden `GET` und `POST` implementiert, alle anderen Methoden rufen die `HttpServlet`-Methode `doGet(HttpServletRequest request, HttpServletResponse response)` auf.

4.6.1.1 Navigation

Die Navigation hält sich an einen Quasi-Standard des Web-Designs. Auf der linken Seite sind alle zugänglichen Menüpunkte aufgelistet. Diese Links rufen das Servlet mit der `GET`-Methode auf, d.h. die Servlet-Methode `doGet()` tritt in Aktion. Anhand der Parameter `link` und `action` wird nun festgestellt, welche Seite der Benutzer gerne sehen würde. `link` ist vom Typ `java.lang.String` und kann folgende Werte annehmen:

1. `link = Admin` führt zur Administration der Benutzer.
2. `link = Forms` führt zum Zweig der Formularverwaltung.
3. `link = Perms` führt zur Administration der Benutzerrechte.
4. `link = Upload` lädt das Upload-Formular.
5. `link = ViewLog` zeigt die Logdatei an.

Jede Veränderung der Werte führt zu einer POST-Anfrage, wobei die Variable `link` denselben, die Variable `action` einen die Aktion beschreibenden Wert hat.

1. `action = update` überschreibt den in der Datenbank vorhandenen Datensatz mit neu eingegebenen Daten.
2. `action = delete` löscht einen Datensatz.
3. `action = Login` wertet die Variablen `username` und `passwd` aus.
4. `action = Logout` löscht die Sessionvariablen.
5. `action = ...`

Weder `link` noch `action` werden in der Session gespeichert, da sich die Werte dieser beiden Variablen mit jedem Zugriff ändern.

4.6.1.2 Verwaltung

Die Verwaltung von Benutzern, Formularen und Rechten beruhen alle auf dem selben Schema. Der Benutzer, sofern eingeloggt, sieht ein Formular mit dem aktuellen Daten. Alle Daten, die der Benutzer ändern darf, werden auch zum Ändern angeboten. Änderungen werden mit dem Button "Ändern" übernommen und sofort in die Datenbank geschrieben.

Das Löschen von Datensätzen wird über den Button "Löschen" vorgenommen. Ein Java-Skript Dialog fordert zur erneuten Bestätigung auf, damit ein Datensatz nicht versehentlich gelöscht wird.

Alle Datenbanktransaktionen werden über die Klasse `Database` (siehe auch Abschnitt 4.6.4, Seite 60) behandelt.

4.6.1.3 Upload

Der Upload einer HTML-Datei und deren Umwandlung in ein Java-DOM-Objekt ist eine der Hauptfunktionalitäten der Klasse `FormGenerator`. Für den Datei-Upload wird ein sogenannter `MultipartRequest` verwendet, da Dateien mit dem MIME-Typ `multipart/form-data` kodiert werden. Die Standardklasse `HttpServletRequest` versteht diesen MIME-Typ nicht, daher wird hier die zusätzliche Bibliothek `Multipart` von O'Reilly verwendet. Die Klasse `MultipartRequest` zerlegt die POST-Request in Dateien und Parameter. [O'R01]

Die Datei ist nun auf dem Server gespeichert. Der Benutzer bekommt einen Dialog zu sehen, ob die Datenbank angelegt werden soll oder nicht. Wird der Dialog bestätigt, so kommt ein XML-Compiler zum Einsatz, der von Enhydra entwickelt worden ist. Der XMLC ist eigentlich eine Stand-Alone Anwendung, mit einem kleinen Trick lässt sich XMLC aber sehr schnell in eine Servlet-Umgebung einbinden. Die Klasse `Compiler` erbt von der Klasse `XMLC` und hat die wichtigsten Methoden implementiert. Mit der Methode `compile(String options)` wird das DOM-Objekt generiert und mit Hilfe der Klasse `FormClassLoader` instantiiert. Diese Instantiierung ist notwendig, um mit der Methode `HTMLObjectImpl.getForms()` die Formularelemente herauszufinden. Jetzt wird eine nach dem Namen des Formulars benannte Tabelle in der Datenbank erstellt. Von jedem Element des Formulars wird das Attribut `NAME` ausgelesen und ein Feld in der eben angelegten Tabelle angelegt. Ausgenommen sind Formularelemente vom Typ `HIDDEN`, `RESET` und `SUBMIT`, da es wenig Sinn macht, nicht von Benutzern eingegebene Daten zu speichern.

Tritt während der gesamten Aktion kein Fehler auf, so wird der Link, der das Formular lädt, angezeigt. Versucht man das Formular zu laden, erhält man allerdings die Fehlermeldung, dass das Formular noch nicht aktiv ist. Standardmäßig werden Formulare nicht sofort aktiviert, sondern müssen vom Formulareigentümer oder von einem Formularadministrator aktiviert werden. Dies soll einen zu frühen Zugriff durch neugierige Benutzer verhindern.

4.6.2 Die Klasse `ProcessForm`

Die Klasse `ProcessForm` lädt gespeicherte Formulare und zeigt diese an. Weiters bildet sie die Schnittstelle zwischen dem Formular und der Datenbank. Wird das Servlet `ProcessForm` ohne Parameter aufgerufen, so werden alle vorhandenen aktiven Formulare in einer Select-Box zur Auswahl angeboten. Wird ein Formular ausgewählt, so wird die Java DOM-Klasse des Formulars über eine Instanz der Klasse `FormClassLoader` geladen und mit einer Methode eines `Show`-Objektes dargestellt.

4.6.2.1 Formulare anzeigen

Die Anzeige eines Formulars wird über mit Hilfe der Klasse `Show` vorgenommen. `Show` bekommt den Namen der Klasse geliefert, instantiiert einen `FormClassLoader`, der die mit Namen bekannte Klasse lädt, ändert die `ACTION` des Formulars auf das Servlet `ProcessForm` und ruft die Methode `toDocument()` auf, die für die Darstellung des DOM-Objektes zuständig ist.

Nachdem der Benutzer Daten eingegeben hat, werden diese vom Servlet `ProcessForm` nocheinmal angezeigt mit der Frage, ob diese Daten gespeichert werden sollen.

Wird das Speichern bestätigt, bemüht sich `ProcessForm` einer Instanz der Klasse `Database`, um die Daten in die entsprechende Tabelle zu schreiben. Zudem wird eine E-Mail mit einem Link auf den Datensatz verfasst und an eine in der Datenbank hinterlegte Adresse geschickt.

4.6.2.2 Datensätze anzeigen

Das Servlet `ProcessForm` beinhaltet auch die Funktionalität, um einen Datensatz ansehen zu können. Die Parameter `link`, `dataid` und `formname` kommen hier zum Einsatz und haben folgende Werte:

```
link    = "ShowData"  
dataid  = "<DATENSATZNUMMER>"  
formname = "<FORMULARNAME>"
```

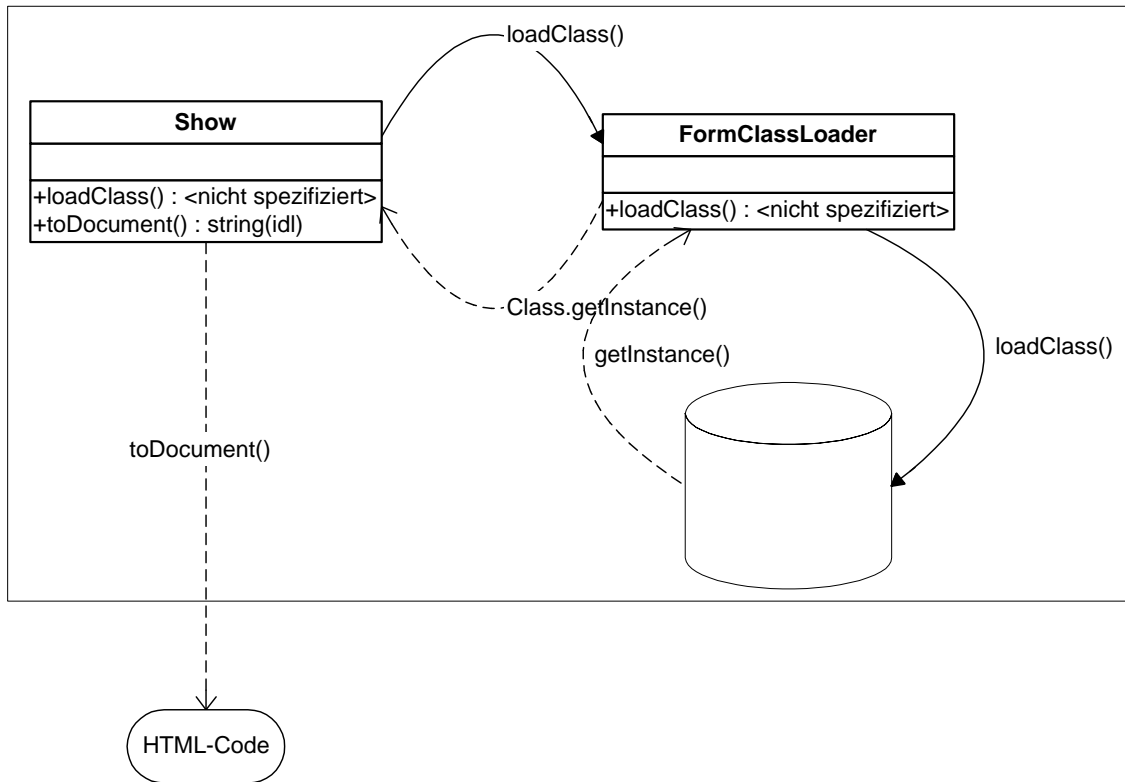


Abbildung 4.7: Anzeigen eines Formulars

Der Datensatz wird aus der Datenbank gelesen und in tabellarischer Form dargestellt.

4.6.3 Die Klasse `SendMail`

Die Klasse `SendMail` beruht auf der `JavaMail 1.2 API [SUN00c]` und kann eine simple Text-Mail verschicken. Es wird eine Nachricht versendet, die einen Link auf einen eingegebenen Datensatz enthält.

Die Methode `writeMail(String dataId, String formName)` baut eine Verbindung zur Datenbank auf, liest die für die Tabelle `formName` hinterlegte E-Mail Adresse und generiert eine Nachricht. Diese Nachricht enthält eine kurze Erklärung, den Namen des Formulars, die Datensatznummer und einen Link auf das Servlet `ProcessForm` mit den Parametern zum Anzeigen des Datensatzes.

4.6.4 Die Klasse Database

Die Klasse `Database` ist die Schnittstelle zwischen dem Servlet und der Datenbank. Hier werden die Verbindung verwaltet und alle Anfragen an die Datenbank geschickt und eventuelle Fehler behandelt.

Die wichtigsten Methoden dieser Klasse sind `connect()` bzw. `connect(String databaseName)` und `query(String databaseQuery)`. Die Verbindung wird über die Java Servlet Engine aufgebaut und von dieser Klasse verwendet (siehe auch Abschnitt 4.3.1, Seite 47).

4.6.5 Die Klasse Compiler

`XMLC` ist eigentlich ein eigenständiges Programm, das auf Kommandozeilenebene ausgeführt wird. Weil es in Java geschrieben wurde, ist es aber möglich, die Klasse dynamisch zu instantiieren bzw. eine Kindklasse zu schreiben, die die wichtigsten Methoden übernimmt und adaptiert.

`Compiler` ist eine Kindklasse von `XMLC` und hat sowohl die Methode `public void compile(String[] args)` als auch die Methode `private MetaData parseArgs(String[] args)` übernommen. Zusätzlich wurde die Methode `compile(String[] args)` durch eine weitere, `public void compile(String options)`, ergänzt, die den `String options` in ein `String[]` umwandelt und die ursprüngliche Methode `public void compile(String[] args)` aufruft.

4.7 Einsatz

Die Verbreitung des Formulargenerators beschränkt sich vorläufig auf die Abteilungen der Firma TIWAG Tiroler Wasserkraftwerke AG und wird dort von den jeweiligen Webmastern eingesetzt, um Daten aus den Online-Formularen zu sammeln. Es sind Überlegungen im Gange, den Sourcecode des Programms unter der GPL⁶ freizugeben.

⁶GPL - GNU Public License

Die Testphase des Formulargenerators wurde erfolgreich abgeschlossen, die Funktionalität und die Bedienung entspricht den Vorstellungen des Auftraggebers.

4.7.1 Auswirkungen

Vom Einsatz des Formulargenerators erwarten sich alle Beteiligten eine einfache Lösung zum Problem mit der händischen Weiterleitung von Kundenabfragen. Auch sollte das Problem, dass immer wieder Datensätze verloren gingen, behoben sein.

Die Installation des Formulargenerators auf den Produktivsystemen der Firma TIWAG Tiroler Wasserkraftwerke AG erwies sich als einfach, da der Entwicklungsrechner eine direkte Abbildung des Produktivsystems darstellt.

Es wurde begonnen, die Intranet-Formulare versuchsweise mit dem Formulargenerator zu verwalten. Eines der wichtigsten Formulare ist der sogenannte *User Helpdesk*, bei dem Fehlermeldungen, Wünsche und Anregungen eingereicht werden. Mit Hilfe des Formulargenerators wurde dieses Formular an eine MySQL-Datenbank angebunden und läuft zur Zeit als Versuch. Sollte dieser Versuch sich als Erfolg erweisen, werden alle anderen Formulare, die Daten sammeln, mit dem Formulargenerator verknüpft werden.

4.8 Zusammenfassung

Der in Java geschriebene Formulargenerator dient dazu, bestehende Websites an eine beliebige Datenbank anzubinden. Dazu werden die Webformulare in DOM-Objekte umgewandelt, die auf Formularelemente untersucht werden. Wird der Parser fündig, wird eine Tabelle in der Datenbank angelegt, in die sämtliche, über das Formular eingegebene Daten gespeichert werden.

Dieser Ansatz wurde gewählt, um die bestehenden Webformulare der Firma TIWAG Tiroler Wasserkraftwerke AG relativ rasch an eine Datenbank anbinden

zu können. Auf diese Weise werden alle Daten in einer zentralen Datenbank gesammelt und können zu einem späteren Zeitpunkt ausgewertet werden.

Kapitel 5

Schlussfolgerungen

Dieses Kapitel beinhaltet eine Zusammenfassung der wichtigsten Punkte des Sessionhandlings und des Usertrackings und zieht die daraus resultierenden Schlüsse.

5.1 Sessionhandling vs. Usertracking

Sessionhandling und Usertracking verfolgen andere Ziele, die aber mit denselben oder ähnlichen Techniken erreicht werden.

Die Ziele des Sessionhandlings sind der Austausch von Statusinformationen über die Dauer der Session zwischen zwei oder mehreren Hosts, unter der Verwendung eines verbindungslosen Übertragungsprotokolls wie zum Beispiel HTTP.

Usertracking hingegen verfolgt längerfristige Ziele. Es dient dazu, das Online-Verhalten von Personen zu studieren, um die gewonnenen Daten für kommerzielle Zwecke zu verwenden. Mit Hilfe dieser Daten können Bannerwerbungen gezielt gesetzt, Spam E-Mails verfasst und das Informationsangebot für einzelne Benutzer gefiltert bzw. aufbereitet werden.

5.2 Sessionhandling in Webanwendungen

5.2.1 Benutzerverwaltung

Webanwendungen bieten immer mehr Interaktivität. Techniken des Sessionhandling machen es möglich, Benutzer zu unterscheiden. Diese Unterscheidung ist unbedingt erforderlich, da ansonsten keine benutzerspezifischen Daten gespeichert werden könnten. Um die Sicherheit einer Webanwendung zu gewährleisten, müssen Benutzerverwaltungssysteme im Web erstellt werden. Eine feine Granulierung dieser Benutzerverwaltung wäre ohne Sessionhandling nicht möglich. Es gäbe genau zwei Möglichkeiten - Zugriff erlaubt oder Zugriff verboten. Mit Hilfe einer Session können Benutzername und Gruppenzugehörigkeiten gespeichert werden und somit eine Feinabstimmung der Zugriffsrechte vorgenommen werden.

5.2.2 Online-Shops

Doch Sessionhandling ist nicht nur für Zugriffsrechte notwendig, sondern auch für die Persistenz von Daten. Ein Online-Shop ohne Shopping-Cart ist unvorstellbar. Der virtuelle Einkaufswagen besitzt Funktionen, mit der Daten am Server für die Dauer des virtuellen Einkaufsbummels gespeichert werden können. Meistens

versteckt sich dahinter eine Datenbank, die die Artikelnummern und die Menge der in den Einkaufswagen gelegten Waren mit der SID des jeweiligen Benutzers in Verbindung setzt.

Werden die Daten nicht mit dem Ende der Session gelöscht, so hinterlässt ein Kunde ein Profil, auf Grund dessen Produktvorschläge gemacht werden können. Wenn der Kunde namentlich bekannt ist (zum Beispiel mit einem Benutzernamen, einer E-Mail Adresse, ...), so kann dieses Profil immer mit dem selben Kunden in Verbindung gebracht und durch zusätzliche Informationen über den Benutzer respektive seiner Interessensgebiete verfeinert werden. Auf diese Weise kann aus einem Überangebot von Waren potentiell interessante Ware gefiltert werden.

5.2.3 Navigation

Viele Websites besitzen ein ausgeklügeltes Navigationssystem, mit dem relativ rasch die gewünschte Information gefunden werden kann. Auf Grund von gespeicherten Sessions können sogenannte *Quicklinks* eingerichtet werden, die Links auf die bisher aufgesuchten Seiten speichern und so ein rasches Navigieren ermöglichen.

Die Benutzerfreundlichkeit einer solchen Site kann dadurch verbessert werden, dass Pfade zu häufig benutzten Informationsangeboten vereinfacht werden. Für eine solche Auswertung müssen die Sessioninformationen ebenfalls längerfristig gespeichert bleiben. Sobald aber Sessiondaten statistisch ausgewertet und zu anderen Zwecken als zum Statusaustausch verwendet werden, spricht man bereits von Usertracking.

5.3 Methoden des Sessionhandlings

Sessionhandling bedient sich einer SID, die am Beginn einer Session generiert wird. Anhand dieser SID ist ein Client jederzeit zu Daten, die auf einem Server gespeichert sind, zuordenbar. Die SID kann über mehrere Wege übermittelt werden.

5.3.0.1 Cookies

Cookies sind kleine, am Client gespeicherte Dateneinheiten, die mit jeder HTTP-Anfrage bzw. -Antwort mitgeschickt werden. Sie enthalten einen Namen und einen Datenteil. Cookies sind an Domänen gebunden, damit sie nicht von Dritten ausspioniert werden können.

In der Vergangenheit sind immer wieder Sicherheitslücken in Browsern gefunden worden, die unbeschränkten Zugriff auf Cookies gewährt haben. Oftmals werden Cookies als Speicher für Passwörter oder Kreditkartennummern verwendet, welche beliebte Angriffsziele für Hacker darstellen.

Moderne Browser überlassen es dem Benutzer die Wahl, ob und welche Cookies angenommen werden sollen.

5.3.0.2 URL-Kodierung

Unter URL-Kodierung versteht man einen mit Daten versehenen URL. Dabei hängt es von der verwendeten Programmiersprache und der Bibliothek ab, wie diese Daten kodiert werden. PHP zum Beispiel übergibt die SID als Parameter, Java hängt die SID mit einem Semikolon getrennt an den URL.

5.3.0.3 weitere Methoden

Die simpelste Methode ist, Sessiondaten clientseitig zu speichern und bei jedem Zugriff auf den Server als HTTP-Parameter mitzugeben. Diese Methode funktioniert immer, erfordert aber einen erhöhten Programmier- und sicherheitstechnischen Aufwand.

Ähnlich der URL-Kodierung kann die SID auch mit Hilfe eines Location-Headers in den URL eingebaut werden. Diese Methode nützt den HTTP-Errorcode 302, der besagt, dass eine Resource temporär unter einer anderen Adresse zu finden ist und sendet diese Adresse als HTTP-Header Location mit. Dieser neue URL beinhaltet die selbe Adresse, die mit einer SID versehen worden ist.

5.3.1 Fazit

Webanwendungen ohne Sessionhandling sind heute nicht mehr denkbar. Statusinformationen können über verschiedene Wege übermittelt werden. Wenn Cookies als Datenspeicher verwendet werden, so sollte ein Fallback-Mechanismus implementiert sein, der das Sessionhandling auch ohne Cookies gewährleistet.

5.4 Usertracking im Web

Usertracking bedient sich der Naivität vieler Webbenutzer. Durch den lockeren Umgang mit Daten wird es vielen Datensammlern leicht gemacht, Benutzerprofile anzulegen, die Interessensgebiete, Altersgruppe, Kaufkraft, etc. beinhalten.

5.4.1 Der Handel mit Daten

Gezielte Werbung ist teuer und Adressen, die mit einem Benutzerprofil gekoppelt sind, können bis zu mehreren 1000 Mark kosten.

Firmen wie DoubleClick haben sich auf das Sammeln und Aufbereiten von Daten spezialisiert und bieten eine Plattform für Online-Werbung. Verträge mit Homepagebetreibern ermöglichen es, gezielte Bannerwerbung zu platzieren. Diese Bannerwerbung führt aber nicht direkt zum Ziel, sondern leitet den Surfer erst über DoubleClick, damit dort neue Daten gespeichert werden können.

Adresshändler wie die Firma Schober befassen sich seit Jahren damit, Adressen nach Zielgruppen einzuteilen und diese Daten an Firmen, die gezielte Werbung platzieren wollen, weiterzuverkaufen.

5.4.2 Methoden des Usertrackings

5.4.2.1 Logdateien

Aus Logdateien lässt sich feststellen, welcher Client zu welcher Zeit welche Resource angefordert hat. Da aber große Netzwerke meist über Proxyserver auf

Internetserver zugreifen, wird als Client nur der Proxy registriert. Dies hat den Nachteil, das nicht das Surfverhalten eines einzelnen Benutzers bzw. Hosts sondern das eines gesamten Netzwerks protokolliert wird.

5.4.2.2 Cookies

Die Firma DoubleClick verwendet Cookies, um Personen zu identifizieren. In so einem Cookie wird u.a. eine ID gespeichert, die einen Host eindeutig identifiziert. Bei jedem Zugriff auf DoubleClick (zum Beispiel über eine Bannerwerbung oder eine Suchmaschine) wird das Cookie ausgelesen und der Link und der Referrer gespeichert. Auf diese Weise wird das Surfverhalten mit jedem Klick mitprotokolliert.

5.4.3 Schutz vor Datensammlern

Den ultimativen Schutz vor Datensammlern gibt es nicht. Alleine durch das Besuchen von Websites können persönliche Interessen herausgefunden und durch die Tageszeit, an der hauptsächlich gesurft wird, zumindest teilweise der Tagesrhythmus bestimmt werden.

Wirksamen Schutz gegen das Ausspionieren persönlicher Daten bieten Anonymizer, Proxyserver und Firewalls und das Nicht-Anklicken von Bannerwerbung.

5.4.4 Fazit

Usertracking steht, im Gegensatz zu Sessionhandling, zwischen zwei Fronten. Die kommerzielle Nutzung von gesammelten Daten ist ein einträgliches Geschäft. Marktforschung und Werbung bedienen sich gesammelter personenbezogener Daten.

Auf der anderen Seite stehen Datenschützer, die durch das Sammeln personenbezogener Daten einen Eingriff in die Privatsphäre von Privatpersonen sehen.

Es existieren wirksame Schutzmechanismen gegen die meisten Datensammelmethoden, aber diese stehen oft im Gegensatz mit Benutzerfreundlichkeit. Jeder

Benutzer muss selber abwägen, wieviel er von sich offen legt, denn einmal ausgesendete Daten sind, wenn überhaupt, nur sehr schwer wieder einzufangen.

5.5 Formulargenerator

Der Formulargenerator soll helfen, Online-Formulare zu verwalten. Mit der Hilfe des Formulargenerators lassen sich bestehende Web-Formulare an eine Datenbank anbinden.

5.5.1 Sessionhandling

Das Sessionhandling des Formulargenerators dient der Sicherheit. Ein Benutzer kann sich anmelden und je nach Berechtigung arbeiten. Damit diese Berechtigungsinformationen nicht verloren gehen, bleiben Benutzername, -ID und Authentifizierungsstatus bis zum Ende der Session erhalten.

Navigationsvariablen ändern sich zu häufig, sodass sich der Overhead einer Speicherung der Werte in der Session nicht auszahlen würde.

5.5.2 Usertracking

Das Usertracking im Formulargenerator beschränkt sich auf eine Logdatei, in der festgehalten wird, wer was wann geändert hat. Dieses Protokoll ist notwendig, um Schritte nachvollziehen bzw. gegebenenfalls rückgängig machen zu können.

5.5.3 Geplante Auswirkungen

Mit Hilfe des Formulargenerators sollten in Zukunft keine Daten mehr verloren gehen, die über Webformulare eingegeben wurden. Der Formulargenerator soll helfen, die Arbeit der Webmaster zu erleichtern, in dem jeder Webmaster seinen Lieblingseditor zur Erstellung von Webformularen verwenden kann und nicht selber die Programmierung einer Datenbankanbindung vornehmen muss.

5.5.4 Eingetretene Auswirkungen

Der Formulargenerator ist zur Zeit nicht produktiv im Einsatz, die Testphase ergab aber ein vielversprechendes Resultat. Die durchwegs positive Resonanz führte dazu, dass der Formulargenerator im Juli bzw. August 2001 im Produktiveinsatz getestet und bewertet wird.

5.5.5 Ausblicke

Das Feedback der Reality-Tests wird in die nächsten Versionen des Formulargenerators einfließen. Die Benutzerführung und der Funktionsumfang wird den Bedürfnissen laufend angepasst werden.

Es sind Überlegungen im Gange, ob der Formulargenerator als OpenSource-Projekt weitergeführt werden oder ganz im Besitz der TIWAG bleiben soll. Diese Entscheidung wird wahrscheinlich erst im Herbst, nachdem der Formulargenerator voll im Produktivbetrieb integriert ist, fallen.

5.5.6 Fazit

Der Einsatz des Formulargenerators macht sich für die TIWAG Tiroler Wasserkraftwerke AG bezahlt, da nicht jeder Webmaster eine eigene Schulung über Datenbankprogrammierung besuchen muss. Des weiteren werden alle online gesammelten Daten zentral gespeichert. Dies erleichtert die Übersicht und die Datensicherung.

Die Webmaster können genau so weiterarbeiten wie bisher, Formulare werden mit dem Formulargenerator auf den Zielrechner geladen.

Die Datenbankunabhängigkeit ermöglicht es die Datenbankplattform auszutauschen, ohne dass eine Zeile Code geändert werden müsste. Dieser Umstand ermöglicht eine rasche Portierung auf eine geänderte Plattform.

Anhang A

Java Klassengerüste

A.1 Admin

```
public class Admin
    extends java.lang.Object
```

This class contains methods for retrieving data to be seen by administrators only

A.1.1 Constructor Detail

Admin

```
public Admin(Database _db)
```

A.1.2 Method Detail

getPerms

```
public java.sql.ResultSet getPerms(java.lang.String userid)
    throws java.sql.SQLException
```

This method collects the users permissions from the database.

Parameters:

userid - the userid

Returns:

a java.sql.ResultSet containing the users permissions

getUserAttributes

```
public java.sql.ResultSet getUserAttributes()
    throws java.sql.SQLException
```

This method collects all users attributes like name, info, expiration date, ... from the database.

Returns:

a java.sql.ResultSet containing all users attributes

getUser

```
public java.sql.ResultSet getUser(java.lang.String userid)
    throws java.sql.SQLException
```

This method collects the specified users permissions

Parameters:

userid - the userid

Returns:

a java.sql.ResultSet containing the specified users attributes

updateUser

```
public void updateUser(java.lang.String userid,
    java.lang.String user,
    java.lang.String passwd,
    java.lang.String isadmin,
    java.lang.String info,
    java.lang.String expires)
    throws java.sql.SQLException
```

This method updates the specified users attributes

Parameters:

userid - the userid, which user should be updated

user - update username

passwd - update password

isadmin - update whether or not the user has administrator's permissions

info - user information

expires - update expiration date

createNewUser

```
public void createNewUser(java.lang.String user,
    java.lang.String passwd,
    java.lang.String isadmin,
    java.lang.String info,
    java.lang.String expires)
    throws java.sql.SQLException
```

This method updates the specified users attributes

Parameters:

user - the username

passwd - the password

isadmin - whether or not the user has administrator's permissions

info - user information

expires - expiration date

deleteUser

```
public void deleteUser(java.lang.String userid)
    throws java.sql.SQLException
```

This method deletes an existing user

Parameters:

userid - the userid

updatePerm

```
public void updatePerm(java.lang.String userid,
    java.lang.String formid,
    java.lang.String perm)
    throws java.sql.SQLException
```

This method updates the specified users permissions

Parameters:

userid - the userid

formid - the id of the online-form

userid - the permission (either write or read)

deletePerm

```
public void deletePerm(java.lang.String userid,
    java.lang.String formid)
    throws java.sql.SQLException
```

This method deletes an existing permission

Parameters:

userid - the userid

formid - the id of the online-form

insertPerm

```
public void insertPerm(java.lang.String userid,
    java.lang.String formid,
    java.lang.String perm)
    throws java.sql.SQLException
```

This method creates a permission

Parameters:

userid - the userid

formid - the id of the online-form

userid - the permission (either write or read)

getAllForms

```
public java.sql.ResultSet getAllForms()
```

```
throws java.sql.SQLException
```

This method collects all existing forms from the database.

Returns:

a java.sql.ResultSet containing a list of attributes of all existing forms

getFormsByUser

```
public java.sql.ResultSet getFormsByUser(java.lang.String userid)
```

```
throws java.sql.SQLException
```

This method returns a java.sql.ResultSet of all existing forms owned by the specified user

Parameters:

userid - the userid

Returns:

a java.sql.ResultSet containing a list of attributes of all existing forms, owned by the specified user

delForm

```
public void delForm(java.lang.String formid)
```

```
throws java.sql.SQLException
```

This method deletes the database entries of the given form

Parameters:

formid - the id of the online-form

updateForm

```
public void updateForm(java.lang.String id,  
                       java.lang.String owner,  
                       java.lang.String active,  
                       java.lang.String email)
```

```
throws java.sql.SQLException
```

This method updates the meta data of the given form

Parameters:

id - the id of the online-form

owner - the owner of the online-form

active - whether or not the form is active, at the very moment

email - the e-mail address, which is used when anybody fills out an online-form

A.2 Compiler

```
public class Compiler
    extends org.enhydra.xml.xmlc.commands.xmlc.XMLC
```

This class compiles an HTML-file or an XML-file into a Java DOM object. It is based on XMLC, a project from Enhydra.

A.2.1 Constructor Detail

Compiler

```
public Compiler()
    Standard constructor
```

Compiler

```
public Compiler(java.lang.String options)
    This constructor takes the compileroptions as argument.
    Parameters:
        options - a java.lang.String containing the compiler options.
```

A.2.2 Method Detail

compile

```
public void compile(java.lang.String[] args)
    throws org.enhydra.xml.xmlc.XMLCException,
           java.io.IOException
    This method actually calls the compiler provided by the XMLC-Project
    Overrides:
        compile in class org.enhydra.xml.xmlc.commands.xmlc.XMLC
    Parameters:
        args - an array containing the compiler options
```

compile

```
public void compile(java.lang.String options)
    throws org.enhydra.xml.xmlc.XMLCException,
           java.io.IOException
    This method builds an array of Strings and calls compile(String[])
    Parameters:
        options - a java.lang.String containing the compiler options
```

A.3 Database

```
public class Database
    extends java.lang.Object
```

This class is the database manager class of the FormGenerator-Project and therefore handles the connection to the database.

A.3.1 Constructor Detail

Database

```
public Database()
```

The standard constructor. A connection to the database is opened. The database identifier `FormGenerator.DBNAME` is used.

See Also:

```
FormGenerator.DBNAME
```

Database

```
public Database(java.lang.String _dbName)
```

This constructor opens a connection to the given data-resource.

Parameters:

`_dbname` - a `java.lang.String` containing the resource name, as defined in the Java Servlet engine's configuration file.

A.3.2 Method Detail

connect

```
public java.sql.Connection connect()
```

```
throws java.lang.ClassNotFoundException,
        java.io.FileNotFoundException,
        java.io.IOException,
        java.sql.SQLException,
        javax.naming.NamingException
```

This method opens a connection to the database, using the standard identifier `FormGenerator.DBNAME`

Returns:

a `java.sql.Connection`.

See Also:

```
FormGenerator.DBNAME
```

connect

```
public java.sql.Connection connect(java.lang.String _dbName)
    throws java.lang.ClassNotFoundException,
           java.io.FileNotFoundException,
           java.io.IOException,
           java.sql.SQLException,
           javax.naming.NamingException
```

This method opens a connection to the database, using `_dbname` as resource identifier.

Parameters:

`_dbname` - a `java.lang.String` containing the resource name, as defined in the Java Servlet engine's configuration file

Returns:

a `java.sql.Connection`

query

```
public java.sql.ResultSet query(java.lang.String q)
    throws java.sql.SQLException
```

This method is used to query the database. It uses a connection opened earlier. If no connection is available, `reset()` is called. If that fails, an exception is thrown.

Parameters:

`q` - a `java.lang.String` containing the query, e.g. a SQL-command.

Returns:

a `java.sql.ResultSet`, containing data collected by the given command or null.

getNextField

```
public java.lang.Object getNextField()
    throws java.sql.SQLException
```

This method is a helper method. It walks through the `java.sql.ResultSet` field by field, returning `java.lang.Object` for each field. It uses an internal counter, which is increased by one, every time this method is called.

Returns:

a `java.lang.Object` specifying the field value.

reset

```
public void reset()
    throws java.lang.ClassNotFoundException,
           java.io.FileNotFoundException,
```

```
java.io.IOException,  
java.sql.SQLException,  
javax.naming.NamingException
```

This method resets the connection and internal counters. When the method is called while a connection exists, this connection is closed and a new one is opened.

destroy

```
public void destroy()
```

This method closes the connection.

A.4 DebugInfo

```
public class DebugInfo  
    extends java.lang.Object
```

This class is for debugging purpose only. If the FormGenerator servlet is called with the parameter debug=true, this class comes into action.

A.4.1 Constructor Detail

DebugInfo

```
public DebugInfo()
```

A.4.2 Method Detail

getParameterString

```
public static java.lang.String getParameterString  
(javax.servlet.http.HttpServletRequest req)
```

This method returns an HTML-table containing all HTTP-parameters given by the request.

Parameters:

req - the HttpServletRequest containing all parameters

Returns:

a java.lang.String containing an HTML-table with all parameter name and value pairs

getMultipartParameterString

```
public static java.lang.String getMultipartParameterString  
(com.oreilly.servlet.MultipartRequest mreq)
```

This method returns an HTML-table containing all HTTP-parameters given by the request. This method is called, when the request uses the MIME-type multipart-formdata, e.g. when a file is uploaded.

Parameters:

mreq - the MultipartRequest containing all parameters

Returns:

a java.lang.String containing an HTML-table with all parameter name and value pairs

getSessionParameterString

```
public static java.lang.String getSessionParameterString  
(javax.servlet.http.HttpSession ses)
```

This method returns an HTML-table containing all HttpSession-variables

Parameters:

ses - the HttpSession containing the session id

Returns:

a java.lang.String containing an HTML-table with all sessionparameter name and value pairs

getApplicationContext

```
public static java.lang.String getApplicationContext  
(javax.servlet.ServletContext application,  
javax.servlet.http.HttpServletRequest req)
```

This method returns an HTML-table containing all initial values.

Parameters:

application - the ServletContext

req - the HttpServletRequest

Returns:

a java.lang.String containing an HTML-table with all initial parameter name and value pairs

getHTTPHeaderString

```
public static java.lang.String getHTTPHeaderString  
(javax.servlet.http.HttpServletRequest req)
```

This method returns an HTML-table containing all HTTP-headers

Parameters:

req - the MultipartRequest containing all headers

Returns:

a java.lang.String containing an HTML-table with all header name and value pairs

A.5 FormClassLoader

```
public class FormClassLoader
    extends java.lang.ClassLoader
```

This class instantiates objects dynamically. It is derived from `java.lang.ClassLoader`

```
\subsection{Constructor Detail}
\paragraph{FormClassLoader}
\scriptsize\begin{verbatim}
public FormClassLoader()
```

A.5.1 Method Detail

loadClass

```
protected java.lang.Class loadClass(java.lang.String name,
                                     boolean resolve,
                                     java.lang.String _basedir)
    throws java.lang.ClassNotFoundException,
           java.lang.NoClassDefFoundError
```

This method loads a class specified by the variable name.

Parameters:

- name - the classname
- resolve - whether to resolve the class or not
- _basedir - directory to look for the class

Returns:

- a `java.lang.Class`-object of the instantiated class

See Also:

- `Class`, `ClassLoader`

A.6 FormError

```
public class FormError
    extends java.lang.Object
```

This class contains all error messages of the `FormGenerator` application.

A.6.1 Constructor Detail

FormError

```
public FormError()
```

A.6.2 Method Detail

getLogInAsAdminError

```
public static java.lang.String getLogInAsAdminError()
```

This method is called when a non-administrator tries to do anything that requires administrator permissions.

Returns:

a java.lang.String containing the error message

getLogInError

```
public static java.lang.String getLogInError()
```

This method is called when anybody tries to do anything that requires logging in.

Returns:

a java.lang.String containing the error message

getSQLExceptionError

```
public static java.lang.String getSQLExceptionError(java.sql.SQLException e,  
                                                    java.lang.String query)
```

This method is called when a java.sql.SQLException is thrown.

Returns:

a java.lang.String containing the error message

getIOExceptionError

```
public static java.lang.String getIOExceptionError(java.io.IOException e,  
                                                  java.lang.String query)
```

This method is called when a java.io.IOException is thrown.

Returns:

a java.lang.String containing the error message

getExceptionError

```
public static java.lang.String getExceptionError(java.lang.Exception e,  
                                                java.lang.String source)
```

This method is called when a java.lang.Exception is thrown.

Returns:

a java.lang.String containing the error message

getNoSuchUserError

```
public static java.lang.String getNoSuchUserError(java.lang.String username)
```

This method is called when anybody tries to log in and the username does not match the password for the specified user or that user does not exist.

Returns:

- a java.lang.String containing the error message

getUserExpiredError

```
public static java.lang.String getUserExpiredError(java.lang.String username)
```

This method is called when a user, whose account has expired, tries to log in.

Returns:

- a java.lang.String containing the error message

getUploadNotHTMLFileError

```
public static java.lang.String getUploadNotHTMLFileError  
(java.lang.String upload_file)
```

This method is called when anybody tries to upload something different than an HTML file

Returns:

- a java.lang.String containing the error message

getCannotLoadCompiledClassError

```
public static java.lang.String getCannotLoadCompiledClassError  
(java.lang.Exception e)
```

This method is called when an error occurred while trying to instantiate a DOM object.

Returns:

- a java.lang.String containing the error message

getCompilerErrorString

```
public static java.lang.String getCompilerErrorString(java.lang.Exception e)
```

This method is called when a compiler error occurs.

Returns:

- a java.lang.String containing the error message

getNoTablenameError

```
public static java.lang.String getNoTablenameError()
```

This method is called when an HTML file is uploaded without a name for the form.

Returns:

a java.lang.String containing the error message

getTablenameExistsError

```
public static java.lang.String getTablenameExistsError(java.lang.String name)
```

This method is called when an HTML file is uploaded and the provided name for the form is already in use.

Returns:

a java.lang.String containing the error message

getNoFileError

```
public static java.lang.String getNoFileError()
```

This method is called when an HTML file is uploaded but no file has been chosen. form is already in use.

Returns:

a java.lang.String containing the error message

getFormNotActiveError

```
public static java.lang.String getFormNotActiveError(java.lang.String formName)
```

This method is called when the form is called but still marked as inactive. form is already in use.

Returns:

a java.lang.String containing the error message

A.7 FormGenerator

```
public class FormGenerator
```

```
extends javax.servlet.http.HttpServlet
```

This is a Java Servlet interface for HTML-forms or XML-forms to any JDBC-compliant database. One may upload any HTML-File containing the <form>-tag and this application sets up the database-connection to store online data.

See Also:

Serialized Form

A.7.1 Field Detail

SERVLETNAME

```
public static final java.lang.String SERVLETNAME
    Defining the servlet's name, currently "/formulargenerator/
    FormGenerator"; It's used by response.encodeURL(SERVLETNAME) to ensure
    sessions without cookies.
```

DBNAME

```
public static final java.lang.String DBNAME
    The databasename for the connection opened by the Java Servlet Engine.
    Currently set to jdbc/formgenerator
```

DEBUG

```
public static final int DEBUG
    Setting the LOGLEVEL to DEBUG
```

WARN

```
public static final int WARN
    Setting the LOGLEVEL to WARN
```

ERR

```
public static final int ERR
    Setting the LOGLEVEL to ERR
```

SILENCE

```
public static final int SILENCE
    Setting the LOGLEVEL to SILENCE
```

LOGLEVEL

```
public static final int LOGLEVEL
    Setting the program logoutput to LOGLEVEL
```

A.7.2 Constructor Detail

FormGenerator

```
public FormGenerator()
```

A.7.3 Method Detail

init

```
public void init()
    Setting up initial values
    Overrides:
        init in class javax.servlet.GenericServlet
```

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
    This method handles all clients communicating via HTTP GET
    Overrides:
        doGet in class javax.servlet.http.HttpServlet
    Parameters:
        request - HTTP-request from the client
        response - HTTP-response to the client
    See Also:
        Java Servlet API
```

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,
                   javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
    This method handles all clients communicating via HTTP POST
    Overrides:
        doPost in class javax.servlet.http.HttpServlet
    Parameters:
        request - HTTP-request from the client
        response - HTTP-response to the client
    See Also:
        Java Servlet API
```

doDelete

```
public void doDelete(javax.servlet.http.HttpServletRequest request,
                    javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
```

This method calls `doGet(request,response)`.

Overrides:

`doDelete` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - HTTP-request from the client

`response` - HTTP-response to the client

See Also:

Java Servlet API

doHead

```
public void doHead(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
```

This method calls `doGet(request,response)`.

Parameters:

`request` - HTTP-request from the client

`response` - HTTP-response to the client

See Also:

Java Servlet API

doOptions

```
public void doOptions(javax.servlet.http.HttpServletRequest request,
                     javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
```

This method calls `doGet(request,response)`.

Overrides:

`doOptions` in class `javax.servlet.http.HttpServlet`

Parameters:

`request` - HTTP-request from the client

`response` - HTTP-response to the client

See Also:

Java Servlet API

doPut

```
public void doPut(javax.servlet.http.HttpServletRequest request,
                 javax.servlet.http.HttpServletResponse response)
```


throws java.io.IOException
This method calls doGet(request,response).
Overrides:
doPut in class javax.servlet.http.HttpServlet
Parameters:
request - HTTP-request from the client
response - HTTP-response to the client
See Also:
Java Servlet API

doTrace

```
public void doTrace(javax.servlet.http.HttpServletRequest request,
                   javax.servlet.http.HttpServletResponse response)
    throws java.io.IOException
```

This method calls doGet(request,response).
Overrides:
doTrace in class javax.servlet.http.HttpServlet
Parameters:
request - HTTP-request from the client
response - HTTP-response to the client
See Also:
Java Servlet API

A.8 Logging

```
public class Logging
    extends java.lang.Object
```

This class provides the logging functionality.

A.8.1 Constructor Detail

Logging

```
public Logging()
```

throws java.io.IOException
This is the standard constructor creating a new instance of Database

A.8.2 Method Detail

write

```
public void write(java.lang.String logstring)
```

```
    throws java.io.IOException
```

This method writes the given parameter logstring into the logging-table anonymously.

Parameters:

logstring - a java.lang.String containing the logging message

write

```
public void write(java.lang.String logstring,
```

```
                java.lang.String userid)
```

```
    throws java.io.IOException
```

This method writes the given parameter logstring into the logging-table.

Parameters:

logstring - a java.lang.String containing the logging message

userid - the userid of the user, whose action caused the log entry

error

```
public void error(java.lang.String msg)
```

This method writes the given parameter msg into the logging-table.

Parameters:

logstring - a java.lang.String containing an error message

viewLogByDate

```
public java.lang.String viewLogByDate(java.util.Date d)
```

```
    throws java.io.IOException
```

This method collects all entries for the provided date.

Parameters:

d - a java.util.Date

Returns:

a java.lang.String containing a part of the log as HTML text.

viewLog

```
public java.lang.String viewLog()
```

```
    throws java.io.IOException
```

This method collects all log entries from the database.

Returns:

a java.lang.String containing the log as HTML text.

A.9 MyDate

```
public class MyDate
    extends java.util.Date
    implements java.util.Comparator
```

This class provides some methods used for date manipulations.

See Also:

Serialized Form

A.9.1 Constructor Detail

MyDate

```
public MyDate()
```

This is the standard constructor, instantiating a `java.util.Calendar` with the current date.

MyDate

```
public MyDate(long date)
```

This is a constructor, instantiating a `java.util.Calendar` with the specified date.

Parameters:

date - a long variable containing the milliseconds since 1972

A.9.2 Method Detail

getMySQLDateString

```
public java.lang.String getMySQLDateString()
```

This method creates a `java.lang.String` containing date information for the database. The date is the current date and the format is `YYYY-MM-DD HH:MM:SS`, as needed for most databases.

Returns:

a `java.lang.String` containing the current date information

getMySQLDateString

```
public java.lang.String getMySQLDateString(java.util.Date d)
```

This method creates a `java.lang.String` containing date information for the database. The format is `YYYY-MM-DD HH:MM:SS`, as needed for most databases.

Parameters:

d - a `java.util.Date`

Returns:

a `java.lang.String` containing the provided date information

getDay

```
public java.lang.String getDay(java.util.Date d)
```

This method creates a `java.lang.String` containing date information for the specified day for the database. The format is `YYYY-MM-DD HH:MM:SS`, as needed for most databases.

Parameters:

d - a `java.util.Date`, specifying the day

Returns:

a `java.lang.String` containing the provided date information

getNextDay

```
public java.lang.String getNextDay(java.util.Date d)
```

This method creates a `java.lang.String` containing date information for the specified day for the database. The format is `YYYY-MM-DD HH:MM:SS`, as needed for most databases.

Parameters:

d - a `java.util.Date`, specifying the day

Returns:

a `java.lang.String` containing the next day from the provided date information

compare

```
public int compare(java.util.Date d1,  
                  java.util.Date d2)
```

This method compares two dates.

Parameters:

d1 - a `java.util.Date`

d2 - a `java.util.Date`

Returns:

an `int` `< 0`, `= 0` or `> 0`, if d1 is earlier than d2, d1 is the same date than d2 or d1 is later than d2

compare

```
public int compare(java.lang.Object a,  
                  java.lang.Object b)
```

This method compares two `java.lang.Objects`.

Specified by:

compare in interface `java.util.Comparator`

Parameters:

a -

b -
Returns:
an int
< 0: if d1 is earlier than d2
= 0: d1 is the same date than d2 (or one of the given parameters
is no java.util.Date at all)
> 0: d1 is later than d2

equals

```
public boolean equals(java.util.Date d)
```

This method checks, whether a given date equals MyDate

Returns:
true, if the given date equals MyDate or false otherwise

before

```
public boolean before(java.util.Date d)
```

This method checks, whether a given date is before MyDate

Overrides:
before in class java.util.Date

Parameters:
d - java.util.Date to check

Returns:
true, if the given date is before MyDate or false
otherwise

after

```
public boolean after(java.util.Date d)
```

This method is the counterpart of before(java.util.Date)

Overrides:
after in class java.util.Date

See Also:
before(java.util.Date d)

A.10 ProcessForm

```
public class ProcessForm  
extends javax.servlet.http.HttpServlet
```

This is a Java Servlet, which was designed to operate as interface between the user and the database. This Servlet loads the forms and presents them to the user, who may fill in some data. This data is then send back to this servlet,

which stores it into the database. It also enables browsing and searching for data, which has been stored earlier.

See Also:

Serialized Form

A.10.1 Constructor Detail

ProcessForm

```
public ProcessForm()
```

A.10.2 Method Detail

init

```
public void init()
```

This method initializes some values, establishes a database connection and starts the servlet.

Overrides:

init in class javax.servlet.GenericServlet

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,  
                 javax.servlet.http.HttpServletResponse response)  
    throws java.io.IOException
```

This method handles all requests, which were transmitted using the GET-method of the HTTP-protocol.

Overrides:

doGet in class javax.servlet.http.HttpServlet

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,  
                  javax.servlet.http.HttpServletResponse response)  
    throws java.io.IOException
```

This method handles all requests, which were transmitted using the POST-method of the HTTP-protocol.

Overrides:

doPost in class javax.servlet.http.HttpServlet

A.11 SendMail

```
public class SendMail
    extends java.lang.Object
```

This class is responsible for sending mails. It uses the JavaMail 1.2 API.

A.11.1 Constructor Detail

SendMail

```
public SendMail(javax.mail.Session s)
```

The standard constructor

A.11.2 Method Detail

writeMail

```
public static void writeMail(java.lang.String dataId,
                             java.lang.String formName)
```

This method creates a mail and sends it to an address, which is stored in the database.

Parameters:

dataId - the id of the data within the database

formName - the name of the form

A.12 Show

```
public class Show
    extends java.lang.Object
```

This class loads a form and presents it to the user by calling its toDocument()-method.

A.12.1 Constructor Detail

Show

```
public Show()
```

```
throws java.lang.NullPointerException,
        java.lang.InstantiationException,
        java.lang.IllegalAccessException,
        java.lang.ClassNotFoundException
```

Wow dude, good guess - the STANDARD CONSTRUCTOR ;-)

Show

```
public Show(java.lang.String name,
            java.lang.String dir,
            java.lang.String action)
throws java.lang.NullPointerException,
       java.lang.InstantiationException,
       java.lang.IllegalAccessException,
       java.lang.ClassNotFoundException
This is another constructor, calling loadClass(String, String, String)
Parameters:
  name - the classname of the DOM object
  dir - the directory, where the DOM objects reside
  action - the action-attribute of the form
```

A.12.2 Method Detail

loadClass

```
public void loadClass(java.lang.String name,
                     java.lang.String dir,
                     java.lang.String action)
throws java.lang.NullPointerException,
       java.lang.InstantiationException,
       java.lang.IllegalAccessException,
       java.lang.ClassNotFoundException
This method loads a class and instantiates it.
Parameters:
  name - the classname of the DOM object
  dir - the directory, where the DOM objects reside
  action - the action-attribute of the form
```

toDocument

```
public java.lang.String toDocument()
throws java.lang.NullPointerException
This method presents the loaded DOM object to the user.
Returns:
  the HTML-code of the form
```


Anhang B

CREATE TABLE Definitionen

B.1 Tabelle forms

```
CREATE TABLE forms (  
  id bigint(21) NOT NULL auto_increment,  
  owner bigint(21) DEFAULT '0' NOT NULL,  
  name varchar(255) NOT NULL,  
  url varchar(255) NOT NULL,  
  active enum('yes','no') DEFAULT 'no' NOT NULL,  
  email varchar(255) NOT NULL,  
  creationtime datetime DEFAULT '0000-00-00 00:00:00' NOT NULL,  
  PRIMARY KEY (id),  
  UNIQUE name (name),  
  KEY form_id (name),  
  KEY creator_id (owner)  
);
```

B.2 Tabelle logging

```
CREATE TABLE logging (  
  id bigint(21) NOT NULL auto_increment,  
  date datetime DEFAULT '0000-00-00 00:00:00' NOT NULL,  
  msg varchar(255),  
  error varchar(255),  
  userid bigint(21) DEFAULT '0',  
  PRIMARY KEY (id)  
);
```

B.3 Tabelle permissions

```
CREATE TABLE permissions (  
  user_id bigint(21) DEFAULT '0' NOT NULL,  
  form_id bigint(21) DEFAULT '0' NOT NULL,  
  perms char(1) NOT NULL,  
  PRIMARY KEY (user_id, form_id),  
  KEY user_id (user_id, form_id)  
);
```

B.4 Tabelle user

```
CREATE TABLE user (  
  id bigint(21) NOT NULL auto_increment,  
  name varchar(255) NOT NULL,  
  passwd varchar(255) NOT NULL,  
  expire date DEFAULT '0000-00-00' NOT NULL,  
  info varchar(255) NOT NULL,  
  creationtime datetime DEFAULT '0000-00-00 00:00:00' NOT NULL,  
  isadmin enum('y','n') DEFAULT 'n' NOT NULL,  
  PRIMARY KEY (id),  
  UNIQUE name (name)  
);
```

Literaturverzeichnis

- [AD99] C. Allen und T. Dierks. *RFC 2246, TLS - Transport Layer Security*.
<http://rfc.net/rfc2246.html>, January 1999.
- [Ano00] Anonymizer.com. *Anonymizer.com - Privacy is your right*.
<http://www.anonymizer.com>, 2000.
- [BFM98] T. Berners-Lee, R. Fielding, und L. Masinter. *RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax*.
<http://rfc.net/rfc2396.html>, 1998.
- [BMM94] T. Berners-Lee, L. Masinter, und M. McCahill. *RFC 1738, Uniform Resource Locators (URL)*.
<http://rfc.net/rfc1738.html>, 1994.
- [BS01] Holger Bleich und Peter Schüler. *Digitale fußspuren. c't Magazin für Computertechnik*, 08:201, 2001.
- [CRS⁺99] Jesus Castagnetto, Harish Rawat, Sascha Schumann, Chris Scollo, und Deepak Veliath. *Professional PHP Programming*. Wrox Press Ltd., 1999.
- [Dou01] DoubleClick. *doubleClick.net Homepage*.
<http://www.doubleclick.net>, 2001.
- [E-M01] E-Media. *Internet aus der Steckdose*.
<http://www.e-media.at/internet/provider/strom/main.strom.asp>, 2001.
- [FGM⁺99] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, und T. Berners-Lee. *RFC 2616, Hypertext Transfer Protocol - HTTP/1.1*.
<http://rfc.net/rfc2616.html>, Juni 1999.

- [Gun96] Shishir Gundavaram. *CGI Programmin on the World Wide Web*. O'Reilly & Associates, Inc., 1996.
- [Hal01] Marty Hall. *Core Servlets and JavaServer Pages*. Prentice Hall PTR, 2001.
- [Jon97] Kevin Jones. *Advancing the Features of Channel Technology*.
http://www.w3c.org/Architecture/9709_Workshop/paper09/html/, 1997.
- [KM97] D. Kristol und L. Montulli. *RFC 2109, HTTP State Management Mechanism*.
<http://rfc.net/rfc2109.html>, 1997.
- [KM00] D. Kristol und L. Montulli. *RFC 2965, HTTP State Management Mechanism*.
<http://rfc.net/rfc2965.html>, 2000.
- [Kna00] Mag. Mario Knapp. Das neue Datenschutzgesetz. *Datagraph*, 01, 2000.
- [Lem01] Lemuria. *Location Poisoning*.
<http://www.lemuria.org/Software/unpoison/>, February 2001.
- [MSB99] Viktor Mayer-Schönberger und Ernst O. Brandl. *Datenschutzgesetz 2000*. Linde, 1999.
- [Mü98] Stefan Münz. *SelfHTML*.
<http://www.teamone.de/selfhtml>, 1998.
- [NCS97] NCSA. *NCSA Mosaic Homepage*.
<http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/>, 1997.
- [Net95] Netscape Communications Corp. *Persistent Client State*.
http://home.netscape.com/newsref/std/cookie_spec.html, 1995.
- [Nor01] Henrik Nordstrom. *Squid Proxyserver*.
<http://www.squid-cache.org>, February 2001.
- [O'R01] O'Reilly Network. *O'Reilly Network Servlet Package*.
<http://www.servlets.com/resources/com.oreilly.servlet>, 2001.
- [PER00] PERL. *PERL.com*.
<http://www.perl.com>, 2000.

- [PHP00] PHP. *PHP.net*.
<http://www.php.net>, 2000.
- [RS99] E. Rescorla und A. Schiffman. *RFC 2660, Secure Hypertext Transfer Protocol - S-HTTP/1.4*.
<http://rfc.net/rfc2660.html>, August 1999.
- [Sch01] Schober Information Group. *Schober Homepage Deutschland*.
<http://www.schober.de>, 2001.
- [SUN00a] SUN Corp. *Java 1.3 API*.
<http://java.sun.com/j2se/1.3/docs/api/index.html>, 2000.
- [SUN00b] SUN Corp. *Java Servlet API*.
<http://java.sun.com/products/servlet/2.2/javadoc/index.html>, 2000.
- [SUN00c] SUN Corp. *JavaMail 1.2 API*.
<http://java.sun.com/products/javamail/index.html>, 2000.
- [TIW01] TIWAG Tiroler Wasserkraftwerke AG. *The Blue Flash*.
<http://www.tiwag.co.at>, 2001.
- [Vog00] Tom Vogt. *Cypherpunks Hyperarchive - Usertracking by URL*.
<http://www.inet-one.com/cypherpunks/dir.2000.02.07-2000.02.13/msg00081.html>, February 2000.
- [W3C01] W3C. *The World Wide Web Consortium*.
<http://www.w3c.org>, 2001.

Stichwortverzeichnis

- CGI, 9
- Cookies, 16–20
 - Nachteile, 20
 - Netscape's Cookies, 19
 - Version 1, 16
 - Version 2, 18
 - Vorteile, 19
- Datenbankpools, 47
 - Java Sourcecode, 48–49
 - SessionDatenbank, 48
- Formulargenerator, 41–62
 - Beispielsession, 51–54
 - Abmelden, 53
 - Anmeldung, 51
 - Formularverwaltung, 53
 - Upload, 52
 - Benutzeradministration, 44
 - Datenbankdesign, 43
 - Einsatz, 60–61
 - Auswirkungen, 61
 - Formularadministration, 45
 - Formularansicht, 47
 - Formularupload, 46
 - Funktionalität, 43
 - Programmierung, 54–60
 - class Admin, A-1
 - class Compiler, 60, A-5
 - class Database, 60, A-6
 - class DebugInfo, A-8
 - class FormClassLoader, A-10
 - class FormError, A-10
 - class FormGenerator, 55–57, A-13
 - class Logging, A-17
 - class MyDate, A-19
 - class ProcessForm, 58–59, A-21
 - class SendMail, 59, A-23
 - class Show, A-23
 - Datenfluss-Diagramm, 51
 - Datensätze anzeigen, 58–59
 - Formulare anzeigen, 58
 - Navigation, 55–56
 - Upload, 57
 - Verwaltung, 56
- Sessionhandling, 47–49
- Usertracking, 50
 - Logdatei, 50
- HTML, 4–5
- HTTP, 5
 - Body, 5
 - Header, 5
 - Server, 6–7
- Java Applets, 2
- Java Servlets, 9–11
- Mosaic, 3
- Multi-Tier-Umgebungen, 11

- Netscape Communications Corp., 3
- Proxy, 7
- QUERY_STRING, 8–9
 - Aufbau, 8
- Schlussfolgerungen, 63–70
- Session, 12–13, 15
 - identifikation, 15
 - variablen, 49
 - Resin Konfiguration, 48
- Sessionhandling, 15–22
 - HTTP, 15
 - SID, 15
 - URL Kodierung, 20–22
 - Nachteile, 22
 - Vorteile, 22
- Sessionhandling in Webanwendungen,
64–65
 - Benutzerverwaltung, 64
 - Navigation, 65
 - Online-Shops, 64–65
- URL, 8
- Usertracking, 23–32
 - Anonymität, 28–30
 - Anonymizer, 29
 - Firewalls, 30
 - Datenschutz, 34
 - Methoden, 24–27
 - Cookies, 26
 - E.T. - Programme, 27
 - Location Poisoning, 27
 - Logdateien, 24
 - Web-Bug, 26
 - Proxies & DNS-Caches, 28
 - rechtliche Aspekte, 33–40
 - Datenerhebung, 38
 - Datenschutz und Internet, 38–39
 - Handel mit personenbezogenen Daten, 36–37
 - Kritik am Datenschutzgesetz, 39
 - Naivität der Benutzer, 38
 - Probleme, 37–38
 - Schutzwürdige Daten, 35–38
 - Verwenden von Daten, 35–36
- Verschlüsselung, 29