Internet

Grundlagen und Anwendungen mit Schwerpunkt World Wide Web

Johann Mitlöhner

März 2009

Inhaltsverzeichnis

1 TCP/IP und die Entwicklung des Internet

Das World Wide Web hat in wenigen Jahren weltweite Verbreitung gefunden und damit auch die Entwicklung des Internet in ungeahntem Ausmaß angetrieben. Internet und WWW sind feste Bestandteile der Informatikausbildung geworden. Ein wirkliches Verständnis der Konzepte hinter Internet und World Wide Web wird aber nur erreicht, wenn wir einen Blick hinter die Kulissen werfen, indem wir nicht nur die Benutzersicht, sondern auch die Entwicklersicht sehen. Daher werden wir uns nicht nur theoretisch mit den Ideen beschäftigen, die Internet und WWW ermöglicht haben, sondern auch selbst eine Webapplikation erstellen, und zwar eine einfache Warenwirtschaft mit Webshop und Datenbankanbindung. Wenn Sie einmal selbst ein solches System entwickelt haben, können Sie die Möglichkeiten und Grenzen der verwendeten Technologien viel besser einschätzen, als wenn Sie nur die Endbenutzersicht kennen.

In der Folge werden nur solche Technologien besprochen, die mit freier **Open Source** Software eingesetzt werden können. Solche Software ist im Source Code verfügbar, dadurch können weltweit viele Personen an der laufenden Verbesserung und Erweiterung mitarbeiten. Diese System sind auch für Sie zu Hause verfügbar, wenn Sie die Datenbank mysql, den Webserver Apache und den Applikationsserver Tomcat auf Ihrem Rechner installieren, z.B.

als fertige Debian Linux Pakete, und dazu JSTL (falls nötig). Gelegentliche Beschreibungen von Menüs u.ä. beziehen sich auf den Gnome Desktop, wie er mit Debian Linux installiert werden kann. Andere Linux-Distributionen ähnlich Debian sind z.B. Ubuntu und Knoppix. Installieren Sie eine dieser Linux-Distributionen auf Ihrem eigenen PC, dann können Sie mehr Dinge ausprobieren als im Schulungsraum und mehr von der Lehrveranstaltung profitieren; siehe u.a. http://debian.org, ubuntu.com, knoppix.net.

1.1 Inter-Net

Mit der zunehmenden Verbreitung von digitalen Computern entstanden auch verschiedenste Netzwerke zu deren Verbindung. Diese waren zunächst sehr Hersteller- und Rechner-spezifisch und durch den Mangel an Standards nicht leicht miteinander zu verbinden. Das Internet entwickelte sich in den 1970er und 80er Jahren aus den Bemühungen zum Zusammenschluß von technisch sehr unterschiedlichen Systemen und schon vorher existierenden kleineren Netzen, daher der Name 'Inter-Net'.
Bei der Entwicklung dieses neuen Inter-Nets waren einige Designentscheidungen wegweisend, vor allem der dezentrale Charakter sowie die Fehler- und Ausfallstoleranz, die dem Stand der Technik angemessen waren, aber auch eine Folge des kalten Krieges waren.

- Das Internet sollte aus kleinen, selbständigen lokalen Netzen bestehen, die über **Gateways** mit anderen lokalen Netzen verbunden sind.
- Daten sollten in Form von Paketen übermittelt werden, die ohne Beschränkungen **weitergeleitet** werden, wenn sie nicht für das eigene lokale Netz bestimmt sind.
- Damit möglichst viele Beteiligte diese Bemühungen unterstützen, sollte das Protokoll zum Zusammenschluß mit anderen Netzen **offen** sein, d.h. für alle verfügbar und nicht herstellerabhängig oder proprietär.

Dieses Design prägt bis heute den weitgehend dezentralen und offenen Charakter des Internet. Ein bestimmtes Unternehmen versucht zwar immer wieder, seine Marktmacht zu benutzen um proprietäre Standards auch im Internet einzuführen, aber glücklicherweise bis jetzt ohne entscheidende Erfolge. In unserer Lehrveranstaltung verwenden wir entsprechend dem Open-Source-Gedanken nur offene Standards und nicht-proprietäre Software, weil damit nicht nur eine hohe Qualität und laufende Verbesserung und Erweiterung sichergestellt sind, sondern auch der Nutzen der Gemeinschaft am höchsten ist und gleichzeitig direkt und indirekt eine große Zahl von interessanten und vielfältigen Entwicklungen unterstützt wird.
⇒ opensource.org, fsf.org, gnu.org, w3.org

1.2 Codes für Zahlen und Zeichen

Zahlen und Zeichen können im Computer nicht direkt verarbeitet werden, sondern müssen vercodiert werden. Das geschieht z.B. beim Eingabegerät wie der Tastatur. Wenn die Zeichen am Bildschirm angezeigt werden sollen, muß die umgekehrte Umwandlung vorgenommen

dez	hex	bin	dez	hex	bin
0	0	0	9	9	1001
1	1	1	10	A	1010
2	2	10	11	B	1011
3	3	11	12	C	1100
4	4	100	13	D	1101
5	5	101	14	E	1110
6	6	110	15	F	1111
7	7	111	16	10	10000
8	8	1000	17	11	10001

Abbildung 1: Zahlensysteme

werden. Die Vercodierung von Zahlen und Zeichen kann auf sehr unterschiedliche Weise geschehen.
Das dezimale Zahlensystem verwendet die Ziffern 0-9 und hat den Stellenwert zehn; wir sind gewohnt, die Zahl 21 als $2 * 10 + 1 = 21$ zu verstehen. Das ist aber nur Konvention; Abb. 1 zeigt für einige dezimale Zahlen ihre hexadezimale und binäre Form.
Das **hexadezimale Zahlensystem** funktioniert sehr ähnlich wie das dezimale System, verwendet aber neben den Ziffern 0-9 auch die Zeichen A-F als Ziffern: A steht für den Wert zehn, B für elf usw. Die hexadezimale Zahl FE wird im dezimalen System zu $15*16+14 = 254$, weil F für 15 und E für 14 stehen. Die Ziffer F steht hier sozusagen an der Zehnerstelle, nur hat diese Stellen eben den Wert 16.
Das **binäre Zahlensystem** kennt nur 0 und 1 als Ziffern. Daher muß schon der Wert zwei als 10 geschrieben werden. Drei ist 11, vier ist 100 usw. Computer rechnen im binären System; die Grundoperationen sind direkt mit elektronischen Schaltkreisen umgesetzt, d.h. es gibt z.B. ein Addierwerk, das aus zwei Folgen von Bits eine weitere Folge von Bits produziert, die der Summe entspricht.
Das Umrechnen zwischen binärem und dezimalem System sowie zwischen hexadezimalem und dezimalem System ist etwas mühsam; zwischen binärem und hexadezimalem System ist es hingegen einfach: jedem binären 4-er Block entspricht eine hexadezimale Stelle: die binäre Zahl 0001 1010 ist 1A im hexadezimalen System.
Ein **Bit** ist die kleinste Einheit für digitale Daten; mit einem Bit können zwei mögliche Zustände gespeichert werden, die meist mit 0 und 1 bezeichnet werden.
Ein **Byte** wird meist als eine Folge von 8 Bit definiert und kann daher $2^8 = 256$ mögliche Zustände speichern. Bytes werden meist nicht mit ihrer Bitfolge angeschrieben, sondern in Hex-Code, z.B. FF für die Bitfolge 1111 1111.
Bitfolgen werden meist als Bytes behandelt, selbst wenn nicht alle Bits ausgenützt werden. So kann z.B. die Bitfolge 100 0001 in dem Byte 0100 0001 gespeichert und in Hex-Code mit 41 bezeichnet werden.
Ein **Text** ist eine Folge von Zeichen, die für die Verarbeitung im Computer in eine Folge von Bytes umgewandelt werden muß. Dafür können verschiedene Codes verwendet werden.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	″	#	$	%	&	’	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	—	}	~	DEL

Abbildung 2: Der ASCII Code; die hexadezimalen Codes der Zeichen (und damit auch die entsprechenden Bitfolgen) können aus Zeile und Spalte abgelesen werden; der Hex-Code für A ist z.B. 41, die entsprechende Bitfolge ist 100 0001.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Abbildung 3: ISO 8859-1 enthält im unteren Teil die ASCII Zeichen und verwendet das 8. Bit, um weitere Zeichen der westeuropäischen Sprachen zu vercodieren; Beispiel: der Hex-Code für Ö ist D6, die entsprechende Bitfolge daher 1101 0110.

Ein solcher Code (**character encoding**) legt fest, welche Zeichen in welche Bytes oder Bytefolgen umgewandelt werden.
Der **ASCII** Code (American Standard Code for Information Interchange, siehe Abb. 2) definiert Bitfolgen für die Zeichen der englischen Sprache sowie einige Sonderzeichen. ASCII ist ein 7-bit Code, es gibt daher 128 Positionen. Die Positionen 00 bis 1F (die ersten 32) sowie 7F sind für die **control characters** bestimmt, wie z.B 08 (backspace), 09 (tab), 0A (line feed), 1B (escape).
Die Zeichen an den Positionen 20 bis 7E (dezimal 32 bis 126) in Abb. 2 werden auch als **printable ASCII** bezeichnet. Ein Text, der nur solche Zeichen enthält, wird **plain text** genannt.
Für Westeuropa interessant sind die Codes ISO 8859-1 und UTF-8. Der **ISO 8859-1** Code ist ein 8-bit Code und definiert neben den ASCII-Zeichen auch die Bitfolgen für die Zeichen, die in den westeuropäischen Schriften verwendet werden; siehe Abb. 3.
Obwohl ISO 8859-1 schon einen großen Fortschritt gegenüber ASCII darstellt (aus nicht-amerikanischer Sicht), bleibt die Beschränkung auf 256 Zeichen doch unzufriedenstellend.

Unicode	0000 0000 1111 1100
2 Byte Form	0000 0yyy yyzz zzzz
UTF-8 Schablone	110y yyyy 10zz zzzz
UTF-8 Code	1100 0011 1011 1100

Abbildung 4: Transformation des Zeichens ü aus seiner Unicode-Position FC d.h. 1111 1100 in den UTF-8 Code 1100 0011 1011 1100. Die Position bestimmt die Form und damit die Schablone; FC liegt in 000080-0007FF, daher die 2 Byte Form.

Im **Unicode** 5.0 Standard werden 98.884 grafische Zeichen definiert; der Großteil sind chinesische, japanische und koreanische Schriftzeichen, daneben finden sich aber auch mathematische und viele andere Symbole; siehe unicode.org, insb. www.unicode.org/charts/PDF/. Die ersten 256 Positionen entsprechen ISO 8859-1. Neben der Definition der Zeichen und ihrer Positionen werden auch Mechanismen zur Umsetzung beschrieben; der wichtigste für das World Wide Web ist UTF-8.
UTF-8 (8-bit Unicode Transformation Format) ist ein Code mit variabler Länge, der Zeichen in ein bis vier Blöcken zu je 8 Bit vercodiert. Die Transformation eines Zeichens geht von der Position im Unicode aus. Die ASCII Zeichen werden mit einem Byte vercodiert (beginnt mit 0), weitere 1920 Zeichen (darunter die europäischen Sonderzeichen wie Umlaute, griechische und cyrillische Zeichen) werden mit zwei Byte vercodiert (erstes Byte beginnt mit 110, das zweite mit 10, siehe Abb. 4), und die restlichen mit drei Byte (Prefix 1110 für das erste Byte und 10 für die weiteren) oder vier Byte (Prefix 11110 und 10).
Trotzdem UTF-8 mit den Prefixen scheinbar einige Bit verschenkt, hat es für viele Anwendungen günstige Eigenschaften:

- Die ASCII-Zeichen werden mit den gleichen Bitfolgen vercodiert wie im ASCII-Code selbst; UTF-8 Text, der nur ASCII-Zeichen enthält, unterscheidet sich daher nicht von ASCII-codiertem Text.
- Durch die Prefixe kann niemals die Bytefolge eines kürzer vercodierten Zeichens in der Bytefolge eines länger vercodierten auftreten; ohne diese Eigenschaft wären Suchalgorithmen für Texte unangenehm kompliziert.
- Die automatische Erkennung von UTF-8 Texten wird durch die Prefixe erleichtert: wenn eine Bytefolge UTF-8 konform ist, dann handelt es sich mit hoher Wahrscheinlichkeit tatsächlich um UTF-8.
- Da die meisten Textdokumente im Web gar keine oder nur relativ wenige Sonderzeichen (Nicht-ASCII-Zeichen) enthalten, ist die Vercodierung auch sehr speichereffizient.

Beim Transfer von Texten wird auf der Senderseite eine Bytefolge mit einer Codeangabe (character encoding) abgeschickt; wenn der Code beim Empfänger bekannt ist, dann kann aus der Bitfolge wieder die richtige Zeichenfolge rekonstruiert werden. Das ist z.B. im Webbrowser und im Email reader relevant, wenn westeuropäische Zeichen dargestellt werden sollen.

Glücklicherweise unterstützen immer mehr Applikationen UTF-8. Trotzdem gibt es derzeit immer noch viele Probleme mit unterschiedlichen Zeichensätzen. Da die Vercodierung der ASCII Zeichen in allen besprochenen Vercodierungsmethoden gleich ist, hat man bei Dokumenten, die nur printable ASCII Zeichen enthalten, die besten Chancen, daß die Übertragung funktioniert und der Text beim Empfänger genauso aussieht wie beim Sender.

1.3 Paketorientierte Verfahren am Beispiel Ethernet

Die zu übertragenden Daten werden in kleine **Pakete** getrennt und unabhängig voneinander übertragen, in etwa vergleichbar mit einem Roman, der über Briefe verschickt wird: auf jedem Brief steht der Empfänger, der Absender, eine laufende Nummer und ein Stück vom Text. Die Briefe gehen nicht unbedingt alle über dieselbe Route und kommen nicht unbedingt in der Reihenfolge an, in der sie verschickt wurden, und manchmal gehen auch einige verloren.
Im **Ethernet**, einer für die Verbreitung von **LANs (local area networks)** und damit auch des Internet historisch sehr wichtigen Netzwerktechnik, variiert die Paketgröße zwischen 64 und ca. 1500 Byte. Die **Netzwerkkarte** im Rechner (NIC, network interface card, auch Ethernet controller genannt) hat eine **Hardware-Adresse** (MAC Adresse, media access control), die aus 6 Byte besteht: die ersten drei Byte bezeichnen den Hersteller, die restlichen vergibt der Hersteller selbst, z.B. als laufende Nummer. Dadurch ist die MAC Adresse weltweit eindeutig, d.h. es gibt keine zwei NICs mit der gleichen MAC Adresse.
Innerhalb eines Ethernet LAN Segments horchen und senden alle NICs auf derselben Leitung (z.B. twisted pair, verdrillter Kupferdraht). Sie erkennen die für sie bestimmten Pakete an der Empfängeradresse und geben diese an Software zur Verarbeitung weiter; Pakete mit anderen Empfängeradressen werden ignoriert. Senden zwei NICs gleichzeitig, so gibt es eine **Kollision**, d.h. das Senden scheitert und wird nach Ablauf einer kurzen, zufälligen Zeitspanne wiederholt.
Ein Paket enthält die Adressen von Sender und Empfänger, z.B. SRC addr: 80:00:20:fa:b2:36, DEST addr: 20:00:15:45:3f:f3, sowie weitere Daten, die von anderen Ebenen (wie IP, TCP, HTTP) verwendet werden. Zur Kontrolle der fehlerfreien Übertragung des Paketes gibt es eine **Prüfsumme**, die der Sender aus übrigen Daten berechnet und im Paket mitschickt. Der Empfänger berechnet nach dem gleichen Verfahren wiederum die Prüfsumme aus den übrigen Paket und kann durch Vergleich mit der mitgeschickten Prüfsumme Fehler erkennen.
Pakete sind verschachtelt d.h. enthalten andere Pakete: ein Ethernet-Paket enthält z.B. ein IP Paket, dieses enthält ein TCP Paket, und dieses ein HTTP oder HTTP Paket. Jede Ebene schickt mit dem enthaltenen Paket weitere Daten mit, um die eigene Funktionalität zu implementieren. Auf der Empfängerseite werden die Pakete wieder ausgepackt, sodaß die darunterliegenden Ebenen für die Applikation unsichtbar sind. Ein Paket kann z.B. folgende Teile enhalten, die nacheinander als Bitfolgen übermittelt werden: Ethernet Hardware-Adressen – IP Adressen – TCP Ports – Applikationsdaten – Prüfsumme.

1.4 HUB, Switch, Router

Ein **HUB** ist ein Verteilerstecker, der Anschlüsse zusammenfaßt. Alle angeschlossenen NICs empfangen alle Pakete.
Ein **Switch** stellt Verbindungen zwischen Partnern her. NICs hören nur mehr Pakete, die an sie gerichtet sind, oder explizit an alle (broadcast).
Ein **Router** erkennt Pakete, die für Empfänger außerhalb des eigenen LAN bestimmt sind; er leitet sie weiter in einen jener Netzbereiche, mit denen er direkt verbunden ist, und den er als brauchbare Route zur Zieladresse wählt (daher der Name Router). Dort kann ein anderer Router die Pakete weiterleiten. Dieser Prozeß wiederholt sich solange, bis das Paket im richtigen Netzbereich angekommen ist. Bei einem internationalen Transfer sind 30 oder mehr 'hops' keine Seltenheit. Die Routenwahl kann sich während einer Sitzung ändern, dann können Pakete auf unterschiedlichen Routen zum Zielrechner gelangen und müssen daher nicht in der gleichen Reihenfolge eintreffen, in der sie abgeschickt wurden.

1.5 IP und TCP

Die Entwicklung des Internet gründete auf der Verbreitung von Protokollen zur Verbindung verschiedener schon vorher existierender, aber technisch sehr unterschiedlicher Netzwerke. Diese Protokolle sind **IP** (Internet Protocol) und **TCP** (Transmission Control Protocol). Ein Protokoll regelt die Kommunikation zwischen Partnern. Darin wird genau festgelegt, wer wann welche Nachrichten an wen verschickt.
IP enthält Regelungen für das Verschicken von einzelnen Paketen und die **Addressierung** von Rechnern im Netz (auch hosts genannt). Eine numerische IP-Adresse besteht aus 4 Zahlen zwischen 0 und 255.
Adressen werden oft im Zusammenhang mit **Domänen** vergeben. So hat z.B. die WU die Domäne wu-wien.ac.at, der numerisch der Bereich 137.208.0.0 bis 137.208.255.255 entspricht. Für Rechneradressen wird 255 normalerweise nicht vergeben.
TCP regelt den **verlustfreien Datentransfer** über das darunterliegende IP Medium. TCP spezifiziert Mechanismen, die garantieren, daß die Pakete beim Empfänger vollständig und in der richtigen Reihenfolge wieder zusammengesetzt werden.
IP und TCP werden meist gemeinsam eingesetzt und konfiguriert, sodaß sie beide zusammen oft als TCP/IP gezeichnet werden.

1.6 Dienste im Internet, die auf TCP/IP aufbauen

TCP/IP ermöglicht die **transparente** weltweite Verbindung von Rechnern, d.h. die darunterliegenen Schichten und ihre technischen Details sind nicht sichtbar. Software kann allein mit den von TCP/IP vorgegebenen Schnittstellen Verbindungen herstellen, unabhängig davon, ob die Übertragung über Ethernet, Modem, Satellit oder andere Medien abläuft, und unabhängig davon, welche Hardware und welche Betriebssysteme auf Sender- und Empfängerseite verwendet werden. Mit dieser Abstraktion wurde eine große Vielfalt an Anwendungen möglich. Einige (wie DNS) existieren in Form von Diensten und bleiben dem Benutzer normalerweise verborgen,

andere sind als Anwendungen bekannt.

EMail: Der erste enorm beliebte und erfolgreiche Dienst im entstehenden Internet war EMail. Dazu zählen die Protokolle wie SMTP (simple mail transfer protocol), die Mail Transfer Programme wie sendmail, und die Mail clients. Mails werden ähnlich wie Pakete zwischen mail hosts geroutet, bis sie den Adressaten erreichen.

Telnet: Damit können Benutzer auf Rechnern arbeiten, als wenn sie mit ihnen direkt verbunden wären. Weil aber das Login-Passwort im Klartext übermittelt wird, und es relativ einfach ist, unverschlüsselte Übertragungen im Internet automatisch mitschreiben zu lassen und auszuwerten, wird Telnet immer mehr durch SSH verdrängt.

FTP: File Transfer Protocol. Damit können Dateien zwischen unterschiedlichen Rechnern übermittelt werden, auch wenn diese unterschiedliche Betriebssysteme verwenden. Auch hier wird das Passwort im Klartext übermittelt, daher unsicher; allerdings gibt es auch die Möglichkeit, Dateien zum anonymen Transfer zur Verfügung zu stellen, was bis heute genutzt wird, und auch ein Vorläufer des WWW war.

DNS: Der Schulungsraumserver hat die numerische Adresse 137.208.107.18 und den hostname balrog.wu-wien.ac.at. Die automatische Übersetzung vom alphabetischen hostname in die numerische IP-Adresse wird über DNS (domain name system) hergestellt. Deshalb wird in der TCP/IP Konfiguration von Rechnern ein DNS-Server angegeben. Oft werden DNS-Server, IP-Adresse und andere Parameter beim Starten des Rechners mit Hilfe von **DHCP** (Dynamic Host Configuration Protocol) automatisch gesetzt, indem ein DHCP-Server im LAN freie IP-Adressen vergibt. Im Zusammenhang mit DNS ist die Namensauflösung innerhalb der eigenen Domäne interessant: dann genügt der erste Teil des hostname, also z.B. balrog statt balrog.wu-wien.ac.at.

WWW: Wie kaum eine andere Anwendung beschleunigte das World Wide Web seit der freien Verfügbarkeit des NCSA Mosaic Web Clients 1994 die Verbreitung des Internet. Mit dem World Wide Web werden wir uns in der Folge genauer beschäftigen.

SSH: Secure Shell. Bietet wie Telnet die Möglichkeit, auf einem Rechner zu arbeiten, mit dem der Benutzer nicht direkt verbunden ist, sowie weitere Dienste, wie z.B. file transfer. Das Login-Passwort wird nicht im Klartext übermittelt. Stattdessen wird mit Hilfe von asymmetrischen kryptographischen Verfahren die Passwortkontrolle lokal erledigt.

2 Kryptographische Grundlagen und einige Anwendungen

Im Zusammenhang mit Internet und World Wide Web werden Sicherheit und kryptographische Verfahren immer bedeutender. Ein zumindest grundlegendes Verständnis dieser Verfahren ist für informierte Entscheidungen im IT Bereich notwendig.

Ein Sender S will eine Nachricht m (message) über ein unsicheres Medium an einen Empfänger R übertragen, sodaß sie für einen Lauscher L unbrauchbar ist. Dazu muß die Nachricht m in eine verschlüsselte Form c (cypher) gebracht werden.
S, R und L werden normalerweise nicht Menschen sein, da die Verschlüsselung sehr aufwendig ist, sondern Programmteile, die von der Kommunikationssoftware aufgerufen werden.
Die Nachricht m ist eine Folge von Bits, die eine Interpretation als Text haben kann, aber auch als Bild oder andere Datentypen.

2.1 Symmetrische Verfahren

Die einfachste Methode verwendet den Schlüssel k sowohl zum Verschlüsseln als auch zum Entschlüsseln: verschlüsselt wird mit einer Funktion e(m,k)=c und entschlüsselt mit d(c,k)=m. Der Schlüssel k ist ebenfalls eine Bitfolge, die eine Interpretation als Bytefolge d.h. als Text haben kann, ähnlich einem Passwort.
Beispiel: mit der Funktion XOR ist eine ganz einfache symmetrische Verschlüsselung möglich, und zwar durch bitweises Anwenden von XOR(m,k)=c und XOR(c,k)=m. Die Funktion XOR ist dem logischen Oder ähnlich: XOR(0,0)=0, XOR(0,1)=1, XOR(1,0)=1; aber XOR(1,1)=0. Ist die Nachricht länger als der Schlüssel, so legen wir den Schlüssel wiederholt an und verschlüsseln so die einzelnen Teile der Nachricht nacheinander.
Hier handelt es sich um eine **block cypher**, weil die einzelnen Blöcke von m unabhängig voneinander verschlüsselt werden. Im Gegensatz dazu steht die **stream cypher**, wo typischerweise die verschlüsselten vorhergehenden Blöcke bei der Verschlüsselung des aktuellen Blocks verwendet werden. Diese Art von cypher ist meist schwerer zu knacken.
Allen symmetrischen Verfahren ist gemeinsam, daß Sender und Emfänger den geheimen Schlüssel k zuerst auf sichere Art vereinbaren müssen. Das ist in manchen Situationen ein großer Nachteil.

2.2 Asymmetrische Verfahren

Hier wird ein Schlüsselpaar erzeugt, das im Hinblick auf bestimmte mathematische Eigenschaften zusammen paßt. Die Schlüssel sind Bitfolgen, die als Zahlen interpretiert werden und in Form von Bytefolgen gespeichert werden. Hier ist z.B. ein **public key** von john.smith@hotmail.com:

```
mQFCBDrSCJERAwDQPtyyNTRdkZmIMy79usAkeKqUbpv1BCvbYhyudiSr9YgdnXKQ
ijoMJZ9+9PAEFalfz9JBZeIZMLefzAZKnMUEYpsDSxux0BBMTzwiDArHuz8ooNRF
XOe69t3qLQXWxU8AoP9CECdJ368XGBL7RLz4SRMK64wDAwCFSf1mkRQ4fiGotiNL
z7gicuY4i5akAckutEjvMFVcBMIdaxLLU/vT8+xmfIQNvPMoXvKI9LO3LA7zKpXq
tv3yVwX+Oxqzy4KK/rINbEctFCqJwFvrdTh/SRLfBQSRICwC/irqjA8sIYPokgA/
7L+Ujsm+9kcViFi+/vGXB42nTeAHbywhqBeHysy7B8PPVwy8+8Z85+JScDMnbG95
rGjC9/evuS/KNyJgOI83vj5UXgN3eu/2WoscRXJrpCYZLIfLiLQjam9obiBzbWl0
aCA8am9obi5zbWl0aEBob3RtYWlsLmNvbT6ITgQQEQIADgUCOtIIkQQLAwIBAhkB
AAoJEAItlGe44SqZXHIAn1T/wcERq1tbNWCYMa+mJ+f4DjhIAKCo8KPzBPBWKmQn
sDd49EvkepVsQ7kAzQQ60giSEAMA3VSPrbRcvXM+a/dU8haaVWuZn6HRElwlNnLj
```

```
Inv+SFRYTlPnw4LqcxXeGWbPmWJ5iBORQ4ORRPwYmCLCyv/Ug+GJoPKCnFYgxP2N
7p/GxuYuVxNgoPjWyOhIxXNOMCijAAICAv4xerEQvi20bSn+CgEBXW1EEI5KAmUI
CxMpTBNZAADhFmKQSICTaZD52KTMNow6lmZT4NV3gaROTzy80XUx5m1oVOQo7kbd
qRc/SgLe5FS9FJJkOspdSul/C9sg4gyjnpCIRgQYEQIABgUCOtIIkgAKCRACLZRn
uOEqmWklAKDQwuqOaiBCp+eJFZYNAfigmm7rGwCg1LG0NzRAzxKPxazmZJCFON5d
7Vo=
=Y2/h
```

Der **public key** ist öffentlich zugänglich, z.B. auf Webseiten oder über Keyserver. Es kann und soll allgemein bekannt sein, daß z.B. john smith@hotmail.com den obigen public key verwendet.
Der zum public key gehörige **private key** muß geheim bleiben; das ist aber möglich, weil er immer nur lokal verwendet und nie kommuniziert wird.
Die **sichere Übertragung** funktioniert durch Verschlüsseln mit dem public key und Entschlüsseln mit dem zugehörigen private key. Nur der Besitzer des zum public key passenden private key kann die Nachricht entschlüsseln.
Die **digitale Signatur** funktioniert durch Verschlüsseln mit dem private key und Entschlüsseln mit dem zugehörigen public key. Nur der Besitzer des zum public key passenden private key konnte die Signatur erzeugen.

2.3 SSL

Leider ist die asymmetrische Verschlüsselung extrem rechenaufwendig und nur für relativ kurze Bitfolgen praktikabel. Daher können nicht ganze Dokumente verschlüsselt werden.
Im **Secure Socket Layer (SSL)** werden symmetrische und asymmetrische Verfahren zusammen verwendet, um eine Nachricht von S and R zu senden:

1. S generiert einen zufälligen Schlüssel k.
2. k ist kurz genug für eine sichere Übertragung an R durch asymmetrische Kryptographie: S verschlüsselt k mit dem public key von R und schickt das Resultat an R.
3. R entschlüsselt mit seinem private key und ist nun im Besitz von k.
4. Nun können Nachrichten durch symmetrische Verschlüsselung mit k sicher übertragen werden.

Für jede Sitzung wird ein neuer Key erzeugt. Die asymmetrisch verschlüsselte Übertragung von k garantiert, daß niemand außer R den Originalschlüssel k rekonstruieren kann.

2.4 Signatur und Digest

Ein **Digest** ist eine Kurzform eines längeren Textes (oder einer anderen Bitfolge) mit bestimmten Eigenschaften; für den idealen Digest-Algorithmus gilt:

1. Unterschiedlichen Dokumenten entspricht immer ein unterschiedlicher Digest.
2. Der Digest kann schnell berechnet werden.
3. Der Digest ist kurz genug für die asymmetrische Verschlüsselung.

Die erste Eigenschaft kann von keinem Algorithmus hundertprozentig erreicht werden, aber eine extrem geringe Wahrscheinlichkeit für gleiche Digests bei unterschiedlichen Dokumenten ist machbar.
Der **SHA1**-Algorithmus wird häufig eingesetzt. Er erzeugt einen Digest mit 160 Bit.
Anstelle des Originaldokuments wird nun der Digest signiert. Die Signatur kann kontrolliert werden, indem der Empfänger ebenfalls den Digest erzeugt und mit dem vergleicht, was er durch Entschlüsseln der Signatur mit dem public key des Unterzeichners erhält.

2.5 Identität und Certificate Authorities

Bei der Kontrolle der Signatur kann nur gesichert werden, daß die verwendeten public und private keys zusammenpassen. Die Identität des Unterzeichners kann so nicht geprüft werden. Im obigen Beispiel wissen wir nicht, ob der angegeben public key wirklich John Smith gehört. Dazu kann die Information über public key und Identität des Ausstellers selbst wieder signiert werden.
Browser und andere Software können eine **Certificate Authority** kontaktieren, um Zertifikate d.h. Informationen über die Identität von Unterzeichnern zu kontrollieren. Die CA verlangt für diesen automatisierten Dienst eine jährliche Gebühr von allen Organisationen und Personen, die solche Zertifikate verwenden wollen.
Eine andere Variante ist das **Web of Trust**. Hier signieren einzelne Personen die public keys und Identitäten von bestimmten anderen Personen, die ihnen persönlich bekannt sind. Da diese dann wiederum andere public keys signieren, können Identitäten meist in wenigen Iterationen geprüft werden. Diese Variante hat (noch?) keine kommerzielle Bedeutung.

2.6 Mathematische Grundlagen

Der RSA-Algorithmus (nach den Autoren Rivest, Shamir, Adleman) wird häufig für die asymmetrische Verschlüsselung verwendet. Im folgenden wird eine etwas vereinfachte Version beschrieben. Eine Nachricht x ist eine Bitfolge und kann als positive ganze Zahl interpretiert werden. Wir wählen zufällig zwei große Primzahlen p und q und berechnen $N = pq$ sowie $F = (p-1)(q-1)$. Nun wählen wir einen public key Exponenten e mit $ggT(e, F) = 1$, d.h. e und F haben nur 1 als gemeinsamen Teiler, z.B. häufig $e = 2^{16} + 1 = 65537$. Den private key Exponenten d wählen wir so, daß $de = kF + 1$ für ein (jeweils zu findendes) k. Der RSA-Modul N wird zusammen mit e als public key veröffentlicht. Bekannt ist also nur das Produkt aus p und q, nicht p und q selbst. Zum Verschlüsseln berechnen wir $x^e \bmod N = y$, zum Entschlüsseln $y^d \bmod N = x$.
Entschlüsseln ist nur möglich, wenn wir den zum public key passenden private key kennen. Beispiel: wir wählen $p = 23$ und $q = 19$. Damit ist $N = 437$ und $F = 396$. Für e wählen

Challenge	Dezimalstellen	Preis	Faktorisiert
RSA-130	130		Apr. 10, 1996
RSA-140	140		Feb. 2, 1999
RSA-150	150		Apr. 16, 2004
RSA-155	155		Aug. 22, 1999
RSA-160	160		Apr. 1, 2003
RSA-200	200		May 9, 2005
RSA-576	174	$10,000	Dec. 3, 2003
RSA-640	193	$20,000	Nov. 4, 2005
RSA-704	212	$30,000	–
RSA-768	232	$50,000	–
RSA-896	270	$75,000	–
RSA-1024	309	$100,000	–
RSA-1536	463	$150,000	–
RSA-2048	617	$200,000	–

Tabelle 1: RSA challenges, vgl. en.wikipedia.org

wir $e = 13$; mit Hilfe von $(kF + 1)/e = d$ ermitteln wir einen Wert für d, den wir mit $k = 2$ finden: $(2F + 1)/e) = d = 61$. Wir können also (437;13) als public key veröffentlichen, der zugehörige private key ist (437;61).
Nun verschlüsseln wir eine Nachricht, z.B. die Bitfolge 101010, die wir als positive ganze Zahl im Binärsystem interpretieren und im Dezimalsystem $x = 42$ erhalten, d.h. wir berechnen zum Verschlüsseln $x^e \bmod N = 42^{13} \bmod 437 = 1265437718438866624512 \bmod 437 = 237$. Die entsprechende Bitfolge ist 10101011101101. Zum Entschlüsseln von $y = 237$ berechnen wir $y^d \bmod N = 237^{61} \bmod 437 = 42$ und erhalten wieder die Originalnachricht.
Das Zwischenresultat 237^{61} hat 145 Dezimalstellen; in Linux steht das Rechenprogramm `dc` zur Verfügung, mit dem exaktes Rechnen auch mit sehr großen Zahlen möglich ist; siehe `man dc`. Damit können Sie weitere Beispiele ausprobieren.
Natürlich kann $N = 437$ sehr schnell in seine Primfaktoren zerlegt werden, in Linux z.B. mit `factor 437`. Die Sicherheit des Verfahrens beruht darauf, daß es zwar relativ einfach ist, zwei große Zahlen miteinander zu multiplizieren, aber sehr aufwendig, das Produkt wieder in seine (unbekannten) Primfaktoren p und q zu zerlegen. Es stellt sich die Frage, wie groß der RSA-Modul sein muß, d.h. wieviele Bit "sicher genug" sind. RSA Security (www.rsa.com) veröffentlicht verschiedene challenges mit Geldpreisen. Zur Zeit (Anfang 2007) sind die challenges bis einschließlich RSA-640 gelöst (640 Bit Modul, das sind 193 Dezimalstellen, vgl. Tab. 1). Das Lösen der RSA-640 challenge d.h. das Faktorisieren einer bestimmten Zahl mit 193 Dezimalstellen dauerte 5 Monate mit 80 2.2 GHz Rechnern. Damit ist natürlich nicht das gesamte Verfahren "geknackt", und für bestimmte Anwendungen kann RSA-640 immer noch sicher genug sein. Derzeit scheint es so, als ob die RSA-Varianten mit 1536 oder 2048 Bit ausreichende Sicherheit bieten; aber solche Spekulationen haben sich in der Vergangenheit schon öfter als völlig falsch herausgestellt.

3 Das World Wide Web

3.1 HTTP

Das HyperText Transfer Protocol ist die Grundlage für WWW. Es handelt sich um ein sehr einfaches Protokoll: der Client (der Webbrowser) schickt einen Request (Anfrage) an den Server (den Webserver), und dieser anwortet mit einer Reply. Grundlage für den Request ist der **URL** (Uniform Resource Locator) mit folgenden Teilen:

Protokoll: meist http, gefolgt von ://

Hostname: z.B. balrog.wu-wien.ac.at

Dokument: z.B. /home/staff/mitloehn/test.jsp

Nach dem hostname kann mit Doppelpunkt noch ein TCP Port angegeben sein, sonst wird für HTTP das Port 80 verwendet. Wenn der Browser einen HTTP Request abschickt, weil der Benutzer auf einen Link geklickt hat, oder den URL in der Location eingetippt hat, dann geschieht typischerweise folgendes:

1. Der Browser versucht einen Verbindungsaufbau über TCP zum Zielrechner. Funktioniert das innerhalb einer bestimmten Zeit nicht, so gibt er eine Fehlermeldung (z.B. request timed out; der **timeout** kann im Browser konfiguriert werden und liegt normalerweise bei 1-3 Minuten). Gründe für Fehler können sein, daß der Rechner überhaupt nicht erreichbar ist, oder daß auf dem Port 80 kein Webserver läuft.

2. Beim Zielrechner läuft (hoffentlich) ein Programm, das auf genau solche requests wartet: der 'Webserver'. Dazu ist der Zielrechner als Internet host normalerweise ununterbrochen in Betrieb. Ca. 70% aller Internet Web Server sind Apache-Installationen, das ist ein frei verfügbarer Open Source Web Server. Immer häufiger läuft dieser Web Server auf einem Rechner, der unter dem ebenfalls freien Open Source Betriebssystem Linux betrieben wird. Als Gründe dafür werden genannt: geringere Kosten, höhere Qualität und Zuverlässigkeit, besserer Support als kommerzielle Produkte.

3. Wenn die Verbindung aufgebaut ist, schickt der Browser im einfachsten Fall eine Zeile Text an den Server, die den Dokumentnamen aus dem URL enthält.

4. Der Webserver schickt das gewünschte Dokument über TCP zurück, Danach wird die TCP Verbindung beendet. Falls der Webserver den Request nicht erfüllen kann, schickt er eine Fehlermeldung mit einem **error code**. Die wichtigsten sind:

   403: Forbidden, weil z.B. dem Webserver der Zugriff auf das gewünschte Dokument vom Betriebssystem nicht erlaubt wurde.

   404: Document Not Found, weil z.B. der Dateiname falsch geschrieben wurde.

500: Internal Server Error, weil z.B. bei der Ausführung von Code für dynamische Inhalte ein Fehler aufgetreten ist.

5. Der Browser stellt das erhaltene Dokument dar, d.h. er führt die im HTML Code gegebenen Layoutanweisungen aus, indem er Absätze, Schriftgrößen, Abstände usw. entsprechend wählt. Das Resultat muß nicht unbedingt dem entsprechen, was sich der Autor der Seite vorgestellt hat; daher empfiehlt es sich, mit verschiedenen Browsern und Betriebssystemen zu testen, und nach Möglichkeit nur Standard-HTML verwenden, ohne Optionen, die nur auf Browsern eines bestimmten Herstellers funktionieren.

3.2 HTTP interaktiv

Wie wir gesehen haben, ist das HTTP Protokoll durchaus nachvollziehbar. Wir wollen nun direkt mit dem Webserver sprechen, um zu sehen, wie er mit dem Webbrowser interagiert. Versuchen Sie in einem Terminalfenster unter Linux (im Gnome-Menü Applications unter Accessories/Terminal) folgendes:

```
telnet balrog 80
GET /index.html HTTP/1.0
```

Das sind *drei* Zeilen Text; nach der GET Zeile folgen zwei Newline, d.h. Sie drücken zweimal auf Enter. Damit übernehmen Sie die Rolle des Webbrowsers und erhalten eine entsprechende Antwort vom Webserver. Blättern Sie im Fenster mit dem Scrollbalken zurück, um auch die führenden Header-Zeilen zu sehen.

```
HTTP/1.1 200 OK
Date: Wed, 05 Oct 2005 12:15:12 GMT
Server: Apache/1.3.33
Last-Modified: Mon, 20 Jun 2005 15:22:55 GMT
ETag: 4c0c4-1404-42b6df4f
Accept-Ranges: bytes
Content-Length: 5124
Connection: close
Content-Type: text/html; charset=iso-8859-1

<h1>Meine Webseite</h1>
...
```

Interessant sind hier Content-Length, die Anzahl der nach den Header-Zeilen folgenden Bytes, sowie Content-Type: text/html, das bedeutet für den Browser, daß er die folgenden Bytes als HTML Code interpretieren und entsprechend formatieren soll. Ist der Content-Type z.B. text/plain, dann tut er das nicht, d.h. er stellt die Zeichen genau so dar, wie sie übermittelt werden. Das passiert normalerweise automatisch bei der Dateiendung .txt. Andere Content-Types sind z.B. image/jpeg und image/gif, die auf die gleiche Weise nach den Headerzeilen als Bytefolge übermittelt werden.

3.3 HTML

Wie der Name HyperText Markup Language nahelegt, wird ein Text mit zusätzlichem Markup versehen, der die Darstellung im Browser bestimmt. In Debian/Linux finden Sie im Gnome-Menü *Applications* unter *Accessories* den *Text Editor*. Erstellen Sie damit eine Datei mit dem folgendem Inhalt, die Sie unter dem Namen `test.jsp` abspeichern.

```
<h1>Eine Webseite</h1>
<p> Ein Absatz Text.  <p> Noch ein Absatz.
Eine Liste:
<ul>
<li> eins
<li> zwei
</ul>
Mehr zu HTML und WWW unter <a href=http://w3.org>w3.org</a>.
```

Beachten Sie, daß alle geöffneten spitzen Klammern auch wieder geschlossen werden müssen. Ausserdem benötigen einige HTML-Elemente (hier h1 und p) einen Abschluß (mit /h1 und /p). Ihre Seite sehen Sie im Browser über einen URL der Form

`http://balrog.wu-wien.ac.at/home/j02/j0212345/test.jsp`

Normalerweise haben HTML-Dateien die Endung .html oder .htm, weil wir aber in der Folge Java Server Pages erstellen werden, verwenden wir gleich die Endung .jsp.
Einige HTML-Elemente:

- `<h1>` Heading Stufe 1
- `<p>` neuer Absatz
- `<ul>` unordered list
- `<li>` list item
- `<a href=...>` Link auf andere Seite
- `<p align=center>` neuer Absatz, zentriert
- `<ol>` ordered list d.h. numeriert
- `<img src=bild1.jpg>` eine Abbildung an dieser Stelle; die Datei bild1.jpg gibt es im gleichen Verzeichnis wie diese Seite, mit diesem Dateinamen; Groß/Kleinbuchstaben bei Dateinamen wird in Unix unterschieden.
- `<div>` division, meist als neue Zeile realisiert
- `<table> <tr> <td> A <td> B <tr> <td> C <td> D </table>` Tabelle

3.4 XML und Semantic Web

Dokumente im WWW waren zunächst nur für den menschlichen Leser gedacht; immer mehr wurde aber klar, daß mit diesem Medium aber auch maschinenlesbare Informationen transportiert werden können. Dazu ist natürlichsprachlicher Text nicht gut geeignet, weil die automatisierte inhaltiche Verarbeitung noch nicht gut funktioniert. Wird der Freitext aber mit inhaltlich orientiertem Markup versehen, dann können auch Programme und nicht nur Menschen damit etwas anfangen. In einem solchen Web enthalten Dokumente nicht nur Markup für das Layout, sondern auch für die Semantik (Bedeutung) des Inhalts. Die Form dieses Markups ist durch XML vorgegeben.

3.4.1 XML

Die Extended Markup Language gibt einen Rahmen zur Definition von beliebigen Markup Sprachen. Dieser Rahmen enthält vor allem folgendes:

- Beschreibungselemente werden mit spitzen Klammern markiert; das ist aus HTML bekannt.
- Optionen werden mit doppelten Hochkomma begrenzt; das ist in HTML nicht notwendig, wenn kein Leerzeichen vorkommt, z.B. ist
  `<p align=center>` nicht XML-konform, hingegen
  `<p align="center">` schon.
- Jedes Element wird mit einem Schrägstrich geschlossen; das bedeutet, daß Elemente wie `<img>` einen Schrägstrich vor der rechten spitzen Klammer bekommen: die in HTML übliche Version
  `<img src="bild1.jpg">` ist nicht XML-konform, aber
  `<img src="bild1.jpg" />` ist XML-konform.
- Elemente dürfen beliebig tief verschachtelt werden, aber nicht überlappend. Beispiel:
  `<wanderweg><start>Rax<ziel>Hochschwab</start>48h</ziel></wanderweg>` ist nicht XML-konform.
  `<wanderweg><start>Rax</start><ziel>Hochschwab</ziel>48h</wanderweg>` ist XML-konform.
  Daß die Zeitangabe nicht mit einem entsprechenden XML-Element markiert ist, stört nicht die XML-Konformität, möglicherweise aber eine sinnvolle Verarbeitung. XML-konforme Dokumente sind natürlich nicht automatisch fehlerfrei in Sinne der gewünschten Anwendung.

Wenn HTML auf diese Weise verwendet wird, dann wird daraus XHTML.
Natürlich genügt dieser Rahmen noch nicht für einen sinnvollen praktischen Einsatz im semantischen Web. Dazu wird für jede Anwendung ein Vokabular definiert; solche Beschreibungen sind Daten über Daten und werden daher als **Metadata** bezeichnet. Das **Resource Description Framework (RDF)** gibt dafür den Rahmen vor. Ein weit verbreitetes Vokabular ist der Dublin Core.

3.4.2 RDF am Beispiel von Dublin Core

Dieses Vokabular dient zur einfachen inhaltlichen Erschließung von Dokumenten; ein Beispiel macht klar, wie diese Beschreibung funktioniert:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://surf.deepblue.org/guides/rush2.html">

    <dc:creator>Rush Randle</dc:creator>
    <dc:title>Advanced Windsurfing Tips</dc:title>
    <dc:date>2007-07-18</dc:date>
    <dc:subject>windsurfing</dc:subject>

  </rdf:Description>
</rdf:RDF>
```

Mit den XML name space Anweisungen (xmlns:rdf und xmlns:dc) werden die Prefixe rdf und dc eindeutig definiert und damit die Elemente dc:creator, dc:title usw. von möglicherweise anderswo verwendeten Elementen abgegrenzt, die ebenfalls creator, title usw. heissen.
Mit der Anweisung rdf:Description beginnt die Beschreibung (es könnten auch weitere Dokumente beschrieben werden), die Option rdf:about gibt den URL des Dokuments an, das beschrieben wird. Darauf folgen die Angaben über Autor, Titel usw.
Hinter dem RDF-Ansatz steht die Idee, daß Aussagen in Form von Tripeln getroffen werden, d.h. Sätze mit den drei Teilen Subjekt-Prädikat-Objekt. In der kürzeren und für den menschlichen Leser angenehmeren Notation N3 könnte das so aussehen:

```
:rush2_html :author "Rush Randle" .
:rush2_html :title  "Advanced Windsurfing Tips" .
:rush2_html :subject "windsurfing" .
```

Nun könnten aber auch weitere Aussagen getroffen werden, z.B. über die Klassen, denen die Satzteile angehören, und die Konsequenzen, die sich daraus ergeben, insbesondere ihre korrekte d.h. hier für die Anwendung sinnvolle Verwendung:

```
:rush2_html a :webpage .
:webpage :subClassOf :document .

:author a rdfs:predicate .
:author rdfs:domain :document .
```

Wird ein Vokabular mit Angaben über Klassen und zulässigen Verwendungen versehen, so spricht man von einer **Ontologie**. Solche Ontologien werden für immer mehr Bereiche

definiert, da sie nicht nur eine eindeutige inhaltliche Erschließung, sondern auch andere weitreichende Möglichkeiten der automatisierten Verarbeitung erlauben. Mehr dazu finden Sie unter w3.org/RDF und http://www.w3.org/2000/10/swap/Primer.html.
Die gesamte RDF-Beschreibung der Windsurfing-Seite wird in einer eigenen Datei gespeichert; stattdessen kann sie auch mit Hilfe von Meta-Tags in die HTML-Seite selbst integriert werden:

```
<html>
<head profile="http://dublincore.org/documents/dcq-html/">
<link rel="schema.DC" href="http://purl.org/dc/elements/1.1/" />
<meta name="DC.title"   content="Advanced Windsurfing Tips" />
<meta name="DC.creator" content="Rush Randle" />
<meta name="DC.date"    content="2007-07-18" />
<meta name="DC.subject" content="windsurfing" />
</head>
<body>
...
</body>
</html>
```

Unter `dublincore.org` sind weitere Elemente definiert. Wenn die Informationen über Titel, Autor usw. im Freitext ebenfalls vorkommen, dann sind sie für den menschlichen Leser problemlos erkennbar, aber für eine zuverlässig fehlerfreie automatisierte Verarbeitung ist ein semantischer Markup Voraussetzung. Damit können Webcrawler und andere Programme eine inhaltliche Erschließung vornehmen; aber auch andere Anwendungen sind möglich, was am Beispiel von Microformats gezeigt werden soll.

3.4.3 Microformats am Beispiel von vCard/hCard

Wie aus den obigen Beispielen zu erkennen ist, kann der Einsatz von RDF für den Autor einer Webseite ein recht großer Aufwand sein, sodaß die Verbreitung dieser semantic web markup Variante eher langsam vorangeht. Einfachere Möglichkeiten sind gewünscht, und Microformats sind eine davon. Dabei werden die für die automatisierte Verarbeitung relevanten Teile des Dokuments nicht extra nochmals in RDF-Syntax oder in Meta-Tags spezifiziert, sondern der Freitext selbst wird mit Markup versehen, der im Browser entweder ohne Wirkung bleibt, oder (idealerweise) die Funktionen von Layout und Sematik kombiniert.
Hier ist eine Adressangabe mit HTML-Markup, der nur auf das Layout abzielt; geben Sie diesen Code in Ihre HTML-Seite ein, um den Effekt im Browser zu sehen:

```
<div>
  <div>Long John Silver</div>
  <div>Treasure Island Diving Expeditions</div>
  <div>604-555-1234</div>
</div>
```

Für den menschlichen Leser sind diese Angaben sofort verständlich, eine automatisierte Verarbeitung ist aber schwierig. Wird hingegen mit der class Option zum Markup für das Layout auch ein semantischer Markup hinzugefügt, dann können die Daten extrahiert und einem persönlicher Adressbuch hinzugefügt werden:

```
<div class="vcard">
  <div class="fn">Long John Silver</div>
  <div class="org">Treasure Island Diving Expeditions</div>
  <div class="tel">604-555-1234</div>
</div>
```

Neben hCard werden eine Reihe anderer Microformats verwendet, wie z.B hCalendar für events und hReview für Bewertungen; weitere werden laufend entwickelt.
Installieren Sie in Ihrem Firefox ein Microformat Plugin, z.B. *tails export* oder *Operator*, und gehen Sie auf die Suche nach Microformats in Webseiten! Natuerlich enthalten zur Zeit nur relativ wenige Seiten solche Microformats, aber das könnte in einigen Jahren anders aussehen.
Die Daten müssen bei Microformats nicht doppelt geführt werden müssen; im Vergleich zu RDF in XML-Dateien sind Microformats wesentlich einfacher zu verwenden, und werden damit möglicherweise bald weite Verbreitung finden. Bei microformats.org können Sie sich über die aktuellen Entwicklungen informieren.

3.5 Formulare und Dateneingabe über HTTP

Bisher haben wir das Web nur als passive Rezipienten von Information verwendet; HTTP sieht aber nicht nur die Abfrage von Daten d.h. das Ansehen von Webseiten vor, sondern auch die Eingabe von Daten, die dann am Server beliebig verarbeitet werden können. Dabei gibt es zwei Schritte: ein Formular zum Eingeben der Daten, und eine Folgeprozedur, die die Daten übernimmt und verarbeitet. Der folgende HTML Code beschreibt ein Formular `artikeleingabe.jsp`:

```
<h2>Artikel Erfassen</h2>
<form action=artikelinsert.jsp method=POST>
<table>
<tr> <td> Artikel-ID:   <td> <input type=text name=aid>
<tr> <td> Bezeichnung:  <td> <input type=text name=bez>
<tr> <td> Preis:        <td> <input type=text name=preis>
<tr> <td> Lagerbestand: <td> <input type=text name=bestand>
<tr> <td>               <td> <input type=submit value=OK>
</table>
</form>
```

Für einen bestimmten Artikel kann damit ein neuer Preis eingegeben werden. Der Webserver kümmert sich darum, die Daten an unsere Folgeprozedur `artikelinsert.jsp` weiterzuleiten.

Dort haben wir dann die vom Benutzer eingegebenen Werte zur Verfügung und können diese nach unseren Wünschen verarbeiten. Bei der Datenübermittlung gibt es zwei Möglichkeiten: **GET** und **POST**.

- Bei der GET Methode werden Name/Wert Paare an den URL angehängt, z.B.

  ```
  artikelinsert.jsp?aid=SD02&bez=SDCard256MB&preis=14.90&bestand=8
  ```

  Eine interessante Konsequenz davon ist, daß wir Daten auch ohne vorhergehendes Formular an Prozeduren übergeben können, wenn wir den URL entsprechend aufbauen. Das werden wir im Abschnitt JSP (Java Server Pages) ausnützen.

- Bei der POST Methode werden die Daten nicht im URL übertragen. Das ist empfehlenswert, wenn viele Werte übergeben werden, oder sensitive Daten wie Passwörter, die dann beim Browser im URL aufscheinen würden. POST ist aber nicht immer vorteilhaft, weil der URL dann nicht als Bookmark verwendet werden kann. Das ist aber oft wünschenswert, z.B. bei Katalogen und Suchresultaten.

In der Folgeprozedur `artikelinsert.jsp` wollen wir die Daten aus dem Formular übernehmen und in einer Datenbank speichern. Das wird die Grundlage für eine ganz einfache Warenwirtschaft mit Webshop. Die Aufgabe lautet:
Ein kleiner Versandhandel überlegt eine datenbankgestützte Lösung für die Auftragsabwicklung über Internet, die bisher mit Email erledigt wurde. Stattdessen soll nun ein Warenwirtschaftssystem erstellt werden, das folgende Prozesse mit WWW als Benutzerschnittstelle unterstützt:
Für jeden Artikel müssen Nummer, Bezeichnung, Preis und Lagerbestand verwaltet werden. Die Artikelnummern sind alphanumerisch d.h. enthalten Zeichen. Die Kunden sollen eine Liste der Artikel im Web sehen können und möglichst einfach und benutzerfreundlich Artikel bestellen können, am besten mit einem Warenkorb. Dabei soll auch eine beliebige Menge angegeben werden können. Von den Kunden sind Name, Adresse und auch die Telefonnummer für eventuelle Rückfragen interessant, weiters sollen sie im Webshop eine eindeutige Kennung und ein Passwort haben. Neue Kunden sollen sich selbst registrieren können. Der Kunde soll sich erst bei der Bestellung identifizieren müssen. Dabei soll neben Kunde und Artikel auch das Datum sowie ein Zeichen als Status gespeichert werden.
Als Grund für die Umstellung wird einerseits eine Verbesserung der Prozesse genannt, andererseits auch eine Unterstützung für kaufmännische Entscheidungen. Insbesondere sollen Umsatzdaten jederzeit nach Datum, Kunden und Artikeln gruppiert zur Verfügung stehen. Möglichkeiten zu weiteren Auswertungen sollen natürlich auch genutzt werden.
Um diese Anforderungen umzusetzen, benötigen wir noch einige Kenntnisse über Datenbanken und Java Server Pages.

4 Einige Grundlagen relationaler Datenbanken

Bei der Speicherung von Daten haben wir grundsätzlich die Wahl zwischen freier und fester Struktur. Der Text, den Sie gerade lesen, ist ein Freitext. Er wird mit einem Texteditor erstellt

und ist im wesentlichen für die menschliche Interpretation vorgesehen. Einige automatische Verfahren sind zwar möglich, z.B. die Suche nach Dokumenten, die bestimmte Wörter enthalten, oder die (nicht ganz fehlerfreie) Zuordnung von Dokumenten zu Sachgebieten, aber im wesentlichen wird die Bedeutung der Daten nur zugänglich, wenn der Text von Menschen gelesen wird, da (zumindest einstweilen) nur Menschen in der Lage sind, die komplizierte und mehrdeutige Struktur der natürlichen Sprache richtig zu interpretieren.
Im Gegensatz dazu ermöglicht uns die Speicherung von Daten in einer Datenbank automatisierte **Abfragen** und auch einfache **Auswertungen**, die für kommerzielle Anwendungen oft ausreichen. Das funktioniert deshalb, weil wir mit der Struktur der Daten auch ihre Bedeutung festlegen: in einer Artikelliste mit den Spalten ArtikelNr, Bezeichnung, Preis und Lagerbestand bleiben keine Unklarheiten über die Bedeutung der Einträge in den einzelnen Zeilen. Natürlich verlieren wir damit die große Flexibilität des Freitextes.
Je nach Ansatz bei der Strukturierung gibt es verschiedene Arten von Datenbanken. Die **relationalen Datenbanken** haben nach wie vor die größte praktische Bedeutung. Hier finden wir als wichtigste Begriffe die Relation und ihre technische Umsetzung, die **Tabelle**. Eine Tabelle hat einen Namen und eine oder mehrere **Spalten**. Jede Spalte hat ebenfalls einen Namen und einen **Datentyp**. In solch eine Tabelle können dann Datensätze eingefügt werden.
Am Schulungsraum-Server arbeiten Sie mit der persönlichen Datenbank-Kennung j Matnr, also z.B. j0012345. Passwort wird in der LV bekanntgegeben. Wir verwenden **mysql**, eine Open Source Datenbank, die auf vielen Unixrechnern verfügbar ist:

```
mysql -u j0012345 -psecret j0012345
```

Dabei steht -u für userid, -p für Passwort, und j0012345 für den Namen der Datenbank (einem userid können mehrere Datenbanken zugeordnet sein).
Innerhalb des mysql Programms als Interface zur Datenbank haben wir eine **history**: mit der Aufwärtstaste (Pfeil oben auf der Tastatur) können wir frühere Befehle wiederholen und ändern (Pfeil links/rechts). Dazu ist es sinnvoll, auch längere Anweisungen auf *einer* Zeile einzugeben. Im folgenden sind längere Anweisungen zur besseren Übersicht auf mehrere Zeilen verteilt.

4.1 SQL

Die Structured Query Language ist eine standardisierte Sprache zur Abfrage und Manipulation von Daten in Datenbanken. Zunächst wollen wir eine Tabelle anlegen. Dazu dient der Befehl **create table**. Es wird der Name der Tabelle sowie ihre Spalten mit deren Namen und Datentypen genannt:

```
create table artikel (
  aid     char(10) primary key,
  bez     char(50),
  preis   float,
```

```
  bestand int
);
```

Damit haben wir definiert, daß ein Artikel bei uns aus einem maximal 10 Zeichen langen alphanumerischen ID, einer Bezeichnung mit max. 50 Zeichen, einem Preis mit Nachkommastellen sowie einem ganzzahligen Lagerbestand besteht. Der Zusatz `primary key` bewirkt zwei Dinge:

- Alle Einträge in der Tabelle müssen eindeutige IDs haben: ein Datensatz mit schon existierender ID kann nicht eingefügt werden.
- Auß;erdem können Einträge bei bekannter ID sehr schnell gefunden werden, auch bei einer sehr großen Zahl von Einträgen, weil auch ein Index angelegt wird.

Nachdem die Tabelle angelegt ist, können mit dem Befehl **insert** Datensätze eingefügt werden:

```
insert into artikel values ('IR01', 'IR Schnittstelle USB', 25.90, 2);
insert into artikel values ('SD12', 'SD Card 128 MB', 5.90, 10);
insert into artikel values ('HD032', 'HD IDE 120 GB 5400rpm', 65.90, 0);
```

Die Datentypen müssen natürlich mit der Definition der Tabelle zusammenpassen. Die Feldlängen für `char` Werte dürfen nicht überschritten werden, und die Artikelummer darf noch nicht vergeben sein. Ungültige Einträge weist das Datenbanksystem mit einer Fehlermeldung zurück.
Mit dem Befehl **select** können nun Datensätze abgefragt werden:

```
select * from artikel;
```

Es werden alle Datensätze angezeigt. Auch Einschränkungen sind möglich:

```
select * from artikel where bestand = 0;
```

Nur Tupel mit entsprechenden Werten für die Spalte `bestand` werden angezeigt.
Die Sortierung kann mit **order by** bestimmt werden:

```
select * from artikel order by preis;
```

Mit dem Befehl **update** können Datensätze geändert werden.

```
update artikel set preis = 24.90 where aid = 'IR01';
```

Wird die Einschränkung `where` weggelassen, so werden alle Datensätze geändert, was normalerweise nicht erwünscht ist.
Der Befehl **delete** dient zum Löschen von Datensätzen:

```
delete from artikel where bestand = 0;
```

Damit der `create table` Befehl von der Datenbank erfolgreich ausgeführt werden kann, darf es noch keine Tabelle mit dem Namen geben. Falls eine solche schon existiert, kann sie mit dem Befehl

```
drop table artikel;
```

gelöscht werden. Im Gegensatz zu `delete` werden Inhalte und Tabellenstruktur gelöscht. Datenbanksystem erlauben in unterschiedlicher Weise einen Überblick über angelegte Objekte und deren Eigenschaften. In mysql bekommen wir mit

```
show tables;
```

eine Liste aller Tabellen. Mit

```
desc artikel;
```

wird eine Beschreibung der Tabelle mit den Namen der Felder und den Datentypen angezeigt. Im weiteren brauchen wir noch zumindest zwei Tabellen, um auch die Kunden und die Bestellungen abspeichern zu können:

```
create table kunde (
  kid char(10),
  passwort char(10),
  name char(40),
  adresse char(60));
```

Für die Bestellungen gibt es verschiedene Varianten der Umsetzung, wir wählen die einfachste mit nur einer Tabelle:

```
create table bestellung (
  kunde char(10),
  artikel char(10),
  menge int,
  datum date,
  status char(10));
```

Die Bestellungen werden in der Datenbank nicht zusammengefaßt, damit ersparen wir uns die Vergabe einer Bestellnummer und eine eigene Tabelle für die Bestellpositionen. Immerhin können wir mit dem Statusfeld verschiedene Prozesse unterstützen, z.B. den Versand einzelner Positionen, weil nur bestimmte Artikel in ausreichender Stückzahl lagernd sind.
Für die folgenden Abfragen brauchen wir einige Datensätze in den beiden Tabellen, z.B.

```
insert into kunde values ('huber', 'mausi1', 'Alois Huber', '1090 Wien, Schlickplatz 3/2');
insert into kunde values ('sb', 'mausi2', 'Sepp Bauer', '1020 Wien, Taborstrasse 4/6');
insert into bestellung ('huber', 'IR01', 1, '2006-07-12', 'versandt');
insert into bestellung ('sb', 'IR01', 2, '2006-07-14', 'offen');
insert into bestellung ('sb', 'HD032', 1, '2006-07-14', 'offen');
```

Der Datentyp **date** erlaubt die Eingabe von Datumswerten in verschiedensten Formaten und das 'Rechnen' mit diesen Werten:

```
select datum, dayofweek(datum), month(datum),
to_days(curdate()) - to_days( datum)
from bestellung;
```

4.2 Abfragen aus mehreren Tabellen

Eine Abfrage aller Bestellungen mit `select kunde, datum from bestellung` liefert die eben eingegebenen Werte. Die Kundenkennungen sind dabei aber nicht unbedingt aussagekräftig; wir würden gerne die Namen der Kunden sehen. Wir können dazu die beiden Tabellen in einer Abfrage verbinden (**Join**).

```
select name, datum from kunde, bestellung where kid = kunde;
```

Als Resultat liefert die Datenbank jene Kombinationen der Tupel aus den Tabellen `kunde` und `bestellung`, wo die Felder `kunde` und `kid` identisch sind.
Der Effekt dieser Abfrage ist die **Expansion** der Kundennummern: statt wenig aussagekräftiger Kunden-IDs sehen wir die Namen der Kunden. Die where-Klausel selektiert aus allen möglichen Kombinationen von Kunde und Bestellung nur jene, die im aktuellen Zusammenhang Sinn machen. Führen Sie die Abfrage ohne where-Klausel durch und vergleichen Sie die Resultate!
Als weiteres Beispiel folgt eine Liste der Bestellungen mit Artikelbezeichnungen:

```
select bez, kunde, datum
from artikel, bestellung
where aid = artikel;
```

4.3 Gruppierungsfunktionen

SQL stellt eine Reihe von Gruppierungsfunktionen für einfache Datenauswertungen zur Verfügung, u.a. `count()`, `sum()`, `min()`, `max()`, `avg()`. Eine solche Funktion wird auf Werte aus mehreren Zeilen angewendet:

```
select count(*) as anzahl from bestellung;
```

zählt die Datensätze in der Tabelle bestellung. Mit der Option `as anzahl` geben wir der Ergebnisspalte einen sprechenden Namen.

```
select max(menge) from bestellung;
```

liefert den höchsten Wert in der Spalte `menge`.
Wirklich interessant werden die Gruppenfunktionen mit der `group by` Klausel: damit wird die Bildung der Gruppen gesteuert, auf die dann die Funktion jeweils angewendet wird. Das Ergebnis enthält soviele Zeilen, wie es Gruppen gibt:

```
select kunde, count(*) as bestellungen
from bestellung group by kunde;
```

Für jeden Wert in der Spalte `kunde` wird gezählt, wieviele Datensätze mit diesem Wert vorhanden sind, m.a.W. wieviele Bestellungen von diesem Kunden stammen. Noch ein Beispiel: die Bestellungen nach Datum und Kunde zusammengefaßt, dazu jeweils Anzahl der Positionen:

```
select kunde, datum, count(*) as positionen
from bestellung group by datum, kunde;
```

Weitere Aufgaben:

- Gesamtbetrag pro Bestellung
- Absatz pro Artikel
- Umsatz pro Artikel
- Umsatz pro Kunde
- Umsatz pro Tag
- Umsatz pro Wochentag, d.h. für alle Montage, Dienstage usw.
- Umsatz pro Jahr und Monat
- Bestellungen der letzten 10 Tage

Überlegen Sie Erweiterungen und Verbesserungen der Warenwirtschaft! So könnte das Bestellwesen mit zwei Tabellen umgesetzt werden: Bestellung mit laufender Nummer, Datum und Kunde, und dazu Bestellpositionen mit Artikel und Menge. Außerdem könnten die Artikel zu Artikelgruppen zugeordnet werden, was für den Katalog und die Auswertungen vorteilhaft wäre.

5 Interaktive Webapplikationen mit JSTL

Dynamische Inhalte werden bei jedem Request neu generiert; **statische** werden von einer Datei gelesen und ändern sich erst, wenn die Datei editiert wird. Es ist naheliegend, daß viele Anwendungen nur dynamisch praktisch realisierbar sind. Denken Sie nur an die Artikelliste: jedesmal, wenn ein neuer Artikel dazukommt oder sich ein Preis ändert, müßte eine statische HTML-Seite editiert werden. Das ist sehr aufwendig und fehleranfällig und bei größerem Datenbestand nicht mehr zu machen. Eine dynamische Lösung mit Datenbankanbindung bietet sich an. Die Artikeldaten werden in der Datenbank gehalten, und alle HTML-Seiten werden aus diesen Daten generiert.
Java Server Pages (**JSP**) ermöglichen uns, dynamische Inhalte in Webseiten einzubinden. Dazu wird in HTML-Seiten Java Code verwendet, der serverseitig bei jedem Request ausgeführt wird und HTML-Code generiert. Die Programmiersprache Java ist aber für unsere Aufgaben unnötig komplex. Mit der JSP Standard Tag Library (JSTL) erreichen wir das gleiche über eine einfachere HTML-ähnliche Syntax.
Der JSP-Code wird nicht vom Webserver ausgeführt, sondern vom Applikationsserver Tomcat; davon merken wir normalerweise nichts, nur wenn wir eine Fehlermeldung bekommen.

Lesen Sie Fehlermeldungen sorgfältig!

Diese Meldungen sind dafür gemacht, um Ihnen bei der Behebung von Fehlern zu helfen. Falls Sie manche Ausdrücke in der Meldung nicht verstehen, verschaffen Sie sich zuerst darüber Klarheit, bevor Sie weiterarbeiten. Das kostet zwar etwas Zeit, aber erspart viel unnötige Arbeit. Gleichzeitig lernen Sie etwas über die Sprache und die Umgebung, mit der Sie arbeiten, und auch das rentiert sich.
Im folgenden Beispiel soll das aktuelle Datum und die Uhrzeit in einem bestimmten Format angezeigt werden. Diese Daten ändern sich natürlich bei jedem Request. In den ersten beiden Zeilen werden `c:` und `fmt:` als Prefix für die JSTL Elemente definiert, damit es keine Namenskonflikte mit HTML-Elementen geben kann. Für viele Aufgaben können wir **Java Beans** als vordefinierte Bausteine verwenden, wie hier für das Datum. Sie können wieder Ihre Datei `test.jsp` mit dem Texteditor bearbeiten, oder auch eine neue JSP-Datei anlegen (Debian/Linux Gnome Menü *Applications/Accessories*).

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>

<jsp:useBean id="now" class="java.util.Date"/>

<p> Datum und Uhrzeit:
<fmt:formatDate value="${now}" pattern="yyyy-MM-dd HH:mm:ss"/>
```

Beachten Sie, wie wir HTML und JSTL Elemente nach Belieben mischen können. Bei jedem Request wird der Inhalt der Datei neu ausgeführt, und zwar vom Applikationsserver Tomcat im Zusammenspiel mit dem Webserver Apache: der Request geht zunächst ans Port 80, dort

erkennt der Webserver an der Endung, daß es sich um eine JSP-Seite handelt und reicht den Request an den Applikationsserver weiter. Dieser führt die Anweisungen in der JSP-Seite aus und schickt den generierten HTML-Code zum Apache zurück, der ihn an den Webbrowser weiterleitet. Wir könnten den Request auch direkt an Tomcat schicken, dann müßten wir allerdings ein anderes Port angeben.
Suchen Sie mit Google nach der genauen Beschreibung von `formatDate` und weiteren nützlichen Beans, die Sie dann in Ihrer Seite ausprobieren.

5.1 Sitzungsvariablen

Wir wollen den Kunden die Möglichkeit geben, sich bei unserem System anzumelden, d.h. sich für die Dauer ihrer Sitzung zu identifizieren. Dazu speichern wir die Kundenkennung in einer Session Variable.
Wir holen uns dazu zunächst aus einem Formular anmelden.jsp die zu speichernde Kennung, die der Benutzer eingibt, um sich zu identifizieren.

```
<h2>Anmeldung</h2>
<p> Geben Sie hier Ihre Kundennummer an:
<form action=setkid.jsp>
<input type=text name=kid>
<input type=submit value=OK>
</form>
```

Im nächsten Schritt setzen wir in der Datei `setkid.jsp` die Sitzungsvariable `kid`, die wir vom Formular als param.kid bekommen.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html>
<c:set var="kid" scope="session" value="${param.kid}"/>
Sie sind nun angemeldet.
</html>
```

Durch die Option `scope=session` steht uns diese Variable nun in allen anderen JSP-Seiten während dieser Sitzung zur Verfügung. Probieren Sie das in einer beliebigen anderen JSP-Seite:

```
<c:out value="${kid}"/>
```

Damit sich unsere Benutzer auch wieder abmelden können, sehen wir auch dafür eine Möglichkeit vor, z.B. in `abmelden.jsp`:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<c:remove var="kid" scope="session"/>
Sie sind nun abgemeldet.
```

Neben den von uns gesetzten Variablen hat jede Sitzung eine Reihe weiterer Werte, u.a. auch ein 32 Zeichen langes Session ID, das die Sitzung identifiziert. Im folgenden Beispiel bringen wir diesen Wert in die Variable sid, die wir im weiteren dann verwenden können, z.B. für den Warenkorb.

```
<c:set var="sid" scope="session"><%=session.getId()%></c:set>
<c:out value="${sid}"/>
```

5.2 Verbindung zur Datenbank

Damit SQL-Befehle verwendet werden können, muß zunächst eine Verbindung zur Datenbank hergestellt werden. Das geschieht z.B. auf der Startseite `start.jsp` der Applikation:

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<h2>Start Page</h2>

<sql:setDataSource
  var="ds"
  scope="session"
  driver="com.mysql.jdbc.Driver"
  url="jdbc:mysql://localhost:3306/j0123456"
  user="j0123456"
  password="secret" />

<p> Sie sind
<c:choose>
  <c:when test="${not empty kid}">
    als Kunde <c:out value="${kid}"/> angemeldet.
  </c:when>
  <c:otherwise>
    nicht angemeldet.
  </c:otherwise>
</c:choose>
```

Die `dataSource` wird als Sitzungsvariable `ds` gespeichert, d.h. sie steht uns danach in allen JSP-Seiten zur Verfügung. Weiter unten zeigen wir an, ob unser Benutzer schon angemeldet ist oder nicht. Dazu prüfen wir mit dem JSTL-Element `choose`, ob die Variable kid leer ist.

5.3 Erfassen

Nachdem die Artikeldaten im bereits erstellten Formular `artikeleingabe.jsp` eingegeben wurden, sollen sie mit `artikelinsert.jsp` in die Tabelle in der Datenbank geschrieben

werden. Dabei prüfen wir, ob ein Fehler aufgetreten ist; in diesem Fall geben wir die Fehlermeldung an den Benutzer zurück. Hier wird das normalerweise eine schon vergebene Nummer sein.

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<c:catch var="msg">
  <sql:update dataSource="${ds}">
    insert into artikel values (
     ? <sql:param value="${param.aid}"/> ,
     ? <sql:param value="${param.bez}"/> ,
     ? <sql:param value="${param.preis}"/>,
     ? <sql:param value="${param.bestand}"/> )
  </sql:update>
</c:catch>

<c:choose>
  <c:when test="${not empty msg}">
    <b>Fehler beim Erfassen:
    <c:out value="${msg}"/>
  </c:when>
  <c:otherwise>
    <p> Artikel wurde erfasst.
  </c:otherwise>
</c:choose>
```

5.4 Listen

Die Datensätze in der Tabelle `artikel` wollen wir natürlich ebenfalls über WWW anzeigen lassen:

```
<%@ taglib prefix="c"   uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql"  %>

<h2>Artikel</h2>

<sql:query var="artikel" dataSource="${ds}">
  select aid, bez, preis from artikel
</sql:query>

<table>
  <c:forEach var="x" items="${artikel.rows}">
    <tr>
```

```
      <td> <c:out value="${x.aid}"/> </td>
      <td> <c:out value="${x.bez}"/> </td>
      <td> <c:out value="${x.preis}"/> </td>
    </tr>
  </c:forEach>
</table>
```

Mit wenigen Änderungen können Sie so auch Listen für Kunden und Bestellungen erstellen. Die Ergebnisse von Auswertungen aller Art werden ebenso über das Web zugänglich, wie z.B. Umsatz pro Tag, Umsatz pro Artikel usw. Erstellen Sie alle sinnvollen Listen und Auswertungen und machen Sie Links auf Ihrer Startseite.

5.5 Dynamisch Zusammengesetzte URLs

Eine Liste aller Bestellungen können wir nun erstellen, wenn wir allerdings nur die Bestellungen eines bestimmten Kunden ausgeben wollen, müssen wir einen Weg finden, der Prozedur die Kundennummer zu übergeben. Das können wir mit einem zusammengesetzten URL erreichen:

```
<a href=bestellungenvonkunde.jsp?kunde=<c:out value="${kid}/>">Bestellungen</a>
```

Die Folgeprozedur bestellungenvonkunde.jsp bekommt nun einen Parameter `kunde`, mit dem wir die SQL-Abfrage einschränken können:

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

Bestellungen von Kunde <c:out value="${param.kunde}"/>:

<sql:query var="bestellungen" dataSource="${ds}">
  select artikel, menge from bestellung
  where kunde = ? <sql:param value="${param.kunde}"/>
</sql:query>

<table>
<c:forEach var="x" items="${bestellungen.rows}">
  <tr>
  <td> <c:out value="${x.artikel}"/>
  <td> <c:out value="${x.menge}"/>
  </tr>
</c:forEach>
</table>
```

Überlegen Sie, wie mit dieser Technik die bereits bestehenden Listen in ihrer Funktion erweitert werden können. Nützlich wäre z.B. in der Liste aller Kunden ein Link bei jedem Kunden, der zu den Bestellungen dieses Kunden führt. Sicher finden Sie noch viele weitere Anwendungen! Fast alle Felder in unseren Listen können mit solchen zusammengesetzten Links unterlegt werden, die dann Detailinformationen zum jeweiligen Wert liefern.

5.6 Weitere Aufgaben

Damit haben wir alle Konzepte kennengelernt, die wir für die weitere Umsetzung der Aufgabe brauchen. Wir wollen folgendes erreichen:

1. Neben der Artikelliste soll auch eine Liste der Kunden und der Bestellungen im Web verfügbar sein.
2. Die gewünschten Auswertungen Umsatz pro Tag usw. sollen ebenfalls im Web verfügbar sein.
3. Kunden sollen sich selbst registrieren sowie Ein- und Ausloggen können.
4. Alle Benutzer unserer Website (auch nicht registrierte) sollen Artikel in einen Warenkorb legen können.
5. Der Inhalt des Warenkorbs soll natürlich angezeigt werden können.
6. Die Mengen im Warenkorb sollen geändert werden können.
7. Der Inhalt des Warenkorbs soll bestellt werden können.
8. Die Kunden sollen Ihre eigenen aktuellen und historischen Bestelldaten ansehen können.

Hinweise auf Fehler in diesem Text und andere Anregungen bitte an mitloehn@wu-wien.ac.at