Prof. Jürgen Sauer

Objektorientierte Programmierung wissensbasierter Systeme

Skriptum zur Vorlesung im WS 2002 / 2003

Inhaltsverzeichnis

4. Objektorientierte Wissensrepräsentation im Internet

4.1 Kommunikations- und Dokumentationsnetze
4.1.1 Vernetztes Wissen
4.1.2 Internet, WWW mit HTML, XML
4.1.3 HTML-Grundlagen
4.1.4 Arbeitsweise eines Web-Servers

4.2 Client/Server und Internet/Intranet
4.2.1 Grundlagen
4.2.2 Grundkonzepte und Anwendungen

4.3 Die Verbindung zur Datenbank über JDBC
4.3.1 Zugriff auf eine relationale DB mit JDBC
4.3.2 Datenbanktreiber für den Zugriff
4.3.3 Verbindungen zur DB
4.3.4 Datenbankabfragen
4.3.5 Hinzufügen von Elementen zu einer DB
4.3.6 Metadaten
4.3.7 Ausnahmen bei JDBC
4.3.8 JDBC-Programmierung
4.3.8.1 Anwendungen/Applets mit JDBC
4.3.8.2 Anwendungen/Applets mit Embedded SQL (SQLJ)

5. Verteilte Objektarchitekturen

5.1 Konzept der Applets
5.1.1 Beschreibung des Konzepts
5.1.1.1 Aufgaben und Erläuterung des Konzepts
5.1.1.2 Lebenszyklus von Applets
5.1.1.3 HTML-Tags für den Einsatz von Applets
5.1.2 Applets und Anwendungen
5.1.3 Methoden zur Ereignisbehandlung in Applets
5.1.3.1 Event-Handling inter JDK 1.0
5.1.3.2 Ereignisbehandlung unter grafischen Benutzerflächen
5.1.3.3 Anwendung lokaler Klassen zur Ereignisbehandlung
5.1.3.4 Dialoge
5.1.4 Multithreading-fähige Applet
5.1.5 Animation in Applets
5.1.6 Laden und Anzeigen von Bildern
5.1.7 Die Ausgabe von Sound
5.1.8 Applets in Archiven

5.2 Konzept des Networking
5.2.1 Networking mit URLs
5.2.2 Kommunikation mit einem CGI-Skript

1. Beschreibung und Präsentation von Wissen

Intelligenz erfordert Wissen. Ohne Wissen gibt es keine Intelligenz. Wissen ist in der Regel sehr umfangreich, schwer zu charakterisieren und ändert sich dauernd. Diese Eigenschaften beeinflussen weitgehend die Beschreibung von Wissen, z.B. in einem Rechensystem.
Wissen ist die in einem System (Computer, Mensch, Bibliothek, usw.) gespeicherte Information. Dabei kann Wissen folgende Eigenschaften besitzen:

- Wissen wird allgemein für wahr gehalten.
- Wissen kann falsch sein
- Allgemein für wahr gehaltenes Wissen kann sich als falsch herausstellen
- Wissen kann Widersprüche umfassen

Wissen kann somit als Informations-Ressource aufgefasst werden.
In der Literartur gibt es keine anerkannte Definition für Wissensverarbeitung. Die fogende Abbildung

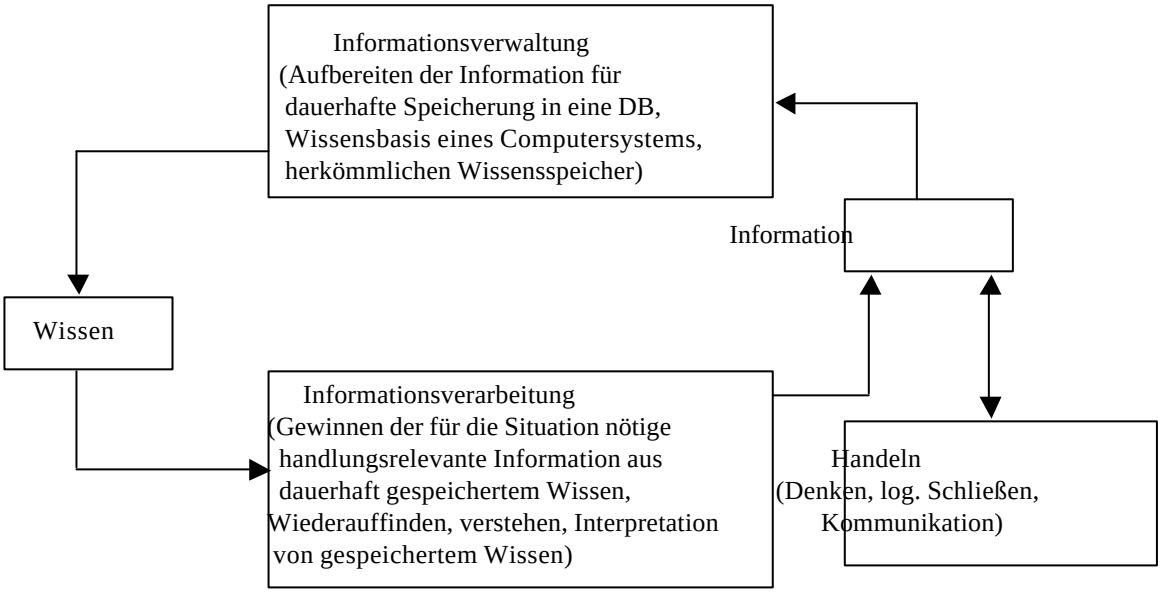


Abb.: Kreislauf von Wissen und Information

zeigt den Kreislauf der Transformation von Wissen in Information und umgekehrt. Darin steht Handeln für Denken, logisches Schließen und Kommunikation, durch das Informationen aus der Umgebung aufgenommen wird. Handeln steht aber auch für das Verhalten in problematischen, unbekannten Situationen, in denen ein System (Mensch, Rechner) Informationen benötigt, um zu handeln.
Wissensverarbeitung befasst sich mit der Beschleunigung, der Rationalisierung und Automatisierung der Transformation von Wissen in Information und umgekehrt. Die Wissensverarbeitung bedient sich dazu der Methoden der KI (Simulation menschl. Intelligenz), der Heuristik (Umsetzen menschl. Problemlösens in die Programmierung) und des automatischen Erkennes (Bild- und Spracherkennung).

1.1 Wissensverarbeitung und Wissensverbreitung

Rechnergestützte Wissensverbreitung

Wissensverarbeitung erfolgt durch einen Kommunikationsprozeß zwischen dem Informationsproduzenten und dem Informationsempfänger. Dieser Prozeß kann direkt oder über den Umweg eines Speichers erfolgen.

"Sender-Kanal-Empfänger"-Schema

Sender und Empfänger stehen zeitgleich miteinander in Verbindung. Bei dem Kanal kann es sich um ein direktes Gespräch, ein Telefonat, ein Chat, eine Wissenskonferenz oder ähnliches handeln.

"Sender-Speicher-Empfänger"-Schema

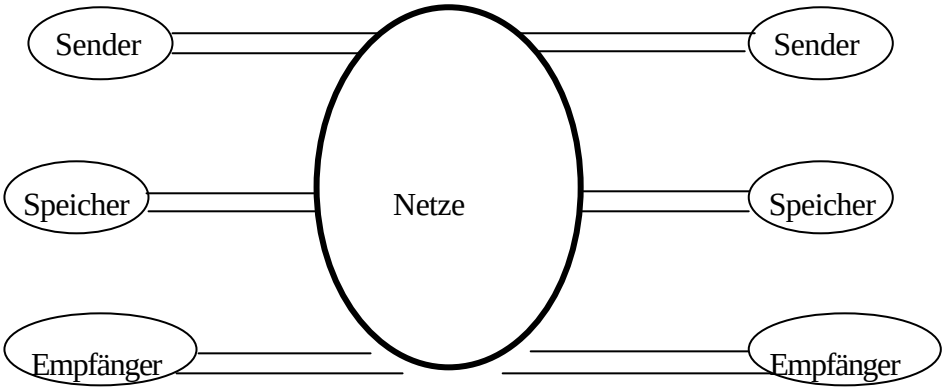


Abb.: "Sender-Speicher-Empfänger"-Schema

Der Sender produziert seine Dokumente für einen Speicher. Aus diesem Speicher können ein oder mehrere Empfänger, je nach Bedarf, Dokumente abrufen. Dokumente stellen die kleinsten Einheiten des Wissensspeichers dar. Die Speicher müssen die einzelnen Dokumente in einer Form aufbewahren, die eine Speicherung über einen größeren Zeitraum gestattet und Möglichkeiten zur Verfügung stellen, gezielt nach bestimmten Dokumenten zu suchen. Bsp. für solche Speicher sind Bibliotheken, das Internet, die Massenmedien, der Buchhandel, usw. Die Kommunikation des Senders bzw. des Empfängers mit dem Speicher erfolgt über Netze. Bei den Netzen handelt es sich um ein elektronisches Netz wie das Vertriebsnetz eines Verlags oder die Fernleihe der Universitätsbibliothek oder allg. das Internet.

Verfahren zur Beschreibung von Wissen

1. Logische Verfahren

Die Wissensbasis wird durch Ausdrücke der formalen Logik präsentiert. In der logischen Programmierung wird Wissen in der Form logischer Ausdrücke (z.B. Fakten, Regeln) beschrieben. Die Ausdrücke werden als Axiome in die Wissensbasis aufgenommen. Zur Verarbeitung des Wissens dienen Inferenzregeln, mit deren Hilfe Beweisverfahren (z.B. Resolutionsverfahren von

Robinson, Prolog-Interpreter) definiert werden können. Mögliche Schlußfolgerungen (Inferenzen) sind Ausdrücke, die über die Inferenzregeln aus den Axiomen ableitbar sind. Das am häufigsten verwendete Darstellungsschema ist das **Prädikatenkalkül** 1. Ordnung. Die Programmiersprache **Prolog** stützt sich auf dieses Kalkül und ist daher für die Implementierung von Wissensbasen mit logischen Repräsentationsverfahren besonders geeignet.
Objekte, die mit diesen Darstellungen beschrieben werden können, sind jedoch recht einfach. Ein großer Teil der komplexen Struktur aus der Problemwelt kann deshalb nicht erfaßt werden. Die Problemwelt enthält bspw. Einzelobjekte, die jeweils mehrere Eigenschaften haben (einschl. der Beziehungen zu anderen Objekten). Es ist nützlich, solche Eigenschaften zusammen zu fassen, um eine umfassende Beschreibung eines komplexen Objekts herzustellen. Ein Vorteil eines solchen Schemas ist, daß es ein System in die Lage versetzt, seine Aufmerksamkeit auf vollständige Objekt zu konzentrieren, ohne dabei alle bekannten Fakten berücksichtigen zu müssen.

2. Netzwerk-Verfahren

Sie präsentieren das Wissen durch einen Graphen, bei dem die Knoten die Objekte und Konzepte des Problemgebiets darstellen, und die Kanten die Beziehung zwischen diesen Objekten oder Objekttypen. **Semantische Netze**[1] sind dafür ein Beispiel. Im Gegensatz zu Datenbankmodellen (Trennung von Schema und Instanz) zählen Objekte (Instanzen) zur Repräsentation von Wissen. Netzwerk-Verfahren sind demnach objektbezogene, graphische Verfahren.

[1] vgl. 1.2.2

1.2 Repräsentationsformen von Wissen

Praktisch kann man eine Wissensrepräsentation als die Abbildung eines Ausschnitts der realen Welt bezeichnen. Die Abbildung muß in der Art und Weise geschehen, dass die darin enthaltenen Informationen automatisiert verarbeitet werden können. Das Wissen kann implizit im Programmcode, somit in der Abfolge der Befehle gespeichert sein, oder explizit an bestimmten Stellen des Systems.

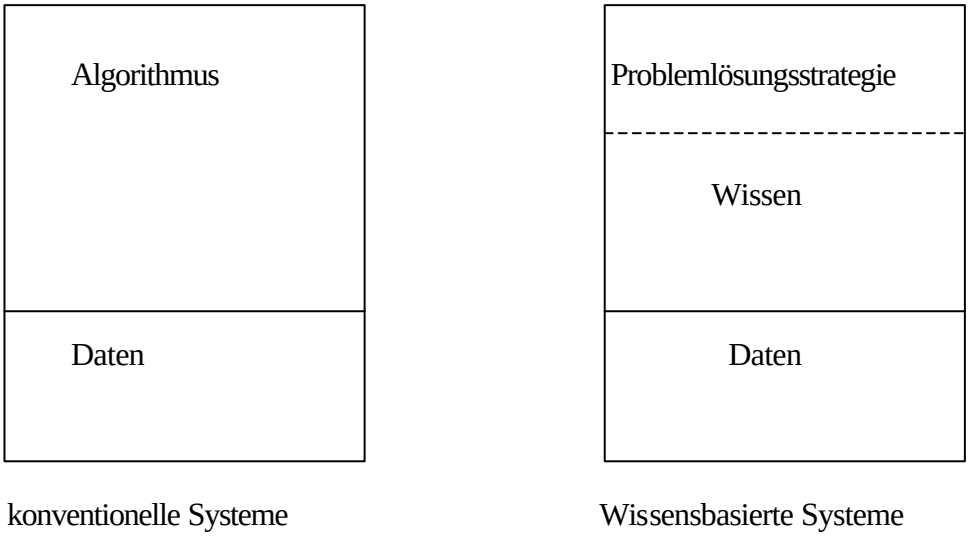


Abb.: Grundsätzlicher Aufbau von konventionellen Programmen und wissensbasierten Systemen

Konventionelle Programme, die Wissen implizit speichern, erschweren es, Wissen nachträglich zu verändern. Dazu wäre ein Eingriff in den Algorithmus notwendig.
Bei wissensbasierten Systemen existiert eine klare Schnittstelle zwischen anwendungsspezifischen Wissen und der allgemeinen Problemlösungsstrategie. Die Komponente "Allgemeine Problemlösungsstrategie" wird auch Inferenzmaschine genannt. Der Vorteil dieses Konzepts ist, dass man eine Problemlösungsstrategie für mehrere Anwendungsgebiete verwenden kann. Es muß nur die entsprechende Wissensbasis ausgetauscht werden.

1.2.1 Prozedurale Methoden zur Wissensrepräsentation

Beim prozeduralen Ansatz wird Wissen in Form einer Abfolge von Anweisungen, den Prozeduren gespeichert. Im Unterschied zu nicht wissensbasierten Programmen muß der Aufruf einer Prozedur aber explizit der Kontrolle des Programms unterliegen. Das bedeutet: das Programm weiß über seine Lösungsmöglichkeiten Bescheid und wendet diese entsprechend den vorliegenden Problemstellungen an.

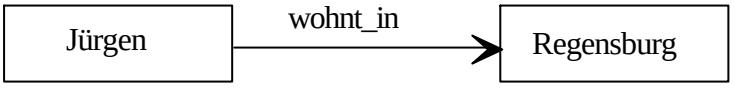
1.2.2 (Objektbezogene) graphische Beschreibung: Semantische Netze

Vereinbarungen zur graphischen Beschreibung

Beschreibungssobjekte sind Konzepte (Objekte, Entitäten, Entitätstypen) und Beziehungen zwischen Konzepten. Konzepte werden graphisch durch benannte Knoten dargestellt, Bezie-hungen zwischen Konzepten durch benannte Pfeile.

Bsp.:

1. Die einfache Gegebenheit "Juergen wohnt in Regensburg" läßt sich durch folgendes semantisches Netz darstellen:



2. Das folgende Netz stellt die Information dar, daß Jürgen einen Wagen der gehobenen Mittelklasse fährt (weißer BMW)

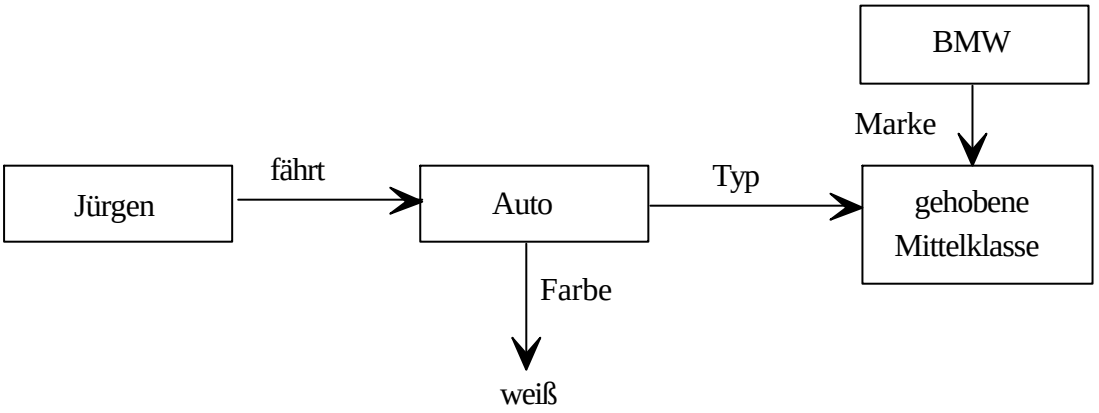


Abb. 1.2-1: Ein einfaches semantisches Netz

Von seiner Struktur her ist ein **semantisches Netz** ein gerichteter Graf mit den Konzepten als Knoten und Beziehungen als Kanten zwischen den Knoten.
Es gibt keine umfassende Übereinkunft darüber, wie die Knoten und die Kanten zu benennen sind. Man hält sich aber generell an die folgenden Vereinbarungen:

1. Knoten werden dazu benutzt, um Objekte und Eigenschaften darzustellen.
   Objekte können sowohl physische Gegenstände (z.B. Student, Dozent, Seminarraum, Fahrzeug) als auch gedankliche Elemente (z.B. Handlungen, Ereignisse, Vorlesungen) sein. Eigenschaften liefern zusätzliche Informationen über Objekte (grün, rot, dumm, intelligent).
2. Eine Kante kann jede Art von Beziehung darstellen.

Kantentypen in semantischen Netzen

Einige Kantentypen haben sich als besonders grundlegend herausgestellt und werden praktisch in allen semantischen Netzen verwendet:

ist_ein (is_a-Kante)

Diese Kante wird benutzt, um eine Beziehung zwischen Klasse und Subklasse darzustellen.
Bsp.: "auto ist_ein fahrzeug"
Damit wird ausgedrückt, daß die Menge der Autos in der Menge der Fahrzeuge enthalten ist.

hat bzw. **ist** (has bzw. is-Kante)
Die Kanten ordnen Objekte Eigenschaften zu.
Bsp.: "auto hat raeder"
Hier wird dem Konzept "auto" die Eigenschaft zugeordnet, daß ein Auto Räder hat.

instanz_von (instance_of bzw. auspraegung_von - Kante)
Diese Kante verbindet ein Individuum mit seinem generischen Typ. Dieser Kantentyp ist eine Hilfe zur Vermeidung möglicher Verwirrungen zwischen Konzepten und Instanzen solcher Konzepte.

Bsp.:

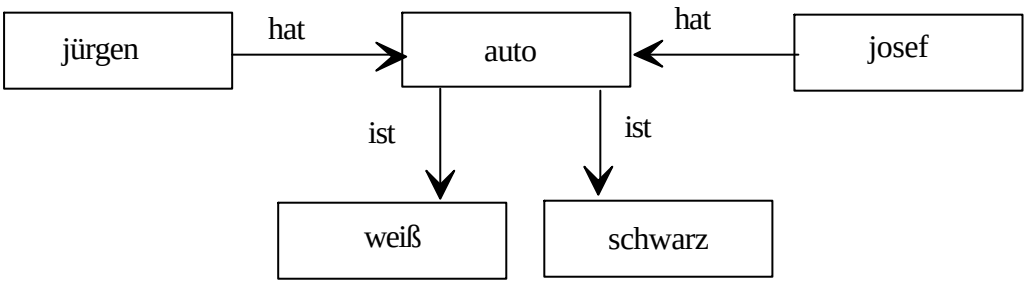


Abb. 1.2-2: Ein semantisches Netz, das Farben von Automobilen beschreibt

Autos sind in der Regel nicht gleichzeitig weiß und schwarz. Auto ist hier demnach kein Konzept, sondern die Instanz eines Konzepts. Eine Instanz verkörpert ein konkretes Individuum einer Klasse. Das vorstehende Netz muß also so richtig beschrieben werden:

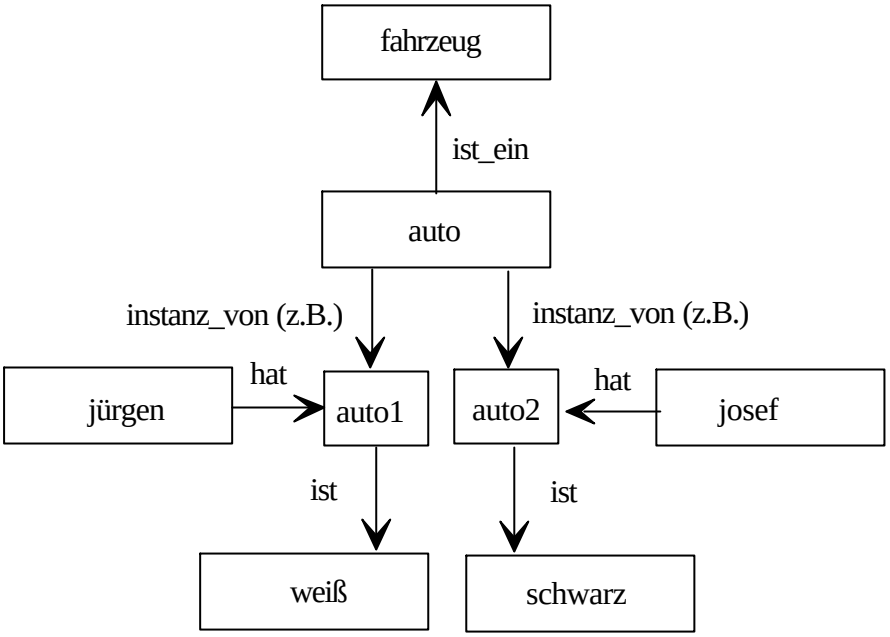


Abb. 1.2-3: Ein semantisches Netz, das die Farben verschiedener Automobile unterscheiden kann

Konzepte sollten allgemeine Eigenschaften von allen (oder den meisten) Instanzen eines Konzepts beschreiben, Instanzknoten sollten nur Eigenschaften individueller Objekte beschreiben. Eine der Hauptvorteile diese Repräsentationsschemas liegt in seiner Flexibilität. Neue Knoten und Glieder können nach Bedarf definiert werden.

Vererbung

Vererbung (inheritance) ist ein weiteres Merkmal semantischer Netze. Die Vererbung von Eigenschaften ist eine Folge der "ist_ein-Relation" und bedeutet: Alle Einzelfälle einer Klasse erben sämtliche Eigenschaften der übergeordneten Klasse, denen sie angehören.
Bsp.: In "auto ist_ein fahrzeug" und "fahrzeug hat raeder" erbt der Knoten "auto" die Eigenschaft "hat raeder" von "fahrzeug".
Der Vererbungsmechanismus bewirkt: sparsames Umgehen mit Speicherplatz. Es müssen nicht alle Fakten explizit gespeichert werden, sondern nur diejenigen, die mit dem allgemeinen Konzept in Verbindung stehen. Dies ist übrigens eine Eigenschaft semantischer Netze, die im menschl. Gedächtnis analog behandelt wird. Die Begriffe "semantische Netze", "Vererbungshierarchien" etc. stammen alle aus dem Zusammenhang über die Informationsspeicherung im menschlichen Gehirn. Psychologen benutzen semantische Netze als Gedächtnisprotokoll. Die Psychologen Coats/Parkin [2] haben bspw. solche einfachen Gedächtnisprotokolle beschrieben.

Modellierung mit Hilfe semantischer Netze

Die Modellierung mit Hilfe eines semantischen Netzes besteht aus zwei Teilen:

[2] vgl. Coats, R.B. und Parkin, A.: "Computer Modells in the Social Sciences", 1977

Der eine Teil enthält die Wissensbasis mit Konzepten und Beziehungen zwischen den Konzepten. Das menschl. Gedächtnis enthält eine große Anzahl solcher Konzepte z.B. "auto", "haus", "freiheit". Viele Bestandteile solcher Wissensstrukturen sind allen Menschen gemein. Manche sind aber nur spezifisch für bestimmte Individuen. So dürfen sich etwa die Wissensstrukturen eines Informatikers und eines beliebigen Durchschnittsmenschen über Hunde nicht unterscheiden, wohl aber über Computer bzw. Software.
Der andere Teil des Modells ist der Interpretationsprozeß, der die Informationen in der Wissensbasis benutzt und verarbeitet. Solche kognitiven Prozesse sind die Aufnahme von Information und die Reorganisation des Gedächtnisses. Aufnahme und Ausgabe von Information dienen zur Kommunikation mit der Welt. Das Modell hat Informationen aufzunehmen und einer geeigneten Form zu repräsentieren.
Die **Semantik semantischer Netze** wird über die Spezifikation von Such- und Inferenzalgorithmen auf diesen Netzen definiert. Dabei strebt man eine geringe Anzahl von Beziehungen an, um die Komplexität der Inferenzprozesse zu reduzieren.
Allgemein sind Semantische Netze "leicht" auf einem Rechner zu bearbeiten. Die Netze bestehen aus Knoten und Kanten, also Graphen. Graphen sind eine weit verbreitete Datenstruktur, die von vielen Programmiersprachen unterstützt wird ("Zeigertypen"in Pascal, "property lists" in Lisp).

Objekt-Attribut-Wert-Tripel

Ein Spezialfall semantischer Netze sind "**Objekt-Attribut-Wert**"-**Tripel** (OAW-Tripel). Das ist eine verbreitete Methode zur Darstellung von Fakten. Nach der unter der Bezeichnung "Objekt-Wert-Attribut"-Tripel vorliegenden Methoden können Schemata für die Beschreibung von Wissen (Daten) bestimmt werden.
In einem solchen Schema können Objekte

- Entitäten (z.B. ein Fachbuch, ein Transistor)
- begriffliche Einheiten (z.B. eine Vorlesung)

sein. Attribute sind die Eigenschaften der Objekte. Werte kennzeichnen die spezifische Beschaffenheit eines Attributs in einer bestimmten Situation.
Das "Objekt-Attribut"-Glied ist eine "**hat_ein**"-Relation, das "Attribut-Wert"-Glied ist eine "**ist_ein**"-Relation.
Objekte können bei der Darstellung als "**O-A-W**"-Tripel hierarchisch geordnet sein, z.B.

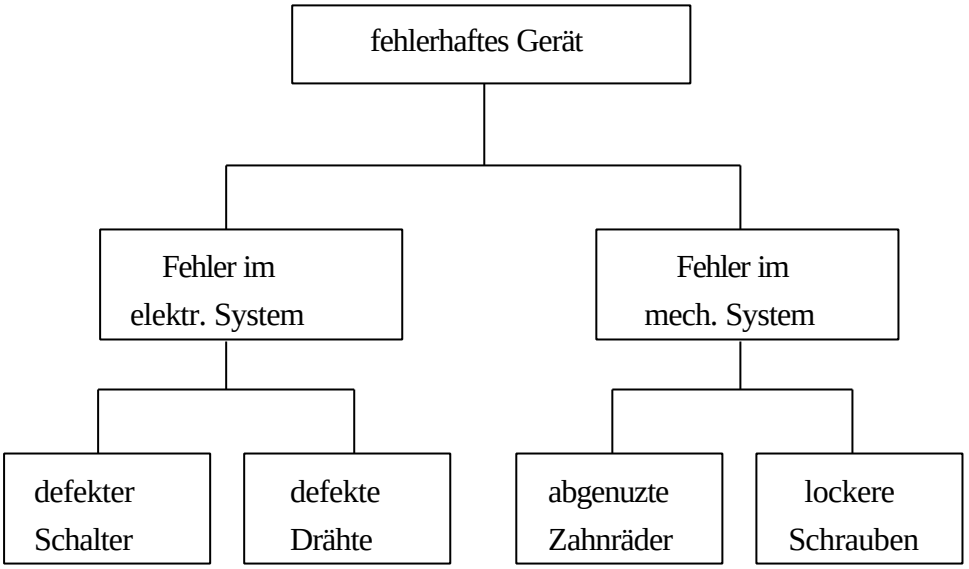


Abb. 1.2-4: Hierarchische Ordnung bei der Darstellung von OAW-Tripeln

Die Objekte in diesem Baum sind statisch. Eine dynamische Instanz dieser Hierarchie kann sich von Fall zu Fall ändern. Der Vorgang, bei dem die spezifischen Werte der in der statischen Wissensbank gespeicherten Attribute festgelegt werden, wird als Instanzierung bezeichnet.

Suchverfahren in Problemlösungs-Zustandsgraphen (Semantische Netze)

Allgemein beschreiben semantische Netze Problem-Lösungszustände und bedienen sich dabei der grafischen Darstellung (Knoten beschreiben einen Problem-Lösungszustand, Verbindungen zwischen den Knoten beschreiben die Übergänge von einem Problem-Lösungszustand zum anderen). Anhand solcher Graphen (semantischer Netze) werden häufig Algorithmen erklärt, die in der Wissensverarbeitung häufig angewendet werden. Die Problemlösungsverfahren bestehen in der Regel aus den folgenden 3 Schritten:

1. Formulierung des Problems durch Festlegen von Start, Ziel und den erlaubten Aktionen. Dabei wird eine sinnvolle Abstraktion der realen Welt vorgenommen
2. Suche: Es wird eine Folge von Aktionen gesucht, die vom Start zum Ziel führen.
3. Ausführung: Es wird die Aktionenfolge, die man als Ergebnis der Suche erhalten hat, Schritt für Schritt abgearbeitet.

Semantische Netze geben durch ihre grafische Repräsentation bei der Suche nach der Lösung wertvolle Hilfestellellungen.
Bsp.: Das folgende semantische Netz kann man als Graphen bezeichnen (mit Startknoten S und dem ZielKnoten Z), der eine Straßenkarte beschreibt. Die Knoten stellen Städte dar, die Verbindungen Straßen und die an den Verbindungsstrecken angegebenen Zahlen (Link Labels) Kosten, die eine Benützung dieser Verbindung verursacht.

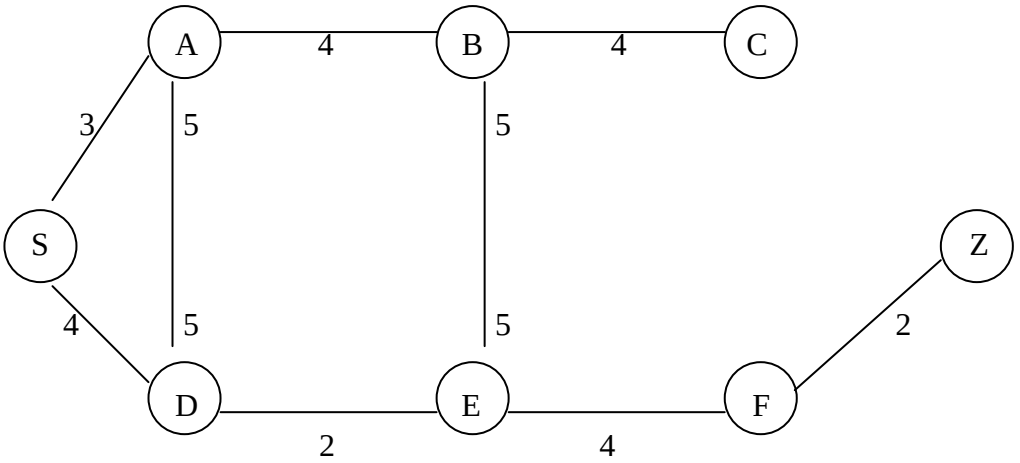


Abb.: Straßenkarte als semantisches Netz dargestellt. Die Städte entsprechen den Knoten, die Verbindungen den Straßen und die Link Labels den Kosten.

Die einfachste Methode einen Weg vom Start zum Ziel zu finden, ist das Betrachten aller möglichen Wege. Wege, die in Schleifen führen, werden nicht beachtet. Für den angegebenen Graphen lassen sich alle möglichen Pfade als Baum darstellen:

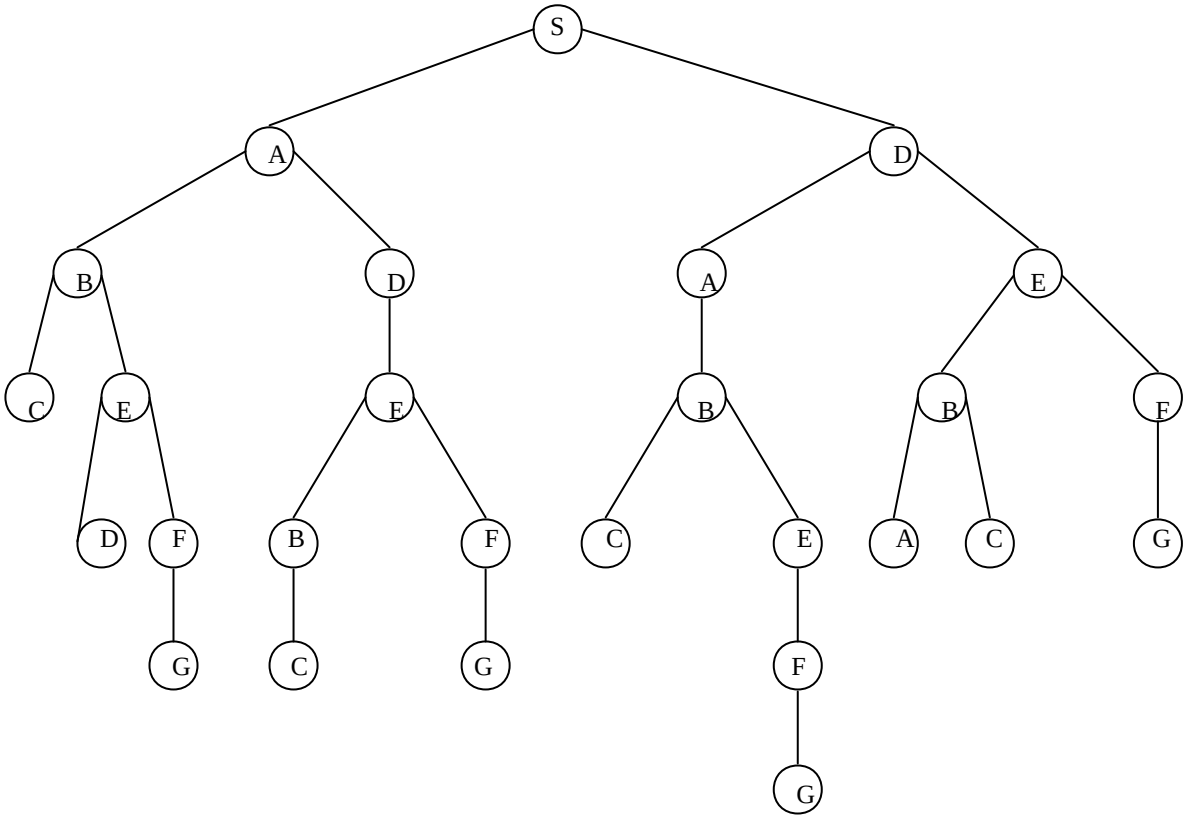


Abb.: Der Suchbaum für das Straßenkartenproblem. Jeder Knoten beschreibt einen Pfad, der vom Start-Knoten ausgeht.

Methoden zum Bestimmen der möglichen Lösungswege (Suchverfahren) sind:

- Blinde Suche
  Tiefensuche (depth first search)
  Breitensuche (breadth first search)
  nicht deterministische Suche

- Heuristisch informierte Suche
  Hill Climbing
  Best First Search
- Optimale Netzsuche
British Museum Procedure
Branch-and-Bound Search mit Abschätzung der Restkosten
Branch-and-Bound Search mit Eliminierung längerer Doppelwege(A*-Algorithmus)

Spielbäume

Spielbäume und die zugehörigen Algorithmen ermöglichen es, auf Rechnern eine bestimmte Art von Strategie-Spielen zu spielen. Bei den Spielen handelt es sich um Zweipersonenspiele, bei denen beide Spieler vollständig wissen, welche Züge gemacht wurden und welche noch möglich sind. Bsp. dafür sind Dame und Schach, Kartenspiele oder Backgammon bei denen der Zufall eine Rolle spielt, gehören nicht dazu.
Die folgende Abbildung zeigt einen Spielbaum, bei dem die Spieler bei jedem Zug zwischen zwei möglichen Zügen wählen können. Die "Nodes" in einem Spielbaum repräsentieren die Brettsituationen, die Links stehen für die erlaubten Züge, die zu neuen Brettsituationen führen.

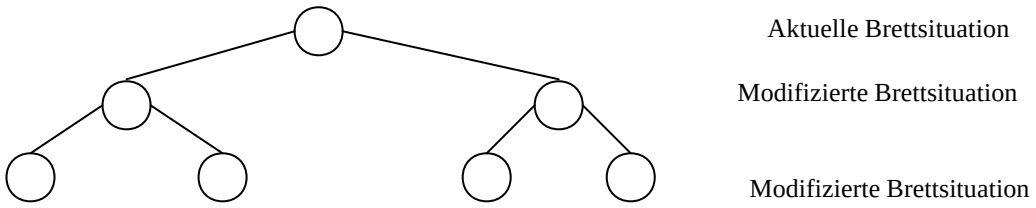
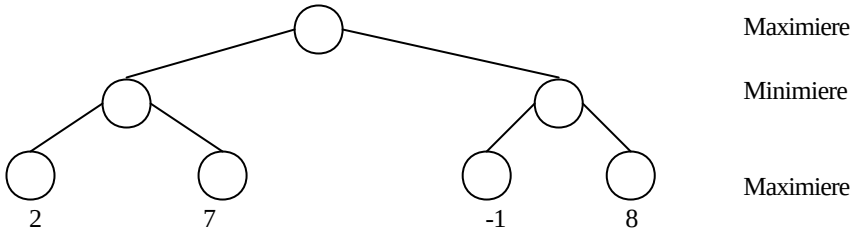


Abb.: Spielbaum, Nodes repräsentieren Brettsituationen, Links gültige Züge

Für ein komplexes System ist es jedoch nicht möglich, den vollständigen Spielbaum aufzubauen. Für Schach werden dazu 10^{120} Züge nötig.
Ein Situations-Analyzer bewertet, ob eine Brettsituation gut oder schlecht ist. Die Bewertung ist in dem Rückgabewert enthalten, der abhängig von der Brettsituation durch einen Zahlenwert ausdrückt, wie gut eine bestimmte Situation ist. Hohe Zahlenwerte (Absolutbetrag) stehen dabei für eine günstige Brettsituation. Das Vorzeichen der Zahlenwerte drückt aus, für welchen der beiden Spieler die Situation günstig ist. Ein Spieler hat somit das Ziel den Zahlenwert zu maximieren (Maximizer), der andere ihn zu minimieren (Minimizer).
Der MiniMax-Algorithmus baut einen Spielbaum bei einer bestimmten Ebene auf und addiert zu jedem Blatt mit Hilfe eines Situations-Analyser einen Zahlenwert.



Der **MiniMax**-Algorithmus beginnt seine Analyse eine Ebene über den Blättern. Befindet er sich in einer Minimisierer-Ebene, ordnet er dem Knoten das Minimum der Bewertungen seiner Kinder zu. In einer Maximierer-Ebene wird das Maximum bestimmt. Nachdem das für alle Knoten dieser Ebene erledigt ist, nimmt er die Bewertung eine Eben höher vor. Bei dem dann dem Root-Knoten

zugeordneten Wert eintspricht die Bewertung der Brettsituation, die der Spieler im schlechtesten Fall erreichen kann.
Da der Situations-Analyzer eine sehr rechenaufwendige Aufgabe darstellt, sollte man überlegen, ob er überhaupt für jedes Blatt aufgerufen werden muß. Der **Alpha-Beta**-Algorithmus zeigt, dass dies nicht nötig ist.

1.2.3 Logikorientierte Methode der Wissensrepräsentation

Aufbauend auf der Theorie der formalen Logik steht die Prädikatenlogik erster Stufe das bekannteste Kalkül zur Darstellung von logischen Sachverhalten dar. Ein Beispiel für eine Formel dieser Präsentation ist:

$$\forall X\,(person(X) \wedge \neg(X = eva) \wedge \neg(X = adam)) \Rightarrow \exists Y(person(Y) \wedge mutter\_von(X) = Y)$$

Jede Person, ausgenommen Adam und Eva, hat mindestens eine Mutter. Eine Formel der Prädikatenlogik erster Stufe setzt sich aus folgenden elementaren Zeichensymbolen zusammen:

- Konstantensymbole, z.B. adam, eva, 10
- Variablensymbole (mit Großbuchstaben beginnende Zeichenfolgen)
- Funktionssymbole zur Darstellung n-ärer Funktionen[3] (hier durch mit Kleinbuchstaben beginnende Zeichenfolgen dargestellt, z.B. mutter_von)
- Prädikatensymbole zur Darstellung n-ärer Prädikate, z.B. person
- Junktoren $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- Quantoren
- Klammern oder Kommas als Hilfssymbole
- "=" (Zeichen für Gleichheit)

Die Prädikatenlogik erster Stufe ist zwar eine sehr ausdrucksvolle Logik, eignet sich aber leider nicht zur Implementierung. So nehmen bspw, Formelmanipulationen sehr viel Rechenzeit in Anspruch. Aus diesen Grunde werden meistens sehr stark reduzierte Teilsysteme in der Praxis verwendet.

1.2.3.1 Einfache Fakten-Regel Systeme

Fakten haben die Form $y\,(c_1,...,c_n)$ mit einem Prädikatensymbol y und Konstantensymbolen c_1, ..., z.B.: mensch(juergen), vater(hans,maria), gerade(2), teiler(2,4).
Regeln sind Implikationen folgender Form:

$$Q(y_1(t_1^1,...) \wedge y_2(t_1^2,...) \wedge y_n(t_1^n,...)) \Rightarrow y_0(t_1^0,...)$$

y_i: Prädikatensymbole

t_j': Variablen-, Konstantensymbole

[3] Funktionen mit n Konstanten oder Variablensymbolen

Q: Quantorpräfix (enthält für genau jedes in der Formel vorkommendes Variablensymbol X einen Allquantor in der Form $\forall X$

Zusätzlich muß jede Variable, die rechts vom Implikationszeichen vorkommt, auch auf der linken Seite vorkommen. Der Junktor $\wedge$ bindet stärker als $\Rightarrow$.
Bsp.:
$\forall X\,(mensch(X) \Rightarrow sterblich(X))$,
$\forall X \forall Y (kind(X,Y) \wedge mensch(X) \Rightarrow mensch(Y))$,
mensch(hans),
kind(hans,ernst),
kind(olga,ernst).

1.2.3.2 Logische Programmiersprache: Prolog

Grundlagen

Grundlage von Prolog (vor allem der in Prolog implementierten Logiksprache) ist eine Teilmenge der Prädikatenlogik, die **Horn-Logik**. Prolog-Programme sind Konjunktionen von sog. Programm-Klauseln (universelle, definierte Horn-Formeln, definite clauses). **Programm-Klauseln** sind Implikationen von nichtnegierten Konditionen (Bedingungen) und einer nichtnegierten Konklusion. Eine **Horn-Klausel** ist eine spezielle Klausel:

$b_1 \leftarrow a_1,\ldots,a_n$

b_1 ist die Konklusion der Horn-KLausel. Enhält die Horn-KLausel Variablen X_1, ... , X_k, so ist das so zu interpretieren:

Für alle X_1 ... X_k gilt: $b_1 \leftarrow a_1 \wedge \ldots \wedge a_n$

Vier Fälle sind zu unterscheiden :

(1) m = 1, n = 0

$b_1 \leftarrow$

Die ist eine einfache atomare Formel, die von keiner Bedingung abhängig ist. In Prolog schreibt man bspw.

teil(e1,einzelteil_1).
struktur(p1,b1,2).

Jede Klausel wird in Prolog durch einen Punkt abgeschlossen. Klauseln, die durch Fall (1) beschrieben werden, sind **Fakten**.

(2) m = 1, n <> 0

$b_1 \leftarrow a_1 \wedge ... \wedge a_n$

Dies ist der übliche DANN - WENN - Fall.

In Prolog schreibt man

- anstelle von "←" das Symbol "**:-**"
- anstelle von "∧" ein Komma.

In dieser Form beschreibt Prolog **Regeln**. Das folgende Prädikat definiert eine derartige Regel:

```
grossvater(Y,X) :- vater(Y,Z), vater(Z,X).
```

Das bedeutet: X ist der Großvater von Y, wenn Z der Vater von Y und X Vater von Z ist. Das Prädikat `grossvater(Y,X)` ist nur beweisbar, wenn die beiden Fakten `vater(Y,Z)` und `vater(Z,X)` vorliegen.

Regeln haben in Prolog folgendes allgemeines Format:

```
schlussfolgerung :- bedingung_1, bedingung_2, ....
```

Eine **Regel** besteht formal aus dem Regelkopf und dem Regelrumpf. Beide sind verbunden über das Symbol „**:-**“.
Die Schlußfolgerung ist dann wahr, wenn alle Bedingungen des Regelrumpfes erfüllt sind.
Regeln definieren den Zusammenhang, in dem die Fakten eines Prolog-Programms interpretiert werden. Regeln und Fakten bilden die Wissensbasis des Programms.

(3) m = 0, n <> 0

$\leftarrow a_1 \wedge ... \wedge a_n$

Die Formel besteht aus Bedingungen. Sie kann so interpretiert werden:

Es ist nicht der Fall, daß gilt $a_1 \wedge ... \wedge a_n$

Diese Ausprägung einer Klausel gibt es in Prolog in der Form einer **Anfrage**. Allerdings wird hier das Symbol "←" durch ein "**?-**" ersetzt.
Anfragen leiten aus Fakten und Regeln Antworten ab. Prolog interpretiert die Horn-klauseln prozedural: Eine Anfrage löst einen Aufruf eines Fakts bzw. des Regelkopfs aus.

(4) m = 0, n = 0

←

Diese **leere Klausel** erscheint im Prolog-Programm selbst nicht. Sie ist beweistechnisch wichtig. Der Ablauf des Programms besteht in dem Versuch, ein Faktum aus dem Programm (der Wissensbasis aus Fakten und Regeln) abzuleiten. Die Ableitung erfolgt nach einem fest vorgegebenen, in Prolog implementierten Schlußfolgerungsmechanismus. Herzuleitende Fakten

werden vom Benutzer in der Form von Implikationen ohne Konklusionsteil (Anfrage, goal) an den Inferenz-mechanismus übergehen, z.B.

?-struktur(p1,b2,X).

mit der Bedeutung: "Gibt es ein X, so daß struktur(p1,b2,X) aus dem Programm folgt". Der in Prolog eingebettete Schlußfolgerungsmechanismus (Resolution) ver-sucht herzuleiten, daß die Anfrage mit den Formeln des Programms im Wider-spruch steht, wenn für X bestimmte Werte eingesetzt werden.

Bsp.: Gegeben ist das folgende Prolog-Programm

```
kind_von(juergen,christian).              /* 1. Fakt */
kind_von(juergen,liesel).                 /* 2. Fakt */
mann(christian).                          /* 3. Fakt */
mann(juergen).                            /* 4. Fakt */
frau(liesel).                             /* 5. Fakt */
mutter_von(X,Y) :-                        /* 1. Regel */
 kind_von(Y,X), frau(X).
vater_von(X,Y) :-                         /* 2. Regel */
 kind_von(Y,X), mann(X).
```

An dieses Prolog-Programm wird die Anfrage

```
?-mutter_von(liesel,juergen).
```

gestellt.

Die Frage ist durch den Kopf der ersten Regel ersetzbar. Prolog durchsucht die Wissensbasis vom Anfang bis zum Ende nach einem passendem Fakt bzw. einem passenden Regelkopf. Die 1. Regel ist an die vorliegende Regel angepaßt, wenn "X durch liesel (X/liesel)", "Y durch juergen (Y/juergen)" ersetzt wird. Diesen Vorgang nennt man in Prolog **Unifikation**. **Unifikation** heißt: Prüfen, ob Literale zusammenpassen!

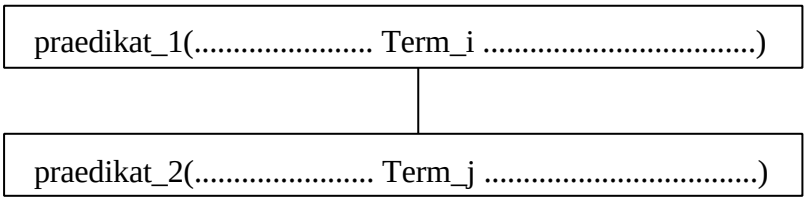


Abb. : Unifikation zweier Prädikate

Etwas vereinfacht besteht **Unifikation** aus 3 Prüfungen:

1. Passen die Prädikate (praedikat_1 = praedikat_2)
2. Stimmen die Anzahl der Terme überein
3. Passen Terme paarweise.

Wenn eine Anfrage beantwortet werden soll, wird in der Wissensbasis (Prolog-Programm) nach einem Faktum bzw. dem Kopf einer Regel gesucht, das bzw. der mit der Anfrage unifiziert (verschmilzt). Anfrage und Klauselkopf unifizieren, wenn die der Unifikation zugrundeliegenden

Prüfungen ergeben: Es ist möglich, Variablen so zu ersetzen, daß die beiden Ausdrücke gleich werden.
Die 1. Regel im vorliegenden Beispiel verweist im Regelrumpf auf weitere Teilziele, die erfüllt sein müssen. Es gilt zunächst das Teilziel "kind_von(juergen,liesel)" zu beweisen. Prolog durchsucht zu diesem Zweck wieder die Wissenbasis vom Anfang bis zum Ende und stößt dabei auf den 2. Fakt. Ein Fakt ist immer wahr, das Teilziel ist erfüllt. Es folgt die Realisierung des weiteren Teilziels "frau(liesel)". Prolog durchsucht die Wissensbasis vom Anfang bis zum Ende und findet eine Bestätigung des Teilziels im 5. Fakt. Die Anfrage wurde damit vollständig bewahrheitet. Der Prolog-Interpreter bestätigt dies durch die Antwort: YES.
Den geschilderten Ablauf kann mit Hilfe einer graphischen Darstellung (Beweisbaum) zusammenfassen. Zur Verdeutlichung des prädikatenlogischen Resolutionsbeweises, werden die Regeln in eine äquivalente Darstellung mit disjunktiv, verknüpften atomaren Formeln überführt. Es ergibt sich

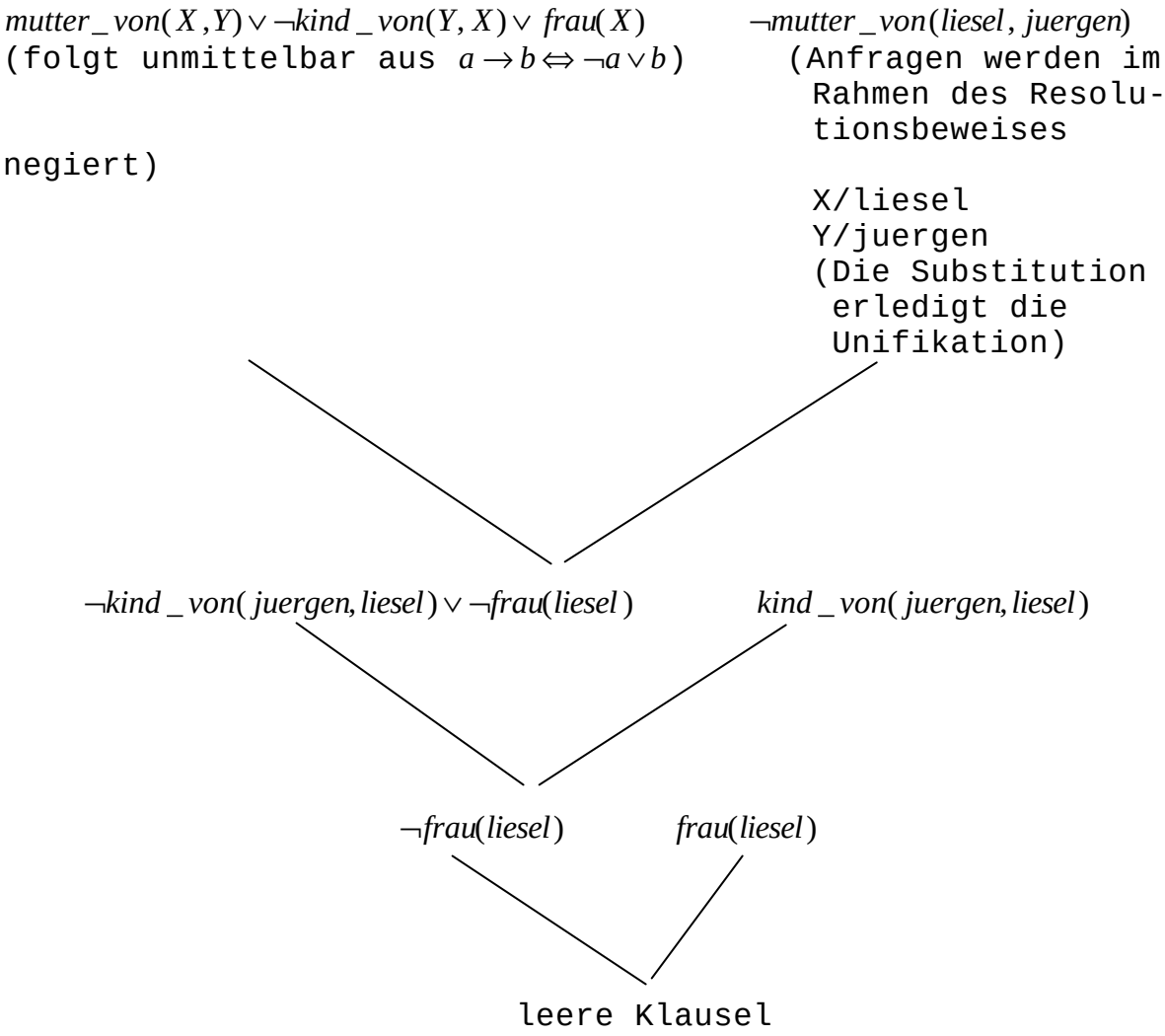


Abb. : Beweisbaum zur Zielvorgabe "?-mutter_von(liesel,juergen)."

Die leere Klausel ist immer falsch, die Anfrage somit wahr.

Der pädikatenlogische Resolutionsbeweis zu der folgenden Anfrage

?-vater_von(X,Y).

kann ebenfalls direkt im Beweisbaum gezeigt werden:

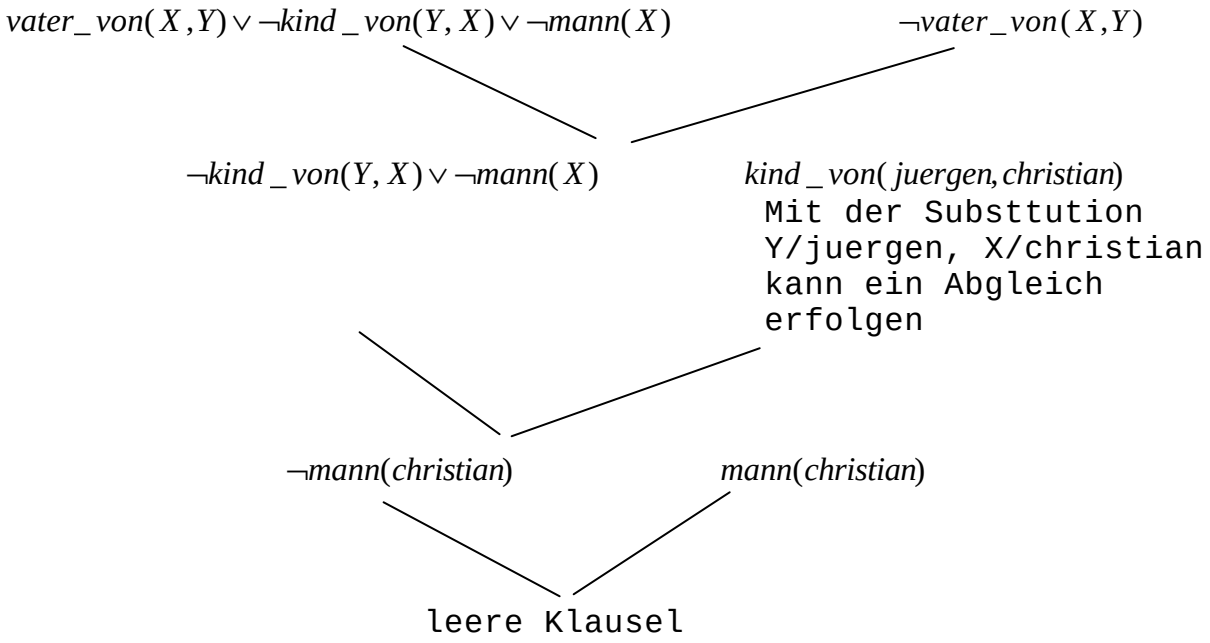


Abb. : Beweisbaum zur Zielvorgabe "?-vater_von(X,Y)."

Der Beweis konnte nur durch die Zuordnung Y/juergen bzw. X/christian gelingen. Man spricht von **Instanziierung der Variablen.**Prolog gibt an, durch welche Bin-dung der Variablen der Beweis gelungen ist:

```
X=christian
Y=juergen
```

Prolog durchsucht Fakten und Regeln immer von links nach rechts. Da links das Ergebnis und rechts die Konjunktion von Ursachen steht, liegt eine **rückwärtsver-kettende Kontrollstruktur** vor.

Aufgabe

Zu welchem Ergebnis führt die Anfrage

?-p(X).

an das folgende Prolog-Programm:

```
p(X) :- q(X).      /* 1. Regel */
q(X) :- p(X).      /* 2. Regel */
q(a).              /* Fakt */
```

Die Anfrage führt zu keinem Ergebnis, sondern zu einer Endlosschleife.

Begründung: Um p(X) zu beweisen, ist q(X) zu beweisen (/* 1. Regel */). q(X) könnte mit X=a bewiesen werden, zuerst wird hier immer die Regel q(X) ÷ p(X) aufgerufen, dh.: Um q(X) zu

beweisen ist wieder p(X) zu beweisen. Das Programm erreicht also wegen dieser Anordnung und dem vorgegebenen, in Prolog fest implementierten Ablauf (von oben nach unten, von links nach rechts) nie den Kandi-daten q(a) und gerät in eine Endlosschleife.

Nach welcher Änderung im vorstehenden Programm kann ein korrekter Ablauf er-zwungen werden?

```
p(X) :- q(X).
q(a).
q(X) :- p(X).
```

Fakten, Regeln, Anfragen bestehen aus Prädikaten, das sind logische Funktionen, die immer nur die Werte "wahr" und "falsch" annehmen können. In Prolog werden Prädikate so beschrieben:

```
praedikat(... Term ....).
```

Ein **Term** kann eine Konstante, eine Variable eine beliebig geschachtelte Struktur oder eine Liste sein.
Die **syntaktischen Elementarformen** von Prolog sind **Terme**. Jede Verknüpfung von Termen ergibt wiederum Terme. Diese rekursive Definition ermöglicht, daß Prolog den Term als einzige syntaktische Form kennt. Zusammengesetzte Terme heißen **Strukturen**. **Variable** beginnen mit einem Großbuchstaben. Ausgenommen davon ist die **anonyme Variable**. Sie ist durch das Symbol "_" gekennzeichnet und definiert eine Variable, deren Wert nicht von Belang ist. **Konstanten** sind Zahlen oder Zeichenketten, die mit Kleinbuchstaben beginnen oder in Anführungszeichen eingeschlossen sind.
Der **lexikalische Geltungsbereich** von Variablennamen ist eine Klausel. Der gleiche Name, der in 2 Klauseln auftritt, bezeichnet 2 verschiedene Variable. Jedes Auftreten einer Variable in derselben Klausel bezeichnet die gleiche Variable. Konstante dagegen verhalten sich anders: Die gleiche Konstante bezeichnet immer das gleiche Objekt in jeder beliebigen Klausel, d.h. im ganzen Programm.

Bsp.:

```
7, 4, fachhochschule,          sind Konstanten
Was, Wie, Kopf, Rest           sind Variable
```

1.2.4 Constraints

Constraints eignen sich besonders zur Darstellung von Randbedingungen, die die Lösung des Problems auf jeden Fall erfüllen muß. Ein Beispiel dafür ist der Wunsch eines Lehrers bei der Stundenplanerstellung, dass er einen bestimmten Tag in der Woche frei haben möchte. Durch jede Constraint wird der Lösungsraum zusätzlich eingeschränkt.
Ein Constraint kann häufig als mathematiscje Gleichung betrachtet werden, ein Constraint-System als Gleichungssystem. Unter Constraint-Programmierung versteht man dann die Berechnung des Gleichungssystems. Es lassen sich aber nich nur mathematische Gleichungen Beschreiben, sondern auch Ungleichungen und nicht numerische Zusammenhänge.
Es existieren unterschiedliche Möglichkeiten Constraints zu realisieren, z.B. lässt sich die Beziehung zwischen Körpüergröße und Idealgewicht folgendermaßen darstellen:

- Tabelle, z.B. mit Hilfe einer Gewichtstabelle
- Funktion, z.B. Normalgewicht = Körpergröße - 100
- Prädikate: Übergewicht = tatsächliches Gewicht/Normalgewicht > 1.2

Ein Propagierungsalgorithmus schränkt den Wertebereich einer Variablen solange ein, bis keine weiteren Einschränkungen mehr möglich sind.
Bsp.: "Hilferuf" eines Studenten an seinen Vater

```
  SEND
+ MORE
------
 MONEY
```

Es soll jedem Buchstaben eine Ziffer zugewiesen werden, so dass die Addition korrekt ist. Wieviel Geld benötigt der Student von seinem Vater.

C1: D + E = Y + 10 * U1
C2: N + R + U1 = E + 10 * U2
C3: E + O + U2 = N + 10 * U3
C4: S + M + U3 = O * 10 *U4
C5: M = U4

D, E, N, O, R, Y $\in \{0,1,2,3,4,5,6,7,8,9\}$
M, S $\in \{1,2,3,4,5,6,7,8,9\}$S
U1, U2, U3, U4 $\in \{0,1\}$

Aus C5 folgt: M und U4 stellen dieselbe Zahl dar und sind deshalb durch die Schnittmenge der beiden Wertebereiche mit M = 1 eindeutig bestimmt.
C4 reduziert sich auf: S + U3 = O + 9. Annahme: U3 = 1. C4 vereinfacht sich zu: S = O + 8. Daraus ergibt sich: $S \in \{8,9\}$ und O = 0. Damit vereinfacht sich C3 zu E + U2 = N + 10, was für keine gültige Wertebelegung erfüllt sein kann, da E < 10 sein muß. U3 = 1 war also falsch.
Neue Annahme: U3 = 0. C4 vereinfacht sich zu S = O + 9 und ist nur für S = 9 und O = 0 erfüllbar. Für C3 ergibt sich: E + U2 = N. Die Annahme U2 = 0 führt zum Widerspruch E = N , also muß U2 = 1 sein. Daraus folgt E + 1 = N. Man ersetzt in C2 N durch E + 1 (symbolisches Propagieren). Für C2 ergibt sich R + U1 = 9. 9 ist nicht mehr im Wertebereich der Variablen R, es folgt R = 8 und U1 = 1.
Für C1 gilt nun D + E = Y + 10. Die aktuelle Wertemengen sind $D, E, N, Y \in \{2,3,4,5,6,7\}$. Aus E + 1 = N folgt, daß für N der wert 2 und E der Wert 7 herausfällt.
Aus D + E = Y + 10 fogt die Ungleichung D + E > 11. Dadurch werden die Wertemengen weiter eingeschränkt $E \in \{5,6\}$ und $D \in \{6,7\}$.

Aus E = 5 folgt D = 7 und N = 7 (Widerspruch). E = 5 muß gelten. Damit ergeben sich: N =6, D = 7 und Y = 2 und insgesamt die folgende eindeutige Lösung des Problems: 9657 + 1085 = 10652.

1.2.5 Objektbezogene texttuelle Darstellung: Frames (Rahmen)

„**frames**“

Frames (Rahmen) liefern ein Organisationsschema für Wissensbasen. Der Zugriff auf das Wissen wird erleichtert, indem das Wissen in Bausteine (den sog. Frames) bereitgestellt wird. Ein Frame (Rahmen) bezeichnet das Zusammenwirken von Wissen, das für ein Objekt, Konzept oder in einer bestimmten Situation von Bedeutung ist. Eine Wissensbasis besteht generell aus vielen Rahmen, wobei jeder Rahmen wiederum mehrere Informationen enthält. Ein Rahmen kann im System ständig gehalten oder während eines Lösungsvorgangs beliebig oft erzeugt und gelöscht werden.
Ein Frame (Rahmen) enthält demnach das problem- und fallspezifische Wissen über ein Objekt. Objekte sind hier in eine Hierarchie eingebettet. **Instanzen** sind die Endknoten (Blätter) der Hierarchie, höhere Knoten in der Hierarchie (, die näher zu den allgemeinen, allen übergeordneten Objekttypen stehen) sind **Klassen**.
Eine **Klasse** kann als Schablone (Schema) für ein Objekt aufgefaßt werden. Sie definiert die Eigenschaften und das erhalten der zugehörigen Objekte. Die einzelnen Ausprägungen einer Klasse heißen Instanzen. Instanzen erhalten grundsätzlich die gleichen Eigenschaften und Verhaltensweisen (z.B. Methoden) wie die Klasse. Klassen, Instanzen, im allg. Objekte erhalten Rahmen zugeordnet.
Frames sind demzufolge „Erwartungsrahmen zur Aufnahme und Abspeicherung von Wissen“. Der „Wissensingenieur“ bzw. Benutzer soll die vorgegebenen, festen Rahmen nur noch mit dem problem- und fallspezifischen Wissensinhalten ausfüllen.
Ein **Frame** hat einen (Objekt-) Namen.

„**slots**“ und „**filler**“

Ein Rahmen besteht gewöhnlich aus zwei Teilen: einem Namen und einer Liste von Attribut-Wert-Paaren. Die Attribute werden auch „Slotnamen“ (slot names) und ihre Werte „filler“ genannt. Ein Eintrag in einem Rahmen kann verschieden interpretiert werden. In manchen Fällen können die entsprechenden Daten unmittelbar im Rahmen gespeichert werden, in anderen Fällen wird eine Prozedur angegeben, mit der Informationen ermittelt werden können.
Slotwerte können demnach umfassen:

- einfache Datenwerte oder Wertmengen
- Zeiger auf andere Frames, die eine Beziehung zwischen Frames beschreiben
- prozedurale Darstellung von Wissen

Auswertung und Interpretation der Daten steuert ein Facet[4]. Hier kann man zusätzliche Informationen über den Slotwert (Typspezifikation, Bedingungen, sog. Dämonen[5]) ablegen.

[4] Die Begriffe Facet, Slot, Frame sind über die deutsche Terminologie des Fachgebiets "Künstliche Intelligenz" eingeführt und werden deshalb hier einfach aus dem Englischen übernommen

So kann ein Frame-System bspw. folgende Facets [6] anbieten:

-- wert (value)
enthält unmittelbar den aktuellen Wert des Slots
-- bei_bedarf
gibt eine Prozedur an, die bei fehlendem Wert-Eintrag (value-Facet) aufgerufen wird und den aktuellen Wert liefert.
-- beim_hinzufuegen
enthält ein prozedurales Anhängsel, das bei der erstmaligen Instanziierung des Werts von einem Slot aufzurufen ist.
-- beim_entfernen
wird beim Entfernen einer Slot-Instanziierung aktiviert
-- beim_aendern
wird beim Ändern eines Slot-Werts aufgerufen
-- default
ermöglicht die Angabe einer Vorbesetzung für den Wert eines Slots. Sie wird gewählt, falls weder "wert-" noch "bei_bedarf"-Facet vorliegt.
-- anforderungen
kann Bedingungen für den Wert des Slots enthalten. Das können bspw. einfache Werte oder Wertintervalle sein.
-- vorziehen
enthält Angaben zu Werten oder Wertebereiche, die nicht obligatorisch, aber wünschenswert sind.

Der bedarfsweise Rückgriff auf höhere Rahmen wird als **Vererbung** bezeichnet. Welche Rahmen jeweils übergeordnet ist und zu welcher Klasse somit ein Objekt gehört, wird in einem **ako**-Slot verzeichnet. Alle Rahmen mit Ausnahme des allerhöchsten müssen einen **ako**-Slot besitzen. Der oberste Rahmen steht gewöhnlich für „allgemein, umfassendes“ Objekt.
Die Zugriffsstrategie für die Veerbung ist durch folgende Regeln bestimmt:

- Gibt es einen Slot für die gesuchten Daten im Rahmen des gerade behandelten Objekts, so gelten die in ihm enthaltenen Informationen
- Andernfalls wird der entsprechende Slot im direkt übergeordneten Rahmen, also in der Beschreibung der nächsthöheren Klasse gesucht und so von den Eltern auf die Kinder vererbt.
- und diese Veerbung wird gegebenenfalls bis zum allerhöchsten Objekt der Hierarchie fortgesetzt.

Slotanhängsel

Ein Slot kann mit einem Wert und / oder einer sich auf ihn beziehenden Information vorbesetzt werden. Da eine solche Information weder dem Wert (, mit dem der Slot gefüllt ist wurde,) noch dem Namen des Slot entspricht, wird sie „slot attachement (Anhängsel)“ genannt.
Anhängsel können sein:

1. Eine Bedingung (constraint), die vom in den Slot gefülten Wert erfüllt wird

---

[5] Dämonen (demons) sind Prozeduren, die automatisch aktiviert werden, falls eine (mit angegebene) Bedingung zutrifft

[6] vgl. Thuy, N.H.C / Schnupp, P.: "Wissensverarbeitung und Expertensysteme", München/Wien , 1989

2. Eine Prozedur, die bei Bedarf zur Bestimmung des Slotwerts eingegesetzt wird („if-needed“ prozedurales Anhängsel).
3. Eine Prozedur die ausgeführt wird, nachdem ein Slot mit einem Wert gefüllt wurde („if-added“ prozedurales Anhängsel). Dadurch wird erreicht, daß Wissen nicht nur passiv gespeichert wird. Es wird möglich, bei jedem Zugriff auf Slots Schlüsse zu ziehen und andere Frames zu modifizieren. Auf diesem Wege kann die Wissensbasis auf einfache Art konsistent gehalten werden.
4. Defaultwerte, die beim Lesezugriff zurückgegeben werden, wenn kein Wert explizit zugeordnet wird.

Schema

Ein Rahmen wird häufig an eine Klasse von Objekten oder Situationen gebunden, z.B.:

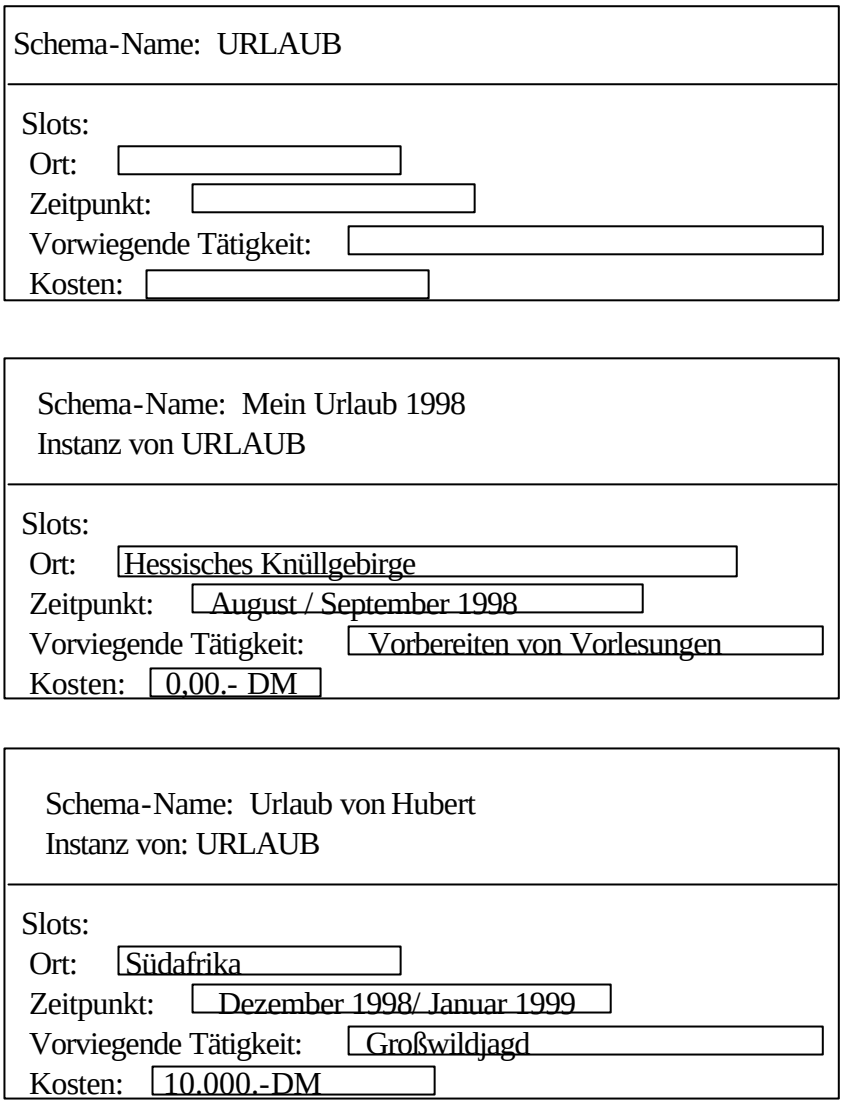


Abb.: Bezeihung zwischen Schema und zwei Instanzen

Ein Rahmen für einen bestimmten Urlaub wird durch Instanzierung des allgemeingültigen Rahmens erzeugt. Ein allgemeingültiger Rahmen wird Schema genannt. Die Instanzierung des Schema verläuft

folgendermaßen: Zunächst wird ein neuer Rahmen erzeugt, indem Speicherplatz für ihn angefordert wird. Danach wird er mit dem Schema verbunden, indem bspw. eine spezieller „Instanz-von“-Slot gefüllt wird.

Anwendungen

Frames sind (passive) Datenstrukturen. Die Manipulation dieser Datenstrukturen erfolgt durch Prozeduren, die nicht von einem „Frame“ ausgewählt und angestoßen wurden. Darin liegt auch der wesentliche Unterschied zum objektorientierten System. Dort sind Zustand und Verhalten gekapselt, und Operationen eine direkte Antwort auf eingegangene Botschaften. Wegen der großen Bedeutung von Frames in der "Künstlichen Intelligenz (KI)" gibt es zahlreiche Implementierungsversuche (FLAVORS, LOOPS, KEE, usw.), die auch objektorientierte Umgebungen besitzen.
Am weitesten fortgeschritten ist KEE[7]. Dieses System deckt ein weites Spektrum von KI-Problemlösungstechniken (u.a.a Frames, Regeln) ab. Das grundlegende Konzept von KEE ist das Framekonzept. Der Grundbaustein innerhalb der KEE-Umgebung ist die Einheit (unit)[8]. Auch eine KEE-Einheit besteht aus Slots, die die Attribute der Einheit speichert. Slots können umfassen: Daten, Datenstrukturen, Regeln und Methoden. Jeder Slot hat seine eigenen Attribute (Facets). Sie erhalten Werte (in KEE VALUES genannt). Ist der Wert eines Slots eine Lisp-Funktion[9], so wird diese Methode (METHOD) genannt. Methoden werden über Botschaften angestoßen. Eine KEE-Einheit kann Eigenschaften und Methoden höherer Ebenen bearbeiten, auch mehrfache Veerbung ist zulässig.

1.2.6 Intelligente Agenten

Für effektiven Informationsaustausch (z.B. im Internet) ist es in vielen Fällen nicht möglich, ein konventionelles Programm zu erstellen, das seine Aufgabenstellung in zufriedenstellender Qualität löst. Die Verwendung von intelligenten Agenten stellt einen Ansatz dar, mit deren Hilfe komplexe, verteilte Probleme bewältigt werden können. Zu diesem Zweck wird das Gesamtproblem in kleinere Teilprobleme zerlegt, die von evtl. verteilten, autonomen Programmen ausgeführt werden.

1.2.7 Unsicheres Wissen

Bisher wurde Wissen immer in Form von eindeutigen Fakten, Regeln, Bedingungen oder Abbildungen gespeichert. Häufig können aber Aussagen und Schlussfolgerungen nur mit einer bestimmten Unsicherheit getroffen werden. Quellen für die Unsicherheit können sein

- inhärente Unsicherheit der Information

[7] Knowledge Engineering Environment von Intellicorp Inc.
[8] vergleichbar mit dem Klassenkonzept
[9] KEE baut auf der Programmiersprache Lisp auf und wird nicht mehr kommerziell vertrieben.

- Unvollständigkeit der Information
- Unsicherheit der Schlußfolgerung
- Zusammenfassung von Informationen aus mehreren Quellen

Unsicherheiten können numerisch oder symbolisch dargestellt werden. bei der symbolischen Repräsentation wird der Grad der Unsicherheit durch Elemente aus einer vorgegebenen Menge von Symbolen ausgedrückt, z.B. dirch Begriffe wie „meisten“, „beinahe“, „immer“.
Bei der numerischen Repräsentation wird die Unsicherheit durch die Zuordnung von einem oder mehreren Zahlenwerten, z.B. der Wahrscheinlichkeit, ausgedrückt.

1.3 Wissensspeicher

Wissensspeicher stellen das zentrale Glied in der Verbindung zwischen dem Informationssender und dem Empfänger dar. Ihnen kommt die Aufgabe zu, jedem Benutzer, egal, ob Mensch oder Maschine, das gewünschte Wissen zur Verfügung zu stellen. Dabei haben Mensch und Maschine unterschiedliche Möglichkeiten und Anforderungen, was Suche und Darstellung des geforderten Wissens betrifft.

1.3.1 Wissensspeicher mit elektronischem Zugang

Beim Internet oder bei Datenbanken efolgt der Zugang elektronisch.

Relationale Datenbanken und objekt-relationale Datenbanken

Relationale Datenbanken eignen sich besonders gut zur Bearbeitung umfangreicher, regelmäßig strukturierter Informationsmengen. In relationalen Datenbanken werden bestimmte Beziehungen abgelegt, aus denen sich Zusammenhänge ableiten lassen. Datenbanksysteme stellen in der Regel zuverlässige Operationen zur Verfügung (Projektion, Selektion, Verbund). Diese lassen sich zu Abfragen und im begrenzten Umfang auch zu Inferenzen einsetzen. Zusammen mit leistungsfähigen Inferenzmethoden (z.B. der Resolution der Prädikatenlogik) ergibt sich ein intelligentes, schnell reagierendes wissensbasiertes System. Relationale Datenbanken unterstützen die logischen Verfahren zur Beschreibung von Wissen.
Wird Wissen in einem Graphen repräsentiert, dann entsprechen die Knoten des Graphen den Objekten des Problembereichs. Die Kanten des Graphen geben die Beziehungen zwischen den Objekten wieder. Eine persistente Ablage, die Struktur und Verhalten des Objekts enthält, kann in objekt-relationalen Datenbanken erfolgen. Bedeutende Hersteller von relationalen Datenbanken (z.B. Oracle) haben die rein relationale Datenstruktur auf objekt-relationale Architektur umgestellt. Es können aber weiterhin auch Tabellen für relationale Datenbanken erstellt werden, zusätzlich aber auch vollständige Spezifikationen von Objekttabellen, reine Indextabellen und komplexe Strukturen, in denen Attribute, Typen, verschachtelte Tabellen eingesetzt werden können.

1.3.2 Dokumente

Da früher Menschen die "einzigen Systeme" waren, die Wissen verarbeitet haben, sind bis heute die meisten Dokumente in einer für den Menschen lesbaren Form gespeichert. Durch das Aufkommen der maschinellen Wissensverarbeitung wird es notwendig, Anforderungen und Fähigkeiten des Rechners bei der Wissensspeicherung zu berücksichtigen.

2. Objektorientierte Systementwicklung

Bei der objektorientierten Systementwicklung werden die Entwicklungsphasen

- Analyse (objektorientierte Analyse, OOA)
- Entwurf (objektorientiertes Design, OOD)
- Realisierung, d.h. Programmierung (OOP)

unterschieden. Die **objektorientierte Analyse** legt die größte Gewichtung auf die Erzeugung von Modellen der realen Welt. In der Software-Entwicklung gibt es mehrere Ansätze für ein Modell. Die beiden häufigsten sind die algorithmische und die objektorientierte Perspektive.
Algorithmische Perspektive: Bei diesem Ansatz ist der Hauptbestandteil jeder Software die Prozedur oder Funktion. Diese Sicht veranlaßt die Entwickler, sich auf Fragen der Steuerung und der Zerlegung großer Algorithmen in kleine (stepwise refinement) zu konzentrieren.
Objektorientierte Perspektive: Bei diesem Ansatz ist der Hauptbestandteil jedes Softwaresystems das Objekt oder die Klasse. Ein Objekt ist im Rahmen dieses Ansatzes etwas, was im allgemeinen aus dem Vokabular des Problems- bzw. oder Lösungsbereichs stammt. Eine Klasse ist eine Menge von gleichartigen Objekten. Jedes Objekt besitzt eine Identität (d.h.: Man kann es benennen oder auf andere Weise von anderen Objekten unterscheiden), ein Zustand und ein Verhalten (d.h.: Man kann etwas mit ihm machen, und es kann seinerseits auch mit anderen Objekten etwas machen).
Objektorientiertes Design umfaßt den Prozeß der objektorientierten Zerlegung und einer Beschreibung der logischen und physikalischen sowie auch statischen und dynamischen Modelle des betrachteten Systems.

2.1 Entwurfsgrundlagen der objektorientierten Technologie

2.1.1 Das Objektmodell

Es wurde im wesentlichen durch objektorientierte Programmiersprachen beeinflußt. Objektorientierte Entwurfsmethoden sollen die Ausdrucksstärke objektorientierter Programmiersprachen (mit Klasse und Objekt als grunlegende Bausteine) ausnutzen. Die durch objektorientierte Programmiesprachen (z.B. Smalltalk, C++, **Java**) bestimmte objektorientierte Programmierung ist die Implementierungsmethode, bei der Programme als kooperative Ansammlung von Objekten angeordnet sind. Jedes dieser Objekte ist Instanz einer Klasse, alle Klassen sind Elemente in einer durch die Vererbungsbeziehungen gekennzeichneten Klassenhierarchie.
Die vier wesentlichen Elemente (Prinzipien) des Objektmodells sind:

- Abstraktion[10]
  Eine Abstraktion gibt die wesentlichen Charakteristika eines Objekts an, die es vor allen anderen Objekten unterscheiden (, wobei klar definierte, konzeptuelle Grenzen gesetzt werden, und zwar unter Bezugnahme auf die Perspektive des Betrachters). Das zentrale Problem beim objektorientierten Design ist die Entscheidung für das richtige Maß an Abstraktion im richtigen

[10] eine der fundamentalen Methoden zur Bewältigung von Komplexität

Bereich. Die Abstraktion eines Objekts soll den Entscheidungen bzgl. einer Implementierung vorausgehen.

- Kapselung[11]
  Die Kapselung ist der Prozeß der Aufgliederung der Elemente einer Abstraktion, die die Struktur und das Verhalten repräsentieren. Die Abstraktion konzentriert sich auf das beobachtbare Verhalten eines Objekts, die Kapselung konzentriert sich auf die Implementierung, die dieses Verhalten schließlich realisiert. So sind nach der Implementierung lokale Daten des Objekts nach außen nicht sichtbar und können nur durch das Objekt selbst (durch Ausführung von Funktionen) behandelt werden. Das entspricht den Prinzipien des „Information Hiding“ und der Modularisierung.
- Modularität
  Modularität ist die Eigenschaft eines Systems, das in eine Menge von in sich geschlossenen und lose gekoppelten Moduln zerlegt wird.
  In objektorientierten Programmiersprachen bilden Klasse und Objekt die logische Struktur eines Systems, die zur physikalische Realisierung des Systems in Modulen angeordnet sind.
- Hierarchie
  ist eine Rangfolge oder Anordnung von Abstraktionen.
  Die beiden wichtigsten Hierarchien in einem komplexen System sind seine Klassenstruktur („is a“-Hierarchie) und seine Objektstruktur („part of“-Hierarchie). Vererbung ist die wichtigste Konsequenz der „is_a“-Hierarchie. Grundsätzlich definiert die Vererbung eine Beziehung zwischen Klassen, wobei eine Klasse (die Unterklasse, Subklasse) Struktur oder Verhalten, das in einer bzw. mehreren Oberklassen (für einfache bzw. mehrfache Vererbung) definiert ist, übernimmt.

Weiter nützliche Elemente des Objektmodells sind:

- Typisierung
  Typisierung erzwingt die Objektklasse, so daß Objekte verschiedener Typen nicht oder zumindest nur unter sehr eingeschränkten Bedingungen ausgetauscht werden können.
- Nebenläufigkeit
  Nebenläufigkeit ist das Attribut, das ein aktives Attribut von einem nicht aktiven Objekt unterscheidet.
- Persistenz
  Persistenz ist die Eigenschaft eines Objekts, durch die seine Existenz eine bestimmte Zeit überdauern kann (d.h. das Objekt existiert weiter, nachdem sein Erzeuger schon nicht mehr existiert).

2.1.2 Klassen und Objekte

Was ist ein Objekt?

[11] Geheimnisse einer Abstraktion

Ein Objekt hat einen Status, ein Verhalten und eine Identität[12]. Die Struktur und das Verhalten ähnlicher Objekte sind in ihrer gemeinsamen Klasse definiert. Die Begriffe Instanz und Objekt sind austauschbar.
Der **Status** eines Objekts umfaßt alle (normalerweise statischen) Eigenschaften des Objekts, zusammen mit den aktuellen (normalerweise dynamischen) Werten dieser Eigenschaften.
Eine **Eigenschaft** ist eine unverwechselbare Charakteristik, eine Qualität oder ein Merkmal, das zur Einzigartigkeit des Objekts beiträgt. Eigenschaften sind normalerweise statisch und haben einen Wert. Dieser Wert kann eine einfache Zahl oder eine anderes Objekt einbeziehen. Objekte werden verwendet und verwenden selbst wieder Objekte.
Das **Verhalten** eines Objekts ist die Art und Weise, wie ein Objekt agiert und reagiert (in Form von Statusänderungen und der Übergabe von Nachrichten). Eine Operation / Nachricht / Methode[13] ist die Aktion, die ein Objekt auf einem anderen Objekt ausführt, um eine Reaktion zu erhalten. In rein objektorientierten Sprachen, z.B. Smalltalk, spricht man davon, daß ein Objekt einem anderen Objekt eine Nachricht übergibt. In C+++ spricht man davon, daß ein Objekt die Elementfunktion (Methode) eines anderen Objekts aufruft.
Eine **Operation** ist ein Service, das eine Klasse ihren Klienten bietet. Normalerweise führt ein Klient etwa fünf Arten von Operationen für ein Objekt aus:

- Modifizierer: Ein Operator, der den Status eines Objekts verändert
- Selektor: Eine Operation, die auf ein Objekt ohne Status-Änderung zugreift.
- Iterator: Eine Operation, die auf Teile des Objekts in wohl definierter Reihenfolge zugreift
- Konstruktor: Eine Operation, die ein Objekt erzeugt und / oder seinen Status initialisiert
- Destruktor: Eine Operation, die den Status eines Objekts freigibt und/oder das Objekt zerstört.

Objekte tragen zum Verhalten des Systems bei, indem sie zusammenarbeiten, d.h. in Beziehung zueinander treten. Die **Beziehung** zwischen zwei beliebigen Objekten beinhaltet die Annahmen, die einzelne Objekte übereinander anstellen. Das betrifft Operationen, die durchgeführt werden können, und auch das resultierende Verhalten. Zwei Beziehungsarten sind von besonderem Interesse:

- Links
- Aggregation (Eltern/Kind-Beziehungen)

Ein Objekt arbeitet mit anderen Objekten mit Hilfe von Links zusammen. Ein **Link** gibt die spezifische Verbindung an, über die ein Objekt die Dienste eines anderen Objekts ausführt, oder über die ein Objekt zu einem anderen gelangen kann.

[12] Identität ist die Eigenschaft des Objekts, die es von allen anderen Objekten unterscheidet.
[13] Die Begriffe Operation, Nachricht, Methode sind austauschbar.

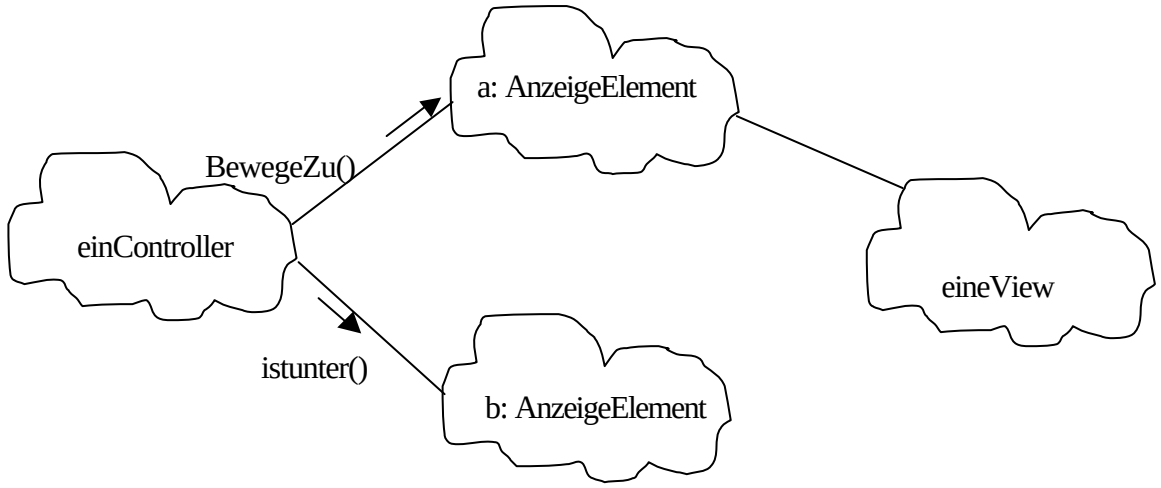


Abb.: Links

Die vorstehende Darstellung zeigt verschiedene Links (Linien zwischen den Object-Icons). Entlang der Links werden Nachrichten (Pfeile, die die Richtung der Nachricht angeben und eine Beschriftung mit dem Namen der Nachricht tragen. Das Objekt „einController“ besitzt Links zu zwei Instanzen von „AnzeigeElement“.
Das Senden von Nachrichten zwischen zwei Objekten ist normalerweise einseitig gerichtet, in manchen Fällen kann es auch zweiseitig gerichtet sein. Als Bestandteil eines Link kann ein Objekt eine der folgenden Rollen übernehmen:

- Aktor: Ein Objekt, das Operationen auf anderen Objekten ausführen kann, auf dem aber andere Objekte keine Operationen durchführen können
- Server: Ein Objekt, das niemals Operationen auf anderen Objekten durchführt. Es wird immer von anderen Objekten verwendet. Im infornationstechnischen Sprachgebrauch heißen Dinge (Objekte, Rechner), die eine Dienstleistung erbringen, Server. Die Dienstleistung wird erbrachr für einen Client (Klient, Kunde), der selbst Rechner oder Objekt sein kann.
- Agent: Ein Objekt, das auf anderen Objekten Operationen ausführen kann und mit dem andere Objekte Operationen ausführen können. Ein Agent wird normalerweise erzeugt, um Aufgaben für einen Aktor oder einen anderen Agenten zu übernehmen.

Damit Objekt A (Client) an Objekt B, die durch einen Link verbunden sind, eine Nachricht senden kann, muß B auf irgendeine Art und Weise für A sichtbar sein. Es gibt vier Möglichkeiten, wie ein Objekt für ein anderes Sichtbar gemacht werden kann:

- Das „Supplier“-Objekt ist global für den Client
- Das „Supplier“-Objekt ist ein Parameter einer Operation eines Client
- Das „Supplier“-Objekt ist Teil des Client-Objekts
- Das „Supplier“-Objekt ist ein lokal deklariertes Objekt in einer Operation des Client

Falls ein Objekt über einen Link eine Nachricht an ein anderes Objek sendet, heißen die beiden Objekte synchronisiert. Für Objekte in einer sequentiellen Anwendung wird die Synchronisation normalerweise durch einen Methodenaufruf erreicht. Gibt es jedoch mehrere Steuerungsprozesse, ist eine komplexe Nachrichtenübermittlung für die Objekte erforderlich zur Überwindung der Probleme mit gegenseitig ausschließenden Operationen, die in konkurrierenden Systemen auftreten können.
Im Gegensatz zu Links (gleichberechtigte Beziehungen) gibt die **Aggregation** eine Hierarchie an, die vom Ganzen bis zu den Teilen (Attribute) führt.

Was ist eine Klasse?

Eine **Klasse** ist eine Menge von Objekten, die eine gemeinsame Struktur und ein gemeinsames Verhalten aufweisen.
Die Schnittstelle einer Klasse besteht vorwiegend aus Deklarationen aller Operationen, die auf Instanzen dieser Klasse angewendet werden können. Sie kann aber auch die Deklaration anderer Klassen, Konstanten, Variablen und Ausnahmen beinhalten.
Die Implementierung einer Klasse besteht vorwiegend aus der Implementierung aller Operationen, die in der Schnittstelle der Klasse definiert sind.
Klassen können, ähnlich wie Objekte, nie isoliert existieren. Es gibt drei grundlegende Arten von Beziehungen: Verallgemeinerung / Spezialisierung („is_a"-Beziehung), Ganzes / Bestandteil („Part of"-Beziehung), Assoziation. In Programmiersprachen haben sich mehrere gemeinsame Ansätze herausgebildet, insbesondere gibt es direkte Unterstützung für die Kombination folgenden Beziehungen:

- Assoziation
  drückt allgemein aus, daß es eine Beziehung zwischen zwei Klassen gibt, z.B.:

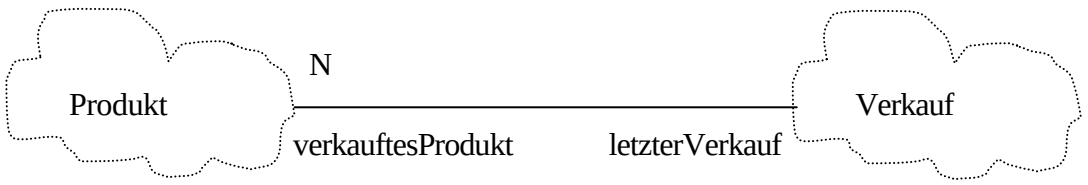


Abb.: Assoziation

Die Bedeutung der Beziehung zwischen den Klassen kann lediglich durch Benennen der Rolle hervorgehoben werden, die einzelne Klassen in ihrer Beziehung zu anderen haben. Das vorliegende Bsp. zeigt eine `1:N` Assoziation. Für jede Instanz der Klase Verkauf können keine Instanz oder mehrere Instanzen der Klasse Produkt vorhanden sein. Für jedes Produkt gibt es genau eine Instanz von Verkauf. In der Praxis unterscheidet man drei Arten von Kardinalität für Assoziationen: `1:1`, `1:N`, `N:N`.

- Vererbung

Die Vererbung ist eine Beziehung zwischen Klassen, wobei eine Klasse Struktur oder Verhalten teilt, das in einer Klasse (Einfachvererbung) oder mehreren anderen Klassen (Mehrfachvererbung) definiert wurde. Unterklassen erben Verhalten und Struktur ihrer Oberklassen

- Aggregation

Aggregations-Beziehungen zwischen Klassen haben eine direkte Parallele zu Aggregations-Beziehungen zwischen Objekten, die diesen Klassen entsprechen. Die Aggregation beschreibt „Ganzes/Bestandteil"-Beziehungen. Falls eine „Ganzes/Bestandteil"-Beziehung zwischen den Objekten besteht, dann muß es eine Aggregations-Beziehung zwischen den Klassen geben.

- Verwendung

„Verwendungs“-Beziehungen zwischen Klassen entsprechen gleichrangigen Links zwischen den entsprechenden Instanzen dieser Klasse.

- Instanzierung

- Generische Klassen

Eine parametrisierte Klasse (auch generische Klasse genannt) ist eine Klasse, die als Schablone für andere Klassen dient. Eine parametrisierte Klasse muß instanziiert werden (d.h. Parameter müssen mit Werten versehen werden), bevor Objekte dieser Klasse erzeugt werden können.

- Metaklasse

Eine Metaklasse ist eine Klasse, deren Instanzen wiederum Klassen sind.

2.1.3 Die Sprache zum Modellieren objektorientierter Systeme

Visualisierung und Spezizierung objektorientierter Softwaresysteme erfolgt mit der Unified Modelling Language (UML). Die UML ist eine Sprache zur Modellierung, deren Vokabular und Regeln sich auf die konzeptionelle und physische Darstellung des Systems konzentrieren.
Die UML ist eine Beschreibungssprache von Softwaresystemen, keine Sprache zur Beschreibung von Software-Entwicklungsprozessen. Daür gibt es verschiedene Modelle, die sich in der Praxis mehr oder weniger gut bewährt haben, z.B.:

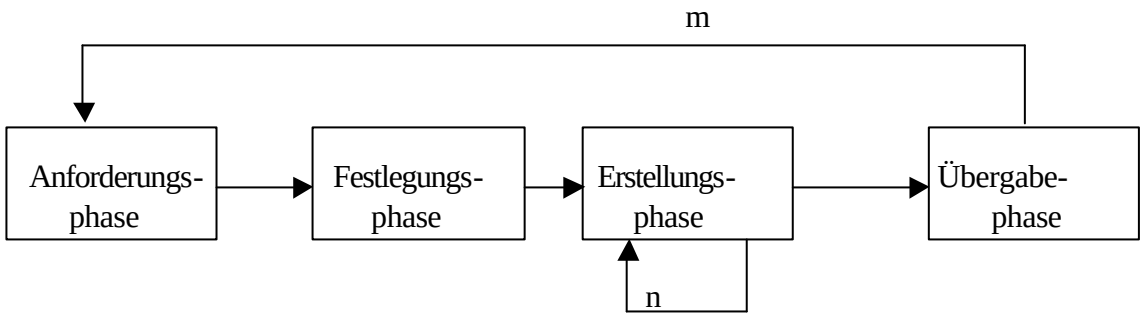


Abb.: Einfacher Software-Entwicklungsprozeß

Insgesamt gesehen verläuft ein Entwicklungsprozeß in mehreren Iterationen, er wird auch als spiralförmig[14] oder iterativ bezeichnet.
Die Entwicklung eines Prozeßmodells, das auf alle realen Prozesse paßt, ist unmöglich. Deshalb wurde der Prozeß und seine Darstellung aus der UML ausgeklammert. Jedoch ist eine Softwareentwicklung ohne Einbettung in den Software-Entwicklungsprozeß unprofessionell.
In der UML wird deshalb versucht ein Rahmenwerk für Prozesse zur Verfügung zu stellen. Rahmenwerk bedeutet: Alle Vorgehensweisen und Notationen von konkreten Software-

[14] Beim spiralförmigen Entwicklungsprozeß werden alle Phasen mehrfach durchlaufen (in der Abb. Pfeil m). Dabei wird jedem Durchlauf eine Teilmenge der Anwender-Anforderungen zugrunde gelegt. Die Spirale wird solange durchlaufen, bis das Endprodukt vorhanden ist. Alle Zwis chenprodukte, die nach einem Durchlauf entstehen, sind voll funktionsfähig.

Entwicklungsprozessen sind mit der UML-Notation darstellbar. Die Grundlage für dieses Prozeß-Rahmenwerk bildet die Beschreibung der OOSE-Methode[15] von Jacobson und der darin beschriebene Prozeß Objectory. Das Prozeßrahmenwerk der UML wird auch Objectory-Prozeß genannt, in Anlehnung an den OOSE-Prozeß.

2.2 Unified Modelling Language (UML)

2.2.1 Übersicht

Unified Modelling Language (UML)

Die UML ist ein Satz von Notationen zur Beschreibung objektorientierter Softwaresysteme. Ihre Autoren sind

Grady Booch: „Object oriented design and applications“
James Rumbaugh: Co-Autor der Object Modelling Technique
Ivar Jacobsen: Use-Cases

Aufgaben und Ziele der UML

Die UML soll nach den Wünschen der Autoren eine Reihe von Aufgaben und Ziele verfolgen:

- Bereitstellung einer universellen Beschreibungssprache für alle Arten objektorientierter Softwaresysteme.
- Vereinigung der bekanntesten Beschreibungsnotationen
- Setzen des Schwerpunkts auf die Produkte des Software-Engineerings und nicht auf den Prozeß

Hintergrund

Zur Einordnung der UML ist eine Betrachtung des Entstehungshintergrunds und der beteiligten Personen interessant:

- die OMT stellte zu Anfang einen Großteil der grundlegenden Notationen
- die UML wird der OMG als Antwort auf einen Aufruf zur Erstellung einer standardisierten objektorientierten Entwurfsmethode vorgelegt.
- Einflüsse von Ada und C++ sind unverkennbar

Richtlinien bei der Entwicklung der UML

Bei der Entwicklung der (hauptsächlich graphischen) Notation sollten u.a. folgende Prinzipien erfüllt werden:

[15] Object Oriented Software-Engineering (OOSE)

- Einfachheit: Verwendung weniger Konzepte und Symbole
- Priorisierung: Einfache Modellierung häufiger Probleme, mehr Aufwand bei ungewöhnlichen Situationen
- Konsistenz: Verwendung gleicher Konzepte in allen Bereichen der Notation
- Orthogonalität: Verwendung unterschiedlicher Notation für unterschiedliche Konzepte
- Schichtenarchitektur: Erweiterbarkeit der Notation um fortgeschrittene Konzepte
- Stabilität: Adaption bekannter Konzepte und Notationen- Druck- und Zeichenbarkeit: Eignung sowohl für manuelle als auch werkzeuggestützte Erstellung

Das Metamodell

Die UML wird formal in einem Metamodell beschrieben. Dabei werden die Elemente der UML selbst benutzt, um das Metamodell zu beschreiben. Ziele sind:

- Beschreibung der Semantik der UML
- Bereitstellung einer formalen Basis
- Unterstützung bei der Erstellung von Tools
- Definition eines Austauschformats für Modelle

Modellierung

Die UML ist eine grafische Notation für diverse objektorientierte Modelle:

- Analyse-Modell (Use-Case-Diagramme)
- Statisches Modell (Klassen und ihre Beziehungen)
- Dynamisches Modell (Sequenzdiagramme, Kolloborations-Diagramme, Zustandsdiagramme, Aktivitäts-diagramme

Diagrammtypen

Die Notation der UML umfaßt Diagramme für die Darstellung verschiedener Ansichten auf das System. Die Diagramme sind vergleichbar mit Bauplänen.

Diagramm	Einsatzgebiet	Phase
Use-Case	Geschäftsprozesse, allgemeine Einsatz-möglichkeiten	Anforderung, Festlegung, Erstellung, Übergabe
Klassendiagramm	So gut wie überall, das Klassendiagramm ist das wichtigste Diagramm der UML	Festlegung, Erstellung
Interaktionsdiagramm	Zeigt den Nachrichtenfluß und damit die Zusammenarbeit der Objekte im zeitlichen Ablauf	Anforderung, Festlegung, Erstellung, Übergabe
- Sequenzdiagramm	Zeitliche Aufrufstruktur mit wenigen Klasse	
- Kollaborationsdiagramm	Zeitliche Aufrufstruktur mit wenigen Nachrichten	
Package-Diagramm	Groborientierung, in welchem Modul welche Klasse zu finden ist. Aufteilung in Unterprojekte, Bibliotheken, Übersetzungseinheiten.	Erstellung

Zustandsdiagramm	Darstellung des dynamischen Verhaltens	Anforderung, Festlegung, Erstellung, Übergabe
Aktivitätsdiagramm	Bei parallelen Prozessen und anderer Parallelität, Geschäftsprozesse	Festlegung, Erstellung,
Implementierungsdiagramm	Besonders für die Darstellung von verteilten Anwendungen und Komponenten; allgemein Darstellung von Implementierungs-einheiten (Übersetzungseinheiten, ausführbare Programme, Hardwarestruktur)	Anforderung, Festlegung, Erstellung, Übergabe
- Komponentendiagramm	Zusammenhänge der Software	
- Deployment-Diagramm	Hardwareaufbau	

Abb.: Einsatzgebiete und Eigenschaften der verschiedenen UML-Diagramme

Jedes der Diagramme besitzt zahlreiche Stilelemente mit verschiedenen Beschriftungsarten[16].

Übergeordnete Konzepte

Als globale Konzepte sind in der UML verfügbar:

- Packages
  Packages erlauben es, als allgemeines Notationselement alle Arten von Diagrammen und Modellen zusammenzufassen. Zwischen Packages können Abhängigkeiten bestehen, so daß bspw. eine Package auf eine andere zugreift. Außerdem lassen sich Packages schachteln.
  Der Name einer Package kann zur Unterscheidung der Herkunft einer Klasse oder eines anderen Elements dienen.
- Stereotypes
  Sie bieten einen Mechanismus zur Kategorisierung jeglicher Elemente in der UML.
  Beispiele sind z.B. Interface, Handler, Exception u.a., welche oft während der Modellierung als Begriffe eingesetzt werden.
  Stereotypen werden gewöhnlich als Text zwischen französischen Anführungszeichen dargestellt, sie können aber auch durch die Definition eines Sinnbilds für den Stereotypen definiert werden.
  Zur Vereinfachung definiert die UML einen Satz von Kern-Stereotypes, die die gängigen Fälle abdecken.

2.2.2 Anwendungsfall und Anwendungsfalldiagramme

Anwendungsfall

Ein Anwendungsfall (use case) ist eine typische Interaktion (ein typischer Arbeitsablauf) zwischen einem Benutzer und einem Computersystem. Zusammenhänge zwischen Anwendungsfällen können in einem Anwendungsfalldiagramm dargestellt werden. Ein Anwendungsfall wird grafisch durch eine Ellipse dargestellt, die den Namen des Anwendungsfalls trägt. Zu jeder Ellipse existiert ein Text (z.B. stichwortartige Beschreibungen), der den Anwendungsfall genauer beschreibt.

- Akteure

[16] Vgl.: Günther Wahl: UML kompakt, OBJEKTspektrum 2/1998

Am Anwednugsfall beteiligte Akteure (Rollen)
- Vorbedingungen
  Zustand des Systems, bevor der Anwendungsfall eintrat
- Nachbedingungen
  Zustand des Systems, nachdem der Anwendungsfall erfolgreich durchlaufen wurde
- Invarianten
  Bedingungen, die im Rahmen des Anwendungsfalles stets erfüllt sein müssen
- Nicht-funktionale Anforderungen
  Zusicherungen, die für Design und Realisierung wichtig sind.
- Ablaufbeschreibung
  Beschreibung des Anwendungsfalles, ggf. gegliedert in numerierte Einzelpunkte.
- Ausnahmen, Fehlersituationen
  Beschreibung der Ausnahme- und Fehlersituationen, die im Rahmen des anwendungsfalles auftreten können.
- Variationen
  Abweichungen und Ausnahmen zum Normalablauf und Beschreibung der alternativen Abläufe für diese Fälle
- Regeln
  Geschäftsregeln, fachliche Abhängigkeiten, etc. die im Rahmen des Ablaufs von Bedeutung sind.

Abb.: Beispielstruktur für einen Anwendungsfall

Akteure

Ein Akteur ist eine außerhalb des Systems liegende Klasse, die an der in einem Anwendungsfall beschriebenen Interaktion mit dem System beteiligt ist. Akteure sind bspw. die Anwender des Systems. Wichtig sind vor allem die Rollen, die die Anwender im Zusammenhang mit dem Anwendungsfall einnehmen. Akteure beschreiben die Rollen der am Anwendungsfall Beteiligten. Diese Rollen können generalisiert und spezialisiert werden. Ein einzelner Akteur kann viele Anwendungsfälle ausführen, ein Anwendungsfall kann über verschiedene Akteure ausgeführt werden.

Anwendungsfalldiagramme

Es zeigt die Beziehungen zwischen Akteuren und Anwendungsfällen.

Bsp.: Anwendungsfalldiagramm für den Geschäftsprozeß „Zeitschriftenumlauf in einem Unternehmen“

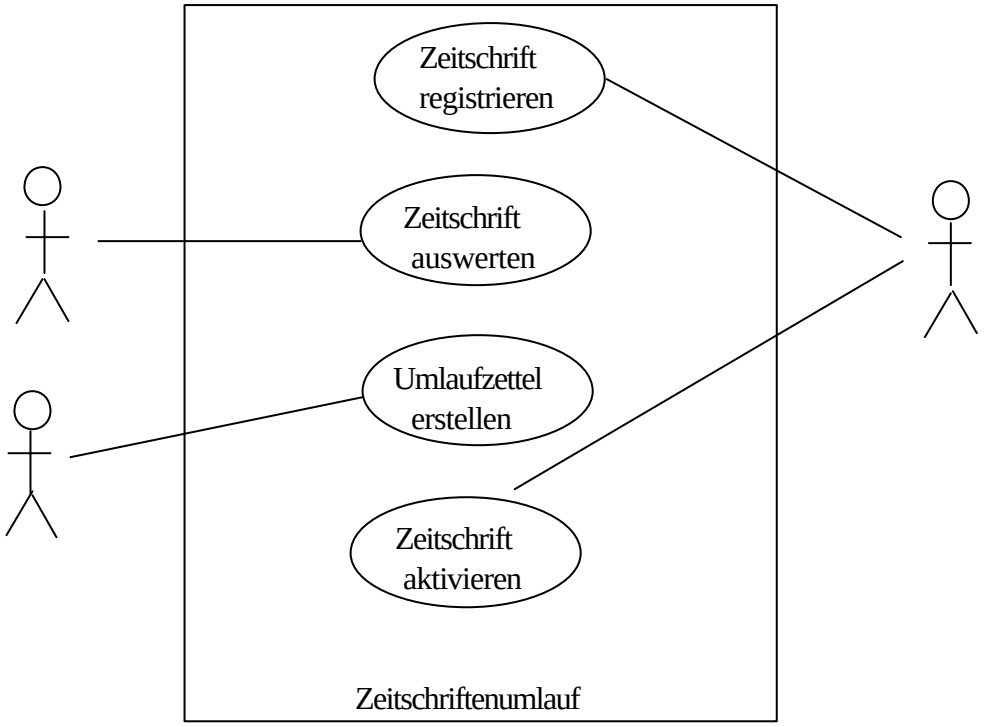


Abb.: Anwendungsfalldiagramm zum Geschäftsprozeß „Zeitschriftenumlauf“

Zu dem vorliegenden Anwendungsfalldiagramm gehören die Anwendungsfälle:

Makierte Artikel registrieren

Ablauf:
1. Zeitschriftenexemplar auswählen
2. Titel, Autor(en), Schlagworte und Kurzbezeichnung des zu registrierenden Artikels eingeben.
3. Speichern

Anmerkungen:
„Zeitschriftenexemplar“: Gemeint sind hier der Name der Zeitschrift und die Angabe der Heft-Nr. oder des Erscheinungsdatums

Umlaufzettel erstellen

1. Aus der Liste der abonierten Zeitschriften eine auswählen
2. Schaltfläche „Drucken“ betätigen.

Die Verbindungstypen „Benutzt (**uses**)“ und „Erweitert (**extends**)“

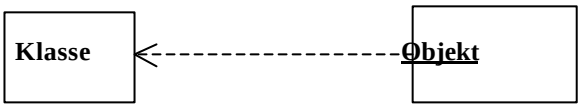
Man benutzt die <<**Erweitert**>>-Beziehung) (extends relationship) bei Anwendungsfällen, die einem anderen Anwendungsfall ähnlich sind, aber noch ein wenig mehr bewerkstelligen. Gezeigt werden spezielle Fehler- und Ausnahmesituationen, spezielle Abweichungen oder Erweiterungen des Standardfalls. Der Pfeil zeigt von der Variante zum Standardanwendungsfall.
Die <<**Benutzt**>>-Beziehung (uses-Relationship) wird verwendet, wenn ein Verhaltensanteil in verschiedenen Anwendungsfällen gleich ist. Mit der <<uses>>-Beziehung wird das gleiche Stück Anwendungsfallbeschreibung, das in verschiedenen Anwendungsfällen vorkommt, in eine andere Anwendungsfallbeschreibung eingebunden. Ein Pfeil zeigt auf den mitbenutzten Anwendungsfall.

2.2.3 Klassendiagramme

Elemente und Darstellung des Klassendiagramms

Das Klassendiagramm beschreibt die statische Struktur der Objekte in einem System sowie ihre Beziehungen untereinander. Die **Klasse** ist das zentrale Element. Klassen werden durch Rechtecke dargestellt, die entweder den Namen der Klasse tragen oder zusätzlich auch Attribute und Operationen. Klassenname, Attribute, Operationen (Methoden) sind jeweils durch eine horizontale Linie getrennt. Klassennamen beginnen mit Großbuchstaben und sind Substantive im Singular.
Ein strenge visuelle Unterscheidung zwischen Klassen und Objekten entfällt in der UML. Objekte werden von den Klassen dadurch unterschieden, daß ihre Bezeichnung unterstrichen ist. Haufig wird auch dem Bezeichner eines Objekts ein Doppelpunkt vorangestellt.. Auch können Klassen und Objekte zusammen im Klassendiagramm auftreten.

Wenn man die Objekt-Klassen-Beziehung (Exemplarbeziehung, Instanzbeziehung) darstellen möchte, wird zwischen einem Objekt und seiner Klasse ein gestrichelter Pfeil in Richtung Klasse gezeichnet:



Defintion einer Klasse

Die Definition einer Klasse umfaßt die „bedeutsamen“ Eigenschaften. Das sind:

- Attribute
  d.h.: die Struktur der Objekte: ihre Bestandteile und die in ihnen enthaltenen Informationen und Daten.. Abhängig von der Detaillierung im Diagramm kann die Notation für ein Attribut den Attributnamen, den Typ und den voreingestellten Wert zeigen:
  Sichtbarkeit Name: Typ = voreingestellter Wert
- Operationen
  d.h.: das Verhalten der Objekte. Manchmal wird auch von Services oder Methoden gesprochen. Das Verhalten eines Objekts wird beschrieben durch die möglichen Nachrichten, die es verstehen kann. Zu jeder Nachricht benötigt das Objekt entsprechende Operationen. Die UML-Syntax für Operationen ist:

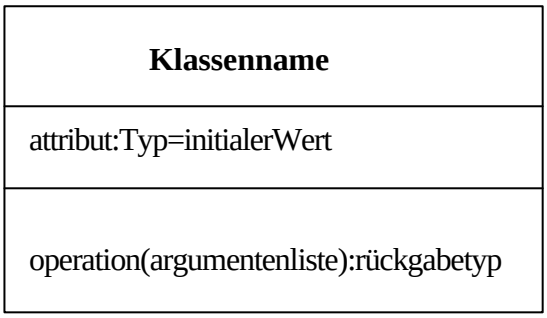
  Sichtbarkeit Name (Parameterliste) : Rückgabetypausdruck (Eigenschaften)

  Sichtbarkeit ist + (öffentlich), # (geschützt) oder – (privat)
  Name ist eine Zeichenkette
  Parameterliste enthält optional Argumente, deren Syntax dieselbe wie für Attribute ist
  Rückgabetypausdruck ist eine optionale, sprachabhängige Spezifikation
  Eigenschaften zeigt Eigenschaftswerte (über String) an, die für die Operation Anwendung finden
- Zusicherungen

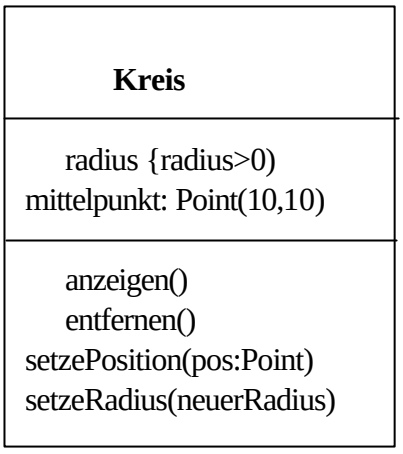
Die Bedingungen, Voraussetzungen und Regeln, die die Objekte erfüllen müssen, werden Zusicherungen genannt. UML definiert keine strikte Syntax für die Beschreibung von Bedingungen. Sie müssen nur in geschweifte Klammern ({}) gesetzt werden.
Idealerweise sollten Regeln als Zusicherungen (engl. assertions) in der Programmiersprache implementiert werden können.

Bsp.: Eigenschaften, Operationen, Zusicherungen eines Kreises, der auf einem Bildschirm dargestellt werden soll.
- Attribute: Radius des Kreises, Position des Kreises auf dem Bildschirm
- Operationen: Anzeigen und Entfernen des Kreises, Verschieben des Kreises, Verändern des Radius des Kreises
- Zusicherungen: Der Radius darf nicht negativ sein und nicht Null sein (radius>0)

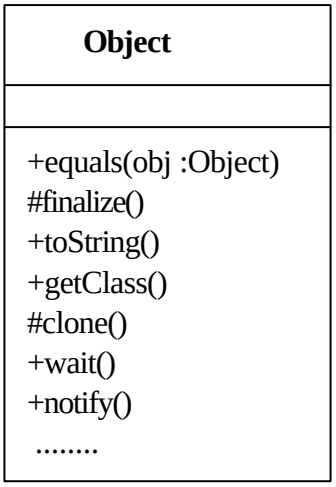
Attribute werden mindestens mit ihrem Namen aufgeführt und können zusätzliche Angaben zu ihrem Typ (d.h. ihrer Klasse), einen Initialwert und evtl. Eigenschaftswerte und Zusicherungen enthalten. Attribute bilden den Datenbestand einer Klasse.
Operationen (Methoden) werden mindestens mit ihrem Namen, zusätzlich durch ihre möglichen Parameter, deren Klasse und Initialwerte sowie evtl. Eigenschaftswerte und Zusicherungen notiert. Methoden sind die aus anderen Sprachen bekannten Funktionen.



1. Bsp.: Eine Klasse Kreis
Eine Klasse Kreis enthält bspw. die Attribute „radius“ und „position“ sowie die Operationen „anzeigen()“, „entfernen()“, „setzePosition(pos)“ und „setzeRadius(neuerRadius)“. Die Zusicherung (radius>0) fordert, daß der Wert des Attributs „radius“ stets größer als 0 sein muß. Die Angabe des Initialwertwerts (10,10) für das Attribut „mittelpunkt“ bedeutet: Beim Erzeugen eines Exemplars wird der Wert des Attributs mit (10,10) vorbesetzt.



2. Bsp.: Die Klasse Object aus dem Paket java.lang



Sämtliche Java-Klassen bilden eine Hierarchie mit `java.lang.Object` als gemeinsame Basisklasse.

Assoziationen

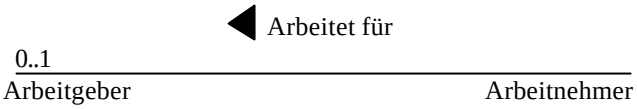
Assoziationen repräsentieren Beziehungen zwischen Instanzen von Klassen.
Mögliche Assoziationen sind:

- einfache (benannte) Assoziationen
- Assoziation mit angefügten Attributen oder Klassen
- Qualifzierte Assoziationen
- Aggregationen
- Assoziationen zwischen drei oder mehr Elementen
- Navigationsassoziationen
- Vererbung

Attribute werden von Assoziationen unterschieden:

Assoziation: Beschreibt Beziehungen, bei denen beteiligte Klassen und Objekte von anderen Klassen und Objekten benutzt werden können.
Attribut: Beschreibt einen privaten Bestandteil einer Klasse oder eines Objekts, welcher von außen nicht sichtbar bzw. modifizierbar ist.

Grafisch wird eine Assoziation als durchgezogene Line wiedergegeben, die gerichtet sein kann, manchmal eine Beschriftung besitzt und oft noch weitere Details wie z.B. Muliplizität (Kardinalität) oder Rollenanmen enthält, z.B.:



Eine Assoziation kann einen Namen zur Beschreibung der Natur der Beziehung („Arbeitet für“) besitzen. Damit die Bedeutung unzweideutig ist, kann man dem Namen eine Richtung zuweisen: Ein Dreieck zeigt in die Richtung, in der der Name gelesen werden soll.

Rollen („Arbeitgeber, Arbeitnehmer) sind Namen für Klassen in einer Relation. Eine Rolle ist die Seite, die die Klasse an einem Ende der Assoziation der Klasse am anderen Ende der Assoziation zukehrt. Die Navigierbarkeit kann durch einen Pfeil in Richtung einer Rolle angezeigt werden.

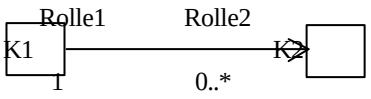


Abb.: Binäre Relation R = C1 x C2

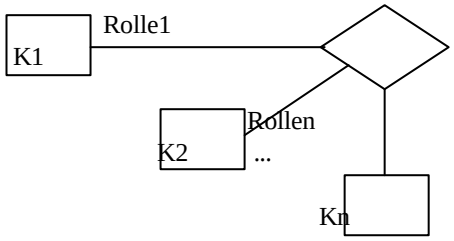
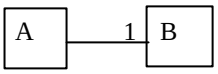
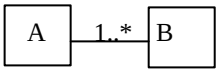
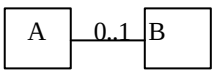
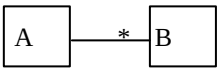


Abb.: n-äre Relation K1 x K2 x ... x Kn

In vielen Situationen ist es wichtig anzugeben, wie viele Objekte in der Instanz einer Assoziation miteinander zusammenhänen können. Die Frage „Wie viele?“ bezeichnet man als Multiplizität der Rolle einer Assoziation. Gibt man an einem Ende der Assoziation eine Multiplizität an, dann spezifiziert man dadurch: Für jedes Objekt am entgegengesetzten Ende der Assoziation muß die angegebene Anzahl von Objekten vorhanden sein.

	Ein A ist immer mit einem B assoziiert	Ein A ist immer mit einem oder mehre-ren B assoziiert	Ein A ist mit keinem oder einem B asso-ziiert	Ein A ist mit kei-nem, einem oder mehreren B asso-ziiert
Unified				

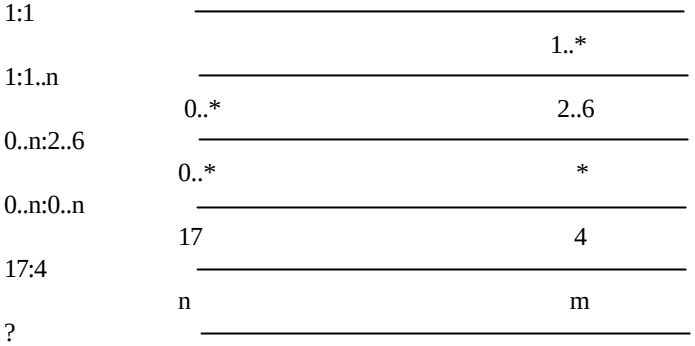
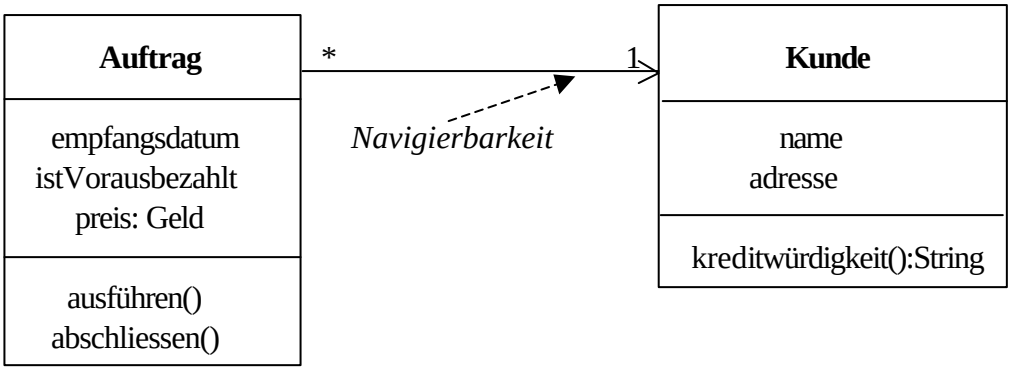
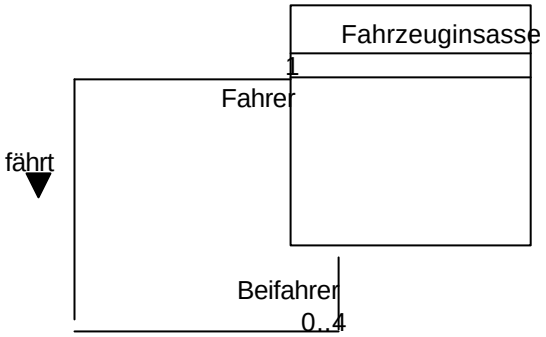


Abb.: Kardinalitäten für Beziehungen

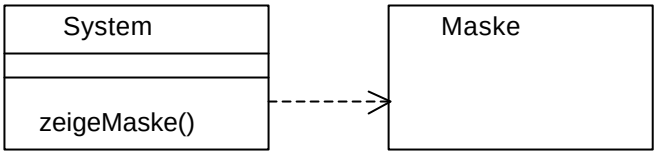
Pfeile in Klassendiagrammen zeigen Navigierbarkeit an. So kann in der folgenden Darstellung



aus einem Auftrag auf den Kunden geschlossen werden, aber einem Kunden fehlt die Fähigkei, über seine Aufträge Auskunft zu geben.
Wenn die Navigierbarkeit nur in einer Richtung existiert, nennt man die Assoziation eine gerichtete Assoziation (unidirectional association). Eine ungerichtete (bidirektionale) Assoziation enthält Navigierbarkeiten in beiden Richtungen. In der UML bedeuten Assoziationen ohne Pfeile, daß die Navigierbarbeit unbekannt oder die Assoziation ungerichtet ist.
Ungerichtete Assoziationen enthalten eine zusätzliche Bedingung: Die zu dieser Assoziation zugehörigen zwei Rollen sind zueinander invers.
Reflexive Assoziationen: Manchmal ist eine Klasse auch mit sich selbst assoziiert. Das kann der Fall sein, wenn die Klasse Objekte hat, die mehrere Rollen spielen können. Ein Fahrzeuginsasse kann entweder Fahrer oder Beifahrer sein. In der Rolle des Fahrers fährt bspw. ein Fahrzeuginsasse „null“ oder „mehrere Fahrzeuginsassen“, die die Rolle von Beifahrern spielen.



Bei einer refleviven Assoziation zieht man eine Linie von der Klasse aus zu dieser zurück. Man kann die Rollen sowie die Namen, die Richtung und die Multiplizität der Assoziation angeben.
Abhängigkeiten: Manchmal nutzt eine Klasse eine andere. Das nennt man Abhängigkeit (dependency). Die UML-Notation ist eine gestrichelte Linie mit einem Pfeil, z.B.:.



Eine **Aggregation** ist eine Sonderform der Assoziation. Sie repräsentiert eine (strukturelle) Ganzes/Teil-Beziehung. Zusätzlich zu einfacher Aggregation bietet UML eine stärkere Art der Aggregation, die **Komposition** genannt wird. Bei der Komposition darf ein Teil-Objekt nur zu genau einem Ganzen gehören.

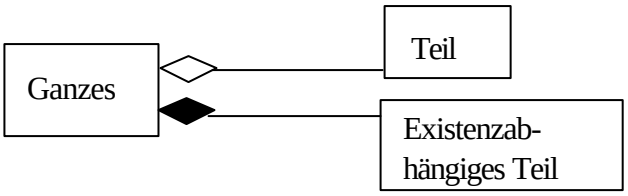
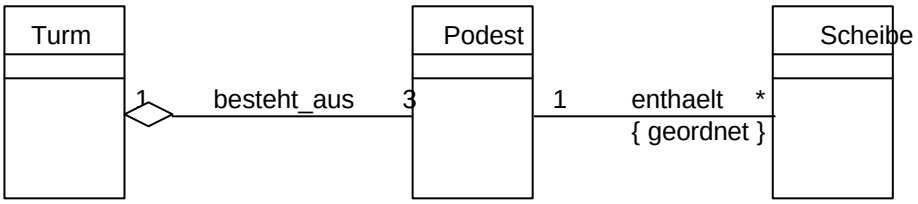


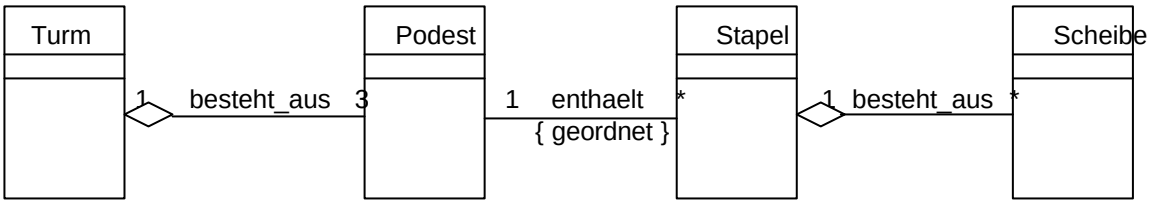
Abb.: Aggregation und Komposition

Eine Aggregation wird durch eine Raute dargestellt. Die Komposition wird durch eine ausgefüllte Raute dargestellt und beschreibt ein „physikalisches Enthaltensein".
Bsp.: Die Darstellung des Problems „Türme von Hanoi" mit Klassendiagrammen[17]
Ziel ist es, einen Stapel von Scheiben, der sich auf einem von drei Podesten befindet, auf ein anderes Podest umzuschichten. Dabei wird das dritte Podest als Hilfspodest benutzt. Jede Scheibe hat eine andere Größe. Die oberste Scheibe des Stapels auf einem Podest darf oben auf den Stapel eines der beiden anderen Podeste gelegt werden, wobei nur eine Scheibe auf einmal bewegt und eine Scheibe niemals auf eine andere gelegt werden darf, die kleiner ist als sie selbst. Je nach Beschreibungstiefe des Problems ergeben sich folgende Klassendiagramme:

1. Ein Turm besteht aus drei Podesten. Auf jedem Podest befinden sich mehrere Scheiben in einer bestimmten Reihenfolge.

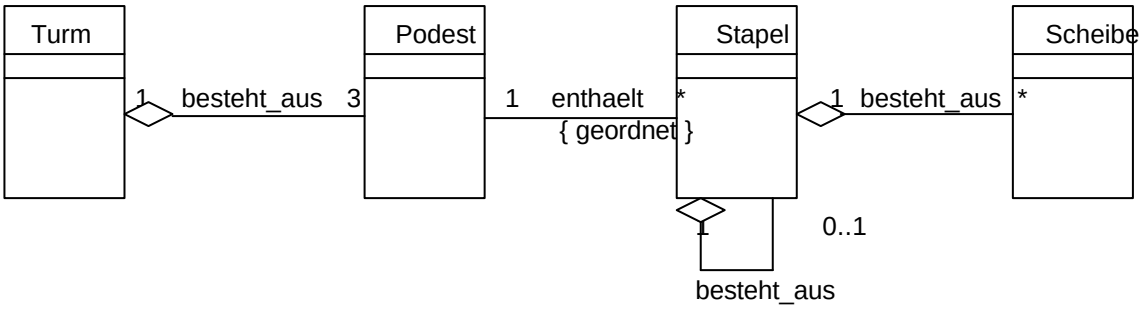


2. Ein Turm besteht aus drei Podesten. Die Scheiben auf den Podesten sind in Untermengen aufgeteilt, die als stapel bezeichnet werden. Ein Stapel ist eine geordnete Menge von Scheiben. Jede Scheibe befindet sich in genau einem Stapel. Auf einem Podest können sich mehrere Stapel befinden, die der Größe nach geordnet sind.

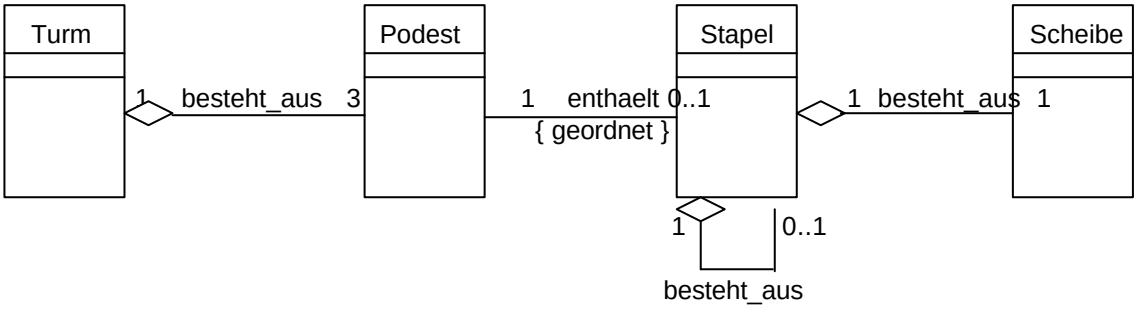


3. Ein Turm besteht aus drei Podesten. Die Scheiben auf den Podesten sind in Untermengen aufgeteilt, die als Stapel bezeichnet werden, wobei sich wie in 2. mehrere Stapel auf einem Pfeiler befinden können. Die Struktur eines Stapels ist jedoch jetzt rekursiv. Ein Stapel besteht aus einer Scheibe (der untersten Scheibe des physikalischen Stapels) und, je nach Höhe des Stapels, null oder einem Stapel

[17] nach einer Vorlage von Torsten Brinda in „LOG IN 20 (2000) Heft 5"



4. Ähnlich wie in 3., außer dass nur ein Stapel mit einem Podest assoziiert ist.



Die **Vererbung** (Spezialisierung bzw. Generalisierung) stellt eine Verallgemeinerung von Eigenschaften dar. Eine Generalisierung (generalization) ist eine Beziehung zwischen dem Allgemeinen und dem Speziellen, in der Objekte des speziellen Typs (der Subklasse) durch Elemente des allgemeinen Typs (der Oberklassse) ersetzt werden können. Grafisch wird eine Generalisierung als durchgezogene Linle mit einer unausgefüllten, auf die Oberklasse zeigenden Pfeilspitze wiedergegeben, z.B.:

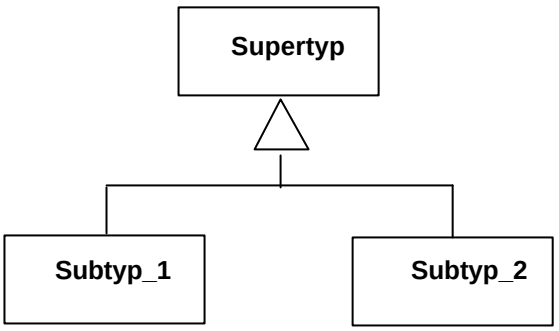
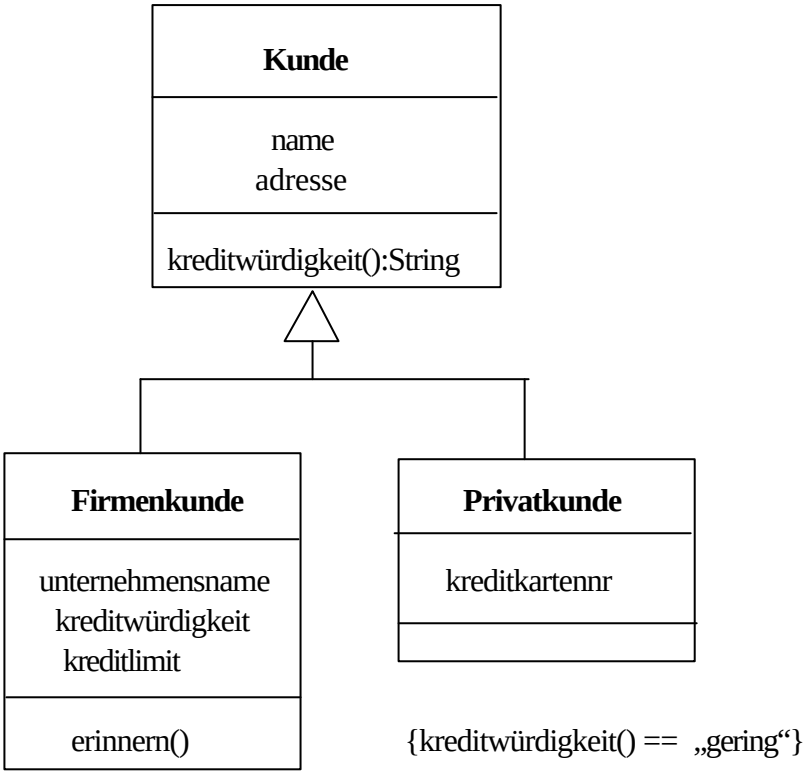


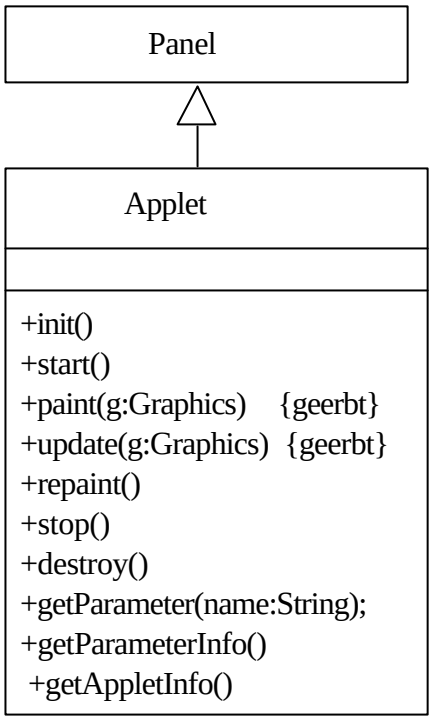
Abb.:

Bsp.: Privat- und Firmenkunden



Es bestehen Unterschiede zwischen Privat- und Firmenkunden, aber auch viele Gemeinsamkeiten. Die Gemeinsamkeiten werden in einer allgemeinen Klasse „Kunde“ gesetzt.

2. Bsp.: Vererbungshierarchie und wichtige Methoden der Klasse Applet



3. Bsp.: Klassendiagramm zur Auftragsbearbeitung

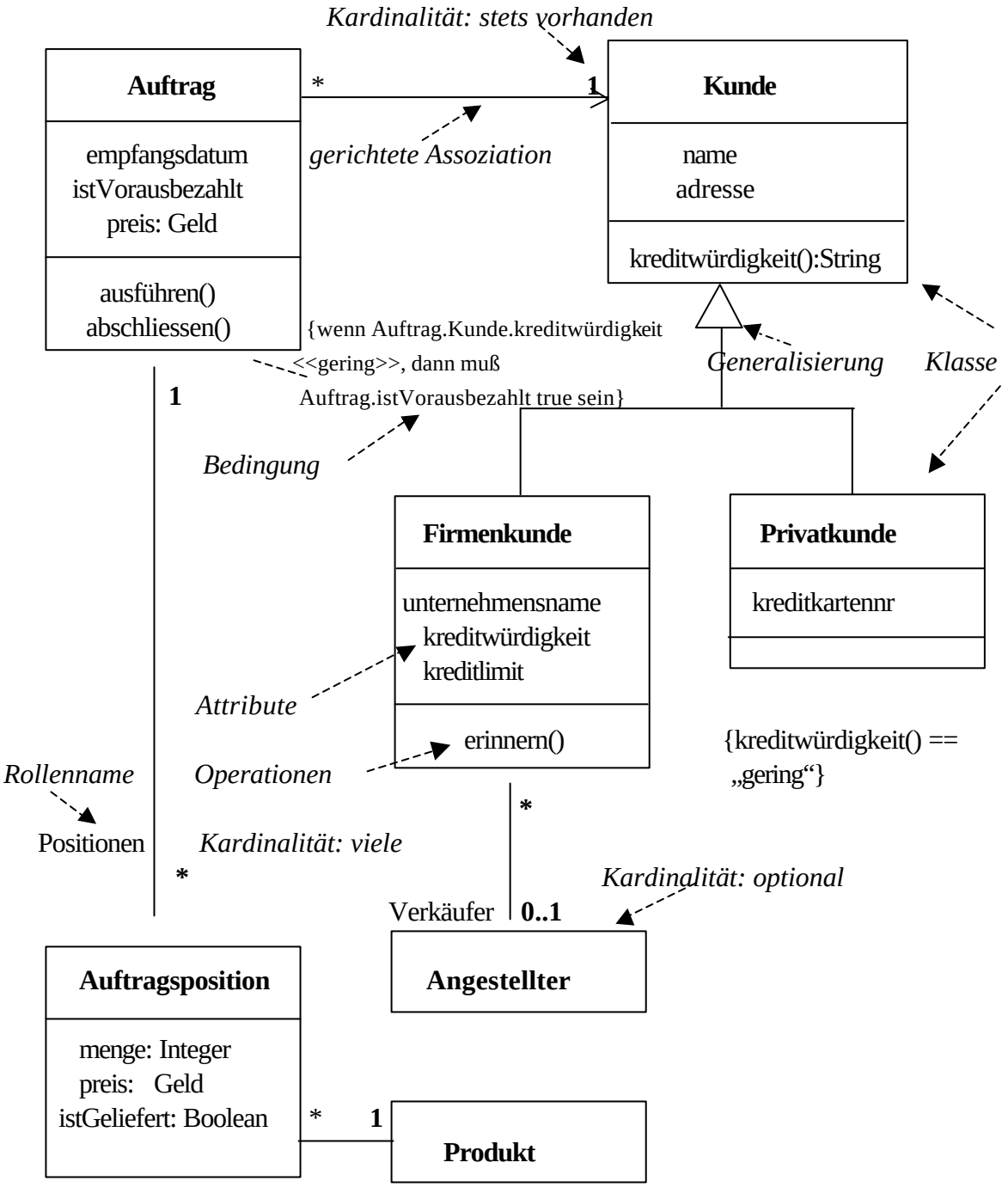


Abb.: Auftragsbearbeitung

Schnittstellen und abstrakte Klassen. Eine Schnittstelle ist eine Ansammlung von Operationen, die eine Klasse ausführt.. Eine Klasse hängt mit der Schnittstelle über die Implementierung zusammen. Eine Schnittstelle modelliert man in der Form einer gestrichelten Linie mit einem großen, unausgefüllten Dreieck, das neben der Schnittstelle steht und auf diese zeigt. Bei der Bildung einer Unterklasse wird beides vererbt. Eine reine Schnittstelle (wie bspw. in Java) ist eine Klasse ohne Implementierung und besitzt daher nur Operationsdeklarationen. Schnittstellen werden oft mit Hilfe abstrakter Klassen deklariert.
Bei abstrakten Klassen oder Methoden wird der Name des abstrakten Gegenstands in der UML u.U. kursiv geschrieben. Ebenso kann man die Bedingung `{abstract}` benutzen.

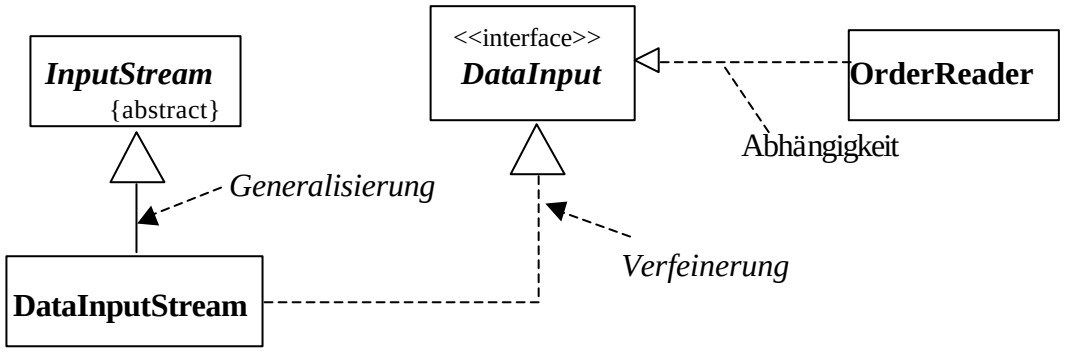


Abb.: Schnittstellen und abstrakte Klassen: Ein Beispiel aus Java

Irgendeine Klasse, z.B. „`OrderReader`“ benötigt die `DataInput`-Funktionalität. Die Klasse `DataInputStream` implementiert `DataInput` und `InputStream`. Die Verbindung zwischen `DataInputStream` und `DataInput` ist eine „Verfeinerung (refinement)“. Eine Verfeinerung ist in UML ein allgemeiner Begriff zur Anzeige eines höheren Detaillierungsniveaus.
Die Objektbeziehung zwischen `OrderReader` und `DataInput` ist eine Abhängigkeit. Sie zeigt, daß „OrderReader“ die Schnittstelle „DataInput für einige Zwecke benutzt.
Abstrakte Klassen und Schnittstellen ermöglichen die Definition einer Schnittstelle und das Verschieben ihrer Implementierung auf später. Jedoch kann die abstrakte Klasse schon die Implementierung einiger Methoden enthalten, während die Schnittstelle die Verschiebung der Definition aller Methoden erzwingt.
Eine andere Darstellung einer Klasse und einer Schnittstelle besteht aus einem (kleinen) Kreis, der mit der Klasse durch eine Linie verbunden ist, z.B.:

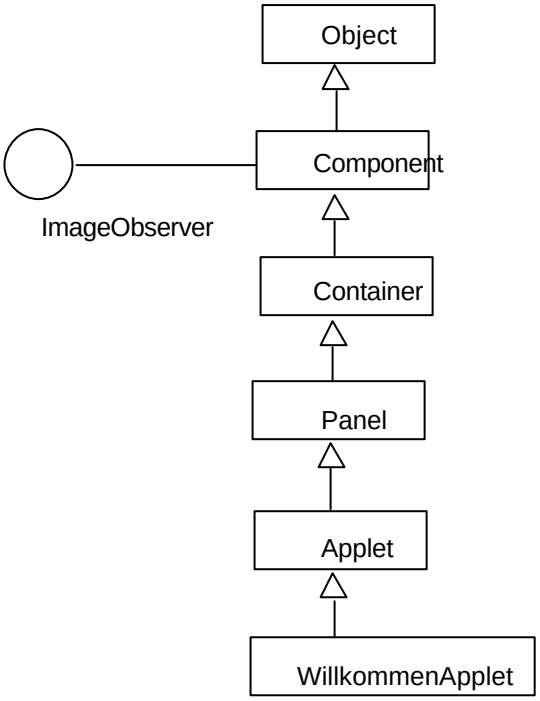


Abb.: Verrebungshierarchie von `WillkommenApplet` einschl. Schnittstelle ImageObserver

2.2.4 Interaktionsdiagramme

Es gibt zwei Arten von Interaktionsdiagrammen:

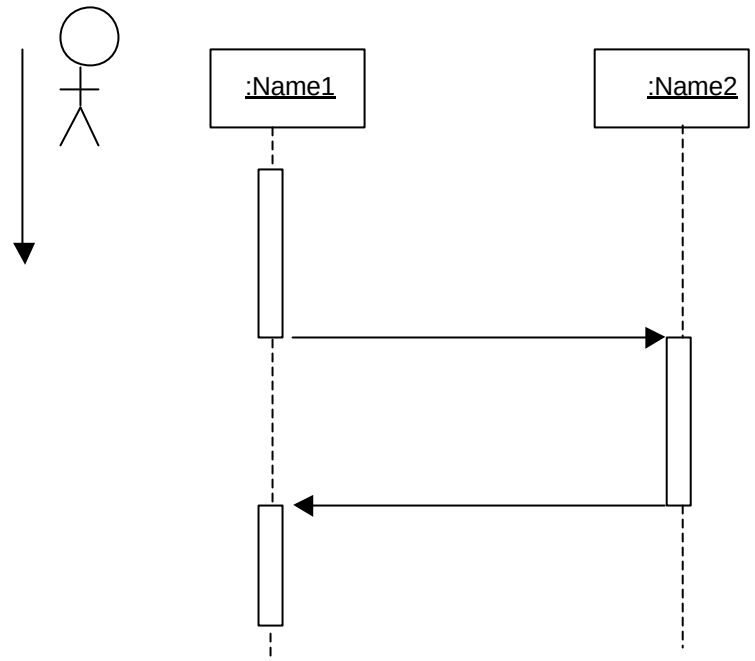
- Sequenzdiagramme
- Kollaborationsdiagramme

Die beiden Diagramme beschreiben zeitliche Abläufe, d.h. Aufrufsequenzen.
Ausgangspunkt von Interaktionsdiagrammen sind Fallbeispiele (Szenarios). Beim Erstellen der Diagramme konzentriert man sich auf die wichtigsten Fälle. Anschließend werden die Sonderfälle mit einbezogen.

1. Sequenzdiagramme

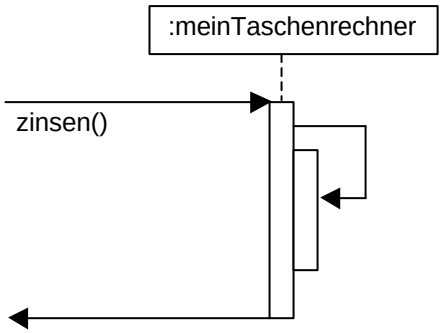
Ein Sequenzdiagramm ist ein Interaktionsdiagramm, bei dem die zeitliche Abfolge der Nachrichten im Vordergrund steht. Sequenzdiagramme visualisieren

- die Lebensdauer von Objekten (gestrichelte Linie, die die Existenz eines Objekts während eines Zeitraums darstellt). Die Zeitachse verläuft von oben nach unten. Objekte sind als Rechtecke am oberen Rand von gestrichelten (Lebens-) Linien dargestellt. Am linken Rand können noch Kommentare stehen.
- die Aktivität von Objekten. Der Focus-of-Control (das langezogene Rechteck) gibt an, wo eine Aktivität ausgeführt wird. Ist ein Objekt ohne Aktivität vorhanden, wird dies durch eine gestrichelte Linie angezeigt. Objekte werden durch einen Pfeil auf das Rechteck erzeugt, ein Löschen wird durch ein Kreuz dargestellt.
- Nachrichtenaustauch zwischen Objekten (beschriftete Pfeile). Über den Pfeilen stehen Operationen, die ausgeführt werden. In eckigen Klammern können Bedingungen angegeben werden, wann die Operation ausgeführt wird. Ruft das Objekt eine Operation von sich selbst auf, dann zeigt der Pfeil auf die Lebenslinie zurück. Eine Nachricht kann einfach (→), synchron (➔) oder asynchron (halbe Pfeilspitze) sein. Verwendet ein Objekt eine synchrone Nachricht, dann wartet es, bevor es mit seiner Arbeit fortfährt, auf eine Antwort. Sendet es eine asynchrone Nachricht, so wartet es keine Antwort ab, bevor es seine Arbeit wieder aufnimmt.
- Das Diagramm zeigt in senkrechter Richtung den Zeitablauf an. Es beginnt oben und schreitet nach unten fort. Das Sequenzdiagramm ist zweidimensional. In der „Rechts-Links“-Dimension werden Objekte angeordnet



Darstellung der Rekursion: Manchmal hat ein Objekt eine Operation, die sich selbst aufruft. Diese wird als Rekursion bezeichnet.

Bsp.: Ein Objekt ist „meinTaschenrechner und eine seine Operationen rechnet Zinsen aus. Zur berechnung der Zinseszinsen, muß die Zinsberechnungsopeartion das Objekt mehrmals selbst aufrufen.



Die in den Anwendungsfällen beschriebenen Objektinteraktionen können in Sequenzdiagrammen verfeinert werden. Für jeden Anwendungsfall können ein oder mehrere Sequenzdiagramm(e) konstruiert werden. Ein Sequenzdiagramm kann entweder eine Instanz (ein einzelnes Szenario) eines Anwendungsfalls zeigen, oder es kann generisch sein und alle Szenarios eines Anwendungsfalls verkörpern. Generische Sequenzdiagramme bieten oft Gelegenheit „if“-Anweisungen oder „while“-Schleifen darzustellen.

1. Bsp.: Sequenzdiagramm für einen einfachen Anwendungsfall, der folgendes Verhalten zeigt:

- das Auftragserfassungsfenster sendet eine Nachricht bereiteVor() an einen Auftrag
- Der Auftrag sendet dann bereiteVor() an jede Auftragsposition des Auftrags
- Jede Auftragsposition prüft den angegebenen Lagerartikel
-- Wenn die Prüfung wahr liefert, löscht die Auftragsposition die entsprechende Menge von Lagerartikel aus dem Lager
-- Sonst ist die Menge der Lagerartikel unter die Bestellgrenze gefallen, und der Lagerartikel fordert eine neue Lieferung an

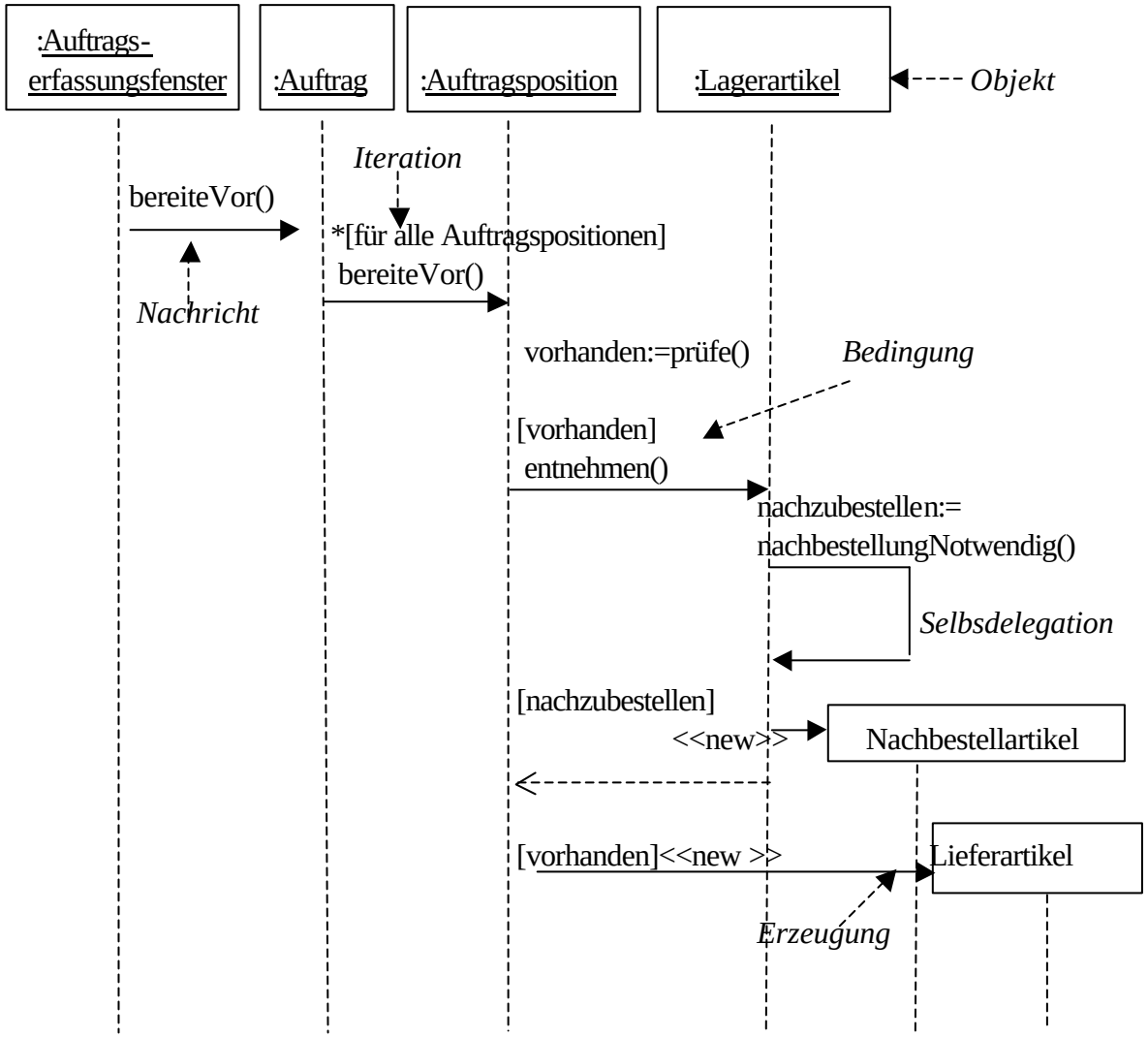


Abb.: Sequenzdiagramm mit Aufträgen

2. Bsp.: Ablauf des „Use Cases“: „Neue Vorlesung anbieten“

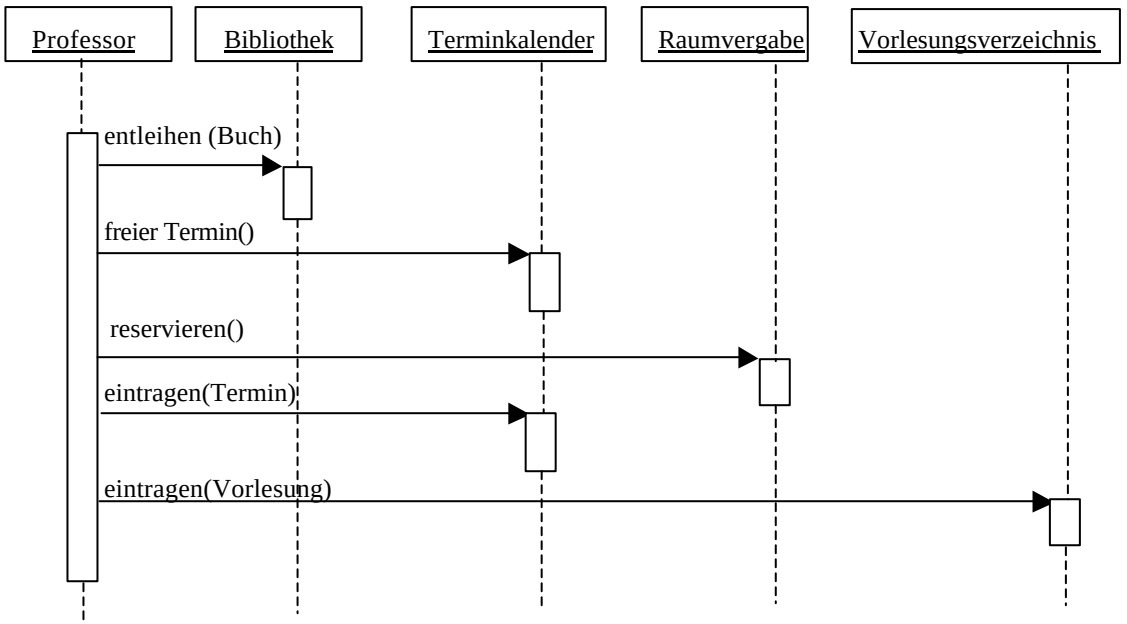


Abb.: Sequenzdiagramm: „Neue Vorlesung anbieten“

3. Bsp.: Lebenszyklus eines Applet

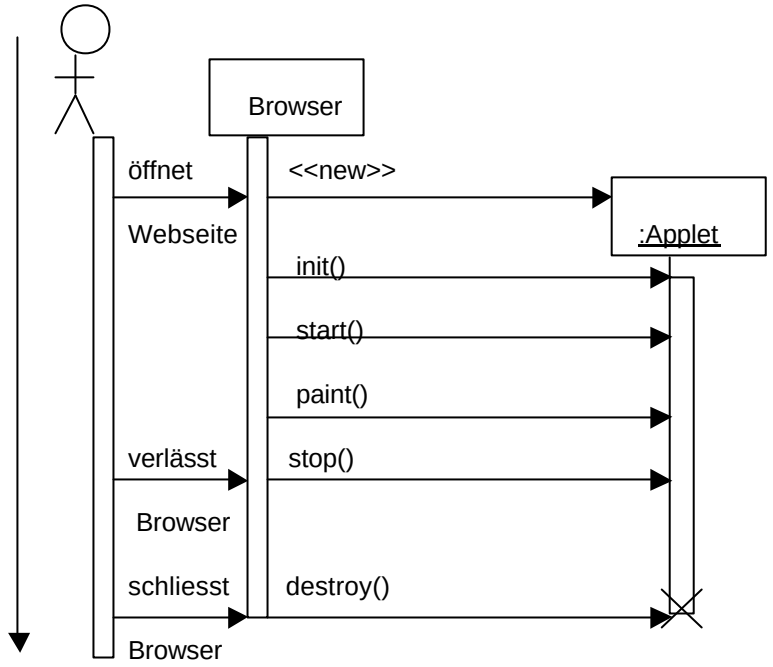


Abb.: Lebenszyklus eines Applets

2. Kollaborationsdiagramme

Ein Kollaborationsdiagramm ist ein Interaktionsdiagramm bei denen die strukturelle Organisation der Objekte im Vordergrund steht, die Nachrichten senden bzw. empfangen.
Sequenz- und Kollaborationsdiagramme sind isomorph, so dass man das eine in das andere umwandeln kann. Sequenzdiagramme haben zwei Merkmale, die sie von Kollaborationsdiagrammen unterscheiden:

1. die Objektlebenslinie (senkrechte gestrichelte Linie)
2. Kontrollfokus (langes, schmales Rechteck)

Kollaborationsdiagramme haben zwei Merkmale, die sie von Sequenzdiagrammen unterscheiden:

1. Ein Objektdiagramm zeigt Objekte und die Beziehungen untereinander. Ein Kollaborationsdiagramm ist eine Erweiterung des Objektdiagramms. Es zeigt neben den Assoziationen zwische Objekten auch die Nachrichten (Darstellung: neben der Assoziationslinie durch einen Pfeil), die diese einander senden.
2. Zur Kennzeichnung der zeitlichen Anordnung schreibt man eine Nummer als Präfix vor die Nachricht (beginnend mit der Nummer 1, monoton wachsend für jede weitere Nachricht für den Kontrollfluss).

In einem Kollaborationsdiagramm werden verwendete Objekte als kleines Sinnbild (icon) dargestellt. Wie in einem Sequenzdiagramm zeigen Pfeile die Nachrichten der vorgegebenen Anwendungsfälle an.

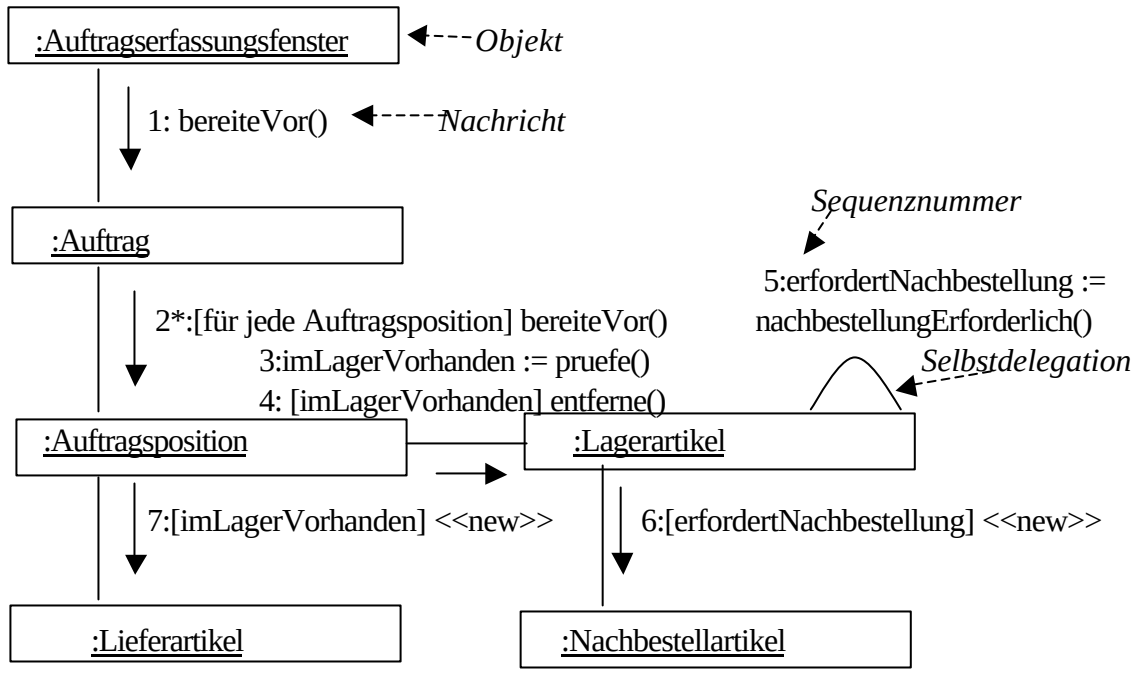
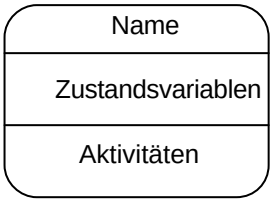


Abb.: Kollaborationsdiagramm mit einfacher Numerierung

2.2.5 Zustandsdiagramme

Ein Zustandsdiagramm zeigt einen Automaten, der aus Zuständen, Zustandsübergängen (Transitionen), Ereignissen und Aktivitäten besteht. Bei Zustandsdiagrammen steht das nach Ereignissen geordnete Verhalten eines Objekts im Vordergrund.
Zustände werden symbolisch über Rechtecke mit abgerundeten Ecken dargestellt.



Durch das Eintreffen von Ereignisssen kann ein anderer Zustand erreicht werden, was durch Pfeile angedeutet wird. Die Pfeile haben eine Beschriftung für das auslösende Ereignis und stellen Übergänge dar. In den Zuständen werden Operationen mit einer gewissen Zeitdauer ausgeführt, die Aktivität genannt werden. Im Gegensatz dazu wird die Zeitdauer von Aktionen mit Null angenommen. Aktionen sind Operationen, die an Übergängen ausgeführt werden. Angezeigt werden Aktionen durch einen Schrägstrich vor dem Namen. Selbstverständlich können Übergänge auch an Bedingungen, die in eckigen Klammern stehen, geknüpft werden.

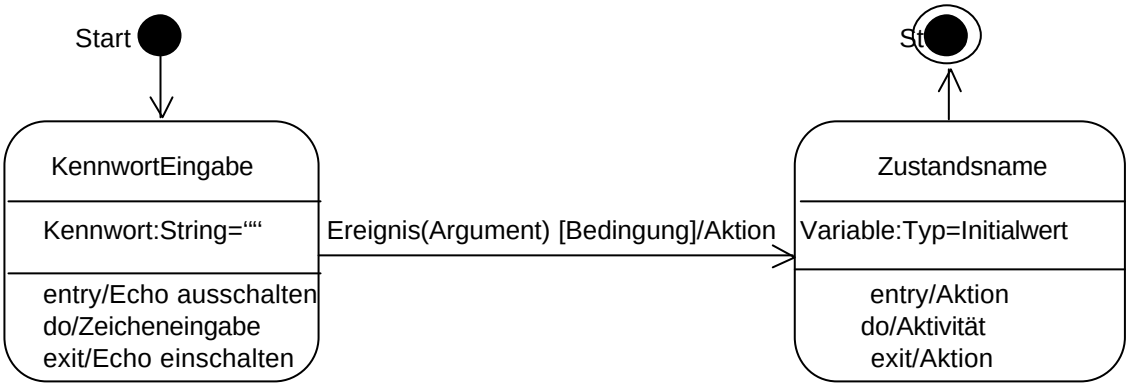


Abb.: Symbole im Zustandsdiagramm

Die Wörter entry, do, exit sind reserviert.

entry gibt das an, was geschieht, wenn das System in den Zustand eintritt.
exit gibt an, welche Aktion beim Verlassen des Zustands ausgeführt wird.
do beschreibt das, was geschieht, wenn sich das System in dem betreffenden zustand befindet.

1. Bsp.: Zustandsdiagramm für einen Getränkeautomaten

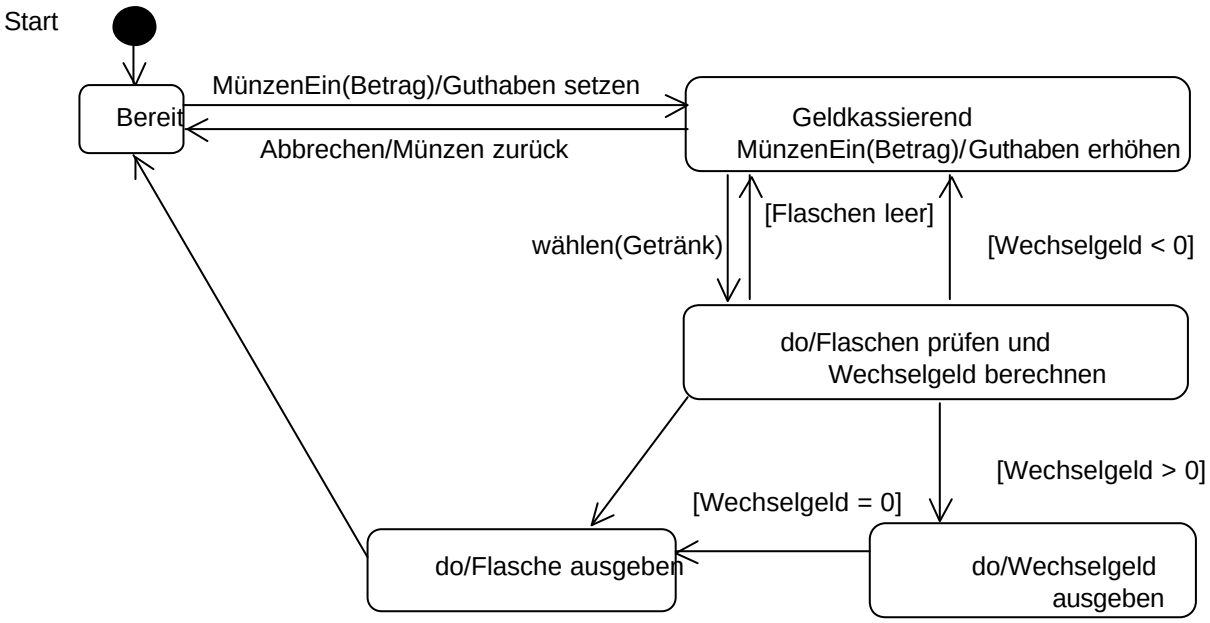


Abb.: Zustandsdiagramm für einen Getränkeautomaten

In den meisten OO-Techniken werden Zustandsdiagramme zur Darstellung des Verhaltens während der Lebenszeit eines einzelnen Objekts für eine einzelne Klasse entworfen.
Aktionen sind mit Transitionen assoziiert und werden als kurzzeitige und nicht-unterbrechbare Prozesse betrachtet. Aktivitäten sind mit Zuständen assoziiert und können länger andauern. Eine Aktivität kann durch ein Ereignis unterbrochen werden.
2. Bsp.: Zustandsdiagramm für ein Auftragsbearbeitungssystem
Die Anfangstransition trägt die Beschriftung „/hole erste Auftragsposition“. Die Syntax einer Transitionsbeschriftung kennzeichnet drei jeweils optionale Teile: *Ereignis [Bedingung] / Aktion*. Sobald die Aktion „/hole erste Auftragsposition“ ausgeführt ist, wird der Zustand „in Prüfung“ erreicht, mit dem die Aktivität „Prüfe Auftragsposition“ verbunden ist (Syntax: do/Aktivität). Vom Zustand „in Prüfung“ gehen drei Transitionen aus. Alle drei haben in ihren Beschriftungen nur Bedingungen. Eine mit einer Bedingung (guard) versehene Transition findet nur statt, wenn diese als wahr ermittelt wird.
Aus einem Zustand kann nur eine Transition ermittelt werden. Dabei ist zu berücksichtigen:

1. Wurden noch nicht alle Auftragspositionen geprüft, dann hole die nächste Auftragsposition und kehre in den Zustand „in Prüfung“ zurück.
2. Wurden alle Auftragspositionen geprüft und sind diese alle vorrätig, dann gehe in den zustand „freigegeben“. Im Zustand „freigegeben“ befindet sich die Aktivität, die eine Auslieferung bewirkt. Sobald „Auslieferung initiieren“ beendet wird, bleibt der Auftrag im Zustand „freigegeben“, bis das Ereignis „ausgeliefert“ auftritt.
3. Wurden alle Auftragspostionen geprüft und sind nicht alle vorrätig, dann gehe in den Zustand „wartend“ über. Die beiden vom Zustand „wartend“ ausgehenden Transitionen sind mit dem Ereignis „Position erhalten“ beschriftet. Das bedeutet: Ein Auftrag wartet, bis er dieses Ereignis empfängt.

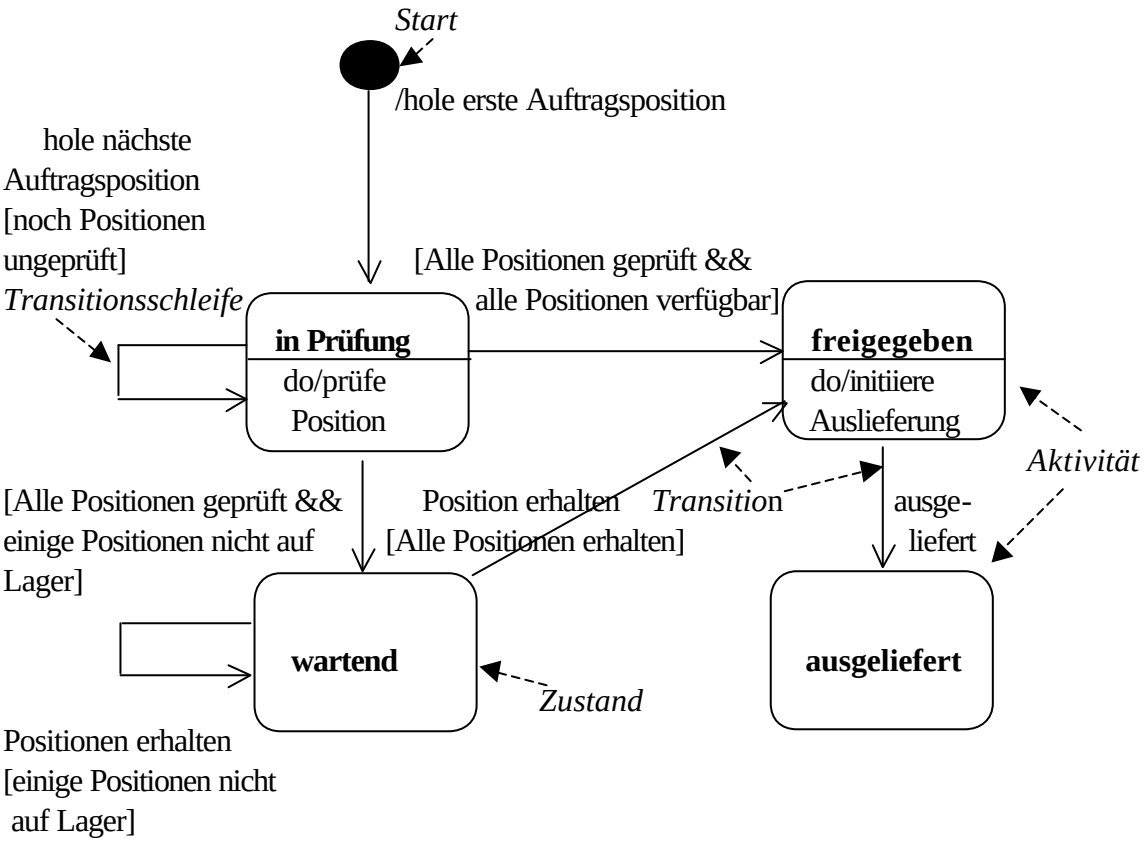
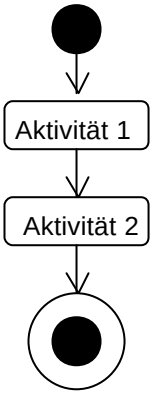


Abb.: Zustandsdiagramm

2.2.6 Aktivitätsdiagramme

Elemente der Aktivitätsdiagramme

Ein Aktivitätsdiagramm ist ein erweitertes Zustandsdiagramm. Ein Zustandsdiagramm zeigt die Zustände eines Objekts und stellt die Aktivitäten als Pfeile dar, die diese Zustände verbinden. Das Aktivitätendiagramm hebt besonders die Aktivitäten hervor.



Das zentrale Symbol ist die Aktivität (activity). Die Interpretation dieses Begriffs ist abhängig von der Perspektive, aus der man ein Diagramm zeichnet. In einem konzeptionellem Diagramm ist eine Aktivität eine von Mensch oder Maschine durchzuführender Aufgabe In Diagrammen aus Spezifikations- oder Implementierungsperspektive ist eine Aktivität eine Methode einer Klasse.
Eine Aktivität ist ein einzelner Schritt in einem Verarbeitungsablauf. Jede Aktivität wird als Rechteck mit abgerundeten Ecken dargestellt. Angestoßen wird sie durch eine eingehende Transition, ihr folgt mindestens eine ausgehende Transition. Die ausgehende Transition implementiert den Abschluß der internen Verarbeitung. Sofern mehrere ausgehende Transitionen vorliegen, sind diese durch definierte Bedingungen (boolsche Ausdrücke) zu unterscheiden. Ein Pfeil repräsentiert den Übergang von einer Aktivität zur nächsten.
Transitionen können synchronisiert und geteilt werden. Außerdem lassen sich Verzweigungs- und Entscheidungspunkte definieren. Es gibt 2 Möglichkeiten, einen Entscheidungspunkt darzustellen, z.B.:

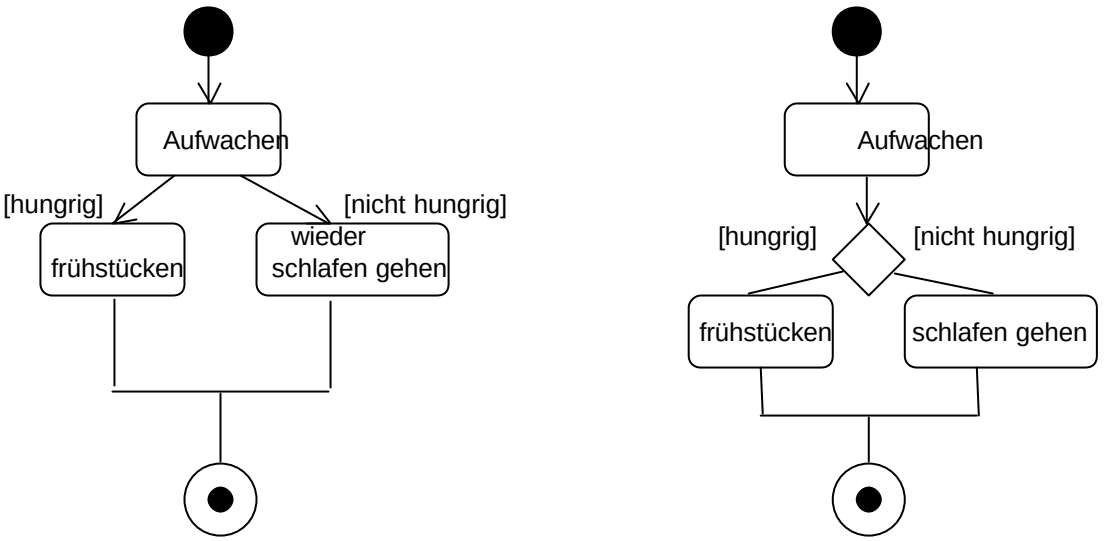


Abb.:

Manchmal gabelt ein Aktivitätsüberganf in zwei getrennte Pfade, die zur gleichen Zeit (d.h. nebenläufig) ablaufen und dann wieder zusammen kommen, z.B.:

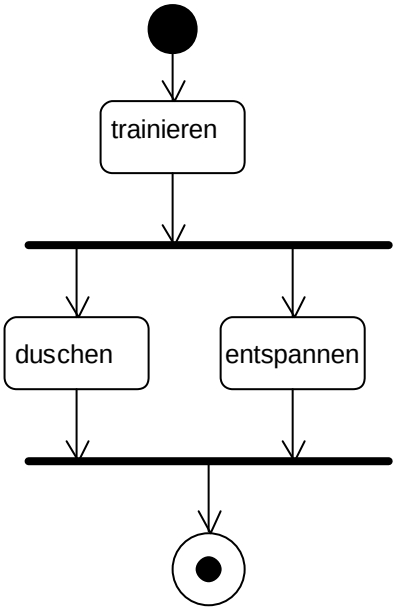


Abb.:

Die Gabelung in zwei nebenläufige Pfade bestimmt die senkrecht zum Aktivitätsübergang fette, durchgezogene Linie.
Der entscheidende Unterschied zu üblichen Ablaufdiagrammen besteht in der Handhabung paralleler Prozesse. Für parallel laufende Aktivitäts-Pfade gilt, daß ihre Reihenfolge irrelevant ist. Sie können nacheinander oder gleichzeitig ablaufen. Jeder Aktivität wird außerdem eindeutig ein Objekt bzw. eine Klasse zugeordnet.
Aktivitätsdiagramme sind somit geeignet, unterschiedliche Arten von Abläufen darzustellen. Sie können die fachlichen Zusammenhänge und Abhängigkeiten, die sich hinter einem Anwendungsfall verbergen (im Gegensatz zum Sequenzdiagramm) vollständig beschreiben. Ein Aktivitätsdiagramm kann sogar anwendungsfallübergreifend sein. Geschäftsregeln und Entscheidungslogik lassen sich ebenfalls abbilden.
Aktivitätsdiagramme enthalten üblicherweise

- Aktivitätszustände und Aktionszustände
- Zustansübergänge (Transitionen)
- Objekte

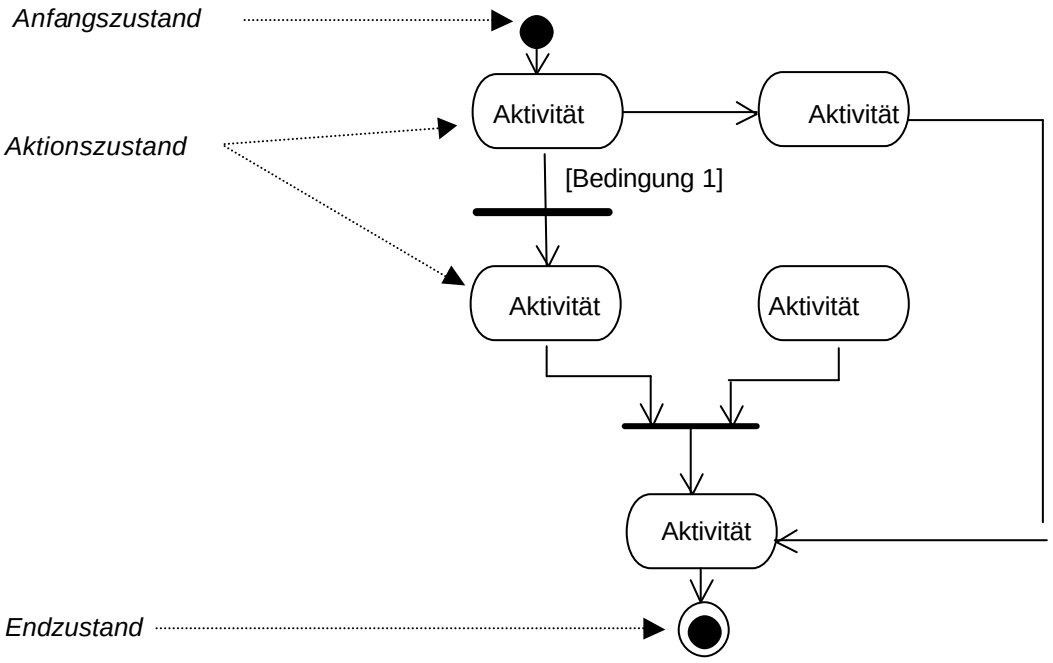


Abb.: Aktivitätsdiagramm

Beispiel: Kaffeekochen

Jede Aktivität kann von einer anderen gefolgt werden (einfache Aneinanderreihung), z.B.: wird in der folgenden Abb. „Kaffeepulver in Filter einfüllen“ von der Aktivität „Filter in Maschine einsetzen“ gefolgt.
Aus „Bestimme Getränk“ kommen zwei Trigger. Jeder Trigger besitz eine Bedingung (guard), ein zu „wahr“ oder „falsch“ auszuwertender logischer Ausdruck. Wird der Kaffeeweg eingeschlagen, dann führt dieser Trigger zu einem Synchronisationsbalken (synchronization bar), an den drei ausgehende Trigger angeschlossen sind. Die darüber erreichten Aktivitäten können parallel stattfinden (Reihenfolge ist unwichtig). Die von diesen Aktivitäten ausgehenden Trigger werden wieder in einem Synchronisationsbalken zusammengafaßt. Ein einfacher Synchronisationsbalken bedeutet: Der ausgehende Trigger tritt erst dann in Erscheinung, wenn alle eingehenden Trigger aufgetreten sind.
In einem anderen Pfad des Beispiels gibt es eine zusammengesetzte Entscheidung. Die erste Entscheidung betrifft den Kaffee. Sollte es keinen Kaffee geben, gelangt man zur zweiten Entscheidung, die durch eine Entscheidungsraute markiert ist.

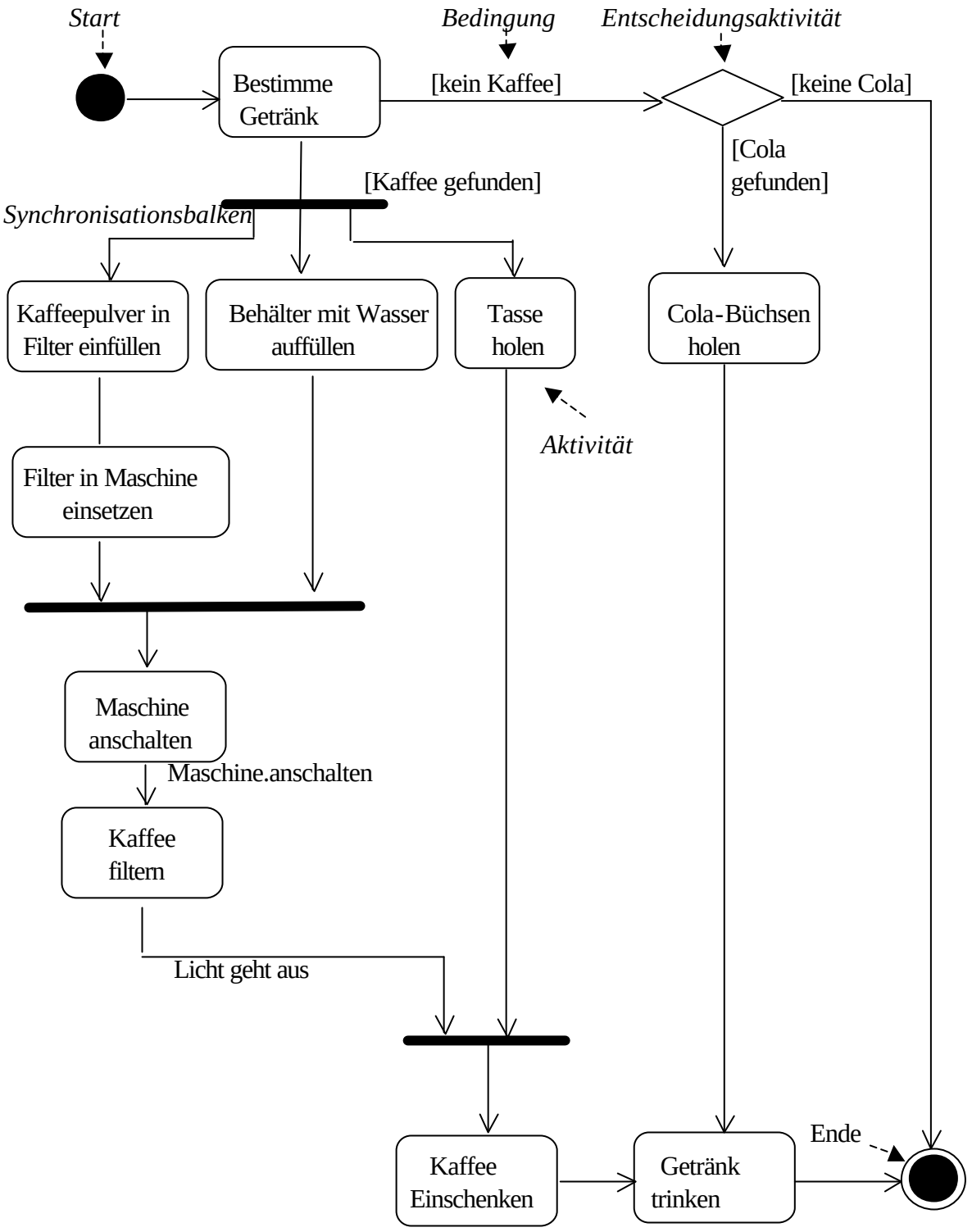


Abb.: Aktivitätsdiagramm

Mit Aktivitätsdiagrammen kann auch die „Visualisierung von Rollen“ in Angriff genommen werden. Zu diesem Zweck spaltet man das Diagramm in parallele Bahnen auf, die man Schwimmbahnen nennt. Jede Schwimmbahn zeigt oben den Namen einer Rolle und präsentiert und präsentiert die Aktivitäten der einzelnen Rolle.

2.2.7 Package-Diagramm

Package-Diagramme dienen zur Strukturierung der verschiedenen Darstellungen. Damit werden Gruppen von Diagrammen oder Elementen zusammengefaßt.

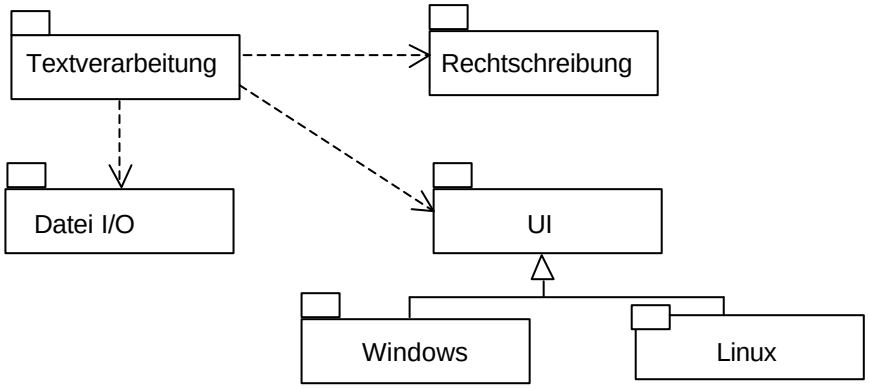


Abb.: Beispiel für ein Package-Diagramm

Ein Package-Diagramm besteht im wesentlichen aus Packages (dargestellt durch große Rechtecke mit kleinem Rechteck links oben) und Abhängigkeiten (den gestrichelten Pfeilen). Eine Abhängigkeit gibt an: Bei einer Änderung des Packages an der Pfeilspitze muß das Package am anderen Ende der gestrichelten Linie evtl. geändert und neu übersetz werden. Packages können weitere Packages enthalten.

2.2.8 Implementierungsdiagramme

Implementierungsdiagramme zeigen Aspekte der Implementierung. Diese umfassen die Codestruktur und die Struktur des Systems zur Ausführungszeit. Es gibt zwei Formen

Komponentendiagramm
Einsatzdiagramm (Deploymentdiagramm)

1. Komponentendiagramm

Das Komponentendiagramm zeigt die Abhängigkeit unter den Softwarekomponenten, d.h.: die Abhängigkeiten zwischen Quellcode, Binärcodekomponenten und ausführbaren Programmen. Einige dieser Komponenten existieren nur während des Übersetzungsvorgangs, einige nur während des „Linkens“, andere zur Ausführungszeit und wieder andere die ganze Zeit über. Im Komponentendiagramm haben die Darstellungen Typencharakter.

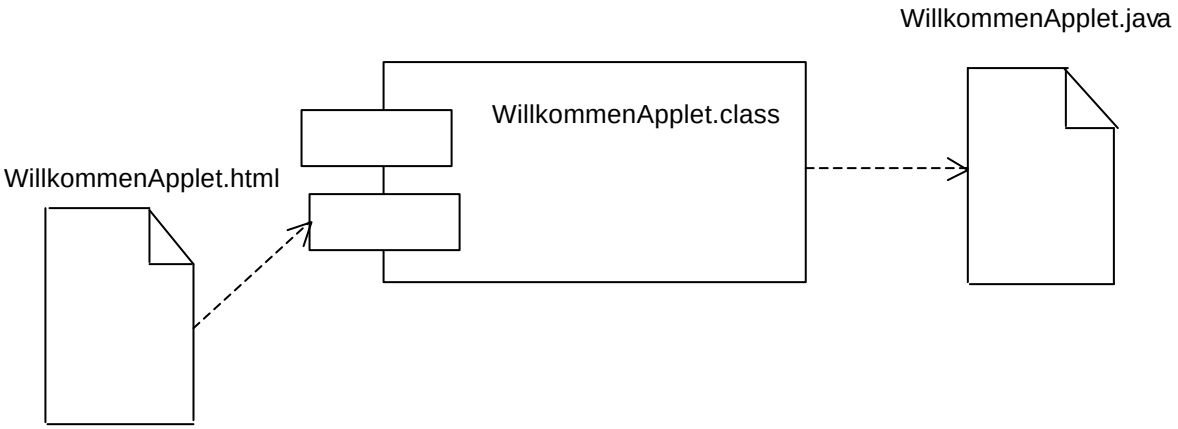


Abb.: Komponentendiagramm

2. Einsatzdiagramm

Ein Einsatzdiagramm (deployment diagram) zeigt die Konfiguration der im Prozeß befindlichen Knoten zur Laufzeit sowie der auf ihnen existierenden Komponenten. Knoten (Quader) stellen eine Hardware- oder Verarbeitungseinheit dar. Unter den Konten existieren Verbindungen. Dabei handelt es sich um die physikalischen Kommunikationspfade, die als Linien gezeichnet werden.

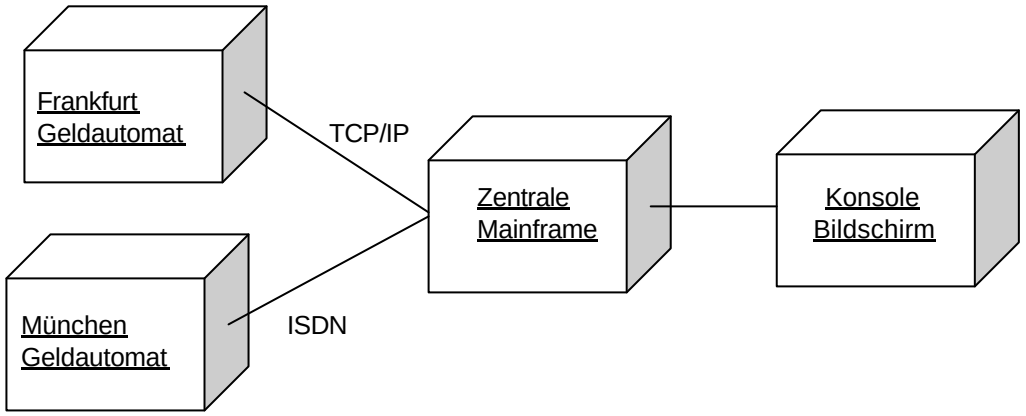


Abb.: Deployment-Diagramm

2.2.9 Schema-Modellierung relationale Datenbanken

Die Klassendiagramme der UML bilden eine Obermenge von Entity-Relationship-Diagrammen. Klassendiagramme lassen auch das Modellieren von Verhaltensweisen zu. In einer physischen Datenbank werden diese logische Operationen im allg. in Trigger oder gespeicherte Prozeduren umgewandelt.

1. Modellierung eines logischen Datenbankschemas

Das Modellieren eines logischen Datenbankschemas[18] umfaßt:

- die Identifikation von Klassen, deren Zustand die Lebensdauer ihrer Anwendungen überdauern muß.
- das Erstellen eines Klassendiagramms, das all diese Klassen enthält, und das Markieren dieser Klassen mit den Standardeigenschaftswert persistent.
- Expansion der strukturellen Eigenschaften dieser Klassen (Spezifikation der Details zu den Attributen, Konzentration auf Assoziationen und Kardinalitäten).
- Beachtung typischer Muster, z.B. rekursive Assoziationen, Eins-zu-eins-Assoziationen, n-äre Assoziationen.
- Betrachtung der Vehaltensweisen dieser Klassen, z.B. durch Expansion von Operationen, die für Datenzugriff und Integrität der Daten wichtig sind.
- Überführung, falls möglich mit Hilfe von Programmen, des logischen in einen physischen Entwurf

[18] Booch, Grady u. a.: Das UML-Benutzerhandbuch, Addison-Wesley, Bonn 1999, Seite 123

Bsp.: Modellierung eines Datenbankschemas für eine Fachhochschule

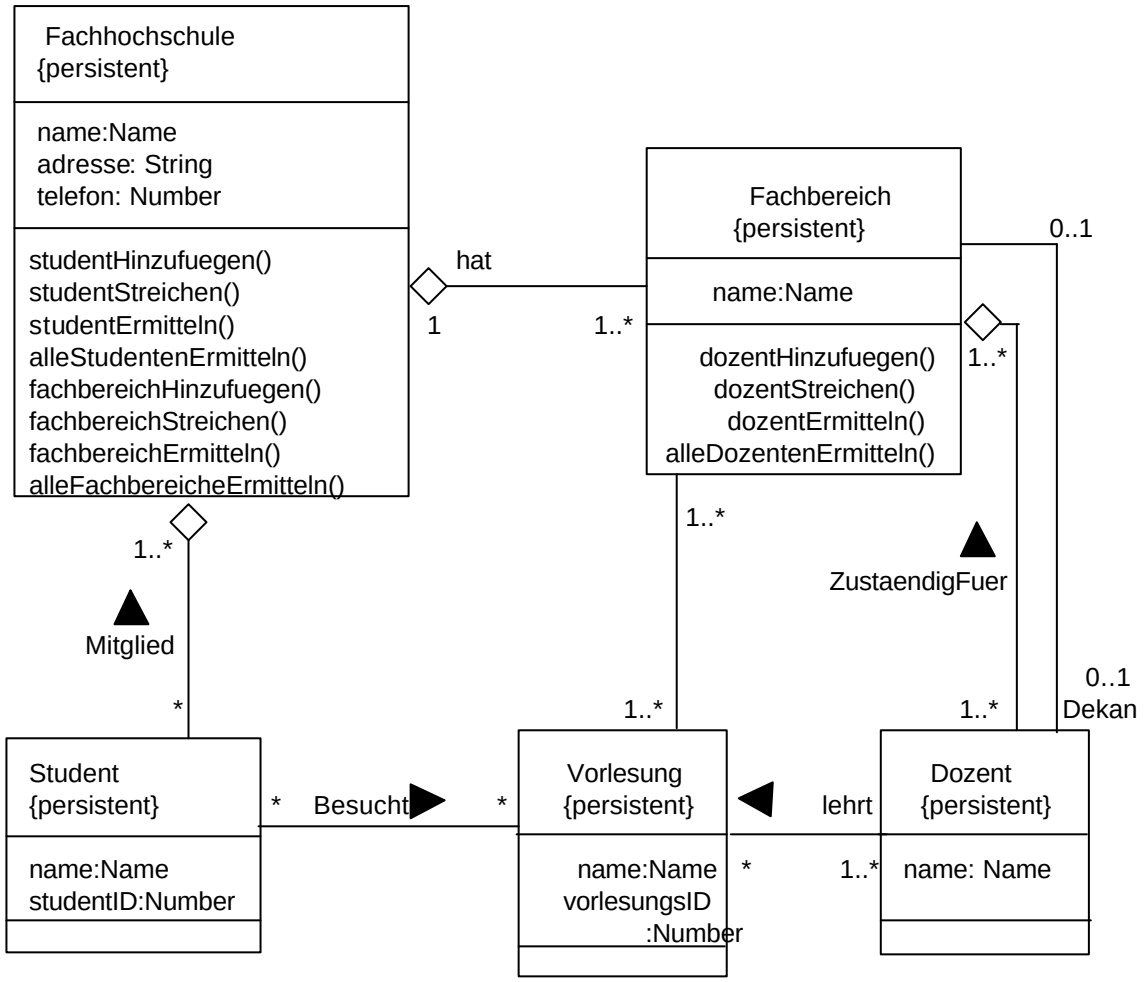


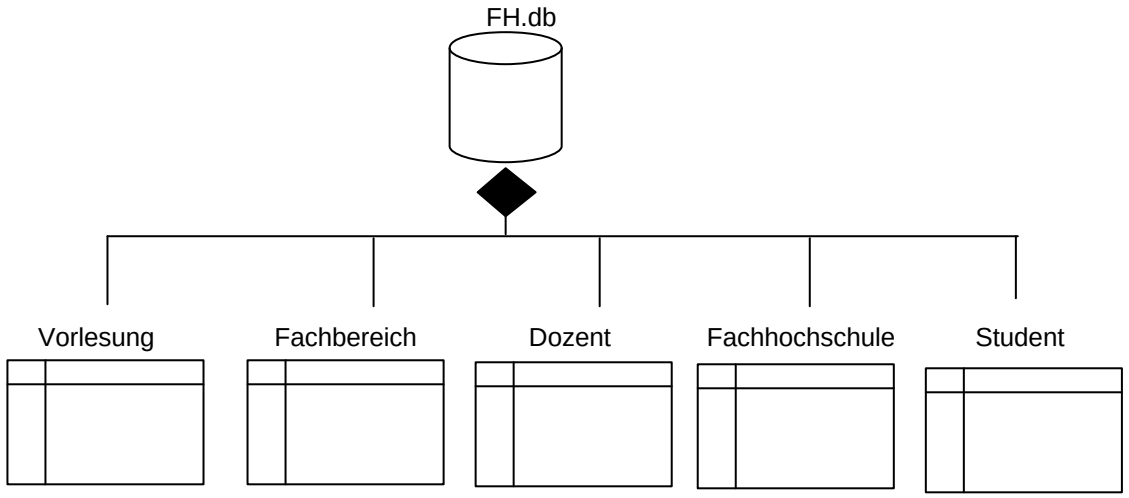
Abb.: Modellieren eines logischen Datenbankschemas

2. Die Modellierung eines physischen Datenbankschemas

Das Modellieren einer physischen Datenbank[19] umfaßt:

- Identifikation der Klassen, die das logische Datenbankschema bilden
- Wahl einer Strategie zur Abbildung dieser Klassen auf Tabbellen.
- Erstellen eines Komponentendiagramms (mit den als Tabellen stereotypisierten Tabellen) zur Visualisierung und Dokumentation

Bsp.: Modellieren der physischen Datenbank FH.db



[19] vgl. Booch, Grady u.a.: Das UML-Benutzerhandbuch, Seite 451.

3. Objektorientierte Programmierung mit Java

1. Objekte, Klassen, Eigenschaften und Verhaltensweisen

   Definition von Klassen
   Anweisungen und Ausdrücke
   Erstellen neuer Objekte und Instanzen
   Referenzen auf Objekte
   Konvertieren
   Manipulieren von Objekten
   Bestimmen der Klasse eines Objekts
   Kopieren von Objekten
   Definition von Eigenschaften
   Variable und Datentypen
   Datenfelder (Arrays)
   Definition von Verhaltensweisen
   Aufruf von Methoden
   Klassenmethoden
   Methoden Overloading
   Methoden Overridung
   Konstruktor-Methoden
   Finalizer-Methoden
   Ausnahmen

2. Vererbung, Schnittstellen, Schutzebenen und Pakete

   Vererbung
   Zugriffskontrolle durch Schutzebenen
   Pakete
   Schnittstellen

3. Java-Klassenbibliothek

   Das Paket java.lang

   Grundlegende Methoden aus dem java.io-Paket

   Das java.awt-Paket
     Komponenten
     Ereignisse
     Graphik und Animation
     Bildverarbeitung

4. Objektorientierte Wissensrepräsentation im Internet

4.1 Kommunikations- bzw. Dokumentationsnetze

4.1.1 Vernetztes Wissen

Ausgangssituation

Unbestreitbar ist: „Eine Wissensexplosion hat die Menschheit mittlerweise voll erfasst“. Die dezentrale Welt verlangt eine intensive Kommunikation des Wissens. Auskunftsysteme müssen die Wege dafür öffnen. Die Vernetzung des Wissens und der Funktionen in der dezentralen Welt verlangt ein neues Denken in der Wissensbereitstellung. Die neuen Methoden und Techniken der grafischen Oberflächen und des Internet spielen den Vorreiter, die Dokumentation als Spiegel des Wissens ist dabei ein wesentlicher Bestandteil.
Ziel ist die Bereitstellung von vernetztem Wissen in einem Wissenspool. In Ansätzen ist dies heute im Internet durch die letzten Entwicklungen (`HTML`, `Java`, `XML`) sichtbar. Die Implementierung erfolgt vorzugsweise als Rechnernetz, das Online-Abfrage und -Pflege erlaubt (z.B. im World Wide Web), denn bei der gegenwärtigen Vernetzung des Wissens ist eine papiergestützte Dokumentation nicht mehr beherrschbar. Das gilt sowohl für die Pflege als auch die gezielte Suche in einer immer wachsenden Informationsflut. Aktuell erforderlich ist der Aufbau einer Online-Wissensvermittlung, die mehr sein muß als ein Handbuch auf dem Bildschirm. Es bietet sich der Aufbau eines internen Kommunikatrions- und Dokumentationsnetztes ("Intranet") an, das auf kompatiblem Internet-Standards (HTML, Java, XML) basiert.

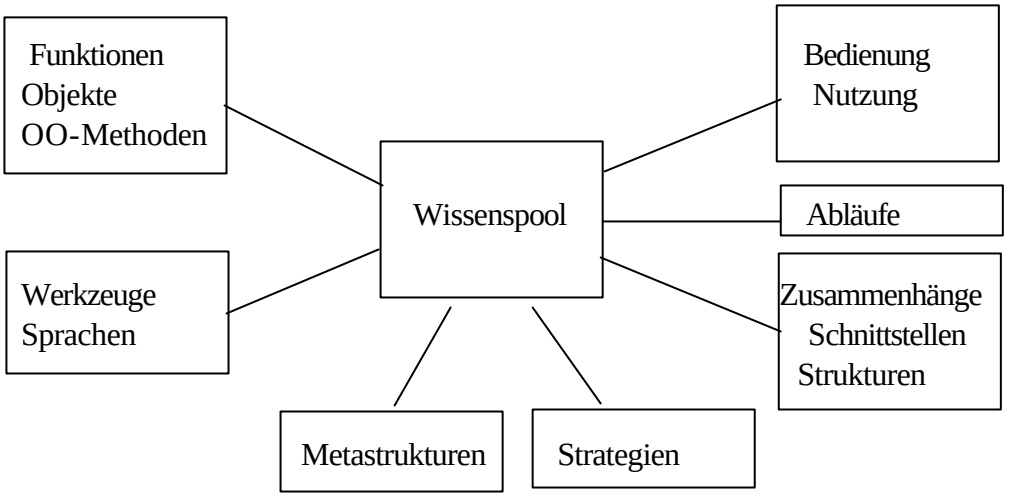


Abb.: Zusammenstellung von vernetztem Wissen in einem Wissenspool

Voraussetzung: Objektorientierung

In einer objektorientierten Welt muß die Dokumentation objektorientierten Prämissen genügen. Anforderungen an Objekte können auf Objekte vom Wissenspool übertragen werden. Insbesondere sind Verfügbarkeit, Verläßlichkeit und Aktualität gefordert,

Einstiegpunkte, Navigationswege

Der Anwender erwartet von einem System, das ihm verläßliches Wissen bereitstellen soll, Verfügbarkeit in allen Situationen. Zum Erreichen dieses Ziels werden verschiedene Einstiegpunkte und Navigationswege benötigt:

Baumstruktur
Die Fülle der Dokumente wird in einer Baumstruktur angeordnet.
Schlagworte
Die Suche über Schlagworte wird für Recherche benötigt.
Fragenkatalog
Fragenkataloge behandeln allgemein interessierende Fragen, Probleme und Lösungswege. Sie vermitteln die Erfahrungen anderer Anwender und unterstützen so die laufende Arbeit.
Querverweise
Sie stellen die Verbindung zwischen verschiedenen Dokumenten her und müssen die Navigation durch einfaches "Anklicken" gewährleisten.

Indizierung

Die schwierigste Arbeit bei der Erstellung der Dokumentation ist die Indizierung durch Schlagworte. Beim Aufbau eines übergreifenden Wissenspools müssen Auswahl und Formulierung der Schlagworte den Anforderungen aller Anwender genügen. Hierfür sind Verfahren und Modelle zur Indizierung zu entwerfen, die allen Beteiligten bekannt sein müssen. Ein weiteres Problem ist die Menge der Dokumente, die in einen Wissenspool einfließen. Die gut aufbereiteten und vollständigen Indexe für die Dokumente können dabei nicht additiv in einen Gesamtindex einfließen. Für jedes Dokument muß eine Untermenge ausgewählt werden, die den Inhalt für den Gesamtindex repräsentiert.

Werkzeuge für den Wissenspool

Das Ergebnis einer gelungenen Wissensbereitstellung muß eine Mischung aus Dokumentation, Nachschlagwerk, Werbebroschüre und Lernprogramm sein. Für diese Aufgabe gibt es kein umfassendes, standardisiertes Werkzeug. Vorhandene Dokumentationswerkzeuge (etwa für die objektorientierte Analyse, für objektorientiertes Design, für Datenwörterbucher (`data dictionaries`) oder Object Browser) decken jeweils nur einen speziellen Teil des Geschehens ab. Sie können evtl. (nach Umformen ihrere Ausgaben) als Zulieferer für den Wissenspool genutzt werden.

Herkömmliche grafische Textsysteme bieten sich als Entwurfs- und Erfassungswerkzeuge an, wenn sie

- die Bildung eines Strukturbaums für die Anordnung der Dokumente
- die Trennung der Schlagworte nach einem Gesmat- und Detailindex
- das Trennen des Inhaltsverzeichnisses als gesondert markiertes Element des Strukturbaums mit automatischer Querverbindung
- das Trennen der Querverweise aus dem Text

ermöglichen.
Vorliegende Hilfesysteme (z.B. die des Office Packets) können als Modell für die Vorgehensweise dienen, wenn eine Weiterentwicklung für die Informationswege eines übergreifenden Wissenspools nötig ist.

4.1.2 Internet, WWW mit HTML, XML

Internet-Dienste ermöglichen die Kommunikation mit anderen Internet-Teilnehmern, die Nutzung von Informationsressourcen im Internet und das Anbieten von Informationen über das Internet. Der Internet-Dienst, dem das Internet das exponentielle Wachstum verdankt, ist das World Wide Web (WWW).
Die Idee des **World Wide Web** (WWW oder einfach nur Web) ist Anfang 1989 am CERN[20] entstanden. Auslöser waren Schwierigkeiten beim Auffinden relevanter Informationen im Internet in Hard- und Software. Zur Lösung des Problems wurde ein auf der Client-/Server-Architektur aufbauendes, hypertext-basiertes System vorgeschlagen. Hypertext wird dabei als Weg verstanden, Informationen derart miteinander zu verknüpfen, daß sich der Benutzer nach eigenem Belieben durch die Informationen bewegen kann. Die Verknüpfungen erscheinen dabei als Knoten in einem verwobenen Netz.

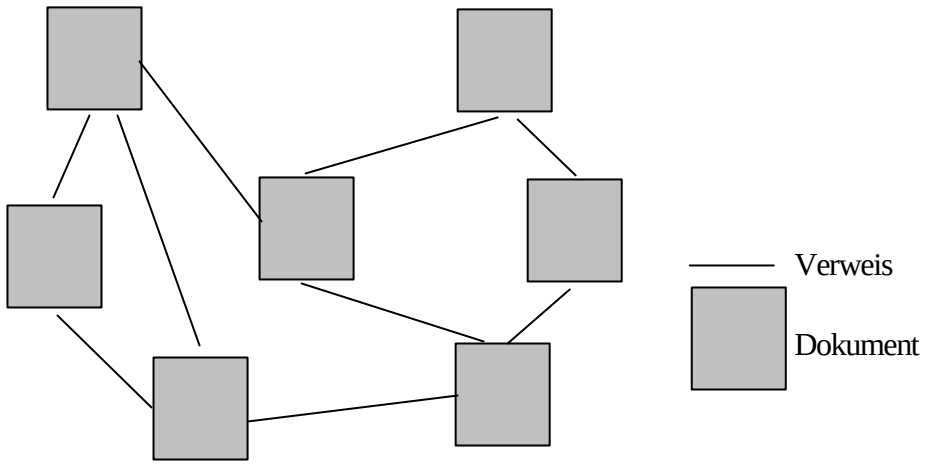


Abb.: Struktur des World Wide Web

Eine verteilte Hypermedia-Anwendung kann als Netz angesehen werden, dessen Knoten Text, Grafik, Ton darstellen. Verbindungen (Links) kennzeichnen die Beziehungen zwischen den Knoten

[20] europäisches Zentrum für Teilchenphysik bei Genf

und werden von der Systemsoftware verwaltet. Die möglichen Verbindungen werden im Dokument (Fenster) an der entsprechenden Stelle gekennzeichnet. Der Benutzer kann durch Anklicken derartiger Markierungen im Dokument oder durch Weiterblättern bzw. Rücksprung zu einem früheren Knoten fortfahren, solange er will.
Dokumente werden in einem speziellen Format, dem Hypertext-Format gespeichert. Zum Erstellen von Dokumenten steht die Seitenbeschreibungssprache HTML (Hypertext Markup Language) zur Verfügung[21].
Grundlage der Webtechnik ist die Sprache HTML, die aus SGML (Standard Generalized Markup Language) abgeleitet wurde. SGML bietet ein hochkomplexes System für die Auszeichnung von Dolumenten. Bei der Entwicklung des WWW brauchte man ein einfaches Auszeichnungs-System, bei dem praktisch jeder schnell zur Könnerschaft gelangen konnte. Man entschloß sich, die neue Auszeichnungssprache an SGML zu orientieren. So entstand die HyperText Markup Language (HTML). HTML ist ein spezieller Dokumenttyp von SGML.
HTML-Dateien sind reine Textdadteien. Steueranweisungen, sog. Tags werden in spitzen Klammern eingeschlossen. HTML beschreibt die Struktur des Texts, z.B. „dies ist eine Überschrift“ oder „hier beginnt ein Absatz“. Neben Tags zur Strukturierung des Texts gibt es auch noch „Image-Tags“ zur ansprechenden Ausstattung der Web-Seiten mit Bildern. Die wichtigsten Tags sind Hyperlinks. Darüber kann man auf Dokumente verweisen, die irgendein anderer Rechner im Internet bereitstellt. Diese Verweise bilden ein weltumspannendes Netz, das WWW.
Das **Hypertext Transfer Protocol** (HTTP) dient zur Übertragung von Informationen aus dem World Wide Web (WWW). HTTP ist ein objektorientiertes Protokoll zur einfachen Übertragung von Hypertext-Dokumenten zwischen Client und Server.
Client-Programme, die HTTP benutzen, werden (in der Regel) als **Web-Browser**, Server-Programme **als Web-Server** bezeichnet. Der Browser schickt an den Server die Aufforderung eine bestimmte HTML-Seite zu übertragen. Falls er in dieser Seite weitere Verweise (z.B. auf Bilder) entdeckt, schickt er weitere Übertragungswünsche hinterher. Das Besorgen der gewünschten Dokumente erfolgt über ein einheitliches Adreßschema, dem Uniform Resource Locator (URL), durch den Internet-Standort und die Art der zu übertragenden Information identifiziert werden. Der URL besteht im wesentlichen aus zwei Teilen:

- Er wird mit der Bezeichnung des Übettragungsprotokolls eingeleitet, z.B.: „http“.
- Danach folgt (mit zwei vorangestellten //) der Name des Internet-Rechners, auf dem die gewünschte Information gespeichert ist. Wird eine bestimmte Datei in einem bestimmten Verzeichnis referenziert, dann werden noch Verzeichnisname und Dateiname angefügt.

Es gibt Strömungen und Tendenzen, die dazu führen, daß Ende der Neunziger Jahre Standards entstehen, die über die Beschränkungen von HTML hinausgehen. Die hohe Komplexität von SGML verhinderte, daß SGML zum Standard im Web wurde. Beim World Wide Web Consortium (W3C)[22] wurde daher eine Arbeitsgruppe eingerichtet, die sich die Aufgabe gestellt hat, eine vereinfachte Varianle von SGML zu schaffen. Das Arbeitsergebnis dieser Gruppe ist **XML**. XML ist ebenso wie SGML eine Metasprache für das Definieren von Dokument-Typen, d.h. XML ist der Oberbegriff für die Regeln (die Syntax) , die angewendet werden, wenn ein neuer Dokument-Typ definiert wird. Das zukünftige HTML hat XML als Grundlage.
Grundsätzlich können XML-Datentypen überall da zum Einsatz kommen, wo es Bedarf für Datentausch gibt

[21] bezieht sich auf die in ISO 1986 standardisierte SGML (Standard Generalized Markup Language).
[22] Standardisierungsorganisation für das Web.

4.1.3 HTML-Grundlagen

HTML-Dokumente sind einfache Textdateien, die mit jedem beliebigen Texteditor erstellt werden können. Damit sie von Web-Browsern optisch ansprechend dargestellt werden können, enthalten sie besondere Befehle, die sog. Tags (oder Marken). Sie beginnen mit einer öffnenden spitzen Klammer < und enden mit einer schließenden spitzen Klammer >. Dazwischen stehen der name der Marke und oft noch zusätzliche Optionen. Viele Marken haben Blockcharakter, d.h.: Sie markieren einen Textbereich wie Überschriften, Titel, spezielle Absätze. Solche Blöcke werden mit einer Endemarke abgeschlossen. Sie sehen, bis auf den Schrägstrich als erstes Zeichen, genau so aus wie die Marke selbst.
Bsp.:
<HTML> und </HTML>
Diese Marken kennzeichnen die datei als HTML-Seite
<HEAD> und </HEAD>
<TITLE> und </TITLE>
Diese Marken stehen im Head-Teil eines Dokuments
<BODY> und </BODY>
<H1> und </H1>
<P> und </P>
<!-- und -->
Mit diesen Marken werden Kommentare eingeschlosen. Sie dienen der Dokumentation und werden vom Browser nicht ausgewertet.
&Uuml
Umlaute und Sonderzeichen müssen besonders definiert werden. Dies geschieht über sog. Escapesequenzen, die mit dem Zeichen „&“ beginnen und mit einem Semikolon enden.

4.1.4 Arbeitsweise eines Web-Servers

1. Das Hypertext Transfer Protocol (http)

„**http**“ definiert eine Kommunikation zwischen Client und einem Server. Typischerweise horcht der Server auf Port 80 (oder 8080) auf Anfragen des Clients. Das Protokoll nutzt eine TCP/IP-Verbindung und ist textbasiert.
Alle HTTP-Anfragen haben folgendes Format:

- Eine Zeile am Anfang
- Einige Kopf-Zeilen
- Einen Körper (Body)

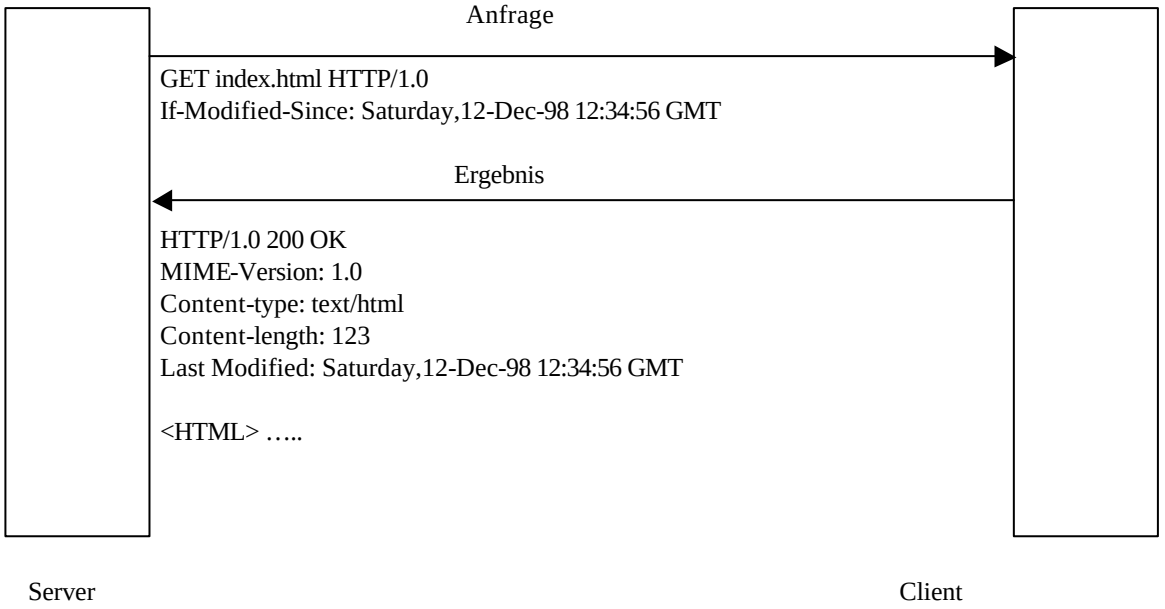


Abb.: Datentransfer zwischen Client und Server

2. Anfragen an den Server

Ist die Verbindung aufgebaut, wird eine Anfrage formuliert, auf welches Objekt (Dokument oder Programm) zugegriffen werden soll. Neben der Anfrage wird auch noch das Protokoll festgelegt, mit dem übertragen wird. HTTP 1.0 (und ebenso HTTP 1.1) definiert mehrere Hauptmethoden. Die 3 wichtigsten Methoden sind:

GET: Ist eine Anfrage auf eine Information, die an einer bestimmten Stelle lokaliserbar ist. Anfrage für eine normale Webseite, die etwa durch Verfolgen eines Links oder Eintrag in der Browserzeile für die URL gestartet wird. GET wird auch manchmal verwendet, wenn Daten über Formulare an den server geschickt werden. In diesem Fall werden die Daten kodiert und an eine Formular-URL angehängt (maximale Größe: etwa 2 K).
POST: Sie elaubt es dem Client, Daten zum Server zu schicken. POST wird vom Browser nur für Formulare innerhalb von <FORM> mit der Anweisung METHOD="POST" verwendet. Es gibt keine Begrenzung des Informationsumfangs.
HEAD: Funtioniert ähnlich wie GET. Nicht das ganze Dokument wird verschickt sondern nur Infos über das Objekt. So sendet er z.B. innerhalb einer HTML-Seite die Informationen, die innerhalb von <HEAD> ... </HEAD> stehen.

Eine Beispielanfrage an einen Web-Server: GET/*directory*/index.html HTTP/1.0
Der erste Parameter ist die Methode des Aufrufs[23]. Der zweite Parameter ist der Dateipfad (relative Pfadangabe). Die Anfrage endet mit einer Zeile, die nur aus Carriage-Return und Linefeed besteht. Nach der Zeile mit der Anfrage können optionale Zeilen gesendet werden.

3. Die Antwort vom Server

[23] Auch Anfrage (engl. request) genannt.

Der Server antwortet mit Statuszeile, einem Header (mit Infos über sich selbst) und dem Inhalt. Der Web-Browser muß sich also um eine Antwort vom Web-Server kümmern.
Eine vom Microsoft WEB-Server generierte Anwort kann etwa so aussehen:

```
HTTP/1.0 200 OK
Server: Micosoft-PWS/2.0
Date: Sat, 09 May 1998 09:52:32 GMT
Content-Typ: text/html
Accept-Ranges: bytes
Last-Modified: Sat, 09 May 1998 09:52:22 GMT
Content-Length: 27

Hier kommt die HTML-Seite
```

Die Antwort ist wieder durch eine Leerzeile getrennt. Der Header vom HTTP setzt sich aus drei Teilen zusammen: dem General-Header (dazu gehört etwa Date), Response-Header (dazu gehört Server) und Entity-Header (Content-Length und Content-Type). Jedes Feld im Header bestheht aus einem Namen gefolgt von einem Doppelpunkt. Dann folgt der Wert.
Die erste Zeile wird Statuszeile genannt und beinhaltet die Version des Protokolls und den Statuscode. Der Stauscode macht eine Aussage über das Ergebnis der Anfrage. Er besteht aus einer Zahl mit 3 Ziffern.
General Header Fields: Zu jeder übermittelten Nachricht gibt es abfragbare Felder (sowohl für den Client als auch für den Server). Zu diesen gehören: Cache-Control, Connection, Date, Pragma, Tranfer-Encoding, Upgrade und Via. Das Datum kann in 3 verschieden Formaten gesendet werden.
Felder im Response Header: Der Response-Header erlaubt dem server die Übermittlung zusätzlicher Infos, die nicht in der Statuszeile kodiert sind. Die Felder geben Auskunft über den Server. Möglich sind: Age, Location, Proxy-Authenticate, Retry-After, Server, Vary, Warning, WWW-Authenticate.
Entity Header Fields: Ein Entity ist eine Info, die auf Grund einer Abfrage gesendet wird. Das Entity besteht aus Metainformationen (Entity-Header) und der Nachricht selbst (überliefert im Entity-Body). Die Metainformationen, die in einem der Entity Header Felder übermittelt werden, sind etwa Infos über die Blocklänge und wann sie zuletzt geändert wurde. Ist kein Entity-Body definiert, so liefern die Felder Infos über die Ressourcen, ohne sie wirklich im Entity-Body zu senden: Allow, Content-Base, Content-Encoding, Content-Language, Content-Length, Content-Location, Content-MD5, Content-Range, Content-Type, ETag, Expires, Last-Modified.
Der Dateiinhalt und der Content-Type: HTTP nutzt die Intermedia-Types im Content-Type. Dieser Content-Type gibt den Datentyp des übermittelten Dateistroms an. Jede über HTTP/1.1 übermittelte Nachricht sollte einen Header mit Content-Type enthalten. Ist dieser Typ nicht gegeben, so versucht der Client anhand der Endung der URL oder durch Betrachtung des Dateistroms herauszufinden, um was für einen Typ es sich handelt. Bleibt dies unklar, dann wird ihm der Typ „application/octet-stream“ zugewiesen.
MIME?: Wenn eine Seite vom Server zum Browser geschickt wird, dann erkennt der Anfrager den Typ der verschickten Datei und kann sie passend anzeigen. Der Hinweis auf den Datentyp kommt vom Server mit einer Mitteilung, die sich MIME[24] nennt. Der Server kann den Datentyp feststellen, in dem er in die Datei schaut, die ersten Bytes untersucht und daraus ein Prognose erstellt. Er kann aber auch einfach über die Dateiendung gehen. Die Lösung mit Dateiendungen wird oft bei Implementierungen von Webservern angewendet. Der Client muß anhand des MIME-Typs das

[24] MIME (Multipurpose Internet Mail Extension) wurde ursprünglich für E-Mail zum Versenden binärer Daten entwickelt

passende Anzeigeprogramm finden. In der Regel kann der Browser die Datei direkt anzeigen, wenn es etwa eine HTML-, Textseite oder Grafik ist[25]. Jeder MIME/Typ besteht aus 2 Hälften: einer Kategorie und einem Datentyp. Wichtige Kategorien sind: text, audio, image, video, application.
Bsp.:

text/html. Das ist der Standardtyp für Webseiten.
text/plain. Textdatei, die nicht vom Browser interpretiert wird.
image. Die Dateitypen für Grafikformate, z.B. image/gif, image/jpeg für GIF und JPEG
audio. Sound-Dateien, z.B. audio/basic oder audio/x-wav
video. Darstellbare Bewegbilder wie video/mgep oder video/quicktime
application. Dazu zählen in der Regel Binärdateien, die von anderen Programmen gelesen oder ausgeführt werden, etwa application/pdf für eine PDF-Datei, application/msword für ein Microsoft Word Dokument oder application/vnd.ms -exel für ein Excel Dokument.

[25] Sonst greift meistens ein Active-X-Control (Microsoft) oder ein Plugin (Netscape) ein.

4.2 Client/Server und Internet/Intranet

4.2.1 Grundlagen

Grundlage der Client-/Server-Architektur ist eine Verteilung zwischen verschiedenen Systemen und Systemteilen. Die Internet-Technologie[26] ist eine Hardware- und Systemstruktur, kombiniert mit einheitlichen Protokollen und Präsentationsformaten. Sie repräsentiert eine geschickte Art der Praktizierung von Client-/Server-Architekturen.

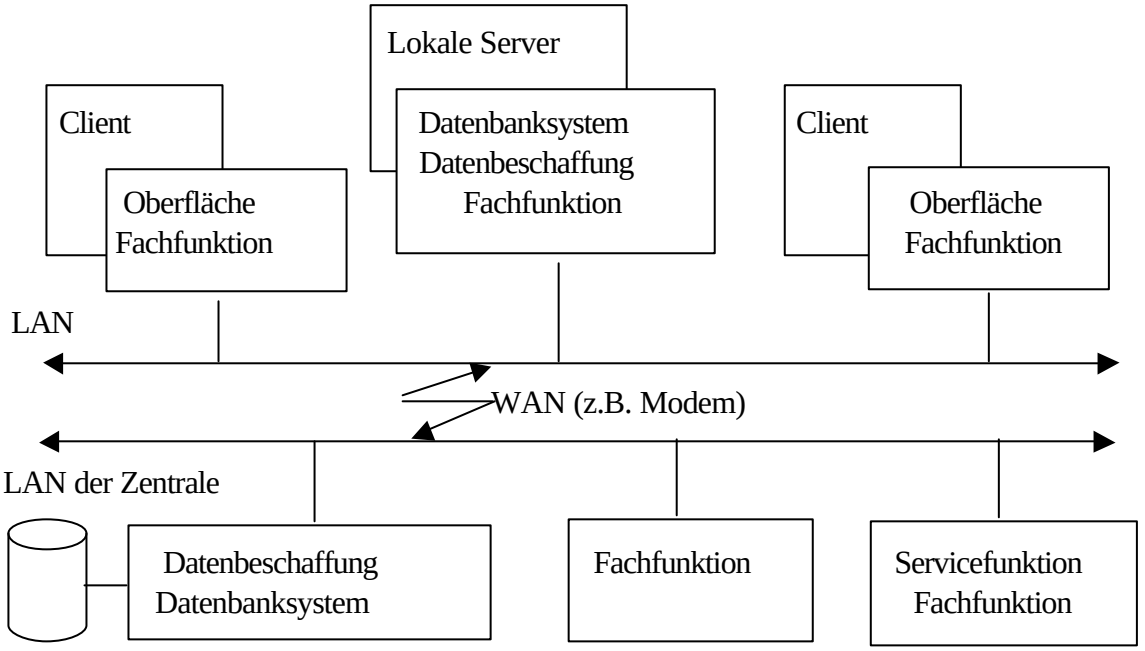


Abb. : Aufgabenverteilung auf verschiedenen Schichten (Präsentation, Funktionen, Datenbeschaffung / Datenbank, Services) des Client-/Server-Modells

Die Abb. zeigt die Verteilung der Aufgaben auf verschiedene Ebenen:
Die Präsentationsebene übernimmt die Darstellung der Information gegenüber dem Benutzer(klassische Frontend-Bearbeitung). Die Funktionsebene umfaßt die fachlichen Funktionen, unabhängig von Darstellung und Speicherung. Die Datenbeschaffungsebene umfaßt die Speicherung der Daten auf den erforderlichen Medien mit den entsprechenden Systemen. Die Service-Ebene stellt Werte und Funktionen in allgemeiner Form bereit, die systembezogen (oder umgebungsbezogen)[27] sind.
Zur Nutzung des Internet steht eine Vielzahl von Anwendungen zur Verfügung, die als Internet-**Dienste** bezeichnet werden. Die Dienste[28] stützen sich auf ein Client-/Server-Modell ab. Bekannte Dienste im Internet sind: Telnet, File Transfer Protocol (FTP), Usenet News, World Wide Web (WWW).

[26] firmenintern als Intranet genutzt

[27] Das fängt beim Tagesdatum an und reicht zu so komplexen Funktionen wie bspw. Zeitsynchronisierung innerhalb eines komplexen Netzes oder der Nachrichtenvermittlung zwischen Objekten (Object Request Broker.

[28] Für eine Vielzahl von Diensten sind Standqardisierungen in sog. RFCs (Request for Comments) niedergelegt

Alle Internet-Dienste stützen sich auf das Client-/Server-Konzept. Das Client-Programm (Client) stellt die Schnittstelle zwischen Benutzer und Server-Programm (Server) dar. Der Server bietet Informations- und Kommunikationsvermittlung an. Aufgabe des Client ist es, Anfragen des Benutzers in eine maschinenverständliche Art „umzuformulieren“ und dem Benutzer die vom Server gelieferte Antwort zu präsentieren. Für die Benutzung eines Internet-Dienstes ist ein Client- und ein Server-Programm nötig. Der Client übernimmt Vorfeldaufgaben (front-end application) der Informationsverarbeitung und erlaubt unterschiedliche Datenquellen unter einer Oberfläche zu integrieren.

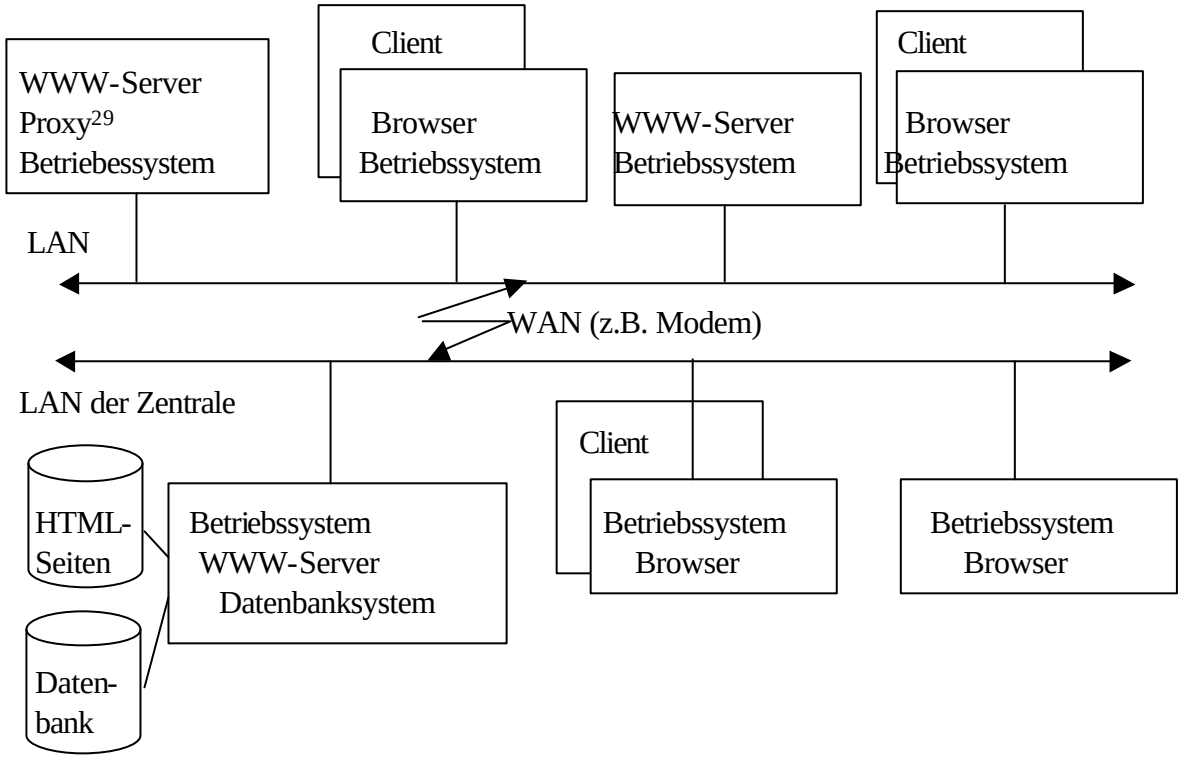


Abb.: Intranet-Architektur

Bei der Client-/Server-Kommuniktaion über das Internet benötigt man eine Server, der ständig in Bereitschaft ist und auf Anfragen von einem Client wartet. Der Client baut eine Verbindung auf, indem er eine Anfrage an den Server richtet und dieser die Verbindung aufnimmt. Das einfachste Bsp. hierfür ist ein **Web-Server**, der auf einem Host-Rechner installiert ist. Der **Web-Browser** fungiert dabei als Client, der eine Verbindung zu einem bestimmten Web-Server für den Datenaustausch herstellt. In erster Linie wird der Client `HTML`-Dokumente vom Web-Server empfangen.
Viele Web-Knoten legen jede Web-Seite als eigene `HTML`-Datei ab. Solche Seiten sind allerdings statisch, d.h. sie erzeugen bei jedem Aufruf denselben Inhalt. In vielen Unternehmen steigt aber das Interesse, vorhandene, in Datenbanksytemen gehaltene Dokumente und Informationen für Web-Clients verfügbar zu machen. Die Anbindung einer Datenbank an einen Web-Server erleichtert die Aktualisierung der Inhalte. Allerdings müssen dazu die Daten mediengerecht aufbereitet, d.h. in

[29] Proxy-Server: Das ist ein zusätzliches lokales Datenbeschaffungsinstrument für einen bestimmten anwenderkreis (Clients des oberen LAN). Die Clients dieses Anwenderkreises gehen nicht direkt ans Netz, sondern beauftragen ihren Proxy über die schnelle lokale Verbindung, eine oder mehrere Informationsseiten zu beschaffen. Der Proxy holt sich die Seite, speichert sie in einem lokalen Cache-Speicher und liefert sie an den lokal anfordernden Client aus.

HTML, der Seitenbeschreibungssprache des Web verpackt werden. Das geschieht mit Hilfe von Programmen, die Daten aus der Datenbank auslesen und aus den so gewonnenen Informationen HTML-Dokumente erzeugen. Die zugehörigen Programme laufen auf dem Web-Server, von dem auch die statischen Dokumente geladen werden.

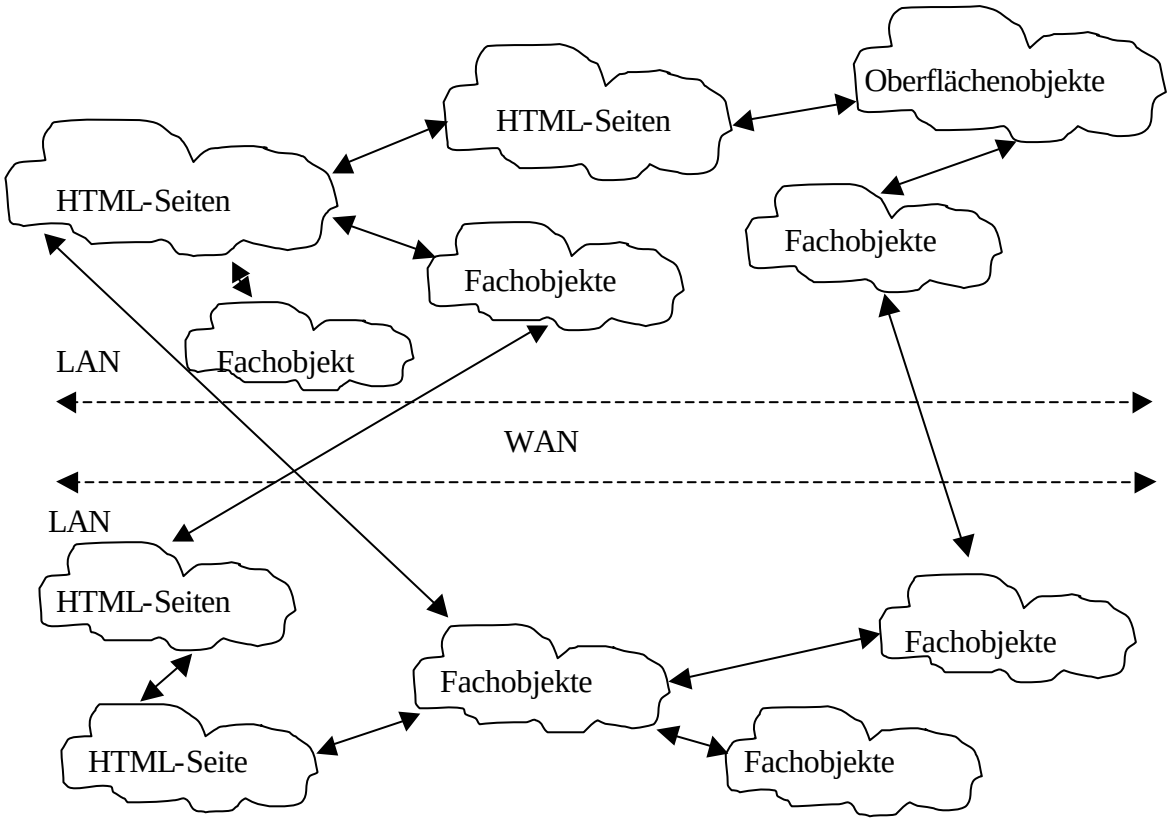
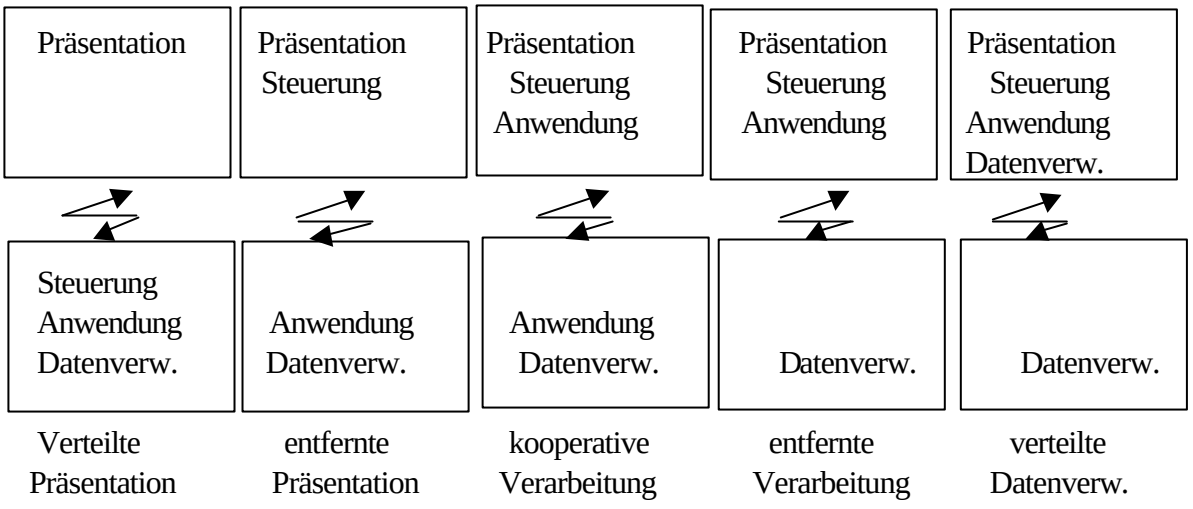


Abb.: Aufbau und Kommunikation der Objekte im Internet

4.2.2 Grundkonzepte der Verteilung von Anwendungen

Typischerweise werden Applikationen in 3 Aufgabenbereiche unterteilt: Datenhaltung, Anwendung(slogik) und Präsentation[30]. Diese 3 Aufgabenbereiche müssen nicht auf einem Rechner oder von einem Prozeß wahrgenommen werden, sondern können auf mehrere Rechner bzw. Prozesse verteilt sein.



[30] Vgl. Client-Server-Architekturen

Abb.: Verteilungsmöglichkeiten im Client-/Server-Modell

Bei der entfernten Repräsentation Common Gateway Interface (CGI) findet die Verarbeitung „Server"-seitig statt, auf dem Client werden nur Repräsentationsaufgaben ausgeführt.

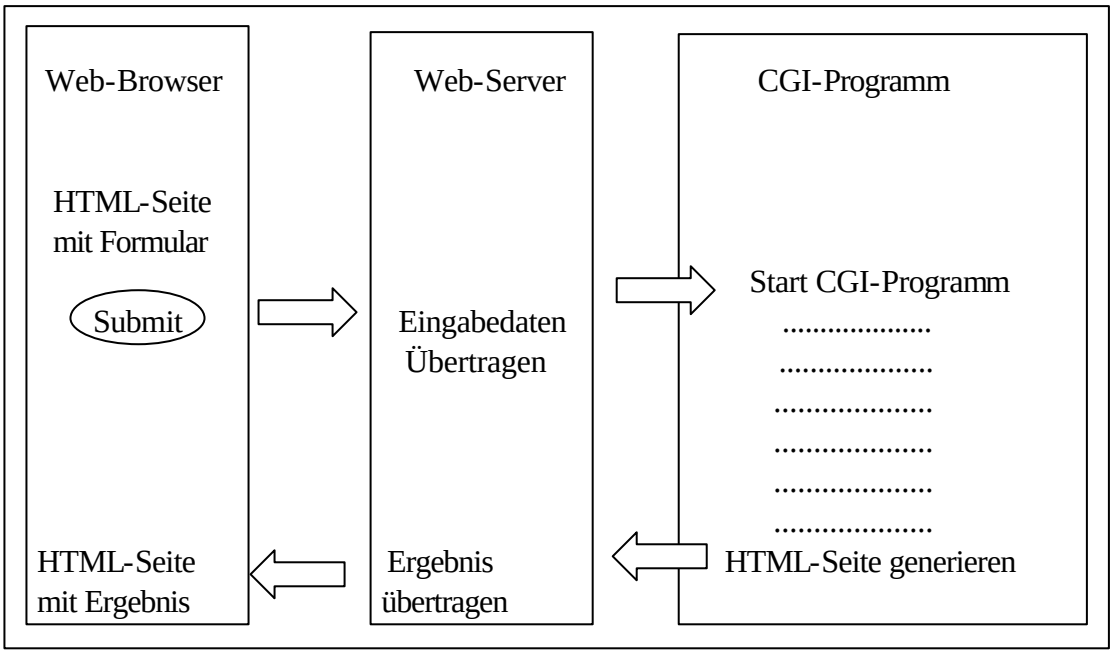


Abb.: Kommunikation bei CGI

CGI stellt eine Schnittstelle zu einem Server dar. Diese Schnittstelle erweitert die Funktionalität eines Server. Damit verfügt die Web-Server Software über eine Schnittstelle, die es erlaubt, HTML-Dokumente dynamisch durch Programme erzeugen zu lassen und die aus der Datenbank erzeugten Daten direkt an den Client zu übergeben. Die CGI-Programme werden von einem Browser durch eine ganz normale URL angesprochen. Der Browser baut eine Verbindung zum Server auf. Dieser erkennt anhand des Pfades in der URL, ob es sich um eine ganz normale Web-Seite handelt oder um ein Skript. Falls es ein Skript ist, dann führt der Server das Skript aus, das eine HTML-Datei erzeugt. Diese wird übertragen und im Browser dargestellt

Bei der verteilten Datenverwaltung werden die Aufgaben zwischen Client und Server aufgeteilt. Beispiele sind Entwicklungsumgebungen, die sowohl auf lokale als auch entfernte Datenbanken zugreifen können. Die verteilte Datenverwaltung geht tief in Theorie und Konzept verteilter Datenbanken ein.

Bei der kooperativen Verarbeitung findet die Datenhaltung Server-seitig statt. Repräsentationsaufgaben werden vom Client wahrgenommen und die Applikations-Funktionsschicht wird zwischen Server und Client aufgeteilt.

Bei der entfernten Verarbeitung wird die Datenhaltung vom Server wahrgenommen. Die Verarbeitung findet „Client"-lastig statt, d.h. der Client übernimmt die Anwendungs- und Repräsentationsaufgaben. Die Anwendungslogik kann somit vollkommen auf der Seite vom Browser, d.h. vom Client ausgeführt werden. Ein solches Programm, das innerhalb vom Browser ausgeführt wird, ist ein **Applet**. Ein besondere Rolle spielt hier die Programmiersprache Java. In Java geschriebene Programme können in einen maschinenunabhängigen Zwischencode übersetzt und Server-seitig abgelegt werden. Wenn eine HTML-Seite auf ein Java-Programm, ein sog. Applet

verweist, dann wird es zum Client übertragen und dort mit einem entsprechenden Interpreter ausgeführt. So steht nichts mehr im Wege, auf dem Client Animationen ablaufen zu lassen, Eingabefelder zu überprüfen oder auch Datenbank-Anweisungen (SQL-) aufzurufen.
Eine besondere Rolle spielt hier die Programmiersprache **Java**. In Java geschriebene Programme können in einen maschinenunabhängigen Zwischencode übersetzt und "Server"-seitig abgelegt werden. Wenn eine HTML-Seite auf ein Java-Programm, ein sog. Applet verweist, dann wird es ggf. zum Client übertragen und dort mit einem entsprechenden Interpreter ausgeführt.
Einen geeigneten Satz von Funktionen für den Datenbankzugriff unter Java enthält die **JDBC** (Java Database Connectivity)-Schnittstelle. JDBC ist die Spezifikation einer Schnittstelle zwischen Client-Anwendung und SQL.
JDBC identifiziert auch eine Datenbank anhand eines URL, z.B.: jdbc:oracle:oci7:@rfhs8012_ora3.

4.3 Die Verbindung zur Datenbank über JDBC

4.3.1 Zugriff auf eine relationale Datenbank mit JDBC

JDBC steht für `Java Database Connectivity` und umfaßt zur Interaktion mit Datenquellen relationale Datenbankobjekte[31] und Methoden. Die `JDBC`-APIs sind Teil der Enterprise APIs, die von JavaSoft spezifiziert wurden und somit Bestandteil der Java Virtual Machine (`JVM`) sind. JDBC-Designer haben die API auf das X/Open SQL-Call Level Interface (CLI) aufgebaut. Auch ODBC basiert auf X/Open SQL-CLI. JDBC definiert API-Objekte und Methoden zur Kommunikation mit einer Datenbank. Ein Java-Programm baut zunächst eine Verbindung zur Datenbank auf, erstellt ein Statement-Objekt, gibt SQL-Statements an das zugrundeliegende Datenbankverwaltungssystem (DBMS) über das Statement-Objekt weiter und holt sich Ergebnisse und auch Informationen über die Resultat-Datensätze. JDBC-Dateien und Java-Anwendung / Applet bleiben beim Client, können aber auch vom Netzwerk heruntergeladen werden. Das DBMS und die Datenquellen liegen auf einem Remote Server. Die JDBC-Klassen befinden sich im `java.sql`-Paket. Alle Java-Programme verwenden zum Lesen und Schreiben von Datenquellen Objekte und Methoden des `java.sql`-Pakets. Ein Programm, das JDBC verwendet, benötigt einen Treiber für die Datenquelle. Es ist die Schnittstelle für das Programm. JDBC besitzt den `DriverManager` zur Verwaltung der Treiber und zum Erstellen einer Liste der in den Anwendungsprogrammen geladenen Treiber. `JDBC-ODBC-Bridge` ist als `JdbcOdbc.class` implementiert und eine native Bibliothek für den Zugriff auf den ODBC-Treiber. Zuerst bildet dieser Treiber JDBC-Methoden auf ODBC-Aufrufe und tritt somit mit jedem verfügbaren ODBC-Treiber in Interaktion.
`JDBC` ist als `java.sql`-Paket implementiert. Dieses Paket enthält alle JDBC-Klassen und Methoden. Klassen im `java.sql`-Paket sind:

`java.sql.Driver`, `java.sql.DriverManager`, `java.sql..PropertyInfo`,
`java.sql.Connection`, `java.sql.Statement`, `java.sql.PrepareStatement`,
`java.sql.CallableStatement`, `java.sql.ResultSet`, `java.sql.SQLException`,
`java.sql.SQLWarning`, `java.sql.Database.MetaData`, `java.sql.ResultSetMetaData`,
`java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`,
`java.sql.Numeric`, `java.sql.Types`, `java.sql.DataTruncation`.

Dem Programmierer gibt JDBC Funktionen für den Aufbau von Verbindungen, zum Lesen oder Aufbau von Datensätzen zur Datenbank. Zusätzlich können Tabellen aktualisiert und Prozeduren auf der Server-Seite ausgeführt werden. Die folgenden Schritte sind für den Zugriff auf eine relationale Datenbank mit JDBC nötig:

1. Installieren der JDBC-Datenbank-Treiber
2. Ein e Verbindung zur Datenbank über den entsprechenden JDBC-Treiber
3. Eine SQL-Anweisung erzeugen
4. Ausführen der SQL-Anweisung
5. Das Ergbenis der Anweisung holen
6. Schließen der Datenbankverbindung

[31] Für objektorientierte DB liegt noch kein fertiges Konzept vor. Mit JDBC-2 Treibern ist jedoch der Übergang zum SQL3-Standard geschaffen, in dem objektorientierte Konzepte eine größere Rolle spielen

Das bearbeiten von Daten einer Datenbank durch ein Java-Programm umfasst im Rahmen der JDBC-Schnittstelle:

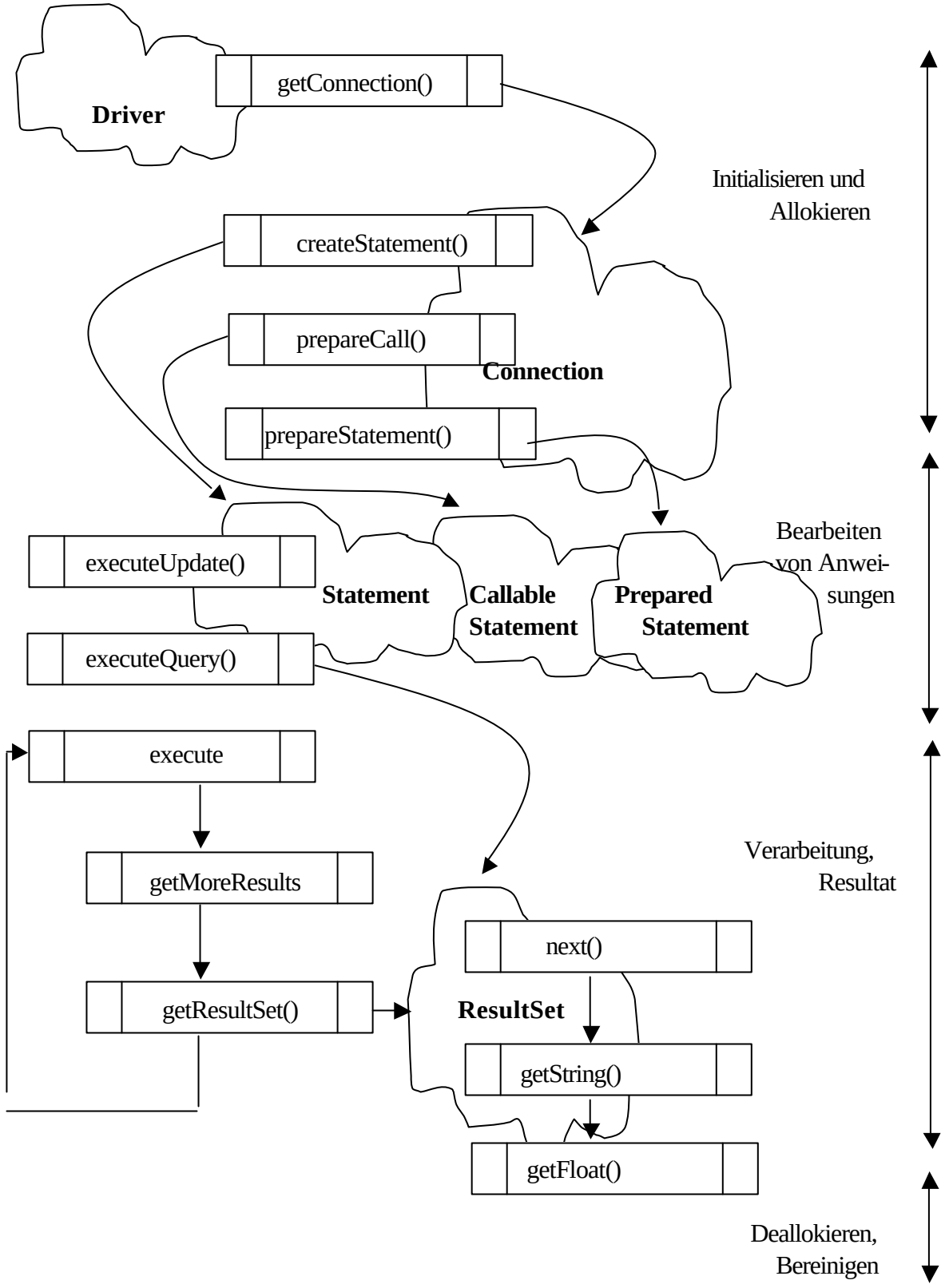


Abb.: JDBC-Objekte, Methoden, Ablauf

Zuerst ruft das Java-Programm die `getConnection()`-Methode auf, um das Connection-Objekt zu erstellen. Dann ergänzt es das Statement-Objekt und bereitet die SQL-Anweisungen vor.

Ein SQL-Statement kann sofort ausgeführt werden (Statement-Objekt) oder ein kompilierbares Statement (Prepared-Statement-Objekt) oder ein Aufruf an eine gespeicherte Prozedur (Callable-Statement-Objekt) sein. Falls die `executeQuery()`-Methode ausgeführt wird, wird ein Resultat-Objekt zurückgeschickt. `SQL`-Anweisungen wie `update` und `delete` verwenden die `executeUpdate()`-Methode. Sie gibt einen ganzzahligen Wert zurück, der die Anzahl der Zeilen anzeigt, die vom SQL-Statement betroffen sind.
Der `ResultSet` enthält Zeilen mit Daten, die mit der Methode `next()` abgearbeitet werden können. Bei einer Anwendung mit Transaktionsverarbeitung können Methoden wie `rollback()` und `commit()` verwendet werden.
Zur Ausführung kann ein Java-Programm über mehrere `JDBC`-Treiber angesteuert werden. Jeder Treiber registriert sich beim `JDBC`-Manager währen der Initialisierung. Beim Versuch, sich mit einer Datenbank zu verbinden, gibt das Java-Programm einen Datenbank `URL` an den `JDBC`-Manager weiter. Der `JDBC`-Manager ruft dann einen der geladenen `JDBC`-Treiber auf. Die `URL`s von JDBC haben die Form "`jdbc:subprotocol:subname`". "`subprotocol`" ist der Name der jeweiligen Datenbank, z.B.: "`jdbc:oracle:oci7:@rfhs8012_ora3`".

4.3.2 Datenbanktreiber für den Zugriff

Zur Nutzung von JDBC braucht man einen passenden Treiber für die Datenbank. JavaSoft unterscheidet 4 Treiber-Kategorien:

1. JDBC-ODBC Bridge Treiber
Die JDBC-ODBC-Brücke wandelt Aufrufe von JDBC in ODBC-Aufrufe der Client-Seite um. Die Methoden sind nativ.
ODBC (Open DataBase Connectivity) ist ein Standard von Microsoft, der den Zugriff auf Datenbanken über eine genormte Schnittstelle möglich macht. JavaSoft und Intersolv haben eine JDBC-ODBC Bridge[32] entwickelt, die die JDBC-Aufrufe nach ODBC umwandelt.

2. Native-API Java Driver

3. Netzprotokoll All-Java Driver

4. Native Protocol All-Java Driver

4.3.3 Verbindung zur Datenbank

Die Verbindung zur Datenbank wird über die Klasse `DriverManager` und die Schnittstelle `Connection` aufgebaut. Vor der Ausführung muß ein passender Datenbanktreiber geladen werden, z.B. die von JavaSoft mitgelieferte JDBC-ODBC Bridge, deren Klasse unter dem Namen `JdbcOdbcDriver` verfügbar ist. Alle Datenbanktreiber müssen an einer Stelle gesammelt werden, dem Treibermanager. Dazu dient die Klasse `DriverManager`.

[32] Leider ist die Brücke nur für Win32- und Solaris -Systeme verfügbar. Das JDK von Sun liefert diese Brücke aus und damit können Java-Programmierer Datenbank-Anwendungen mit der Unterstützung einer Vielzahl existierender ODBC-Treibern programmieren

Methode	Parameter	Return-Type
getConnection	(String *url*,java.util.Properties *info*)	Connection
getConnection	(String *url*,String *user*, String *password*)	Connection
getConnection	(String *url*)	Connection
getDriver	(String *url*)	Driver
registerDriver	(java.sql.Driver *driver*)	void
deregisterDriver	(Driver *driver*)	void
getDrivers	()	java.util.Enumeration[33]
setLoginTimeout	(int *seconds*)	void
getLoginTimeout	()	int
setLogStream	(java.io.Printstream *out*)	void
getLogStream	()	Java.io.PrintStream
println	(String *message*)	void

Abb.: Methoden der Klasse `java.sql.DriverManager`

Der Treibermanager bietet eine Methode `getConnection()` an, mit der die Verbindung zur DB hergestellt wird. Lädt man bspw. den JDBC-ODBC-Treiber mit `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")`, dann kann eine Verbindung mit Hilfe des Connection-Objekts bspw. so aufgebaut werden:

```
Connection conn =
  DriverManager.getConnection ("jdbc:odbc:" + database,user,password);
```

Die Verbindung wird mit speziellen Optionen parametrisiert, u.a. mit dem Treiber, der die Datenquelle anspricht.. Datenquellen sind durch eine besondere **URL** mit folgendem Format qualifiziert: `jdbc:Subprotokoll`[34]`:Datenquellenname`
Die Methode `public static Connection(String url,String user,String password)` erwartet bis zu Paramter (Pflichtparameter, Anmeldename und Paßwort).

Methode	Parameter	Return-Type
createStatement	()	Statement
prepareStatement	(String sql)	PreparedStatement
prepareCall	(String sql)	CallableStatement
nativeCall	(String sql)	String
close	()	void
isClosed	()	boolean
getmetaData	()	DatabaseMetaData
setReadOnly	(boolean readOnly)	void
isReaddOnly	()	boolean
setCatalog	(String catalog)	void
getCatalog	()	String
setAutoClose	(boolean autoClose)	void
getAutoClose	()	boolean
getWarnings	()	SQLWarning
setAutoCommit	(boolean autoCommit)	void
getAutoCommit	()	boolean
commit	()	void
rollback	()	void
setTransactionIsolation	(int level)	void

[33] Die statische Methode getDrivers() liefert eine Aufzählung der angemeldeten Treiber

[34] Für ODBC-Datenquellen ist das Subprotokoll mit `odbc` anzugeben

getTransactionIsolation	()	int

Abb.: `java.sql.Connection`-Methoden und Konstanten

Die `TransactionIsolation`-Konstanten werden in `java.sql.Connection` als Integer mit folgenden Werten definiert:

TransactionIsolation-Konstantenname	Wert
TRANSACTION-NONE	0
TRANSACTION-READ-UNCOMMITTED	1
TRANSACTION-READ-COMMITTES	2
TRANSACTION-REPEATABLE-READ	4
TRANSACTION-SERIALIZABLE	8

4.3.4 Datenbankabfragen

Abfragen über das Statement-Objekt

Für Abfragen ist ein Statement-Objekt mit der Methode createStatement()[35] anzulegen, z.B.:

```
Statement stmt = conn.createStatement ();
```

SQL-Anweisungen ohne Parameter werden normalerweise über das Statement-Objekt ausgeführt. Wird das gleiche Statement mehrfach ausgeführt, lohnt es sich ein `PreparedStatement` zu konstruieren.
In JDBC gibt es drei Typen von Statement-Objekten zur Interaktion mit SQL: Statement, PreparedStatement[36], CallableStatement[37].

Ausführen von SQL-Anweisungen

Zum Auslesen von Informationen benutzt man SELECT-befehle aus SQL. Zur Angabe der SQL-Befehle dient die execute()-Methode des Statement-Interface. „executeQuery()“ benutzt man für Abfragen, executeUpdate() bei UPDATE und DELETE, z.B.:

```
ResultSet rset = stmt.executeQuery ("select 'Hello World' from dual");
```

[35] `public Statement createStatement() throws SQLException`

[36] Beim `PreparedStatement`-Objekt bereitet das Anwendungsprogramm ein SQL-Statement mit der `java.sql.Connection.prepeareStatement()`-Methode vor. Eine `PreparedSatetement`-Methode nimmt eine SQL-Zeichenkette, die an das zugrundeliegende DBMS weitergegeben wird. Das DBMS führt die SQL-Anweisung aber nicht aus. Die Methoden executeQuery(), executeUpdate() und execute() nehmen keine Parameter auf, sondern nur Aufrufe für das DBMS, das (bereits optimierte) SQL-Statement auszuführen.

[37] Ein `CallableStatement`-Objekt wird von der `prepared`-Methode eines `Connection`-Objekts erzeugt.

`public ResultSet executeQuery(String sql) throws SQLException` des Java-Interface `java.sql.Statement` führt ein SQL-Statement aus, das ein einzeilnes ResultSet-Objekt zurückgibt.

Methodenname	Parameter	Rückgabetyp
executeQuery	(String *sql)*	ResultSet
executeUpdate	(String *sql)*	int
execute	(String *sql)*	boolean
getMoreResults	()	boolean
close	()	void
getMaxFieldSize	()	int
setMaxFieldSize	(int max)	void
getMaxRows	()	int
setMaxRows	(int max)	void
setEscapeProcessing	(boolean enable)	void
getQueryTimeout	()	int
setQueryTimeout	(int seconds)	void
cancel	()	void
getWarnings	()	Java.sql.SQLWarning
clearWarnings	()	void
setCursorName	(String name)	void
getResultSet	()	ResultSet
getUpdateCount	()	int

Abb.: Methoden des Interface java.sql.Statement

Das Ergebnis einer Abfrage durch `executeQuery()` wird von einer Ergebnistabelle vom Typ `ResultSet` lassen sich unterschiedliche Spalten ansprechen und die zeilen auswerten.

Methodenname	Parameter	Rückgabetyp
next	()	boolean
close	()	void
wasNull	()	boolean

Beschaffen der Datenwerte über die Position

Methodenname	Parameter	Rückgabetyp
getAsciiStream	(int *columnIndex*)	java.io.Input.Stream
getBinaryStream	(int *columnIndex*)	Java.io.Input.Stream
getBoolean	(int *columnIndex*)	boolean
getByte	(int *columnIndex*)	byte
getBytes	(int *columnIndex*)	byte[]
getDate	(int *columnIndex*)	Java.sql.Date
getDouble	(int *columnIndex*)	double
getFloat	(int *columnIndex*)	float
getInt	(int *columnIndex*)	int
getLong	(int *columnIndex*)	long
getNumeric	(String *columnIndex*,int *scale*)	Java.sql.Numeric
getObject	(String *ColumnIndex*)	Object
getShort	(int *columnIndex*)	short
getString	(int *columnIndex*)	String
getTime	(int *columnIndex*)	Java.sql.Time
getTimeStamp	(int *columnIndex*)	Java.sql.Timestamp
getUnicodeStream	(int *columnIndex*)	java.io.InputStream

Beschaffen der Datenwerte über den Spalten-Name

Methodenname	Parameter	Rückgabetyp
getAsciiStream	(String *columnName*)	java.io.Input.Stream
getBinaryStream	(String *columnName*)	Java.io.Input.Stream
getBoolean	(String *columnName*)	boolean
getByte	(String *columnName*)	byte
getBytes	(String *columnName*)	byte[]
getDate	(String *columnName*)	Java.sql.Date
getDouble	(String *columnName*)	double
getFloat	(String *columnName*)	float
getInt	(String *columnName*)	int
getLong	(String *columnName*)	long
getNumeric	(String *columnName*,int *scale*)	Java.sql.Numeric
getObject	(String *ColumnName*)	Object
getShort	(String *columnName*)	short
getString	(String *columnName*)	String
getTime	(String *columnName*)	Java.sql.Time
getTimeStamp	(String *columnName*)	Java.sql.Timestamp
findColumn	(String *ColumnName*)	int
getWarnings	()	SQLWarning
clearWarnings	()	void
getCursorName	()	String
getMetaData	()	ResultSetMetaData

Methoden des Interface „`java.sql.ResultSet`“

Alle Methoden können eine SQLException in dem Fall auslöen, das etwas mit der DB nicht stimmt. Meistens geben SQL-Anweisungen mit Daten gefüllte Tabellen zurück. Bei JDBC kann das Programm nur eine Datenzeile zum aktuellen Bearbeitungszeitpunkt sehen. Das Programm verwendet zum Erreichen der nächsten Zeile die `next()`-Methode. JDBC besitzt keine Methoden, um im `ResultSet` zurückzugehen oder eine bestimmte Position[38] aufzusuchen. Falls ein Programm Zugriff auf eine Zeile besitzt, kann es über den Positionsindex (1 für die erste Spalte, 2 für die zweite Splate, usw.) oder den Spaltennamen auf Feldwerte der Tabellenspalten zu gelangen.
Die wichtigsten Methoden sind `executeQuery`, `executeUpdate()` und `execute()`. Falls ein Statement-Objekt mit einem SQL-Statement erzeugt wird, dann nimmt `executeQuery()` einen SQL-String an. Sie gibt die SQL-Zeichenkette über den Treibermanager an die zugrundeliegende Datenquelle weiter und bekommt das `ResultSet` für das Anwendungsprogramm. `executeQuery()` gibt nur ein `ResultSet` zurück. In den Fällen, in denen mehr als ein `ResultSet` erhalten wird, sollte `execute()` verwendet werden.
Bsp.: Check zur Überprüfung einer JDBC-Installation. Das JDBC-Programm gibt auf dem Textbildschirm „`Hello World`“ aus.

```
// You need to import the java.sql package to use JDBC
import java.sql.*;
// We import java.io to be able to read from the command line
import java.io.*;

class JdbcCheckup
{
  public static void main (String args [])
       throws SQLException, IOException
```

[38] Bookmarks in ODBC

```
{
  // Load the Oracle JDBC driver

  try {
       // Class.forName("oracle.jdbc.driver.OracleDriver");
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      }
  catch (Exception e)
    {
     System.out.println ("Fehler: " + e.getMessage () + "\n");
    }

  // DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
  // Prompt the user for connect information
  System.out.println (
   "Please enter information to test connection to the database");
  String user;
  String password;
  String database;
  user = readEntry ("user: ");
  int slash_index = user.indexOf ('/');
  if (slash_index != -1)
  {
    password = user.substring (slash_index + 1);
    user = user.substring (0, slash_index);
  }
  else
    password = readEntry ("password: ");
  database = readEntry ("database (a TNSNAME entry): ");
  System.out.print ("Connecting to the database...");
  System.out.flush ();
  System.out.println ("Connecting...");
  Connection conn =
    DriverManager.getConnection ("jdbc:odbc:" + database,
         user, password);
  // Create a statement
  Statement stmt = conn.createStatement ();
  // Do the SQL "Hello World" thing
  ResultSet rset = stmt.executeQuery ("select 'Hello World' from dual");
  while (rset.next ())
    System.out.println (rset.getString (1));
  System.out.println ("Your JDBC installation is correct.");
  // close the resultSet
  rset.close();
  // Close the statement
  stmt.close();
  // Close the connection
  conn.close();
}
// Utility function to read a line from standard input
static String readEntry (String prompt)
{
  try
  {
    StringBuffer buffer = new StringBuffer ();
    System.out.print (prompt);
    System.out.flush ();
    int c = System.in.read ();
    while (c != '\n' && c != -1)
```

```
      {
        buffer.append ((char)c);
        c = System.in.read ();
      }
      return buffer.toString ().trim ();
    }
    catch (IOException e)
    {
      return "";
    }
  }
}
```

4.3.5 Hinzufügen von Elementen zu einer DB

`public int executeUpdate(String sql) throws SQLException` führt eine SQL-Anweisung aus, die Manipulationen an der DB vornimmt. Die SQL-Anweisungen sind INSERT, UPDATE oder DELETE. Zurückgegeben wird die Anzahl der veränderten Zeilen bzw. 0, falls eine SQL-Anweisung nichts verändert hat.

4.3.6 Metadaten

Metadaten können für jede Abfrage angefordert werden. So lassen sich u.a. leicht herausfinden

- wie viele Spalten in einer Tabelle abgefragt werden können
- wie der Name der Spalte ist
- wie der SQL-Typ der Spalte ist
- wie viele Dezimalzeichen eine Spalte hat.

Die Methode `getMetaData()` des Interface `java.sql.ResultSet` gibt die Metainformationen über ein `ResultSet` zurück. Metadaten sind auch für die gesamte DB abfragbar.
Bsp. Für diese Art von Informationen sind

- Wer ist mit der DB verbunden?
- Kann die DB nur gelesen werden?
- Sind gespeicherte Prozeduren erlaubt?

Sind Informationen über die DB gefragt, so lassen sich über Metadaten eines `DatabaseMetaDat`a-Objekts bspw. Datenbankeigenschaften des Herstellers herausfinden. Ein DataBaseMetaData-Objekt erhält man über die Methode getConnection() des Treibermanagers:

```
Connection con = DriverManager.getConnection(”jdbc:odbc:daten”,”user”,”password”);
DataBaseMetaData meta = con.getMetaData();
```

`getMetaData()` gibt ein `DatabaseMetaData`-Objekt zurück, das eine große Anzahl von Methoden erlaubt.

Methodenname	Parameter	Rückgabetyp
allProceduresAreCallable	()	boolean
allTablesAreSelectable	()	boolean
getURL	()	String
getUserName	()	String
isReadOnly	()	boolean
nullsAreSortedHigh	()	boolean
nullsAreSortedLow	()	boolean
nullsAreSortedAtStart	()	boolean
nullsAreSortedAtEnd	()	boolean
.....	...	...

Abb.: Methoden eines DatabaseMetaData-Objekts

4.3.7 Ausnahmen bei JDBC

Unter JDBC sind 3 Arten von Ausnahmen möglich:

`SQLException`

Die Klasse SQLException ist die Basisklasse aller jdbc-Exceptions: Sie enthält über den Fehler folgende Informationen:

- eine Fehlerbeschreibung
- eine weitere Fehlerbeschreibung, die den XOPEN SQL Status (beschrieben in der SQL-Spezifikation) angibt
- eine Ganzzahl (, die vom Datenbanktreiber kommt)

Methodenname	Parameter	Rückgabetyp
SQLException	(String *reason*,String *SQLState*,int *vendorCode*)	SQLException
SQLException	(String *reason*,String *SQLState*)	SQLException
SQLException	(String *reason*)	SQLException
SQLException	()	SQLException
getSQLState	()	String
getErrorCode	()	int
getNextException		SQLException
setNextException	SQLException e)	void

`SQLWarnings`

Warnungen können ermittelt werden über die Funktionen getWarnings() der Klassen Connection, ResultSet und Statement. Werden die Meldungen nicht geholt, dann werden sie durch die nachfolgenden Operationen mit Connection, ResultSet oder Statement überschrieben.
„Warnings“ werden an das Objekt, das die „Warning“ verurscht, angehängt. Die Überprüfung auf Warnungen erfolgt mit der Methode getWarning(), die für alle Objekte verfügbar ist.

Methodenname	Parameter	Rückgabetyp
SQLWarning	(String *reason*, String *SQLState*,int *vendorCode*)	SQLWarning
SQLWarning	(String reason,String *SQLState*)	SQLWarning
SQLWarning	(String *reason*)	SQLWarning
SQLWarning	()	SQLWarning
getNextWarning	()	SQLWarning
SetNextWarning	(SQLWarning *w*)	void

`Data Truncation`

Das ist ein spezieller Typ einer SQL-Warnung. Sie wird immer dann erzeugt, wenn Daten während Schreib-/Lese-Operationen verlorengingen. Mit instanceof DataTruncation kann dann geprüft werden, ob es sich um eine DataTruncation handelt.

4.3.8 JDBC-Programmierung

4.3.8.1 Anwendungen / Applets mit JDBC

```
/*
 * This sample applet just selects 'Hello World' and the date from the
 * database
 */

// Import the JDBC classes
import java.sql.*;

// Import the java classes used in applets
import java.awt.*;
import java.io.*;
import java.util.*;

public class HalloJdbcApplet extends java.applet.Applet
{
  // The driver to load
  static final String driver_class = "oracle.jdbc.driver.OracleDriver";

  // The connect string
  static final String connect_string =
              "jdbc:oracle:thin:saj39122/saj39122@lap-sauer:1521:pcora81";

  // This is the kind of string you woudl use if going through the
  // Oracle 8 connection manager which lets you run the database on a
  // different host than the Web Server.  See the on-line documentation
  // for more information.
  // The query we will execute
  static final String query = "select 'Hallo JDBC: ' || sysdate from dual";
  // The button to push for executing the query
  Button execute_button;
  // The place where to dump the query result
  TextArea output;
  // The connection to the database
  Connection conn;
  // Create the User Interface
  public void init ()
  {
    this.setLayout (new BorderLayout ());
    Panel p = new Panel ();
    p.setLayout (new FlowLayout (FlowLayout.LEFT));
    execute_button = new Button ("Hello JDBC");
    p.add (execute_button);
    this.add ("North", p);
    output = new TextArea (10, 60);
    this.add ("Center", output);
  }
  // Do the work
  public boolean action (Event ev, Object arg)
  {
     if (ev.target == execute_button)
    {
      try
      {
```

```
      // Clear the output area
      // Diese setText Anweisung initialisiert TextArea mit leerem String!
      //output.setText ("");

      // See if we need to open the connection to the database

      if (conn == null)
      {
        // Load the JDBC driver
        output.append ("Loading JDBC driver " + driver_class + "\n");
        Class.forName (driver_class);

        // Connect to the databse
        output.append ("Connecting to " + connect_string + "\n");
        DriverManager.registerDriver
                        (new oracle.jdbc.driver.OracleDriver ());
        conn = DriverManager.getConnection (connect_string);
        output.append ("Connected\n");
      }
        // Create a statement
      Statement stmt = conn.createStatement ();
        // Execute the query
      output.append ("Executing query " + query + "\n");
      ResultSet rset = stmt.executeQuery (query);
        // Dump the result
      while (rset.next ())
        output.append (rset.getString (1) + "\n");
      // We're done
      output.append ("done.\n");
      }
      catch (Exception e)
      {
      output.append (e.getMessage () + "\n");
      }
      return true;
    }
    else
      return false;
  }
}
```

4.3.8.2 Anwendungen / Applets mit Embedded SQL (SQLJ)

```
/*
 * This applet extends the AppletInterface class that contains the
 * user interface component of the applet.
 * This applet connects to a database to select and display
 * employee information.  It will also delete a select employee
 * from the database.
 */

// SQLJ-specific classes
import java.sql.SQLException;
import java.sql.DriverManager;
import sqlj.runtime.ExecutionContext;
import sqlj.runtime.ref.DefaultContext;
import oracle.jdbc.driver.OracleDriver;
import oracle.sqlj.runtime.Oracle;

// Event handling classes
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class AppletMain extends AppletUI
                        implements ActionListener
{

  // Declare a named iterator with several columns from the EMP table

  #sql public static iterator
  EmpIter(String empno, String ename, String job, String sal, String comm);

// Applet initialization

  private DefaultContext m_ctx = null;

  public void init ()
  {

    // Create the User Interface

    super.init();

    // Activate the buttons

    Query_button.addActionListener(this);
    Query_button.setActionCommand("query");

    Delete_button.addActionListener(this);
    Delete_button.setActionCommand("delete");

    // Open a connection to the database

    if (m_ctx == null)
    {
      // Connect to the database
      String url      = null;
      String user     = null;
      String password = null;
```

```
    try
    {
      url      = getParameter("sqlj.url");
      user     = getParameter("sqlj.user");
      password = getParameter("sqlj.password");
    }
    catch (NullPointerException exn) { };
    // permit to call as an application

    try
    {
       if (   url==null || url.equals("")
           || user==null || user.equals("")
           || password==null || password.equals(""))
       {
     // If the connect properties are not passed as parameters from the
     // HTML file, we pull them out of the connnect.properties resource.
     output.append("Connecting using the connect.properties resource\n");
     m_ctx = Oracle.getConnection(getClass(),"connect.properties");
       }
       else
       {
     output.append("Connecting using the PARAMeters in the HTML file\n");
     output.append("User " + user + " to " + url + "\n");

     DriverManager.registerDriver(new OracleDriver());
     m_ctx = Oracle.getConnection(url, user, password);
       }
       output.append("Connected\n");
    }
    catch (SQLException exn)
    {
       output.append("A SQL exception occurred: "+exn.getMessage()+"\n");
    }
  }
  else
  {
    output.append("Re-using connection.\n");
  }
}

// Perform the work

public void actionPerformed(ActionEvent ev)
{
  String command = ev.getActionCommand();

  try
  {

    if (command.equals("query"))
    {
       int numRecords = 0;

       EmpIter ecur;

       // Clear the output area
       output.setText("");
```

```
   String x = query_name_field.getText();
   if (x==null || x.equals("") || x.equals("%"))
   {
      // Execute the query
      output.append("Executing: SELECT * FROM EMP\n");
      #sql [m_ctx] ecur = { SELECT * FROM EMP };
      while (ecur.next ())
      {
       output.append(ecur.empno() + "     " + ecur.ename() + "     "
                       + ecur.job() + "     $" + ecur.sal() + "\n");
       numRecords++;
      }
   }
   else
   {
      output.append("Executing: SELECT * FROM EMP WHERE ENAME = '"+
                          query_name_field.getText() + "'\n\n");
      #sql [m_ctx] ecur =
 { SELECT * FROM EMP WHERE ENAME = :(query_name_field.getText()) };
      while (ecur.next())
      {
        output.append("Employee's Number:   " + ecur.empno() + "\n");
        output.append("Employee's Name:     " + ecur.ename() + "\n");
        output.append("Employee's Job:      " + ecur.job() + "\n");
        output.append("Employee's Salary:   " + ecur.sal() + "\n");
        output.append("Employee's Commison: " + ecur.comm() + "\n");
        numRecords++;
      }
   }

   // we're done
   output.append(numRecords + " record"+( (numRecords==1)?"":"s" )+
                 " retrieved.\n");
   query_name_field.setText("");

   // ensure that iterator is closed
   ecur.close();
}
else if (command.equals("delete"))
{
   output.setText("");

   // Use an execution context to get an update count
   ExecutionContext ectx = new ExecutionContext();
   #sql [m_ctx, ectx]
   { DELETE FROM EMP WHERE ENAME = :(delete_name_field.getText()) };
   int numDeleted = ectx.getUpdateCount();
   if (numDeleted==1)
   {
 output.append("Deleted employee "+delete_name_field.getText()+ ".");
   }
   else if (numDeleted==ExecutionContext.EXCEPTION_COUNT)
   {
     output.append("An exception occurred during deletion.");
   }
   else
   {
     output.append("Deleted "+numDeleted+" employees.");
   }
```

```
            delete_name_field.setText("");
          }
        }
        catch (SQLException e)
        {
          // Report the error
          output.append("A SQL exception occurred:\n"+e.getMessage () + "\n");
        }
      }

      // it is important to rollback (or commit) at the end of the day, and
      // not leave the connection dangling

      public void stop()
      {
        if (m_ctx != null)
        {
          try
          {
             System.out.println("Closing the applet.");
             #sql [m_ctx] { ROLLBACK };
             // or, if you prefer: #sql [m_ctx] { COMMIT };

             m_ctx.close();
          }
          catch (SQLException exn) { }
          finally { m_ctx = null; }
        }
      };

      // Provide a main entry point so this works both, as an applet, and as
      // an application.
      public static void main(String[] args)
      {
        AppletFrame f = new AppletFrame(new AppletMain(), 600, 350);
      }

    }
```

5. Veteilte Objektarchitekturen

5.1 Konzept der Applets

5.1.1 Beschreibung des Konzepts

Applets sind Programme, die in der Regel nicht als eigener Prozeß auf dem Rechner iniziiert werden und dort laufen, sondern innerhalb eines Web-Browsers ausgeführt werden. Zur Ausführung eines Applets ist es demnach nötig, das Applet in einem HTML-Dokument[39] einzubetten. In diesem HTML-Dokument werden dem javafähigen Browser die Informationen mitgeteilt, die er zur Ausführung des Applets benötigt. Der Browser lädt die Klassendatei und führt das Applet automatisch aus.
Aus der Sicht des Programmierers ist das Applet eine Klasse, die von der Applet-Klasse abgeleitet wird.

5.1.1.1 Aufgaben zur Erläuterung des Konzepts

1. Aufgabe: Erstelle ein Applet, das in einem Fenster den Text „Herzlich Willkommen in der Java-Welt“ ausgibt.
Lösungsschritte:
1) Erstelle die folgende Datei mit dem Namen „`WillkommenApplet.java`“ mit Hilfe eines Dateiaufbereites (Editor):

```
/* Das erste Java-Applet */

import java.awt.Graphics;

public class WillkommenApplet extends java.applet.Applet
{
 public void paint (Graphics g)
 {
  g.drawString("Herzlich willkommen in der Java Welt!",5,25);
 }
}
```

Durch die „import“-Anweisung können Entwickler Klassen verwenden, die in anderen Dateien definiert sind. Compiler bzw. Interpreter greifen auf die class-Dateien zu. Über „import“ wird bestimmt, wo diese Dateien liegen. Java importiert immer das Paket „`java.lang`“, denn hier ist die Klasse **Object** enthalten, von der alle Java-Klassen abgeleitet sind. Die Graphics-Klasse enthält Methoden zum Zeichnen von Textzeichen und Zeichenketten. Mit der „`drawString`“-Methode der Graphics-Klasse können Textzeichen auf den Bildschirm gemalt werden.
Die folgende Abbildung zeigt die Modellierung[40] des vorliegenden Quellcodes:

---

[39] eine entsprechende Referenz innerhalb einer HTML-Seite mit einem speziellen Tag, dem <APPLET>-Tag erledigt die Einbettung in den Browser

[40] Die Modellierung erfolgt nach den Regeln der Unified Modelling Language (UML). Die UML ist eine grafische, standardisierte Sprache zum Spezifizieren, Konstruieren, Visualisieren und Dokumentieren.

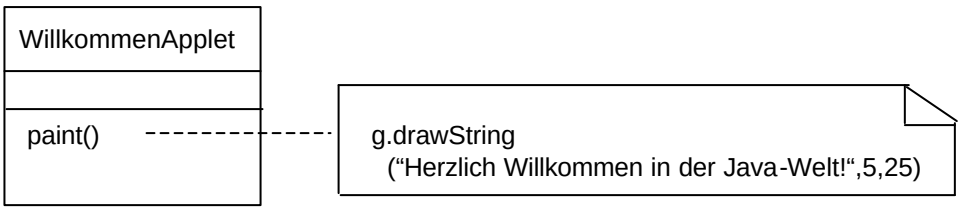


Abb.: Klassendiagramm zu WillkommenApplet

Die Klasse `WillkommenApplet` läßt sich grafisch als rechteckiges Symbol darstellen. Die `paint()`-Methode wird ohne formale Parameter beschrieben, ihre Implementierung wird durch die beigefügte Notiz gezeigt.
Die unmittelbare Superklasse wird im Quelltext direkt in der „extends"-Klausel angegeben. „`extends java.applet.Applet`" bestimmt das die angegebene Applet-Klasse von der Applet-Klasse des Abstract Windowing Toolkit (AWT) abgeleitet ist. Der andere Teil der Klassendefinition enthält das Schlüsselwort „`public`" mit der Bedeutung: Die Klasse ist nach dem Laden für das gesamte Java-System verfügbar. Applets müssen „`public`" deklariert werden.
Die folgende Abbildung zeigt die Beziehungen der Klasse `WillkommenApplet` zu ihren unmittelbaren Nachbarn:

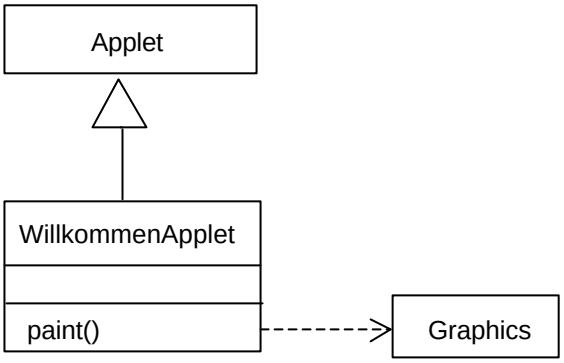


Abb.: Die unmittelbare Umgebung von `WillkommenApplet`

Die gerichtete Linie mit der unausgefüllten Pfeilspitze von `WillkommenApplet` zu `Applet` repräsentiert eine Generalisierung, d.h.: `WillkommenApplet` ist eine Unterklasse von `Applet`. Der gestrichelte Pfeil repräsentiert eine Abhängigkeitbeziehung: `WillkommenApplet` verwendet `Graphics`.
Ein eigenes, vom Benutzer erstelltes Applet überschreibt gewöhnlich Methoden, die in der Superklasse Applet definiert sind. Diese Methoden übernehmen Aufgaben zur Initialisierung des Applet vor der Ausführung (`public void start()`), zur Reaktion auf Mauseingaben, zum Anhalten des Applet (`public void stop()`) und zu Aufräumungsarbeiten (`public void destroy()`), wenn das Applet beendet wird. Eine dieser Methoden ist `paint()`, die sich um die Anzeige des Applet in einer Webseite kümmert. Die `paint()`-Methode besitzt ein einziges Argument, eine Instanz der Klasse `Graphics`. Die Klasse `Graphics` stellt Verhaltensweisen zur Darstellung von Schriften, Farben, zum Zeichnen von Linien[41], von Ellipsen

[41] `public void drawLine(int x1, int y1, int x2, int y2);` `(x1,y1)` bestimmt den Anfangspunkt, `(x2,y2)` bestimmt den Endpunkt der Linie.

bzw. Kreisen[42], von Rechtecken[43] und anderen Formen zur Verfügung. In der `paint()`-Methode wurde hier der String "Herzlich Willkommen in der Java Welt!" bei den (x,y)-Koordinaten (5,25) ausgegeben. Der Ursprung des Koordinatensystems liegt in der linken oberen Ecke des Darstellungsbereichs. Der String wird in einer defaultmäßig festgelegten Schrift und Farbe angezeigt.
Die Untersuchung der Java-Bibliotheken zu Applet und Graphics zeigt: Die beiden Klassen sind Teil einer größeren Hierarchie. Verfolgt man die von Applet erweiterten und implementierten Klassen, dann kann man das folgende Klassendiagramm erhalten:

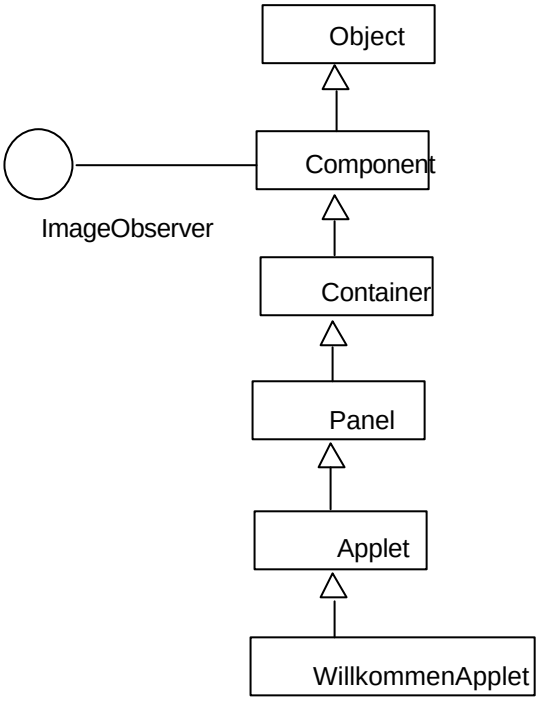


Abb.: Vererbungshierarchie von `WillkommenApplet`

Die Beziehung zwischen ImageObserver und Component ist eine Schnittstelle. ImageObserver wird von Component implementiert.
`WillkommenApplet` arbeitet mit den Klassen Applet und Graphics unmittelbar zusammen. Diese beiden Klassen bilden lediglich einen kleinen Ausschnitt aus der Bibliothek mit vordefinierten Java-Klassen. Die Verwaltung dieser Klassen und Schnittstellen organisiert Java in mehreren verschiedenen Paketen. Das Wurzelpaket in der Java-Umgebung heißt `java`. In dieses Paket sind mehrere weitere Pakete geschachtelt, die wiederum andere Pakete, Schnittstellen und Klassen enthalten. Object existiert im Paket `lang`, Panel, Container, Component existieren im Paket `awt`, und die Klasse Applet im Paket `applet`. Die Schnittstelle `ImageObserver` existiert im Paket `image`, das liegt wiederum im Paket `awt` (qualifizierter Name: `java.awt.ImageObserver`). Die Paketstruktur[44] kann in einem Klassendiagramm visualisiert werden:

---

[42] `public void drawOval(int x, int y, int width, int height);` `(x,y)` gibt die Koordinaten der oberen linken Ecke des die Ellipse umschreibenden Rechtecks mit der Höhe `height` und der Breite `width` am

[43] `public void drawRect(int x, int y, int width, int height);` bestimmt die obere linke Ecke des Rechtecks, `(width, height)` legen Breite und Höhe des Rechtecks fest.

[44] Pakete werden in der UML als Akten mit Reitern dargestellt. Die gestrichelten Pfeile repräsentieren die Abhängigkeiten zwischen den Paketen.

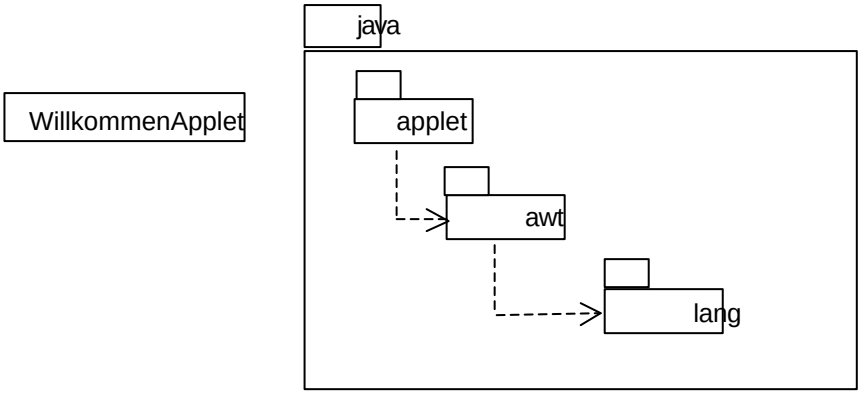


Abb.: Paketstruktur im `WillkommenApplet`

2) Speichern der Datei mit dem Namen „WillkommenApplet.java“ unter einem beliebigen Verzeichnis.
3) Aufruf des Java-Übersetzers über die folgende Befehlszeileneingabe: „`javac WillkommenApplet.java`“. Der Compiler gibt bei erfolgreicher Übersetzung keine Rückmeldung. Zwei Dateien müssen nach erfolgreicher Übersetzung vorliegen: „`WillkommenApplet.java`“ und „`WillkommenApplet.class`“.
4) Erstellen der folgenden HTML-Datei „`WillkommenApplet.html`“, anschließend Speichern dieser Datei.

```
<HTML>
<HEAD>
<TITLE>Seid gegruesst!</TITLE>
</HEAD>
<BODY>
<P>Mein Java Applet sagt:<BR>
<APPLET CODE="WillkommenApplet.class" WIDTH=200 HEIGHT=50>
</APPLET>
</BODY>
</HTML>
```

Der Bezugspunkt in HTML-Dateien auf ein Applet erfolgt mit dem <Applet>-Tag. Das Code-Attribut dient zur Angabe von dem Namen der Klasse, die das Applet enthält. Die Attribute WIDTH und HEIGHT dienen zum Bestimmen der Größe des Applets. Der Browser benutzt diese Werte zur Zuteilung des Raums, der für das Applet auf der Seite freigehalten werden muß. Hier wurde eine Box mit einer Breite von 200 und einer Höhe von 50 Pixeln definiert.
WillkommenApplet ist als Applet implementiert. Es kann nie isoliert stehen, sondern ist normalerweise Teil einer Webseite.

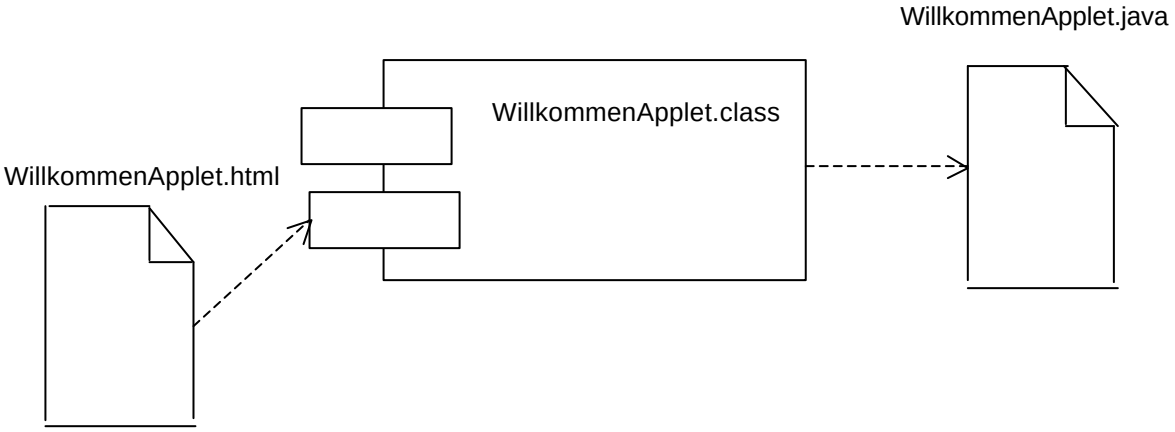


Abb.: Die Komponenten von WillkommenApplet

5) Ausführung des Applet mit einem javafähigen Web-Browser bzw. dem Appletviewer, z.B. mit dem Aufruf „appletviewer WillkommenApplet.html“. Das Resultat müßte so aussehen:

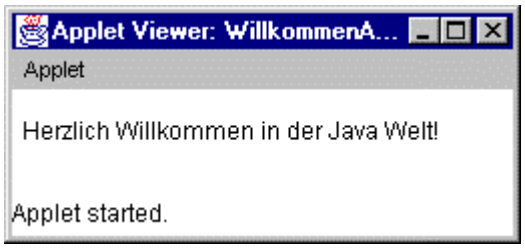


Abb.: Darstellung des Fensters mit dem Appletviewer

In einem Browser würde zusätzlich der Text rund um das Applet („Mein Java Applet sagt:“) gezeigt werden.

2. Aufgabe: Verändern von Schrift und Farbe für den Text „Herzlich Willkommen in der Java-Welt“.
Lösungsschritte:
1) Erweitere die Datei mit dem Namen „WillkommenApplet.java“ mit Hilfe eines Dateiaufbereites (Editor).
Die erste Erweiterung soll die Schrift verändern, in der der Text ausgegeben wird. Es wird ein Objekt der Klasse `java.awt.Font` über folgende Anweisung erzeugt: `Font f = new Font("TimesRoman",Font.BOLD,12);`
Objekte der Klasse Font dienen zum Bereitstellen verschiedener Schriftarten für die Methode drawString() und repräsentieren den Namen, den Stil und die Größe einer Schrift. Mit einem spezifischen Objekt der Klasse Font kann man eine Schrift aufrufen, die sich von der standardmäßig in Applets benutzten Schrift unterscheidet. Dem Font-Objekt wird hier die Schrift „TimesRoman“, fett in „12-Punkt“ Größe zugewiesen. Das neue Objekt wird anschließend der Instanzvariablen f zugewiesen und ist so der Methode `paint()` zugänglich:

```
public void paint (Graphics g)
{
 // Mitteilung: Die Schrift zur Anzeige von Text befindet sich in der
 // Instanzvariablen f
 g.setFont(f);
 // Mitteilung: Die Farbe fuer die Ausgabe ist gelb
 g.setColor(Color.yellow);
```

```
  // Die glebe Farbe dient zum Fuellen eines Rahmens fuer den Text, der
  // aus einem Rechteck mit abgerundeten Ecken besteht
  g.fillRoundRect(0,0,225,30,10,10);
  // Mitteilung: die Farbe fuer Textausgaben ist
  // eine Instanz der Klasse Color fuer die Farbe rot
  g.setColor(Color.red);
  // Mitteilung: Text wird in festgelegter Schrift und Farbe bei den
  // (x,y)-Koordinaten (5,25) ausgegeben.
  g.drawString("Herzlich Willkommen in der Java Welt!",5,25);
 }
```

Die Klassen `Font` und `Color` werden über die folgenden `import`-Anweisungen bereitgestellt:

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;
```

Dafür kann man auch „`import java.awt.*`“[1] schreiben.
2) Kompiliere die Datei „`WillkommenApplet.java`“ in eine „.class“-Datei.
3) Ausführung des Applet, z.B. über den Aufruf `appletviewer WillkommenApplet.html`.

3. Aufgabe: Zeichnen von 100 verschieden eingefärbten Rechtecken in ein Applet-Fenster

```
// zeichne  Rechtecke
import java.applet.*;
import java.awt.*;
public class RechteckeAppl3 extends Applet
{
 // Instanzvariable
 int appletHoehe;
 int appletBreite;
 int rechteckHoehe;
 int rechteckBreite;
 int rechteckTop;
 int rechteckLinks;
 Color rechteckFarbe;
 int anzRechtecke = 100;
 // Methoden
 public void init()
 {
  setBackground(Color.lightGray);
  Dimension groesse = getSize();
  appletHoehe  = groesse.height;
  appletBreite = groesse.width;
  // repaint();
 }
 public void paint(Graphics g)
 {
  // setBackground(Color.white);
  g.setColor(Color.black);
  g.drawRect(0,0,appletBreite - 1,appletHoehe - 1);
  for (int i = 0; i < anzRechtecke; i++)
  {
   rechteckTop    = bestimmeZufallszahl(appletHoehe);
```

[45] Diese Anweisung stellt alle Klassen des Pakets `java.awt` zur Verfügung

```
   rechteckLinks  = bestimmeZufallszahl(appletBreite);
   rechteckHoehe  = bestimmeZufallszahl(appletHoehe  - rechteckTop);
   rechteckBreite = bestimmeZufallszahl(appletBreite - rechteckLinks);
   rechteckFarbe  = new Color(bestimmeZufallszahl(255),
                              bestimmeZufallszahl(255),
                              bestimmeZufallszahl(255));
   g.setColor(rechteckFarbe);
   g.fillRect(rechteckLinks,rechteckTop,rechteckBreite-1,
              rechteckHoehe - 1);
  }
 }
 private int bestimmeZufallszahl(int bereich)
 {
  double ausgangsgroesse;
  ausgangsgroesse = Math.random();
  return (int) (ausgangsgroesse * bereich);
 }
}
```

4. Aufgabe: Entwickle aus der vorliegenden Aufgabe eine allgemeine Applet-Schablone, die als Vorlage für das Erstellen von Applets dienen kann.

Das folgende Gerüst beschreibt ein Muster für alle Applets:

```
// Name der Klasse:

// Beschreibung:

// Import der Pakete
// import java.lang.*;
// import java.applet.*;
// import java.awt.*;

// Top-Level-Klassen-Deklaration bzw. Definition des Applets

public class Klassenname extends java.applet.Applet
{
 // Variablen-Deklarationen bzw. Definitionen
 // ...
 // Eigene Methoden
 // ...
 // Methoden, die ueberschrieben werden
 //
 public void init()
 {
  // ...
 }
 public void start()
 {
  // ...
 }
 public void stop()
 {
  // ...
 }
 public void destroy()
 {
  // ...
```

```
 }
 // Optional: die Ausgabemethode
 public void paint(Graphics g)
 {
  // ..
 }
}
```

Ein Applet erbt Methoden und Variablen der Applet-Klasse. Die Applet-Klasse erbt wiederum von einer Reihe anderer Klassen. Eine Klasse, die `java.applet.Applet` erweitert, kann Variablen und Methoden aus `java.lang.Object`, `java.awt.Component`, `java.awt.Container` sowie `java.awt.Panel` verwenden.

5.1.1.2 Lebenszyklus von Applets

Ein Java-Applet besitzt im Gegensatz zu einer Java-Anwendung keine main()-Methode, die beim Laden gestartet wird und das Programm solange am Leben hält, bis es der Benutzer beendet. In einem Applet sorgen vier Methoden dafür, daß sich das Applet in seiner Umgebung korrekt verhält.

1. Das Erstellen eines Applets

Zum Erstellen eines Applet muß immer eine „Subklasse“ der Klasse Applet[46] erzeugt werden. Java setzt voraus, daß eine Applet-Subklasse `public` deklariert wurde. Erkennt Java ein Applet auf einer Web-Seite, dann wird die Applet-Ausgangsklasse und die Hilfsklasse, die diese erste Klasse evtl. benutzt, über das Netz geladen. Java erstellt eine Instanz dieser Klasse, alle systembezogenenen Methoden werden an diese Instanz geschickt. Mehrere Applets auf der gleichen oder auf unterschiedlichen Seiten verwenden andere Instanzen, so daß sich jedes Applet auf dem gleichen System evtl. anders verhält.

2. Applet-Methoden

Applets können zahlreiche, unterschiedliche Aktivitäten umfassen, die verschiedenen wichtigen Ereignissen im Lebenszyklus eines Applet entsprechen, z.B. Initialisieren, Zeichnen, Mausereignisse. Jeder Aktivität ist eine entsprechende Methode zugeordnet, d.h.: Falls eine Ereignis stattfindet, ruft der Browser (oder ein javaähnliches Werkzeug) diese spezifische Methode auf.
Zum Reagieren auf solche Ereignisse sind bestimmte Verhaltensweisen vorzusehen. Das geschieht durch Überschreiben der jeweiligen Methode in der Applet-Subklasse. Unterschiedliche Applet-Verhalten bedeutet: Jeweils andere Methoden müssen überschieben werden. Die folgenden Methoden bestimmen den Lebenszyklus eines Applet:

[46] Alle Applets müssen java.applet.Applet in die Datei mit der Definition der Applet-Klasse importieren. „java.applet.*“ vollzieht das Einbunden von „java.applet.Applet“ automatisch. Fast alle Applets (die mit grafischen schnittstellen) benötigen auch java.awt.*

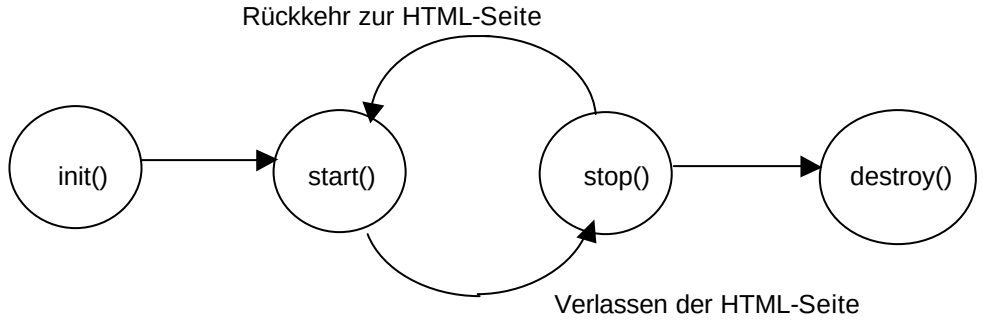


Abb.: Lebenszklus eines Applet

Die Methode `init()` wird nach dem Laden des Applet ausgeführt. Sie dient zur Initialisierung.
Die Methode `start()` wird automatisch nach Aufruf der Methode init() aufgerufen bzw. dann, wenn das Applet in den Zustand „aktiv“ versetzt wird. Applets können in zwei Zuständen sein: aktiv und inaktiv. Nach dem Laden eines Applets ist dieses zunächst inaktiv. Das Applet wechselt in den Zustand aktiv, wenn es erstmalig auf dem Bildschirm erscheint. Von dort aus wechselt es seinen Zustand zwischen aktiv und inaktiv. Wodurch dieser Zustandswechsel genau ausgelöst wird, ist abhängig vom Kontext des Applel, d.h. in der Regel vom verwendetet Web-Browser.
Die Methode `stop()` wird aufgerufen, wenn die HTML-Seite, in der das Applet eingebunden ist, verlassen wird bzw. das Applet in den Zustand inaktiv versetzt wird.
Die Methode `destroy()` zerstört das Applet, nachdem es gestoppt wurde und der Kontext des Applet sich für eine Zerstörung entscheidet. Die Methode `destroy()` sorgt dafür, daß alle vom Applet belegte Ressourcen wieder freigegeben werden. Vorhandene, vom Applet erzeugte Threads werden ebenfalls zerstört.
Bsp.: Überwachen des Lebenszyklus eines Applet.
Ein Applet führt keine Aktionen aus eigener Initiative aus, sondern empfängt Ereignisse vom System und liefert Ergebnisse zurück. Beim Start eines Applet ruft der Webbrowser die Methode init() des Appletobjekts auf. Beim Beenden eines Applets wird die Methode destroy() aufgerufen. init() und destroy() werden im Lebenszyklus eines Applet genau einmal ausgeführt, nämlich beim Erzeugen bzw. beim Beenden eines Objekts. Zusätzlich sind zwei weitere Methoden start() und stop() vorgesehen, die an verschiedenen Stellen zur Aktivierung aufgerufen werden können. Wählt ein Anwender bei gestartetem Applet z.B. im Browser eine andere Internetseite an, so ruft der Browser die stop()-Routine auf und hält das Applet an, bis der Anwender wieder auf die Appletseite zurückkehrt. Dann ruft der Browser die start()-Routine zum ordnungsgemäßen Fortsetzen des Applet auf.

```
import java.awt.*;
import java.applet.*;

public class AppletDemo extends Applet
{
 // Instanzvariable
 String s;
 int    initialisierungen = 0;
 int    startpunkte       = 0;
 int    haltepunkte       = 0;
 // Methoden
 public void init()  { initialisierungen++; }
 public void start() { startpunkte++; }
 public void stop()  { haltepunkte++; }
 public void paint(Graphics g)
 {
  // Zur Ausgabe einer geeigneten Nachricht über das Verhalten
  // des Applet wird der String s mit Hilfe des Opertors + aus
  // Teilstrings zusammengesetzt. Ganze Zahlen werden dabei in Zei-
```

```
  // chenketten konvertiert.
  s = "Initialisierungen: " + initialisierungen +
      ", Startpunkte: " + startpunkte +
      ", Haltepunkte: " + haltepunkte;
  g.drawString(s,10,10);
 }
}
```

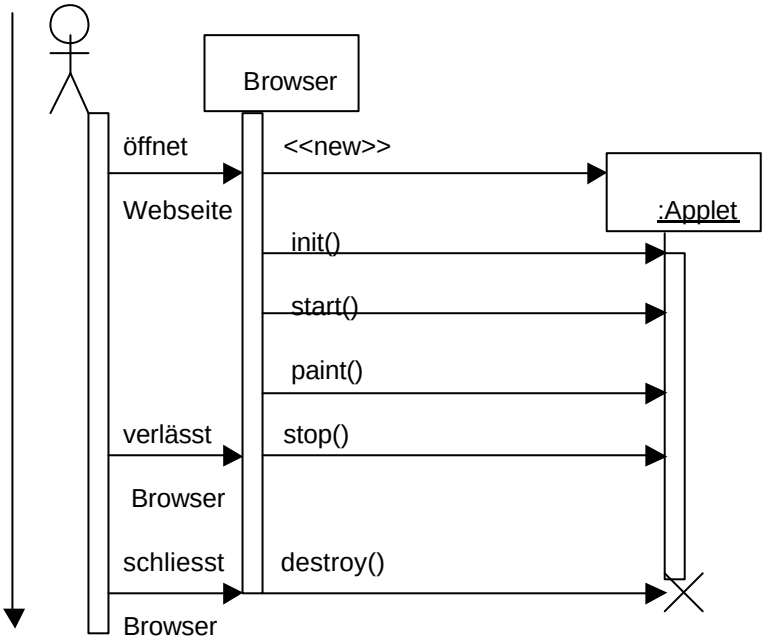


Abb.: Lebenszyklus eines Applets

Die `paint()`-Methode wird in Java immer aufgerufen, wenn ein Applet gezeichnet werden muß, z.B. bein erstmaligen Zeichnen des Applet, beim Verschieben des Applet-Fenster, beim Überlagern des Applet-Fenster durch ein anderes Fenster. Die `paint()`-Methode hat folgende Gestalt:

```
public void paint(Graphics g)
{
 ....
}
```

`paint()` besitzt ein Argument: eine Instanz der Klasse `Graphics`. Dieses Objekt wird vom Browser erstellt und an paint() abgegeben. Es muß sichergestellt sein, daß die Graphics-Klasse in den Code der Applet-Subklasse importiert[47] wird.
Bsp.: „Aller Anfang ist schwer!. Dieser Spruch soll mit Hilfe eines Applet gezeigt werden.. Die zugehörige Quellcodedatei „AllerAnfangApplet.java“ umfaßt

```
import java.applet.*;
import java.awt.*;

public class AllerAnfangApplet extends Applet
{
 Font f;
 String spruch;
 //
```

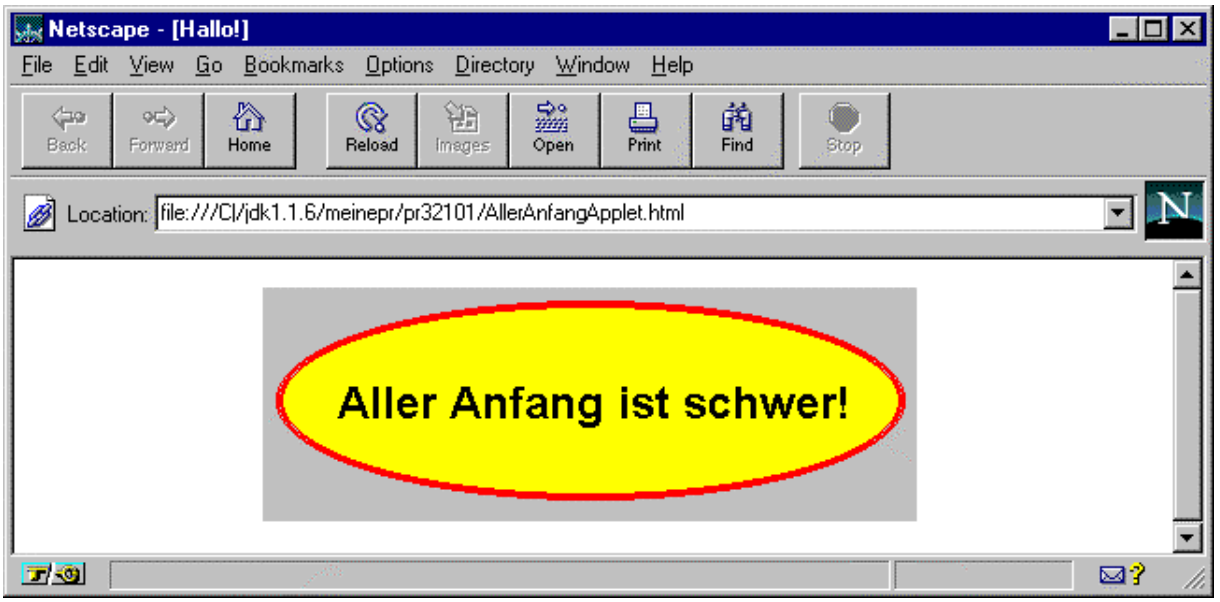
[47] Normalerweise geschieht dies über: `import java.awt.Graphics`

```
 public void init()
 {
  f = new Font(„Helvetica“,Font.BOLD,24);
  this.spruch = „Aller Anfang ist schwer!“;
 }
 //
 public void paint(Graphics g)
 {
  // Oval mit Farbe yellow
  g.setColor(Color.yellow);
  g.fillOval(10,10,330,100);
  // Roter Rahmen; da Java keine Linienbreite kennt,
  // wird die Linienbreite durrch 4 Ovale, (die sich
  // um Pixelbreite unterscheiden,) simuliert
  g.setColor(Color.red);
  g.drawOval(10,10,330,100);
  g.drawOval( 9, 9,332,102);
  g.drawOval( 8, 8,334,104);
  g.drawOval( 7, 7,336,106);
  g.setColor(Color.black);
  g.setFont(f);
  g.drawString(this.spruch,40,70);
 }
}
```

Die zugehörige HTML-Datei AllerAnfangApplet.html umfaßt:

```
<HTML>
<HEAD>
<TITEL>Hallo!</TITLE>
<BODY>
<CENTER>
<APPLET CODE=“AllerAnfangApplet.class“ WIDTH=350 HEIGHT=125>
</APPLET>
</CENTER>
</BODY>
</HEAD>
</HTML>
```

In einem Browser führt das zu der folgenden Darstellung:



Java-Applets zeichnen sich durch Überschreiben der „paint“-Methode selbst. Wie wird die „paint“-Methode aufgerufen?
Es gibt drei verschiedene Methoden zum Neuzeichnen eines Applet:

`public void paint(Graphics g)`
Sie zeichnet tatsächlich die Grafik des Applets in den Zeichenbereich. Sie wird immer aufgerufen, wenn ein Applet neu gezeichnet[48] werden muß. Das in der „paint“-Methode abgebene Graphics-Objekt enthält den Grafikstatus, d.h. die aktuellen Merkmale der Zeichnungsoberfläche.
`public void repaint()`
Sie kann jederzeit aufgerufen werden, wann auch immer das Applet neu gezeichnet werden muß. Sie ist der Auslöser, die „paint“-Methode sobald wie möglich aufzurufen und das Applet neu zu zeichnen. Sollten die repaint()-Anweisungen schneller ablaufen, als Java diese verarbeiten kann, werden evtl. einige übersprungen. In vielen Fällen ist die Verzögerung zwischen dem Aufruf von repaint() und der eigentlichen Aktualisierung des Fensters vernachlässigbar.
`public void update(Graphics g)`
Sie wird von repaint() aufgerufen. Die „update“-Methode löscht den vollständigen Zeichenbereich[49] und ruft anschließend die „paint“-Methode auf, die dann das Applet vollständig neu zeichnet. Der Aufruf der „paint“-Methode erfolgt also nicht direkt über die „repaint“-Methode, sondern indirekt über die „update“-Methode.

5.1.1.3 HTML-Tags für den Einsatz von Applets

Die konkrete Referenzierung eines Java-Applet in einer HTML-Seite wird mit dem Applet-Tag `<applet ...>` eingeleitet. Zwischen dem einleitenden Tag und dem abschließenden Applet-Tag (</applet>) können benötigte Parameter oder beliebiger Text eingegeben werden. Die einfachste Form der Applet-Referenz. Ohne irgendwelche Parameter wird ein Java-Applet eingebunden mit

`<APPLET CODE = “`*`klassenelement`*`“ WIDTH = `*`Wert`*` HEIGHT = `*`Wert`*`>`
`</APPLET>`
klassenelement: Applet-Klasse
WIDTH = *Wert*: Breite des Applet in Pixel

[48] Dies ist immer beim ersten Aufruf des Applets der Fall, aber auch jedesmal dann, wenn das Applet-Fenster verschoben oder zwischenzeitlich von einem anderen Fenster überlagert wurde.
[49] Das ruft oft den unangenehmen Flimmereffekt bei schnellen Bildsequenzen hervor.

HEIGHT = *Wert*: Höhe des Applet in Pixel

Das <APPLET>-Tag erzeugt keinen Absatz, deshalb sollte es in einem allgemeinen Text-Tag stehen, z.B. <P> oder in einem Überschriften-Tag (<H1>, <H2> usw.).

Optionale Parameter des <Applet>-Tag

Parameter	Bedeutung
CODEBASE	Hier kann ein alternatives Verzeichnis[50] für das Laden von Klassendateien angegeben werden. Fehlt diese Angabe, wird das Dokumentenverzeichnis genommen.
ARCHIVE	Angabe des JAR-Archivs, aus dem die Klassendateien und sonstige Resourcen des Applet genommen werden
OBJECT	Name der Datei, die den serialisierten Inhalt des Applet enthält.
ALT	Alternativer Text für Browser, die das Applet verstehen, aber Java nicht unterstützen
NAME	Eindeutiger Name für das Applet. Er kann zur Unterscheidung mehrerer kommunizierender Applets auf einer Web-Seite verwendet werden.
ALIGN	Vertikale Anordnung des Applets in einer Textzeile. Hier kann einer der Werte left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom angegeben werden.
VSPACE	Rand über und unter dem Applet
HSPACE	Rand links oder rechts vom Applet

Neben den Parametern des „Applet-Tag“ können auch Parameter an das Applet selbst übergeben werden. Jeder Parameter kann durch ein <PARAM>-Tag über zwei Attribute für Name (*name*) und Wert (value) festgelegt werden. Das <PARAM>-Tag steht zwischen einem öffnenden und schließenden <APPLET>-Tag.
Die Parameter werden beim Laden an das Applet weitergereicht. Innerhalb des Applets können sie mit der Methode `public String getParameter(String name)` abgefragt werden.

```
import java.applet.*;
import java.awt.*;

public class GraphApplet extends Applet
{
  int x0, xN, y0, yN;
  double xmin, xmax, ymin, ymax;
  int appletHoehe, appletBreite;

  public void init()
  {
    // Wie groß ist das Applet?
    Dimension groesse = getSize();
    appletHoehe   = groesse.height;
    appletBreite  = groesse.width;
    x0 = 0;
    xN = appletBreite-1;
    y0=0;
    yN=appletHoehe-1;
    xmin = -10.0;
    xmax =  10.0;
    ymin = -1.0;
```

[50] Pfadname, in dem sich die Klassen befinden

```
    ymax =  1.0;
  }

  public void paint(Graphics g)
  {
    double x1, y1, x2, y2;
    int i, j1, j2;
    // Achsen
    g.drawLine(0,appletHoehe / 2,appletBreite,appletHoehe / 2);
    g.drawLine(appletBreite / 2,0,appletBreite / 2,appletHoehe);
    g.drawRect(0,0,appletBreite - 1,appletHoehe - 1);
    // Kurve
    j1 = yWert(0);
    for (i = 0; i < appletBreite; i++) {
      j2 = yWert(i+1);
      g.drawLine(i, j1 ,i+1, j2);
      j1 = j2;
    }
   }

  private int yWert(int iWert)
  {
    // Berechne x, y
    double x, y;
    int jWert;
    x = (iWert * (xmax - xmin)/(appletBreite - 1)) + xmin;
    // Berechnen Funktionswert
    y = Math.sin(x) * Math.cos(x) / x;
    // Umsetzen y in Bildpunktwerte
    jWert = (int) ((y - ymin)*(appletHoehe - 1)/(ymax - ymin));
    // Umrechnen auf das Koordinatensystem des Fensters
    jWert = appletHoehe - 1 - jWert;
    return jWert;
  }
}
```

```
<HTML>
<HEAD>
<TITLE>Graphische Darstellung einer Funktion</TITLE>
</HEAD>
<BODY>
<APPLET CODE="GraphApplet.class" WIDTH=400 HEIGHT=300>
<PARAM NAME=”AppletParameter1” VALUE=Uebergabewert>
<PARAM NAME=AppletParameter2 VALUE=Uebergabewert>
….
</APPLET>
</BODY>
</HTML>
```

5.1.2 Applets und Anwendungen

Da in Applets alle Java-Sprachkonstrukte und -Klassenbibliotheken zur Verfügung stehen, unterscheidet sich der Quellcode zwischen Applets und Applikationen nur unwesentlich. Einer der wesentlichsten Unterschiede besteht darin, daß ein Applet keine `main()`-Methode besitzt und statt dessen von der Klasse Applet abgeleitet werden muß. Ferner ist anstelle eines Konstruktors die `init()`-Methode zu verwenden. Diese Methode wird vom Browser aufgerufen, wenn ein Objekt der Applet-Klasse erzeugt wird.
Wird von einer Anwendung implizit das Rahmen-Layout verwendet, so muß in der init()-Methode bei der Konvertierung einer Anwendung in ein Applet explizit angegeben werden, da der Standard-Layoutmanager für ein Applet das `FlowLayout` ist.
Umwandeln eines Applet-Codes in eine Anwendung. Eine main()-Methode muß in die Anwendung eingebracht werden:

```
public static void main(String args[])
{
 // Erzeuge einen Rahmen (Frame) fuer die Anwendung
 Frame meinFenster = new Frame();
 //Erzeuge eine Instanz
.....meineApplikation = new ......();
 // Fuege die aktuellen Applikation zu dem Frame
 meinfenster.add("Center",meineApplikation);
 // Erweitere das Fenster auf die gewuenschte Groesse
 // und mache das Fenster sichtbar
 meinFenster.resize(200,200);
 meinFenster.show();
 // Rufe die Methoden auf
 meineApplikation.init();
 meineApplikation.start();
}
```

Schließlich besitzen Applets weder eine Titelzeile noch Menüleisten. Sollten diese in einer Anwendung, die in ein Applet konvertiert werden soll, vorkommen, müssen sie gelöscht werden. Ein Titel kann für die HTML-Seite mit dem <TITLE>-Tag erzeugt werden, Menüs müssen durch andere GUI-Komponenten (Schaltflächen o.ä.) modelliert werden.

5.1.3 Methoden zur Ereignisbehandlung von Applets

5.1.3.1 Event-Handling unter JDK 1.1

Ein Applet kann auch auf Ereignisse wie Mausbewegungen reagieren. Für soche Ereignisse (z.B. Drücken der Maustaste) stellt Java Ereignisbehandlungs-Methoden des JDK 1.0 bzw. JDK 1.1 zur Verfügung.

Bsp. Ein Applet zum Zeichnen von Punkten an den Stellen, an denen eine Maustaste gedrückt wurde.

```
// Import der Pakete
import java.awt.*;
import java.awt.event.*;
// Top-Level Klassen-Deklaration des Applets
public class MausDownPunktApplet extends java.applet.Applet
{
 // Variablen-Deklarationen
 private int mausX, mausY;
 // private boolean mausKlick = false;
 // Methoden, die ueberschrieben werden
 public void init()
 {
  setBackground(Color.yellow);
  addMouseListener(new MouseAdapter()
  {
   public void mouseClicked(MouseEvent e)
   {
    mausX = e.getX(); mausY = e.getY();
    repaint();
   }
  });
 }
 /*
 public boolean mouseDown(Event e, int x, int y)
 {
  mausX = x; mausY = y;
  mausKlick = true;
  repaint();
  return true;
 }
 */
 // Ausgabemethode
 public void paint(Graphics g)
 {
  g.setColor(Color.blue);
  // if (mausKlick)
  // {
   g.fillOval(mausX,mausY,20,20);
  // mausKlick = false;
  // }
 }
}
```

Nach jedem Mausklick wird ein blauer Punkt in die Zeichenfläche gebracht. Das Bild wird neu gezeichnet. Die `repaint()`-Methode ruft vor der Ausführung der `paint()`-

Methode die `update()`-Methode auf, die anschließend `paint()` aufruft. „`update()`“ leert in Originalform den Anzeigebereich des Applets. Wird `update()` überschrieben, z.B. durch

```
public void update(Graphics g)
 {
  paint(g);
 }
```

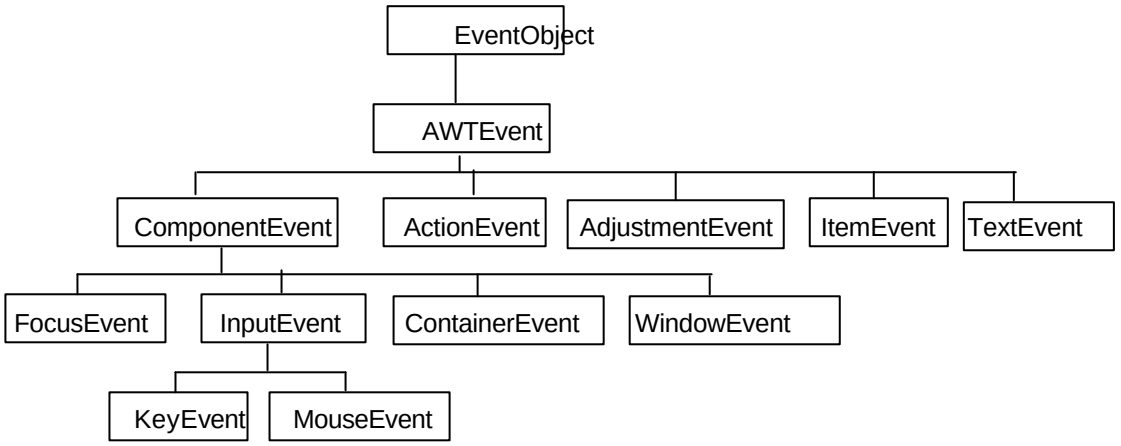
entfällt das Leeren des Anzeigebereichs.

```
// Import der Pakete
import java.awt.*;
import java.awt.event.*;
// Top-Level Klassen-Deklaration des Applets
public class MausDownPunkteApplet extends java.applet.Applet
{
 // Variablen-Deklarationen
 private int mausX, mausY;
 // private boolean mausKlick = false;
 // Methoden, die ueberschrieben werden
 public void init()
 {
  setBackground(Color.yellow);
  addMouseListener(new MouseAdapter()
  {
   public void mouseClicked(MouseEvent e)
   {
    mausX = e.getX(); mausY = e.getY(); repaint();
   }
  });
 }
 /*
 public boolean mouseDown(Event e, int x, int y)
 {
  mausX = x; mausY = y;  mausKlick = true;  repaint();
  return true;
 }
 */
 // Ausgabemethode
 public void paint(Graphics g)
 {
  g.setColor(Color.blue);
  // if (mausKlick) {
  g.fillOval(mausX,mausY,20,20);
  //  mausKlick = false;
  // }
 }
 public void update(Graphics g)
 { paint(g); }
}
```

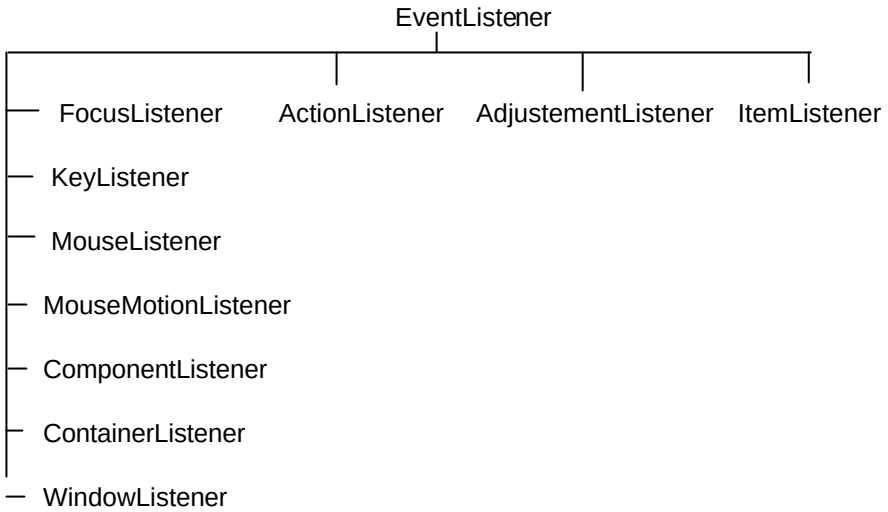
Eine überschriebene update()-Methode kann den Flimmereffekt erheblich senken.

5.1.3.2 Ereignisbehandlung unter grafischen Benutzeroberflächen

Die Reaktion auf Nachrichten (z.B. Mausklick, Tastatureingabe, Veränderungen an Größe und Lage der Fenster) erfolgt in speziellen Ereignisempfängern (EventListeners), die das zum Ereignis passende Empfänger-Interface implementieren. Damit ein Ereignisempfänger Nachrichten einer bestimmten Ereignisquelle erhält, muß er sich bei der Quelle registrieren lassen, d.h.: Es muß eine spezielle Event-Klasse geschrieben, instanziert und bei der Ereignisquelle registriert werden. Zum Empfang von Nachrichten muß ein Objekt eine Reihe von Methoden implementieren, die von der Nachrichtenquelle, bei der es sich registriert hat, aufgerufen werden können. Die Ereignisempfänger stellen Methoden durch Implementierung von Interfaces bereit, die aus der Klasse EventListener des Pakets `java.util` abgeleitet sind.



Die Hierarchie der AWT-spezifischen Ereignisklassen beginnt mit der Klasse AWTEvent und befindet sich im Paket java.awt. AWTEvent ist Superklasse aller Ereignisklassen des AWT, die sich im Paket java.awt.event befinden. Dieses Paket ist in jede Klasse einzubeziehen, die sich mit dem Event-Handling von AWT-Anwendungen beschäftigt.
EventListener-Interface. Je Ereignisklasse gibt es ein EventListener-Interface. Es definiert eine seperate Methode für jede Ereignisart der Ereignisklasse. So besitzt bspw. das Interface MouseListener die Methoden mouseClicked, mouseEntered, mouseExited, mousePressed und mouseReleased, die beim Eintreffen des jeweiligen Ereignis aufgerufen werden.



Jede der Methoden eines Listener-Interface enthält als einziges Argument ein Objekt vom zugehörigen Ereignistyp. Alle Methoden sind vom Typ `void`.

Das JDK 1.1 impliziert eine Vielzahl von unterschiedlichen Möglichkeiten, Ereignishandler zu implementieren. Fogende Entwurfsmuster bieten sich an:

- Die Fensterklasse implementiert die erforderlichen EventListener-Interfaces, stellt die erforderlichen Call-Back-Methoden zur Verfügung und registriert sich selbst bei der Ereignisquelle
- In der Fensterklasse werden lokale und anonyme Klassen definiert, die einen EventListener implementieren oder sich aus einer Adapterklasse ableiten. Eine Adapterklasse implementiert ein Interface mit mehreren Methoden und erlaubt es somit abgeleiteten Klassen, nur noch die Methoden zu überlagern, die tatsächlich von Interesse sind.
- GUI-Code und Ereignisbehandlung werden vollkommen getrennt und auf unterschiedlicheerteilt Klassen verteilt.
- In der Komponentenklasse werden Methoden überlagert, die für das Empfangen und Verteilen von Nachrichten erforderlich sind.

5.1.3.3 Anwendung lokaler Klassen für die Ereignisbehandlung

Im GUI-Objekt, das einen Event-Handler benötigt wird eine lokale Klasse zur Implementierung des passenden Interface angegeben. Lokale Klassen werden lokal zu einer anderen Klasse erzeugt. Sie sind nur innerhalb dieser Klasse definiert und sichtbar. Objekte der lokalen Klasse können nur aus der erzeugenden Klasse produziert werden. Die lokale Klasse kann aber auf alle Instanzmerkmale der erzeugenden Klasse zugreifen.
Eine Variante lokale Klassen sind anonyme Klassen. Sie werden ebenfalls lokal zu einer anderen Klasse erzeugt, kommen aber ohne Klassennamen aus. Dazu werden sie bei der Übergabe eines Objekts an eine Methode oder als Rückgabewert einer Methode innerhalb einer einzigen Anwendung definiert und instanziert. Damit einer anonymen Klasse überhaupt eine sinnvolle Aufgabe zugeführt werden kann, muß sie aus einer anderen Klasse abgeleitet sein oder ein bestehendes Interface implementieren.

Bsp.: ActionListener-Interface für die Übernahme eines Textfeld-Inhalts in ein Textfeld zur Ausgabe

Falls das Textfeld für die Ausgabe den im Textfeld für die Eingabe angegebenen Text wiedergeben soll, muß ein ActionListener (eine zusätzlich innere, anonyme Klasse) mit der Methode `public void actionPerformed (ActionEvent a)` definiert werden. Diese Methode umfaßt Aufrufe der Methoden `getText()` und `setText()` der Klasse `TextField`. Darüber kann in das Eingabetextfeld eingegebener Text in das Ausgabefeld ausgegeben werden. Da die "Methode actionPerformed" der anonymen inneren Klasse auf "eingabeTextFeld" bzw. "ausgabeTextFeld" zugreift, ist außerhalb von main() anzugeben:

```
private static TextField eingabeTextFeld = new TextField(20);
private static TextField ausgabeTextFeld = new TextField(20);
```

Zum Schließen des Fensters wird über eine anonyme innere Klasse ein **WindowAdapter** (eine Art WindowListener) mit der Methode "public void windowClosing(WindowEvent e)" definiert und an den "Frame" eingekettet. Wird der Frame geschlossen, dann wird diese Methode aufgerufen, die über System.exit(0) die Applikation beendet.
Eine Adapterklasse implementiert ein Interface mit mehreren Mehoden und erlaubt es somit abgeleiteten Klassen, nur noch die Methoden zu überlagern, die tatsächlich von Interesse sind. Passende Adapterklasssen stellt das Paket java.awt.event bereit, z.B. FocusAdapter, KeyAdapter, MouseAdapter, MouseMotionAdapter, ComponentAdapter, ContainerAdapter, WindowAdapter. Die Registrierung erfolgt mit der Methode addWindowListener, die in den Klassen Dialog und Frame zur Verfügung steht.

Bsp.: Textfeldbearbeitung mit "ActionListener-Interface" und WindowAdapter

```
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
public class EchoMitBeno extends Object
{
 // Einlesen und Ausgeben von Zeichenketten ueber
 // eine grafische Benutzeroberflaeche
 private static TextField eingabeTextFeld = new TextField(20);
 private static TextField ausgabeTextFeld = new TextField(20);
 public static void main(String args[])
 {
  Frame fenster = new Frame("Echo");
  fenster.addWindowListener(new WindowAdapter()
  {
   public void windowClosing(WindowEvent e)
   {
    System.exit(0);
   }
  });
  // TextField eingabeTextFeld  = new TextField(20);
  Label eingabeTextFeldLabel = new Label("Eingabestring:");
  eingabeTextFeld.setEditable(true);
  eingabeTextFeld.addActionListener(new ActionListener()
  {
   public void actionPerformed(ActionEvent ae)
   {
    String s = eingabeTextFeld.getText();
    ausgabeTextFeld.setText(s);
   }
  });
  // TextField ausgabeTextFeld  = new TextField(20);
  Label ausgabeTextFeldLabel = new Label("Ausgabestring:");
  ausgabeTextFeld.setEditable(false);
  Panel panel = new Panel();
  panel.add(eingabeTextFeldLabel);
  panel.add(eingabeTextFeld);
  panel.add(ausgabeTextFeldLabel);
  panel.add(ausgabeTextFeld);
  // Frame fenster = new Frame("Echo");
  fenster.add(panel);
  fenster.pack();
  fenster.setVisible(true);
 }
}
```

Alternativ zur vorliegenden Realisierung der Ereignisbehandlung über eine anonyme Klasse, kann auch ein Listener über eine lokale Klasse implementiert werden. Gewöhnlich ist die Anwendung eine Subklasse von Frame.

```
import java.lang.*;
import java.awt.*;
import java.awt.event.*;

public class Vorl11b extends Frame
{
```

```
  private static Label eingabeTextFeldLabel = new Label("Eingabestring:");
  private static TextField eingabeTextFeld = new TextField(20);
  private static Label ausgabeTextFeldLabel = new Label("Ausgabestring:");
  private static TextField ausgabeTextFeld = new TextField(20);
  public Vorl11b()
  {
  eingabeTextFeld.setEditable(true);
  ausgabeTextFeld.setEditable(false);
  Panel panel = new Panel();
  panel.add(eingabeTextFeldLabel);
  panel.add(eingabeTextFeld);
  panel.add(ausgabeTextFeldLabel);
  panel.add(ausgabeTextFeld);
  eingabeTextFeld.addActionListener(new TFL());
  add(panel);
  pack();
  setVisible(true);
  }
  class TFL implements ActionListener
  {
   public void actionPerformed(ActionEvent ae)
   {
    String s = eingabeTextFeld.getText();
    ausgabeTextFeld.setText(s);
   }
 }
 public static void main(String args[])
 {
  Vorl11b vorl11b = new Vorl11b();
  vorl11b.addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
     System.exit(0);
    }
  });
 }
}
```

Schließlich kann das Interface auch direkt implementiert werden. Zweckmäßigerweise ist dann die Anwendung eine Subklasse von Frame.

```
import java.lang.*;
import java.awt.*;
import java.awt.event.*;

public class Vorl11a extends Frame implements ActionListener
{
  private static Label eingabeTextFeldLabel = new Label("Eingabestring:");
  private static TextField eingabeTextFeld = new TextField(20);
  private static Label ausgabeTextFeldLabel = new Label("Ausgabestring:");
  private static TextField ausgabeTextFeld = new TextField(20);
  public Vorl11a()
  {
  eingabeTextFeld.setEditable(true);
  ausgabeTextFeld.setEditable(false);
  Panel panel = new Panel();
  panel.add(eingabeTextFeldLabel);
  panel.add(eingabeTextFeld);
```

```
  panel.add(ausgabeTextFeldLabel);
  panel.add(ausgabeTextFeld);
  eingabeTextFeld.addActionListener(this);
  add(panel);
  pack();
  setVisible(true);
  }

   public void actionPerformed(ActionEvent ae)
   {
    System.out.println(ae.getActionCommand());
    String s = eingabeTextFeld.getText();
    ausgabeTextFeld.setText(s);
   }

 public static void main(String args[])
 {
  Vorl11a vorl11a = new Vorl11a();
  vorl11a.addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
     System.exit(0);
    }
  });
 }
}
```

5.1.3.4 Dialoge

Der Anwender verkehrt im Rahmen grafischer Benutzeroberflächen über vom Programm vorgegebene Dialoge. Diese bestehen aus einem Fenster und einer Reihe von Dialogelementen (z.B. Textfelder, Buttons, Listboxen) zur Darstellung und Erfassung programmspezifischer Daten. Das Design der Dialoge wird von Layoutmangern unterstützt, die sich um Größe und Anordnung der einzelnen Dialogelemente kümmern.

Zum Erstellen eines Dialogs sind 4 Schritte nötig:

- Anlegen eines Fensters
- Zuordnen eine Layoutmanagers
- Einfügen von Dialogelementen
- Anzeigen des Fensters

Anlegen eines Fensters. Ein Dialogfenster kann wahlweise aus der Klasse Frame oder Dialog abgeleitet werden. "Dialog" erlaubt "modale Dialoge[51]" und verhindert das Verändern der Fenstergröße durch den Anwender. Im Gegensatz zum Frame kann ein "Dialog"-Fenster keine Menüleiste erzeugen und dem Fenster kein Icon[52] zuordnen.

[51] Modale Dialaloge verhindern die Interaktion des Anwenders mit anderen Fenstern der Anwendung bis zum Schließen des Dialogfensters.

[52] Falls ein Fenster (unter Windows) minimiert wird, zeigt es ein Icon an. Mit Hilfe eines Doppelklicks auf das Icon kann eine ursprüngliche Größe des Fensters wiederhergestellt werden. Mit Hlife der Methode "`public`

Zuordnen eines Layoutmanagers. Es wird über die Methode "public void setLayout(LayoutManager mgr)" der Klasse Container realisiert. Java stellt fünf Layoutmanager bereit: FlowLayout BorderLayout, GridLayout GridBagLayout und CardLayout.
Neben den Fähigkeiten eines Layoutmanagers bestimmt in der Regel die Reihenfolge der Aufrufe von der "add"-Methode des Fensters die tatsächliche Anordnung der Komponenten auf dem Bildschirm.
Schachteln von Layoutmanagern. Dazu wird an die Stelle, die das Sublayout erhalten soll, einfach ein Objekt der Klasse Panel eingefügt, das einen eigenen Layoutmanager erhält. Dieses Panel kann mit Dialogelementen bestückt werden, die entsprechend dem zugeordneten Sublayout formatiert werden, z.B.:

```
import java.awt.*;
import java.awt.event.*;
public class PR14171 extends Frame
{
 public PR14171()
 {
  addWindowListener(new WindowAdapter()
  {
   public void windowClosing(WindowEvent e)
   {
    setVisible(false);
    dispose();
    System.exit(0);
   }
  });
  // Layout festelegen und Komponenten hinzufuegen
  int i = 0;
  Panel p1 = new Panel();
  p1.setLayout(new GridLayout(3,1));
  p1.add(new Button("Schaltflaeche " + ++i));
  p1.add(new Button("Schaltflaeche " + ++i));
  p1.add(new Button("Schaltflaeche " + ++i));
  Panel p2 = new Panel();
  p2.setLayout(new BorderLayout());
  p2.add("North",new Button("Schaltflaeche " + ++i));
  p2.add("South",new Button("Schaltflaeche " + ++i));
  p2.add("West",new Button("Schaltflaeche " + ++i));
  p2.add("East",new Button("Schaltflaeche " + ++i));
  p2.add("Center",new Button("Schaltflaeche " + ++i));
  // Hauptfenster
  setLayout(new GridLayout(1,2));
  add(p1);
  add(p2);
  pack();
 }
 public static void main(String args[])
 {
  PR14171 f = new PR14171();
  f.setVisible(true);
 }
}
```

`void setIconImage(Image bild)`" kann einem Fenster ein Icon zugeordnet werden, das beim minimieren angezeigt wird.

Das vorliegende Programm zeigt nach dem Aufruf das folgende Fenster:

Einfügen von Dialogelementen. Es erfolgt über

```
public Component add(Component komponente)
public Component add(Component komponente, int pos)
public Component add (String name, Component komponente)
// erwartet einen String-Parameter, der bei bestimmten Layout-Managern
// (z.B. BorderLayout) Informationen zur Positionierung der Elemente ("bei
// BorderLayout: "South", "East", "West", "North", "Center") angibt.
```

der Klasse Container. Mit "`public void remove(Component komponente)`" können bereits an das Fenster übergebene Komponenten gelöscht werden.

Anzeigen eines Dialogfensters. Es erfolgt durch einen Aufruf von "setVisible". Zweckmäßig sollte zuvor "public void pack()" der Klasse Window zur Anpassung der Fenstergröße an den für die Darstellung der Dialogelemente erforderlichen Platz ausgeführt werden.

5.1.4 Multithreading fähige Applets

Mit Threads können in Java Applets so erstellt werden, daß alle oder auch einzelnen Codeteile in ihrem eigenen Thread laufen, ohne andere Teile des Systems zu beeinflussen. Ein Applet kann im wesentlichen über vier Schritte Multithreading-fähig gemacht werden:

1. Erweitern der Signatur des Applets um implements Runnable
2. Hinzufügen einer Instanzvariablen, die den Thread des Applet enthält
3. Reduktion der start()-Methode, so daß sie außer dem Start des Threads keine weiteren Threads enthält
4. Hinzufügen der run()-Methode, die den eigentlichen Code enthält, den das Applet ausführen soll.

Bsp.: Ein Applet zur Anzeige von Datum und Uhrzeit, jede Sekunde wird aktualisiert Nach den bisher vorliegenden Erkenntnissen müßte das zugehörige Applet folgende Gestalt haben:

```
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;
//
// Top Level Deklaration des Applets
//
public class DigitalUhr extends java.applet.Applet
{
 // Variablen-Deklaration
 Font einFont = new Font("TimesRoman",Font.BOLD,24);
```

```
 Date datum;
 // Eigene Methoden
 // Methoden, die ueberschrieben werden
 public void start()
 {
  // Ausfuehrung des Applet
  while (true)
  {
   datum = new Date();
   repaint();  // Aufruf der repaint()-Methode
   try {Thread.sleep(1000); } // Pause von 1000 Millisekunden
   catch(InterruptedException e) {}
  }
 }
 // Optional, aber sehr wahrscheinlich – die Ausgabemethode
 public void paint(Graphics g)
 {
  g.setFont(einFont); // Setzen des aktuellen Font
  g.drawString(datum.toString(),10,50); // Ausgabe Datum
  // Da paint() wiederholt mit jeweils dem aktuellen Wert von
  // „datum“ aufgerufen wird, wird die Zeichenkette jede Sekunde
  //zur Ausgabe des neuen Datums aufgerufen
 }
}
```

In der `start()`-Methode nimmt die `while`-Schleife alle Systemressourcen für sich in Anspruch (einschl. der Anzeige am Bildschirm). Deshalb funktioniert die digitale Uhr nicht. Außerdem kann das Applet nicht gestoppt werden, da die `stop()`-Methode nicht aufgerufen werden kann. Die Lösung des Problems liegt im erneuten Schreiben des Applets mit Threads. Das Applet muß dazu mit den vorgegebenen vier Arbeitsschritten erweitert werden.

```
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;
//
// Top Level Deklaration des Applets
//
public class DigitalThreadUhr extends java.applet.Applet
                               implements Runnable
{
 // Variablen-Deklaration
 Font einFont = new Font(“TimesRoman“,Font.BOLD,24);
 Date datum;
 Thread faden;
 // Eigene Methoden
 // Methoden, die ueberschrieben werden
 public void start()
 {
  if (faden == null)
  {
   faden = new Thread(this);
   faden.start();
  }
 }
 public void stop()
 {
  if (faden != null)
```

```
      {
       faden.stop();
       faden = null;
      }
     }
     public void run()
     {
      // Ausfuehrung des Applet, hier findet die Animation statt
      while (true)
      {
       datum = new Date();
       repaint();  // Aufruf der repaint()-Methode
       try {Thread.sleep(1000); } // Pause von 1000 Millisekunden
       catch(InterruptedException e) {}
      }
     }
     // Optional, aber sehr wahrscheinlich – die Ausgabemethode
     public void paint(Graphics g)
     {
      g.setFont(einFont); // Setzen des aktuellen Font
      g.drawString(datum.toString(),10,50); // Ausgabe Datum
      // Da paint() wiederholt mit jeweils dem aktuellen Wert von
      // „datum“ aufgerufen wird, wird die Zeichenkette jede Sekunde
      // zur Ausgabe des neuen Datums aufgerufen
     }
    }
```

Das folgende Gerüst umfaßt ein Muster für „multithreading“-fähige Applets:

```
// Name der Klasse:

// Beschreibung:

// Import der Pakete
// import java.lang.*;
// import java.applet.*;
// import java.awt.*;

// Top-Level-Klassen-Deklaration bzw. Definition des Applets

public class Klassenname extends java.applet.Applet
{
 // Variablen-Deklarationen bzw. Definitionen
 // ...
 // Eigene Methoden
 // ...
 // Methoden, die ueberschrieben werden
 //
 public void init()
 {
  // ...
 }
 public void start()
 {
  // ...
 }
 public void stop()
 {
  // ...
```

```
 }
 public void destroy()
 {
  // ...
 }
 // Optional: die Ausgabemethode
 public void paint(Graphics g)
 {
  // ..
 }
 // Bei Multithreading: Verwendung der run-Methode
 public void run()
 {
  // ...
 }
}
```

5.1.5 Animation in Applets

Animationsschritte

Eine Animation umfaßt in Java zwei Schritte:

1.Aufbau und Ausgabe eines Animationsrahmens (-fenster).
2. Entsprechende häufige Wiederholung der Zeichnung, um den Eindruck von Bewegung zu vermitteln (Abspielen einer Animation).

Aufbau eines Animationsrahmens

Dazu gehört alles das, was die Animation vorbereitet, z.B.:

- Ermitteln der Größe des Ausgabebereichs
- Positionieren der Animation
- Erstellen oder Laden der einzelnen Animationsbilder
- Aufbau von einzelnen Animationssequenzen

Abspielen einer Animation

Die `paint()`-Methode wird von Java aufgerufen, wenn ein Applet gezeichnet werden muß. Java kann aber auch aufgefordert werden, ein Bild zu einem bestimmten Zeitpunkt nachzuzeichnen. Tut man das wiederholt und schnell genug mit der `repaint()`-Methode, dann entsteht eine Animation. `repaint()` ist eine Anfrage an Java, das Applet so schnell wie möglich zu zeichnen.

```
import java.awt.*;
import java.util.*;

public class PendelAppl1 extends java.applet.Applet implements Runnable
{
 // Instanzvariable
 // Position vom Zentrum des schwingenden Pendels
 int x, y;
 //
 double thetaMax = (double)  0.35;
 double thetaMin = (double) –0.35;
 // Die initiale Position vom Pendel
 double theta = (double) 0.;
 //
 double wechsel = (double) 0.01;
 //
 int xStart = 150, yStart = 20;
 // Radius des Pendels
 double r = (double) 200;
 // Durchmesser des Balls
 int d = 20;
 Thread faden;
 // Methoden
 public void init()
 {
  setBackground(Color.yellow);
```

```
 }
 public void start()
 {
  if (faden == null)
  {
   faden = new Thread(this);
   faden.start();
  }
 }
 public void stop()
 {
  if (faden != null)
  {
   faden.stop();
   faden = null;
  }
 }
 public void run()
 {
  while (true)
  {
   x = xStart + (int)(r * Math.sin(theta));
   y = yStart + (int)(r * Math.cos(theta));
   if ((theta >= thetaMax) | (theta <= thetaMin))
    wechsel = -wechsel;
   theta += wechsel;
   repaint();
   try { Thread.sleep(10); }
   catch(InterruptedException e) {}
  }
 }
 public void paint(Graphics g)
 {
  g.setColor(Color.blue);
  g.drawLine(xStart,yStart,x,y);
  g.setColor(Color.red);
  g.fillOval(x-d/2,y-d/2,d,d);
 }
}
```

Reduktion von Flimmereffekten in Animationen

Die Methode `update()` ist due Ursache für das Flimmer-Problem. Da das Applet-Fenster zwischen den Einzelbildern gelöscht wird, springen die Bereiche des Applet-Fenster, die sich ändern, kurz zwischen dem Zustand Löschen und Neuzeichnen hin und her, d.h. sie flimmern. Zwei Verfahrensweisen können das Flimmern von Java-Applets einschränken:

- Überschreiben der update-Methode so, daß sie entweder den Bildschirm nicht löscht oder nur Teile löscht, die geändert wurden.
- Überschreiben der Methoden update() und paint(), Verwenden doppelter Pufferung.

1. Überschreiben der `update()`-Methode

`update()` löscht den Bildschirm durch Füllen mit der aktuellen Hintergrundfarbe, setzt die aktuellen Farbe auf die Vordergrundfarbe und ruft anschließend `paint()` auf. Das Überschreiben von `update()` muß sicherstellen, daß so etwas Ähnliches geschieht.

```
import java.awt.*;
public class PendelAppl2 extends java.applet.Applet implements Runnable
{
 // Instanzvariable
 // Position vom Zentrum des schwingenden Pendels
 int x, y;
 //
 double thetaMax = (double)  0.35;
 double thetaMin = (double) –0.35;
 // Die initiale Position vom Pendel
 double theta = (double) 0.;
 //
 double wechsel = (double) 0.01;
 //
 int xStart = 150, yStart = 20;
 // Radius des Pendels
 double r = (double) 200;
 // Durchmesser des Balls
 int d = 20;
 Thread faden;
 int xAlt, yAlt;
 // Methoden
 public void init()
 {
  setBackground(Color.white);
 }
 public void start()
 {
  if (faden == null)
  {
   faden = new Thread(this);
   faden.start();
  }
 }
 public void stop()
 {
  if (faden != null)
  {
   faden.stop();
   faden = null;
  }
 }
 public void run()
 {
  while (true)
  {
   x = xStart + (int)(r * Math.sin(theta));
   y = yStart + (int)(r * Math.cos(theta));
   if ((theta >= thetaMax) | (theta <= thetaMin))
    wechsel = -wechsel;
   theta += wechsel;
   repaint();
   try { Thread.sleep(10); }
   catch(InterruptedException e) {}
  }
 }

 public void update(Graphics g)
```

```
 {
  g.setColor(Color.Yellow);
  g.drawLine(xStart,yStart,xAlt,yAlt);
  g.fillOval(xAlt-d/2,yAlt-d/2,d,d);
  g.setColor(Color.blue);
  g.drawLine(xStart,yStart,x,y);
  g.setColor(Color.red);
  g.fillOval(x-d/2,y-d/2,d,d);
  paint(g);
 }

 public void paint(Graphics g)
 {
  xAlt = x;
  yAlt = y;
 }
}
```

2. Double Buffering

Mit „double buffering“ wird eine zweite Oberfläche geschaffen, in der alles vorgezeichnet und dann auf einmal in die Zeichnungsoberfläche des Applet ausgegeben wird.

```
import java.awt.*;
public class Pendel extends java.applet.Applet implements Runnable
{
 // Instanzvariable
 // Position vom Zentrum des schwingenden Pendels
 int x, y;
 //
 double thetaMax = (double)  0.35;
 double thetaMin = (double) –0.35;
 // Die initiale Position vom Pendel
 double theta = (double) 0.;
 //
 double wechsel = (double) 0.01;
 //
 int xStart = 150, yStart = 20;
 // Radius des Pendels
 double r = (double) 200;
 // Durchmesser des Balls
 int d = 20;
 Thread faden;
 int xAlt, yAlt;
 Image backgroundImage;
 Graphics backgroundGraphics;
 // Methoden
 public void init()
 {
  setBackground(Color.white);
  backgroundImage = createImage(this.size().width,
                                this.size().height);
  backgroundGraphics = backgroundImage.getGraphics();
 }
 public void start()
 {
  if (faden == null)
  {
```

```
   faden = new Thread(this);
   faden.start();
  }
 }
 public void stop()
 {
  if (faden != null)
  {
   faden.stop();
   faden = null;
  }
 }
 public void run()
 {
  while (true)
  {
   x = xStart + (int)(r * Math.sin(theta));
   y = yStart + (int)(r * Math.cos(theta));
   if ((theta >= thetaMax) | (theta <= thetaMin))
    wechsel = -wechsel;
   theta += wechsel;
   repaint();
   try { Thread.sleep(10); }
   catch(InterruptedException e) {}
  }
 }
 public void update(Graphics g)
 {
  paint(g);
 }
 public void paint(Graphics g)
 {
  backgroundGraphics.setColor(Color.white);
  backgroundGraphics.drawLine(xStart,yStart,xAlt,yAlt);
  backgroundGraphics.fillOval(xAlt-d/2,yAlt-d/2,d,d);
  backgroundGraphics.setColor(Color.blue);
  backgroundGraphics.drawLine(xStart,yStart,x,y);
  backgroundGraphics.setColor(Color.red);
  backgroundGraphics.fillOval(x-d/2,y-d/2,d,d);
  g.drawImage(backgroundImage,0,0,this);
  xAlt = x;
  yAlt = y;
 }
}
```

5.1.6 Das Laden und Anzeigen von Bildern

Den Umgang mit Bildern ermöglicht die Klasse `Image` des Pakets `java.awt`. In einem Applet können Methoden der Klassen Applet und Graphics zum Laden und Anzeigen von Bildern herangezogen werden. Bilder werden als seperate Dateien außerhalb der .class-Dateien von Java gespeichert. Falls die Image-Klasse verwendet wird, muß das Bild im Format .GIF oder .JPG vorliegen.

Laden von Bildern. Es erfolgt mit der Methode `getImage()` aus der Applet-Klasse, die mit einem oder zwei Argumenten aufgerufen werden kann:

- Aufruf von getImage() mit einem Argument (ein Objekt vom Typ URL[53])
- Aufruf mit zwei Argumenten (Basis URL des Bilds (URL-Objekt)) und ein String, der den relativen Pfad oder den Dateinamen des aktuellen Bilds angibt.

Die Klasse Applet besitzt zwei Methoden zum Erzeugen einer Basis-URL ohne Angaben fester Adressen im Programm:

- die Methode getDocumentBase() gibt ein URL-Objekt zurück, das den Ordner (das Verzeichnis) repräsentiert, die die Webseite mit dem Applet enthält.
- die Methode getCodeBase() gibt ein Verzeichnis (Ordner) zurück, das das Verzeichnis repräsentiert, in dem sich die .class-Datei der Hauptklasse des Applet befindet.

Ausgabe von Bildern. Mit der Methode drawImage() der Graphics-Klasse kann ein Bild, das in ein Image-Objekt geladen wurde, angezeigt werden. drawImage() hat 4 Argumente:

- das Image-Objekt, das angezeigt werden soll
- die x- und y-Koordinate
- das Schlüsselwort this

Mit paint() kann das Bild zur Anzeige gebracht werden:

```
public paint(Graphigs g)
{
 g.drawImage(imageObjekt, xKoord, yKoord, this);
}
```

Bsp.:

```
      import java.awt.*;
      import java.applet.*;
      public class ZeichneBild extends Applet
      {
       private Image bild;
       public void init()
       {
        bild = getImage(getDocumentBase(),"B04240900.jpg");
        resize(250, 200);
       }
       public void paint(Graphics g)
       {
        int xPos = 10;  g.drawImage(bild,xPos,10,this);
       }
}
```

[53] Adressen im World Wide Web werden durch URL-Objekte repräsentiert. Die Klasse URL (Uniform Resource Locator) ist Teil des Pakets java.net

5.1.7 Die Ausgabe von Sound

Sound-Ausgabe in Applets

Das JDK bietet Möglichkeiten zur Ausgabe von Sound[54] an. Die Ausgabe von Sound kann über zwei Methoden der Klasse Applet erfolgen:

```
public void play(URL url)
public void play(URL url, String name)
```

hier kann entweder die URL einer Sound-Datei oder die Kombination von Verzeichnis-URL und Dateinamen angegeben werden. Die Übergabe der URLs geschieht über die Applet-Methoden:

```
public URL getCodeBase()
public URL getDocumentbase()
```

Die Methoden liefern eine URL des Verzeichnisses, aus dem das Applet gestartet wurde bzw. in dem die aktuelle HTML-Seite liegt. Der Nachteil dieser Vorgehensweise ist, daß die Sound-Datei bei jedem Aufruf neu geladen werden muß.

```
public getAudioClip(URL url, String name)
```

Hier wird ein Objekt der Klasse AudioClip beschafft, das dann abgespielt werden kann. AudioClip stellt drei Methoden zur Verfügung:

```
public void play()
```

startet die zuvor geladene Sound-Datei und spielt sie genau einmal ab.

```
public void loop()
```

startet die zuvor geladene Sound-Datei und spielt den Sound in einer Endlosschleife immer wieder ab.

```
public void stop()
```

Darüber kann die zuvor mit loop() iniziierte Schleife beendet werden.

Bsp.

```
import java.net.*;
import java.applet.*;
public class PR33602
{
 public static void main(String args[])
 {
  if (args.length >= 1)
  {
   try {
        URL url = new URL(args[0]);
        AudioClip clip = Applet.newAudioClip(url);
        clip.play();
        try {
```

[54] Das JDK 1.2 ermöglicht die Soundausgabe in Applets und Applikationen. Frühere Versionen gestatten Soundausgabe nur für Applets. Die Ausgabe war auf Sound beschränkt, die im AU-Format (stammt aus der Sun-Welt und legt ein Sample im Format 8 Bit Mono, Sampling-Rate 8 kHz, m-lawKompression ab) vorliegen mußte. Seit dem JDK 1.2 werden auch die Sample-Formate WAV und AIFF sowie die Midi-Formate Typ 0 und Typ 1 und RMF unterstützt. Zudem gibt es einige Shareware- oder Freeware-Tools, die zwischen verschieden Formaten konvertieren können (z.B. CoolEdit oder Gold-Wave).

```
            Thread.sleep(10000);
           }
        catch (InterruptedException e)
        {}
        }
        catch (MalformedURLException e)
        {
         System.out.println(e.toString());
        }
  }
 }
}
```

5.1.8 Applets in Archiven

Bisher wurden Applet-Klassen auf dem Server abgelegt. Alle Klassen, die das Applet benötigt, werden dann vom Server geladen, wenn sie zum ersten Mal verwendet werden. Das führt zu lästigen Verzögerungen. In Java gibt es eine Möglichkeit, Applets in komprimierte Archive zu laden. Solche Archive werden mit einem Zip-Programm gepackt und enthalten alle Klassen, die zum Ablauf nötig sind. Die Archive werden komplett vor dem Start des Programms übertragen. Das JDK stellt ein werkzeug mit dem Namen **`jar`** bereit[55]: `jar cf archive.jar *.class ...`
Falls das Applet eigene Pakete verwendet, so sollten diese Klassen in den entsprechenden Unterverzeichnissen ebenfalls mit eingepackt werden. Das Applet-Tag in der HTML-Datei muß dann folgendermaßen erweitert werden:
<APPLET CODE = “Appletname.class“ WIDTH=“...“ HEIGHT=”…” ALLIGN=”BOTTOM”
ARCHIVE=”archive.jar”>
Archive können auch für locale Applikationen herangezogen werden. Dazu setzt man den CLASSPATH auf das Archiv und startet das Programm, z.B.:

```
javac HalloWelt.java
jar cf Hallowelt.jar HalloWelt.class
set CLASSPATH=HalloWelt.jar
del HalloWelt.class
java HalloWelt
```

Die Applikation bezieht dann alle Klassen aus dem Archiv. Diese Methode eignet sich besonders, um Applikationen auf anderen rechnern zu installieren. Es ist nichts anderes zu tu, als das jar-File auf dem Rechner zu kopieren und vor dem Programmstart den CLASSPATH zu setzen.

[55] Man kann auch andere Komprimierer verwenden, die das „zip“-Format benutzen, z.B. das freie Paket von InfoZip, das die Programme zip und unzip enthält. Unter Unix und Windows sieht dann die Komprimierung so aus: zip archive.jar *.class. Der Nachteil solcher Komprimierer ist: Zum Unterschied zu jar wird keine „Manifest“-Datei erzeugt, die alle Informationen über Klassen enthält.

5.2. Konzept des Networking

Das Konzept des Networking beruht auf dem Client/Server-Modell. Beim Networking sind meistens zwei Prozesse beteiligt:

- Der Client-Prozeß, der die Kommunikation zwischen Prozessen initiiert.
- Der Server-Prozeß, der vom Client-Prozeß zur Ausführung bestimmter Dienste benötigt wird.

Es können auch mehrere Client- als auch mehere Server-Prozesse an der Kommunikation beteiligt sein. Die Kommunikation zwischen Prozessen muß nicht zwangsläufig über ein physisches Netzwerk erfolgen, die Prozesse können auch auf einem Rechner in Ausführung sein[56]. Die Kommunikation zwischen Client und Server kann auf 4 verschiedene Arten erfolgen:

- Kommunikation mit einem CGI-Skript
- Kommunikation über Sockets und Datagramms
- Kommunikation mit Hilfe von CORBA (Common Object Request Broker Architecture)
- Kommunikation mit Hilfe von RMI (Remote Method Invocation)

Das Paket `java.net` stellt Klassen für die Implementierung von Internet-Anwendungen zur Verfügung. Zentraler bestandteil des Pakets sind die Socket-Klassen zum Aufbau von Verbindungen über Sockets und Klassen, mit denen Verbindungen unter Verwendung von URLs aufgebaut werden können.

5.2.1 Networking mit URLs

Neben der Mödlichkeit eine Kommunikation über Sockets, Datagramms, CGI, RMI oder CORBA[57] zu initiieren, besteht in Java die Möglichkeit mit URLs zu arbeiten. Hierfür bildet die Klasse **URL** des Pakets `java.net` mit ihren Methoden die Basis:
Im WWW werden Ressourcen über URLs (Universe Resource Locator) identifiziert. Eine URL besteht aus:

Einem Protokollnamen, z.B. http (HTML), file (lokale Dateien), ftp (Dateitransfer), rmi (Remote Method Invocation) oder jdbc (Java Database Connectivity), dem ein Dopplelpunkt, zwei Schrägstriche und optional ein Hostname bzw. eine Hostadresse und eine Portnummer folgt. Fehlt die Angabe des Rechners, wird der aktuelle Rechner (localhost) benutzt, bei Ports bekannte Standardnummern. Schließlich folgt eine Bezeichnung der Ressource, typischerweise unter Angabe eines Pfads.

Java implementiert das Konzept eines **Uniform Resource Locator** durch eine eigene Klasse URL.
Erzeugen von URL-Objekten. Am einfachsten ist es, über eine String-Repräsentation der URL-Adresse zu gehen, z.B.: `URL fhURL = new URL("http://www.fh-regensburg.de/");`
Diese URL wurde mit dem Konstruktor

```
public URL(String urlAddr) throws MalformedURLException
```

erzeugt. Ein anderer Konstruktor ist

[56] In jedem Fall muß jedoch die Netzwerksoftware (TCP/IP-Protokoll) auf dem Rechner vorhanden sein, die auch ohne Existenz eines physischen Netzwerks diese Software für die Kommunikation benötigt wird.
[57] Common Request Object Broker Architecture

```
public URL(URL urlObj, String … ) throws MalformedURLException
// erzeugt relativ zur URL ein neues URL-Objekt.
```

Bsp.: Zugiff auf die FH-Homepage

```
import java.net.*;
import java.io.*;
public class OeffneURLStrom
{
 public static void main(String args[])
 {
  try {
       String s;
       URL fhURL = new URL("http://www.fh-regensburg.de/");
       BufferedReader ein = new BufferedReader(
              new InputStreamReader(fhURL.openStream()));
       while ((s = ein.readLine()) != null)
          System.out.println(s);
       ein.close();
      }
  catch(MalformedURLException e)
  { System.out.println("MalformedURLException: " + e); }
  catch(IOException e)
  { System.out.println("IOException: " + e); }
 }
}
```

Die Klasse URL besitzt auch Konstruktoren, die die Angabe der Komponenten von der Adresse (also Zugriffsart, Hostname und Dateiadresse getrennt) akzeptieren:

```
public URL(String protocol, String host, int port, String file) throws
                                                    MalformedURLException
public URL(String protocol, String host, String file) throws
                                                     MalformedURLException
```

Jeder der Konstruktoren wirft eine `MalformedURLException`, wenn der Parameter im Konstruktor entweder null ist oder er ein unbekanntes Protokoll beschreibt.
Zugriff auf Daten über eine URL. Es gibt zwei Möglichkeiten über eine URL bzw. über eine URLConnection. Beide Wege benutzen Streams. Jedes URL-Objekt besitzt die Methode openStream(), die einen InputStream zum Weiterverarbeiten liefert.

```
final InputStream openStream() throws IOException
// öffnet eine Verbindung zum Server und liefert einen InputStream zurück
URLConnection openConnection() throws IOException
// liefert ein URLConnection-Objekt, das die Verbindung zum entfernten
// Server vertritt. openConnection() wird vom Protokoll-Handler immer
// dann aufgerufen, wenn eine neue Verbindung geöffnet wird.
```

Verweist die URL auf eine Textdatei, dann erweitert man den InputStream zu einem `BufferedReader`, da dieser die `readLine()`-Methode besitzt.
Bsp.: Eine Antwort (HTML-Seite) auf eine Suchfrage

```
import java.io.*;
```

[58] pr52100

```
import java.net.*;
public class GoogleSucher
{
 public static void main(String args[]) throws IOException,
                                                MalformedURLException
 {
  String s = "";
  if (args.length == 0) s = "Stephan Meier in Regensburg";
  else
   for (int i = 0; i < args.length; i++) s += args[i] + " ";
  s.trim();
  s = "p=" + URLEncoder.encode(s);
  System.out.println(s);
  URL u =  new URL("http://de.google.yahoo.com/bin/query_de?" + s);
  BufferedReader ein = new BufferedReader(
     new InputStreamReader(u.openStream()));
  String zeile, antwort = null;
  while ((zeile = ein.readLine()) != null) antwort += zeile + "\n";
  System.out.print(antwort);
 }
}
```

URLs in Applets. Die Klasse Applet hat zwei Methoden mit denen man eine Basis-URL erzeugen kann, ohne eine feste Adresse im Programm anzugeben:

- Die Methode `getDocumentBase()` gibt ein URL-Objekt zurück, welches das Verzeichnis repräsentiert, das die Webseite mit dem Applet enthält.
- Die Methode `getCodeBase()` gibt ein URL-Objekt zurück, das den Ordner repräsentiert, in dem sich die .class-Datei der Hauptklasse des Applet befindet.

Die Applet-Klasse bietet eine Methode mit dem Namen `getImage()` an, mit der ein Bild in ein Image-Objekt geladen werden kann. Es gibt zwei Möglichkeiten, diese Methode zu verwenden:

- Die Methode `getImage()`, aufgerufen mit einem Argument (ein Objekt vom Typ URL), lädt das Bild mit dieser URL
- Die Methode `getImage()`, aufgerufen mit zwei Argumenten (der Basis-URL und einem String, der den relativen Pfad oder Dateinamen des aktuellen Bilds ausgibt).

Bsp.:

```
import java.applet.Applet;
import java.awt.*;
import java.net.*;
public class AppletURL extends Applet
{
 Image bild;
 public void init()
 {
  URL u1 = getDocumentBase(); System.out.println(u1);
  try { URL u2 = new URL(u1,"images/scratch1.gif");
        System.out.println(u2);
        bild = getImage(getCodeBase(),"images/B04240900.jpg");
      }
  catch (MalformedURLException e) { System.out.println(e); }
 }
```

[59] vgl. pr51800

```
 public void paint(Graphics schirm)
 {
   int iBreite = bild.getWidth(this); int iHoehe  = bild.getHeight(this);
   schirm.drawImage(bild,10,10,iBreite/4,iHoehe/4,this);
 }
}
```

5.2.2 Kommunikation mit einem CGI-Skript

CGI ist die Beschreibung einer Schnittstelle, mit der externe Programme mit Informations-Servern, meistens Web-Servern, Daten austauschen. Die ausgeführten Programme werden CGI-Programme genannt und können in den unterschiedlichsten Programmiersprachen verfasst sein. Häufig sind es Shell- oder Perl-Skripte[60].
Die CGI-Programme werden von einem Server durch eine URL angesprochen. Der Browser baut eine Verbindung zum Server auf, und dieser erkennt an Hand des Pfads in der URL, ob es sich um eine normale Web-Seite handelt oder um ein Skript. Falls es ein Skript ist, dann führt der Server das Skript aus, das eine HTML-Datei erzeugt. Diese wird übertragen und im Browser dargestellt. Der Aufrufer einer URL merkt keinen Unterschied zwischen erstellten, also dynamischen und statischen Seiten.

5.2.3 Kommunikation über Sockets und Datagramms

Socket[61]: streambasierte Schnittstelle zur Kommunikation zweier Rechner in einem TCP/IP-Netz. Das Übertragen von Daten über eine Socket-Verbindung ähnelt dem Zugriff auf eine Datei:

- Zunächst wird eine Verbindung aufgebaut
- dann werden Daten gelesen / oder geschrieben
- Schließlich wird die Verbindung wieder abgebaut.

Werden Rechner verbunden, so implementiert jeder Rechner einen Socket. Derjenige, der Daten empfängt (Client), öffnet eine Socket-Verbindung zum Horchen und derjenige, der sendet, öffnet eine Verbindung zum Senden (Server). Damit der Empfänger den Sender auch hören kann, muß dieser durch eine eindeutige Adresse als Server ausgemacht werden. Er bekommt also eine IP-Adresse im Netz und eine ebenso eindeutige Port-Adresse. Der Port ist so etwas wie eine Zimmernummer: Die Adresse bleibt dieselbe, aber in jedem Zimmer sitzt einer und macht seine Aufgaben.
Für jeden Dienst (Service), den ein Server bereitstellt, gibt es einen Port[62]. Eine Port-Nummer ist eine 16-Bit-Zahl und in die Gruppen System und Benutzer eingeteilt.

---

[60] Oft wird die Bezeichnung CGI-Skript verwendet. Die Unterscheidung „Skript oder Programm“ ist in CGI schwammig. Eine compilierte Quelldatei wird üblicherweise als Programm bezeichnet. Programme, die mit einem Interpreter arbeiten, nennt man Skript.

[61] Sockets wurden Anfang der 80er Jahre für die Programmiersprache C entwickelt und mit Berkeley UNIX 4.1/4.2 allgemein eingeführt.

[62] Diese Adressen werden von der IANA (Internet Assigned Numb er Authority) vergeben. Die IANA ist der zentrale Koordinator für die IP-Adressen, Domain Namen, MIME Typen und viele andere Parameter, u.a.a. Portnummern.

Sockets aus der Sicht einer Client- bzw. Server-Anwendung werden durch die beiden Klassen `Socket` und `ServerSocket` repräsentiert.

StreamSockets

Ein Stream baut eine feste Verbindung zu einem Rechner auf. Das besondere ist: Die Verbindung bleibt für die Dauer der Übertragung bestehen.
Bsp.:

1. Ein einfacher Server, der alles „nachplappert“, was der Client sendet.

```
// MultiPlapperServer.java
// Einfacher Server, der nachplappert,
// was der Client sendet
import java.io.*;
import java.net.*;
public class PlapperServer
{
 // Auswahl von einem Port, ausserhabl des Bereichs 1 bis 1024
 static final int PORT = 8080;
 public static void main(String args[]) throws IOException
 {
  // ServerSocket fuer einen bestimmten Port
  ServerSocket s = new ServerSocket(PORT);
  System.out.println("Start: " + s);
  try {
       // Blockbildung bis eine Verbindung hergestellt ist
       Socket socket = s.accept();
       // accept() blockiert solange, bis sich ein Client bei
       // der Serveranwendung anmeldet
       try {
            System.out.println(
             "Verbindung akzeptiert: " + socket);
            BufferedReader ein =
             new BufferedReader(new InputStreamReader(
              socket.getInputStream()));
            PrintWriter aus =
             new PrintWriter(
              new BufferedWriter(new OutputStreamWriter(
              socket.getOutputStream())),true);
            while (true)
            {
             String str = ein.readLine();
             if (str.equals("ENDE")) break;
             System.out.println("Echo: " + str);
             aus.println(str);
            }
           }
       finally {
                System.out.println("Schliessen...");
                socket.close();
               }
       }
  finally {
            s.close();
           }
 }
```

```
}
```

2. Ein einfacher Client, der Zeilen zum Server sendet und einliest

```
// PlapperClient.java
// einfacher Client, der Zeilen zum Server sendet
// und einliest
import java.io.*;
import java.net.*;
public class PlapperClient
{
 public static void main(String args[]) throws IOException
 {
  // "null" als Parameter produziert einen speziellen lokale
  // Rueckgabe und dient zum Test fuer die Netzverarbeitung
  InetAddress adr = InetAddress.getByName(null);
  // Alternativen:
  // InetAddress adr = InetAddress.getByName("localhost");
  // InetAddress adr = InetAddress.getByName("127.0.0.1");
  System.out.println("Adresse = " + adr);
  Socket socket = new Socket(adr,PlapperServer.PORT);
  try {
      BufferedReader ein =
       new BufferedReader(new InputStreamReader(
        socket.getInputStream()));
      PrintWriter aus =
       new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(
         socket.getOutputStream())),true);
      for (int i = 0; i < 10; i++)
      {
       aus.println("Wie sterben " + i);
       String str = ein.readLine();
       System.out.println(str);
      }
      aus.println("ENDE");
      }
  finally {
           System.out.println("Schliessen ...");
           socket.close();
          }
 }
}
```

5.2.4 Kommunikation mit Hilfe von CORBA

CORBA (Common Object Request Object Architecture) ist ein Standard zur Integration von Anwendungen und Entwicklung verteilter Systeme. CORBA dient dazu, Anwendungen in einer heterogenen, verteilten Umgebung zu integrieren. Eine Client/Server-Kommunikation kann in Java auch über CORBA erfolgen. Hierzu bietet SunSoft einen komplett in Java implementierten Object Request Broker an. Ein Object Request Broker dient als Vermittler („Postverteiler") zwischen Objekten. Zur Object Management Architecture (OMA) der Object Management Group (OMG)[63] gehört auch eine Schnittstellen-Definitionssprache (IDL[64]). Java überstellt die Angaben in der Schnittstellensprache über `idltojava.exe` in Java-Quellcode.

Ein einfaches Beispiel.
1. Die Angaben in der IDL befinden sich in der Datei HalloApp.idl

```
module HalloApp {
interface Hallo
{
 string sageHallo();
};
};
```

2. Aufruf von idltojava.exe in folgender Form: `idltojava HalloApp.idl`
Im Unterverzeichnis HalloApp wurden folgende Dateien erzeugt:

```
Hallo.java
HalloHelper.java
HalloHolder.java
_HalloImplBase.java
_HalloStub.java
```

3. Übersetzen der Client-Anwendung[65]: `javac HalloClient.java HalloApp\*.java`
4. Übersetzen der Server-Anwendung[66]: `javac HalloServer.java HalloApp\*.java`
5. Ablauf der Client/Server -Anwendung
   1. Start des Java IDL Nameserver von einem DOS-Fenster: `tnameserv –ORBInitialPort 1050`
   2. Aufruf von einem 2. DOS-Fenster: `java Halloserver –ORBInitialPort 1050`
   3. Aufruf von einem 3. DOS-Fenster: `java HalloClient –ORBInitialPort 1050`
   4. Der Client gibt aus: `Hallo Welt!`

[63] OMG: gegründet von 8 Hardware-Herstellern und Software-Firmen mit dem Ziel der Standardisierung.
[64] Interface Definition Language
[65] vgl. pr52401
[66] vgl. pr52401

5.2.5 Kommunikation über RMI

Ähnlich wie die Kommunikation mit CORBA verfolgt auch **RMI** (Remote Method Invocation) den Ansatz, eine verteile Objektarchitektur zu realisieren. Im Gegensatz zu CORBA beruht RMI jedoch auf reiner Java-Sprachbasis und relisiert eine verteilte Objektarchitektur für eine homogene Java-Welt.

Remote-Methoden-Aufruf

RMI erlaubt die Erzeugung von Java-Objekten, derem Methoden von der virtuellen Maschine auf einem anderen Rechner aufgerufen werden können. Das System ist dem Remote Procedure Call (RPC) –Mechanismus ziemlich ähnlich.

Erstellen eines Remote-Objekts

Folgende Schritte dienen zum Aufbau des Systems:

1. Erstellen und Kompilieren von Java Code für „Interfaces“.
2. Erstellen und Kompilieren von Java Code für die Implementierungs-Klassen
3. Generieren von Stubs und Skeleton „class files“ aus den Implementierungs-Klassen
4. Erstellen des Java Code für das „request service host program“.
5. Entwicklung von Java Code für das „RMI client program“.
6. Installation und Ablauf vom RMI-System.

Bsp.:
1. Definition der Schnittstelle „Calculator Interface“. Das Calculator Interface definiert alle „remote features“, die vom Dienst angeboten werden. Die Methoden müssen alle ein „throws Remote-Exception-Statement“ besitzen, das etwaige Netzwerkprobleme auffängt, die zur Verhinderung der Kommunikation zwischen Client und Server führen können.

```
public interface Calculator extends java.rmi.Remote
{
 public long add(long a, long b) throws java.rmi.RemoteException;
 public long sub(long a, long b) throws java.rmi.RemoteException;
 public long mul(long a, long b) throws java.rmi.RemoteException;
 public long div(long a, long b) throws java.rmi.RemoteException;
}
```

Das „Calculator Interface“ definiert alle `remote features`, die vom Dienst angeboten werden. Es wird übersetzt mit: `javac Calculator.java`.
2. Implementierung des `remote service` CalculatorImpl über: `javac CalculatorImpl.java`

```
public class CalculatorImpl
  extends
 java.rmi.server.UnicastRemoteObject
  implements Calculator
{
 // Implementations must have an
 // explicit constructor
 // in order to declare the
```

```
 // RemoteException exception
 public CalculatorImpl()
  throws java.rmi.RemoteException
 {
  super();
 }
 public long add(long a, long b)
   throws java.rmi.RemoteException
 {
  return a + b;
 }
 public long sub(long a, long b)
  throws java.rmi.RemoteException
 {
  return a - b;
 }
 public long mul(long a, long b)
  throws java.rmi.RemoteException
 {
  return a * b;
 }
 public long div(long a, long b)
  throws java.rmi.RemoteException
 {
  return a / b;
 }
}
```

3. Generieren von Stub- und Skeleton „class files“ aus den Implementierungsklassen. Stubs und Skeletons werden erzeugt mit dem RMI-Compiler **`rmic`**: `rmic CalculatorImpl`

4. Server „CalculatorServer“

```
import java.rmi.Naming;

public class CalculatorServer
{
 public CalculatorServer()
 {
  try {
        Calculator c = new CalculatorImpl();
        Naming.rebind("rmi://localhost:1099/CalculatorService", c);
      } catch (Exception e)
        {
         System.out.println("Trouble: " + e);
        }
 }

 public static void main(String args[])
 {
  new CalculatorServer();
 }
}
```

5. Client: „CalculatorClient“

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
```

```
import java.rmi.NotBoundException;

public class CalculatorClient
{
 public static void main(String[] args)
 {
  try {
       Calculator c = (Calculator)
       Naming.lookup("rmi://localhost/CalculatorService");
       System.out.println( c.sub(4, 3) );
       System.out.println( c.add(4, 5) );
       System.out.println( c.mul(3, 6) );
       System.out.println( c.div(9, 3) );
      }
  catch (MalformedURLException murle)
  {
   System.out.println();
   System.out.println("MalformedURLException");
   System.out.println(murle);
  }
  catch (RemoteException re)
  {
   System.out.println();
   System.out.println("RemoteException");
   System.out.println(re);
  }
  catch (NotBoundException nbe)
  {
   System.out.println();
   System.out.println("NotBoundException");
   System.out.println(nbe);
  }
  catch (java.lang.ArithmeticException ae)
  {
   System.out.println();
   System.out.println("java.lang.ArithmeticException");
   System.out.println(ae);
  }
 }
}
```

6. Start im 1. Konsolenfenster aus dem Verzeichnis, in dem sich die Klassen befinden:
   `rmiregistry`

5.3 Konzept der Servlets

Das Java Server API dient ähnlich CGI zur Server-seitigen Programmausführung. Offiziell werden diese Anwendungen als **Servlet** bezeichnet. Im Gegensatz zur Erstellung eines eigenständigen Servers dienen Servlets dazu, die Funktionalität eines bestehenden Servers zu erweitern[67].
Ein Java-Servlet ähnelt einem CGI-Skript, weil es serverseitige Aufgaben übernimmt, ist aber auch mit einem Java-Applet vergleichbar. Nur agiert ein Servlet nicht in einem Web-Browser sondern auf einem Web-Server. Es besitzt keine grafische Oberfläche, da auf der Server-Maschine, auf der das Servlet läuft, niemand mit ihm agiert.
Seine Clients sind in der Regel Web-Browser. Sie können über einen Link im anzuzeigenden HTML-Code HTTP-Requests stellen. Das Servlet nimmt die Client-Anfragen entgegen und beantwortet sie. Das Servlet wird entweder beim Start des Web-Servers geladen oder bei seinem ersten Client-Aufruf geladen und gestartet und bleibt dann im Speicher. Dort wartet es auf Anfragen von Clients (z.B. HTTP-GET-Requests), um diese zu bearbeiten. Für diese Bearbeitung stehen alle Features von Java zur Verfügung. Ergebnisse werden dann den Clients zugesandt. Beendet werden Servlets in der Regel nie. Gründe für das Beenden eines Servlets können bspw. ein Shutdown des Web-Servers oder das Compilieren einer neuen Version sein.
Zur Entwicklung und zum Test von Servlets benötigt man einen Servlet fähigen Webserver[68] bzw. einen Servlet-Container.

Apache Tomcat.

Tomcat[69] ist die offizielle Referenzimplementierung. Tomcat kann zum Testen der Servlets und Java Server Pages sowohl als „stand-alone“-Applikkation eingesetzt oder als Ergänzung zum Apache-Server eingebunden werden.
Download und Installation. Der Tomcat Server liegt unter http://jakarta.apache.org/tomcat als komprimiertes Archiv zum Laden bereit. Nach dem Auspacken liegen im Pfad `jakarta-tomcat` unterschiedliche Verzeichnisse, wobei unter `/bin` die ausführbaren Skripte[70] zum Verwalten der Servers sind.
Damit der Server gestartet werden kann, muß die Umgebungs-Variable JAVA_HOME so gesetzt sein, dass sie auf das JDK zeigt.

```
set TOMCAT_HOME=d:\Programme\jakarta-tomcat-3.2.3
set JAVA_HOME=d:\jdk1.3.1
cd bin
startup
```

Ohne Veränderung der Voreinstellung installiert sich der Webserver auf dem lokalen Rechner auf Port 8080. Der aufruf `bin\startup` aktiviert direkt den Server.

---

[67] Hierfür stehen im JDK die Pakete java.servlet und java.servlet.http (spezifische Erweiterung von http) bere

[68] Sun verzeichnet auf ihrer Webseite http://java.sun.com/products/servlet/industry.html eine Liste von Servlet fähigen Servern. Ein Server ist genau dann Servlet fähig, wenn er die Java Servlet und Java Server Pages Spezifikation erfüllt. Die Servlet-Komponete kann dabei integraler Bestandteil des Servers sein oder auch ein Zusatz, der nachträglich zu installieren ist.

[69] Webadresse: http://jakarta.apache.org

[70] Batch-Dateien unter Windows

Ein Blick im Browser auf die lokale Adresse http://localhost:8080 zeigt die Startseite. Hier befinden sich Beispiele und die APIs für das Paket.

5.4 Java Server Pages (JSP)

5.4.1 Was sind Java Server Pages?

Sun Micosystems hat die Java Server Pages (JSP) für die Realisierung einer Präsentationsschicht einer Webanwendung entwickelt. Die JSP-Technik basiert auf dem Java-Servlet-API und dient im wesentlichen zur einfachen Erzeugung von HTML- und XML-Ausgaben eines Webservers. Wie PHP oder Perl können Autoren direkt HTML oder XML mit einer Skriptsprache mischen. Natürlich hat sich Sun für Java entschieden.
Servlets sind Server-Programme, die Webseiten erstellen. Das machen sie, in dem sie HTML-Anweisungen mit `println()` (oder ähnlichem) in den Ausgabestrom senden. JSP (Java Server Pages) geht das Problem genau anders herum an. Nicht ein Servlet kümmert sich um die Ausgabe des HTML-Codes, sondern eine HTML-Datei wird mit Java Code ergänzt, z.B.:

```
<HTML><BODY>
Hallo Anwender. Heute ist der
<% out.print(new java.util.Date()); %>
</BODY></HTML>
```

Der Server kann JSP von normalen HTML-Seiten unterscheiden und kompiliert mit Hilfe des JSP-Übersetzers[71] daraus ein Servlet und stellt dieses Serlet dar. Der Übersetzungsvorgang von JSP in ein Servlet muß nur einmal getätigt werden, dann nimmt der Servlet-Container direkt die übersetzte Klasse.
Eine JSP ist eigentlich ein spezielles Servlet, das durch eine JSP-Engine erzeugt wird. Der Servlet-Container nimmt eine HTTP-Anfrage an und ermittelt anhand eines URL-Mapping, dass es sich um eine Java Server Page handelt. Eine JSP-Engine wird aufgerufen und schaut in der Verwaltung nach, ob vielleicht schon eine kompilierte aktuelle JSP für die Anfrage vorliegt. Falls nicht, lädt sie aus dem Dateisystem die entsprechende Seite.
In der JSP-Datei befinden sich spezielle Tags, in denen die sog. Scriplets stehen. Diese beschreiben die Anwendungslogik der Servlets. Der Programmcode steuert dabei die Ausgabe. Er kann die gesamte Java-API nutzen, die serverseitig verfügbar ist. Daneben gibt es Vereinfachungen zum Ansprechen von JavaBeans. Da das einführen von eigenen Tags problematisch ist, hat Sun spezielle XML-Tags im Namensraum des JSP eingeführt, um mit speziellen XML-Tools den Seitenaufbau zu gestalten. Neben dem Inhalt der HTML-Datei (der Template genannt wird) lassen sich 3 unterschiedliche JSP-Konstrukte in eine Seite einbinden:

- Skript-Elemente. Sie enthalten java-Programmcode, der direkt in das Servlet wandert. Es gibt unterschiedliche Typen für Ausdrücke, Anweisungen und Deklarationen. Normale Anweisungen werden **Scriplets** genannt.
- Direktiven. Sie steuern die Struktur der Seite.

[71] Der JSP-Übersetzer ist ein Schwachpunkt, den hier muß ein Compiler eingebunden werden. Tomcat löst das Problem, in dem der Sun-Compiler in tools.jar verwendet wird. Für „nicht“-Compiler-Hersteller ist es schwierig eine Servlet / JSP-Umgebung anzubieten.

- Aktionen. Nutzen von vergefertigeten Komponenten wie Beans. Einbinden von externen Seiten und Weiterleitung an andere Seiten.

JSPs können mit Tomcat genutzt werden. Tomcat definiert zwei Teilprojekte mit den Namen `Catalina` und `Jasper`. Catalina ist für Servlets zuständig und Jasper für die Java Server Pages. Bei einer Installation sind beide Teile aktiv.
Bsp.: „Snoop“ unter `http://localhost:8080/examples/jsp/snp/snoop.jsp`.
Dieser Aufruf bezieht sich auf die Datei: `<jakarta-tomcat-Pfad>\webapps\examples\jsp\snp\snoop.jsp`.
Zum Test eigener JSP-Seiten kann das Verzeichnis benutzt werden, um den Pfad nicht anpassen zu müssen.

5.4.2 Skript-Elemente

Die Spezifikation unterscheidet unterschiedliche Skript-Elemente für Ausdrücke, Scriplets und Deklarationen. Allen gemeinsam ist, dass sie mit dem Tag `<%` beginnen und mit `%>` enden. Script-Tags können nicht geschachtelt werden.

`<% .. %>`. Eingebettete Javastücke besitzen dieses Format. Hier lassen sich beliebige Deklarationen tätigen und Ausgaben machen. Scriplets liegen zwischen den Tags <% und %>.
`<%! .. %>`. In dieser Umgebung werden variablen und Methoden definiert.
`<%= .. %>`. Hier steht ein Ausdruck, der in die Seite eingebaut wird.

5.4.3 Entsrechende XML-Tags

Für die entsprechenden Tags gibt es äquivalente XML-Anweisungen.
Für Scriplets:

```
<jsp:scriplet>
Programmcode
<jsp:scriplet>
```

Für Ausdrücke:

```
<jsp:expression>
Programmcode
</jsp:expression>
```

Für Deklarationen

```
</jsp:declaration>
Programmcode
</jsp:declaration>
```

5.4.4 Direktiven

Eine Direktive in JSP hat Einfluß auf die Struktur von Seiten. Die Direktiven werden in den Tags <%@ und %> eingeschlossen.

5.4.5 Aktionen

Aktionen in JSP verändern das Verhalten des Servlet-Containers. Externe Daten können eingebunden werden, Seiten umgeleitet werden, es kann auf externe Objekte verwiesen werden. Für einzelne aktionen existieren keine Tags, sie werden in XML-Anweisungen gesetzt. Dabei ist die interessanteste Möglichkeit die Nutzung von JavaBeans. Dafür wurde `jsp:useBeans:setProperty`, `jsp:getProperty` definiert.