Kapitel 10: Objektorientierte Datenbanksprachen und Erweiterungen von SQL

Von relationalen zu objektorientierten DBS

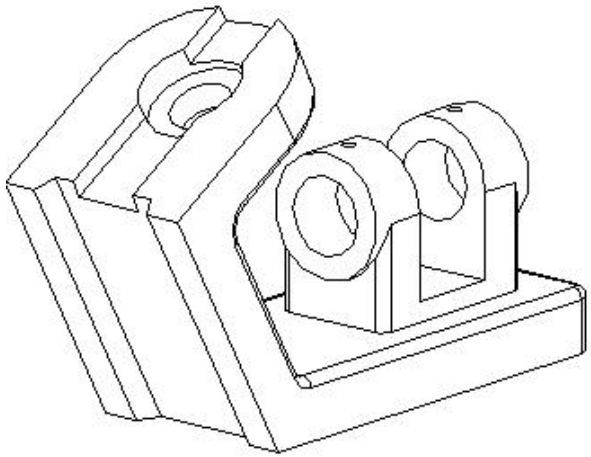
Schwachpunkte relationaler DBS
bei technisch-wissenschaftlichen Daten, z.B. bei:

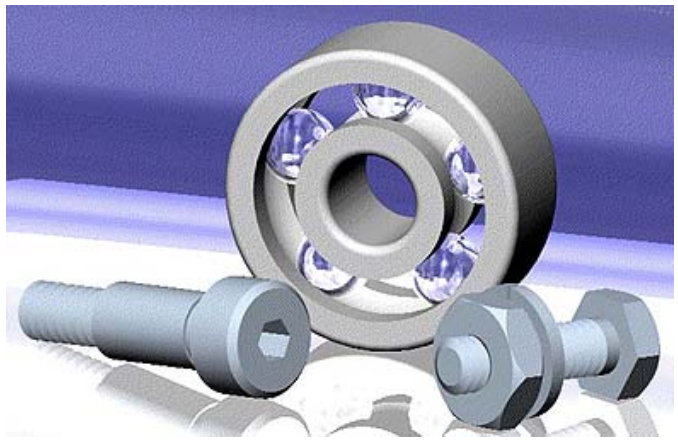
- CAD
- Geowissenschaften
- Desktop Publishing

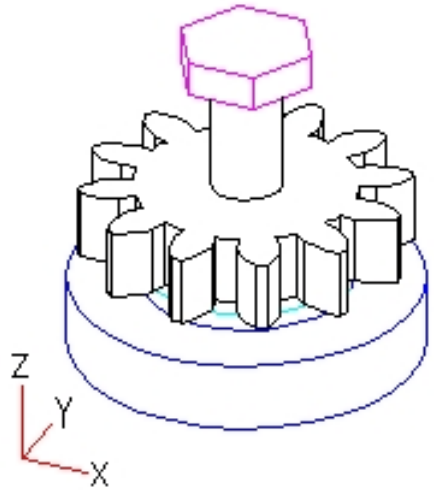
Forderungen an „postrelationale“ DBS:

- Unterstützung komplexer Objektstrukturen
- Erweiterbarkeit des DBS um anwendungsspezifische Methoden
- Wiederverwendbarkeit von Datenschemata und Methoden

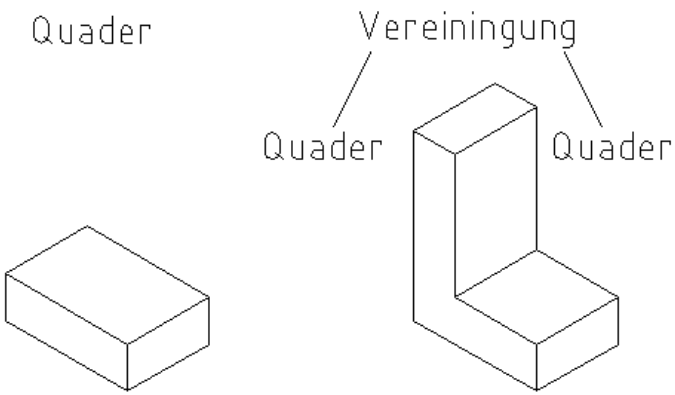
Beispiel 1: CAD (1)

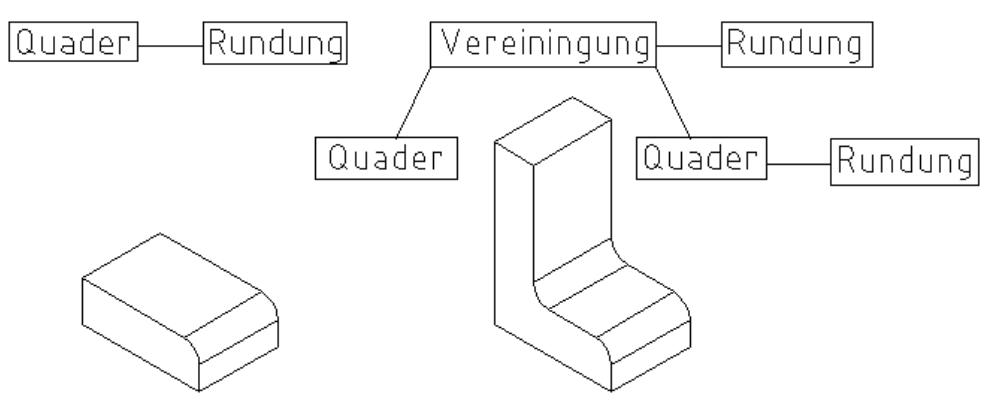




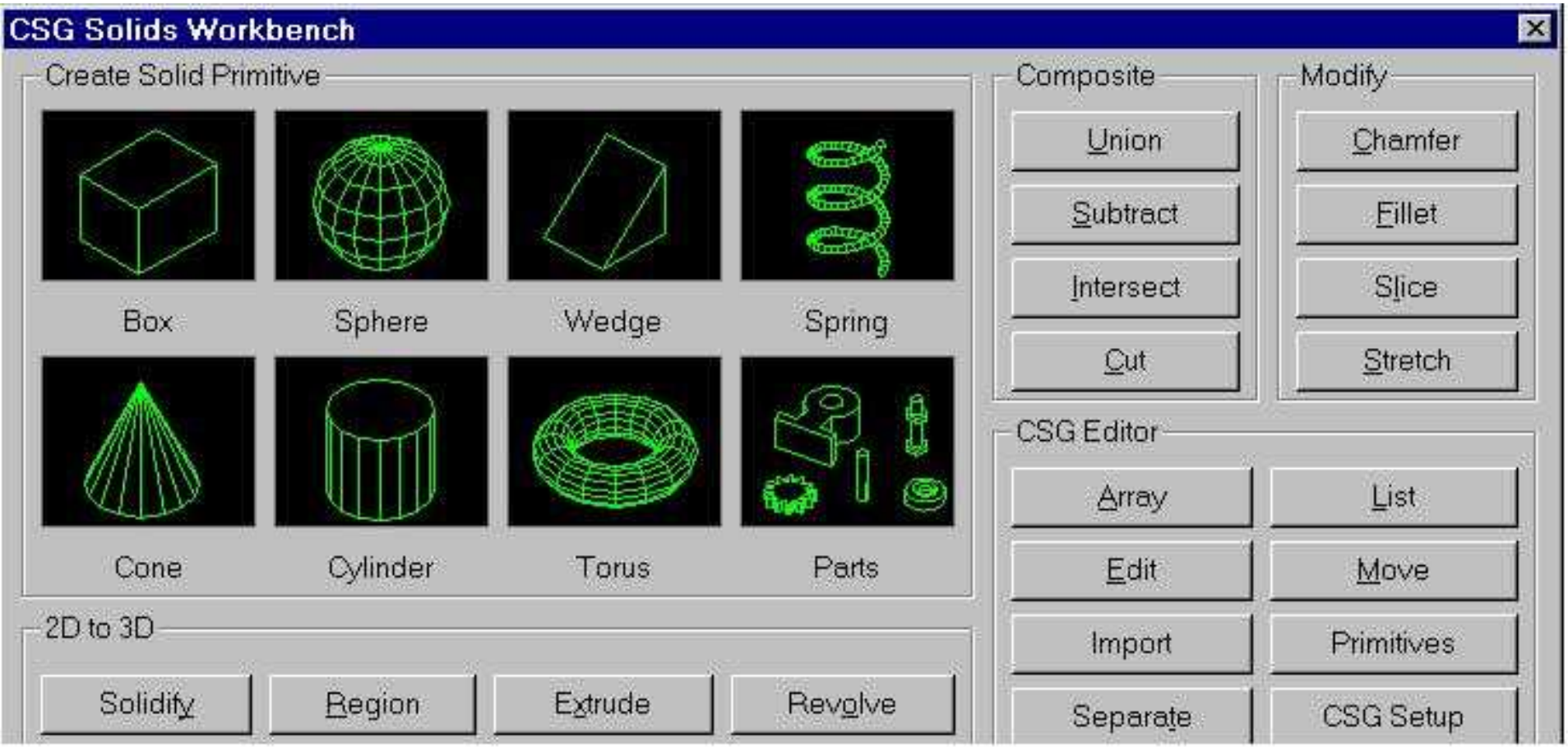


CSG-Modell (Constructive Solid Geometry):

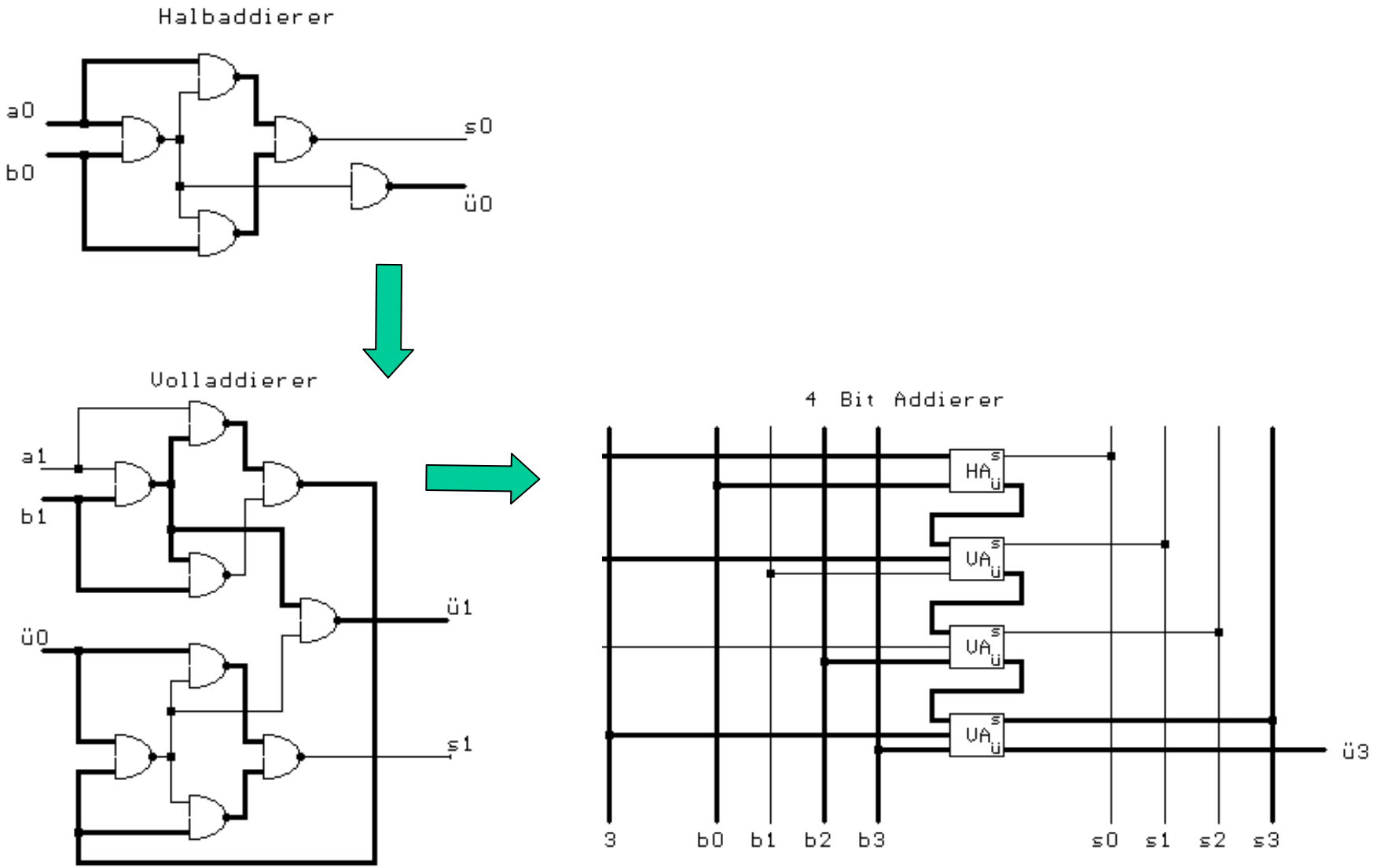




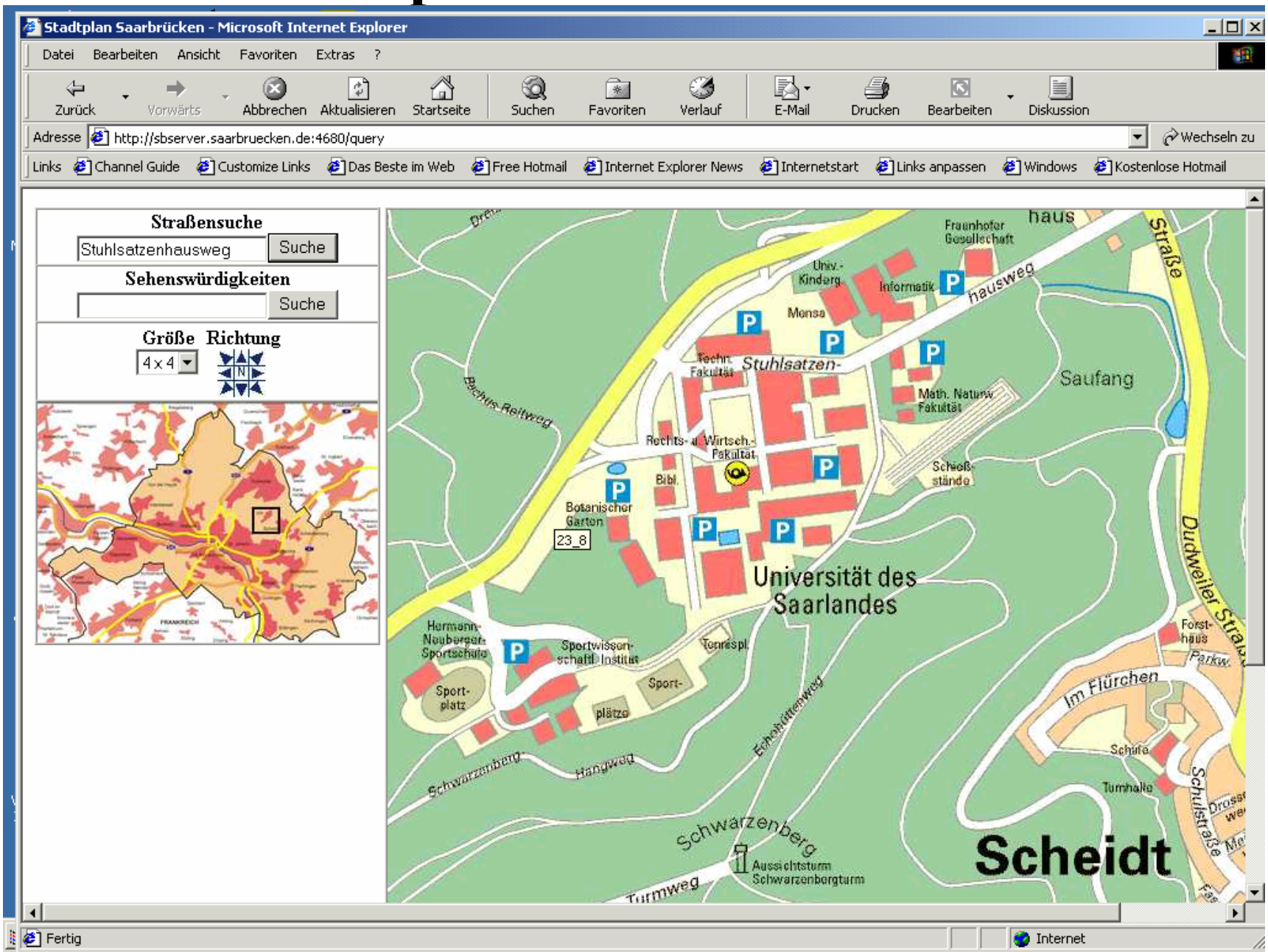
Beispiel 1: CAD (2)



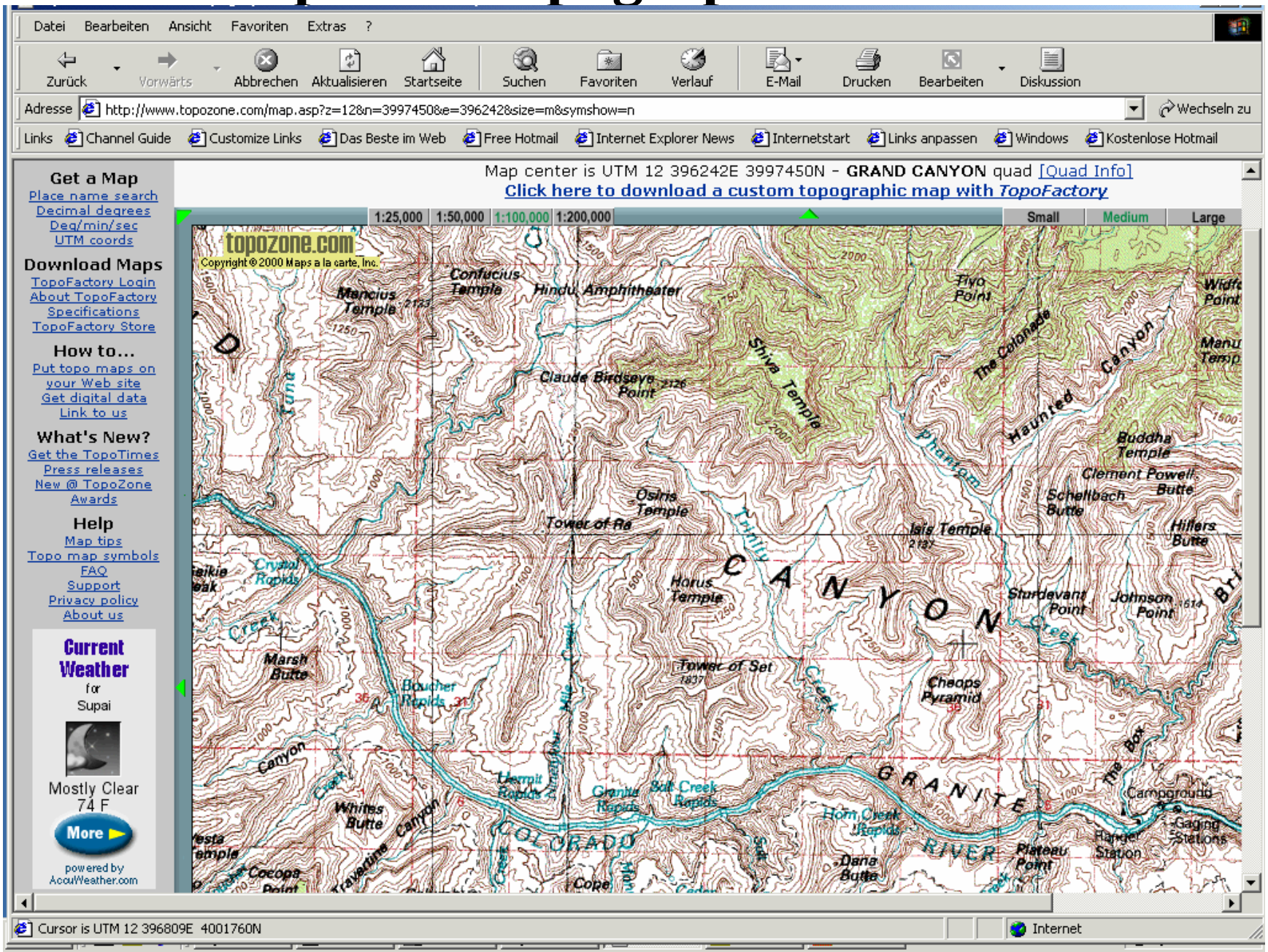
Beispiel 2: Schaltkreisentwurf



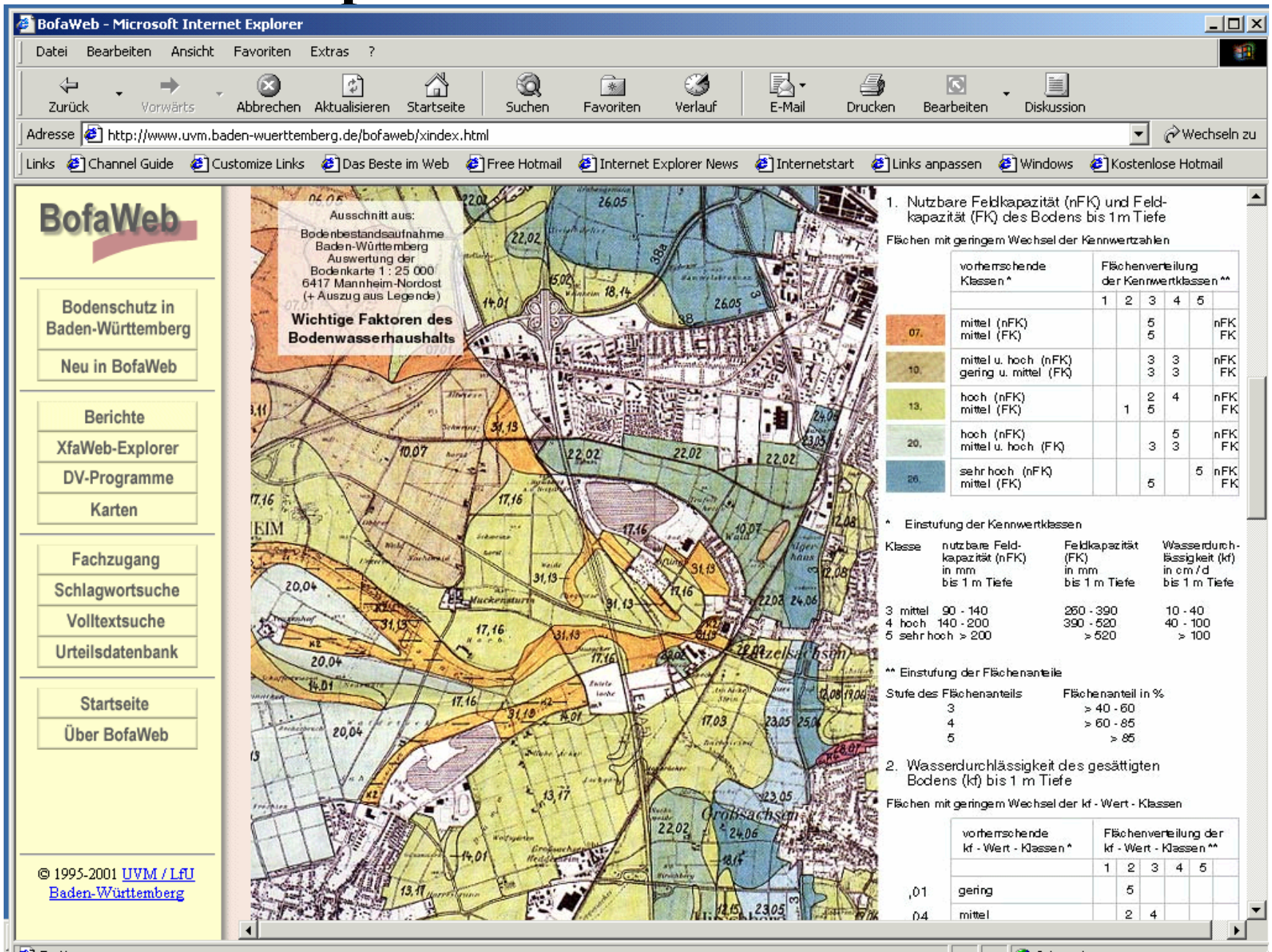
Beispiel 3: Straßenkarten



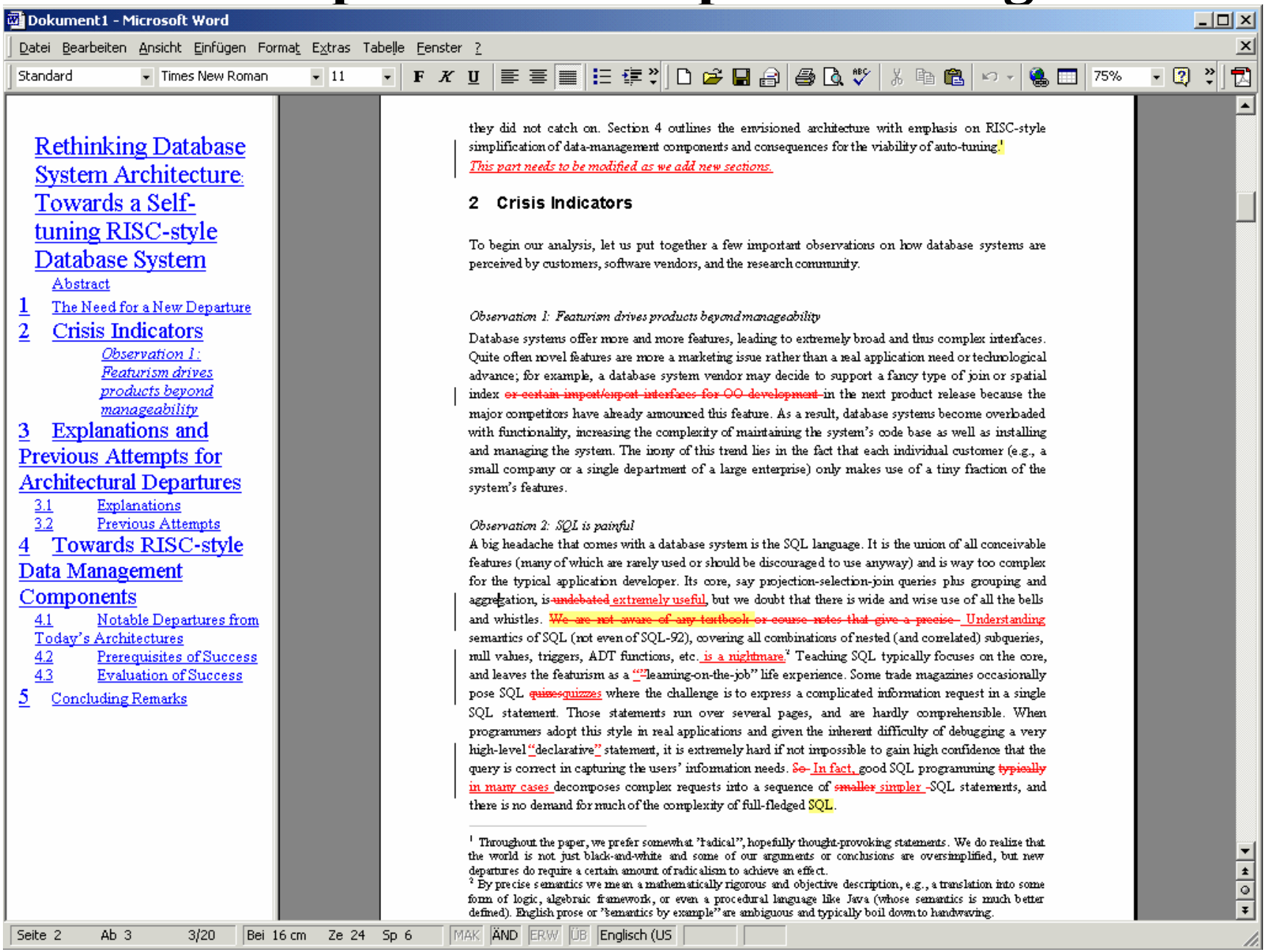
Beispiel 4: Topographische Karten



Beispiel 5: Thematische Karten



Beispiel 6: Desktop Publishing

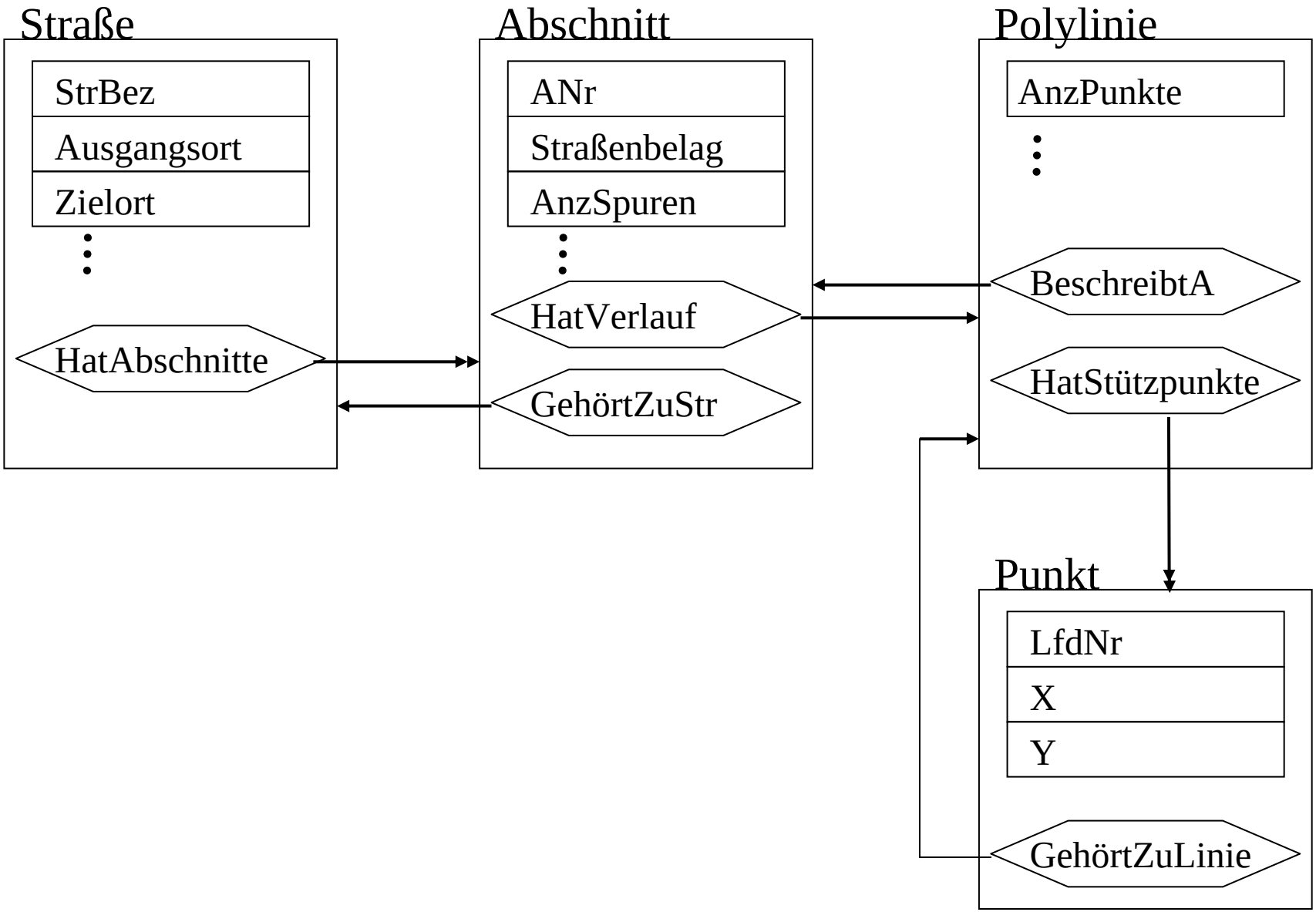


10.2.1 Komplexe Objekte

Objekt: Menge von Attributen (attribute, property, member)
Typ eines Objekts: Menge der Attribute des Objekts
Klasse: Menge von Objekten desselben Typs
OO-DB: Menge von Klassen

Typ eines Attributs:
- elementar (Integer, Real, String, Boolean)
- erzeugt mit Typkonstruktoren (mit Parametertypen):
    - Relationship T: Referenz auf Objekt vom Typ T
    - Struct<T1, ..., Tk>: Tupel von Attributen vom Typ T1, ..., Tk
    - Set<T>: Menge von Elementen des Typs T
    - Bag<T>: Multimenge von Elementen vom Typ T
    - List<T>: Liste von Elementen vom Typ T
    - Array<T>: Abbildung (eines Intervalls) von natürlichen Zahlen auf Elemente vom Typ T

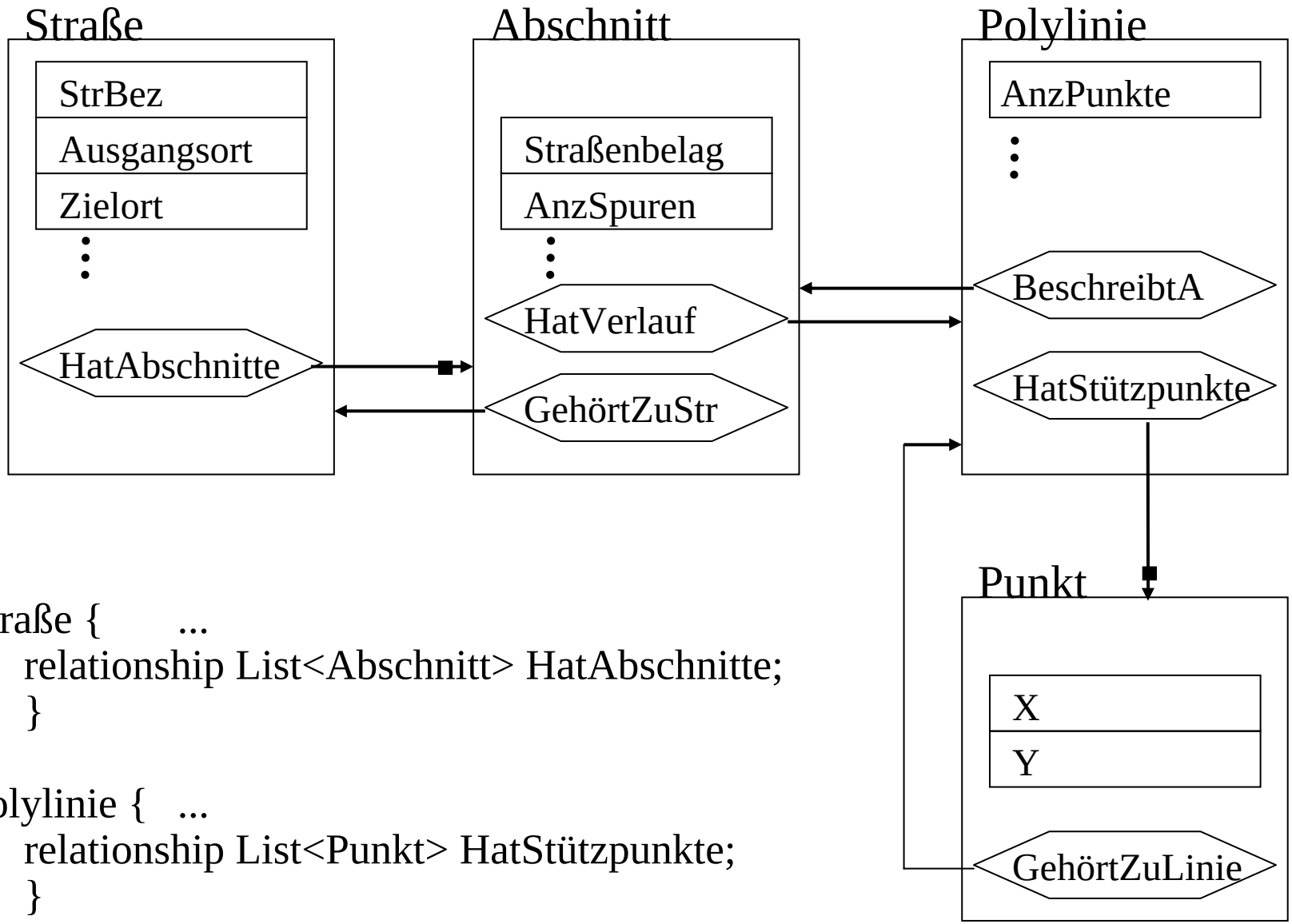
Beispiel – Alternative 1



Beispiel – Alternative 1 (textuell)

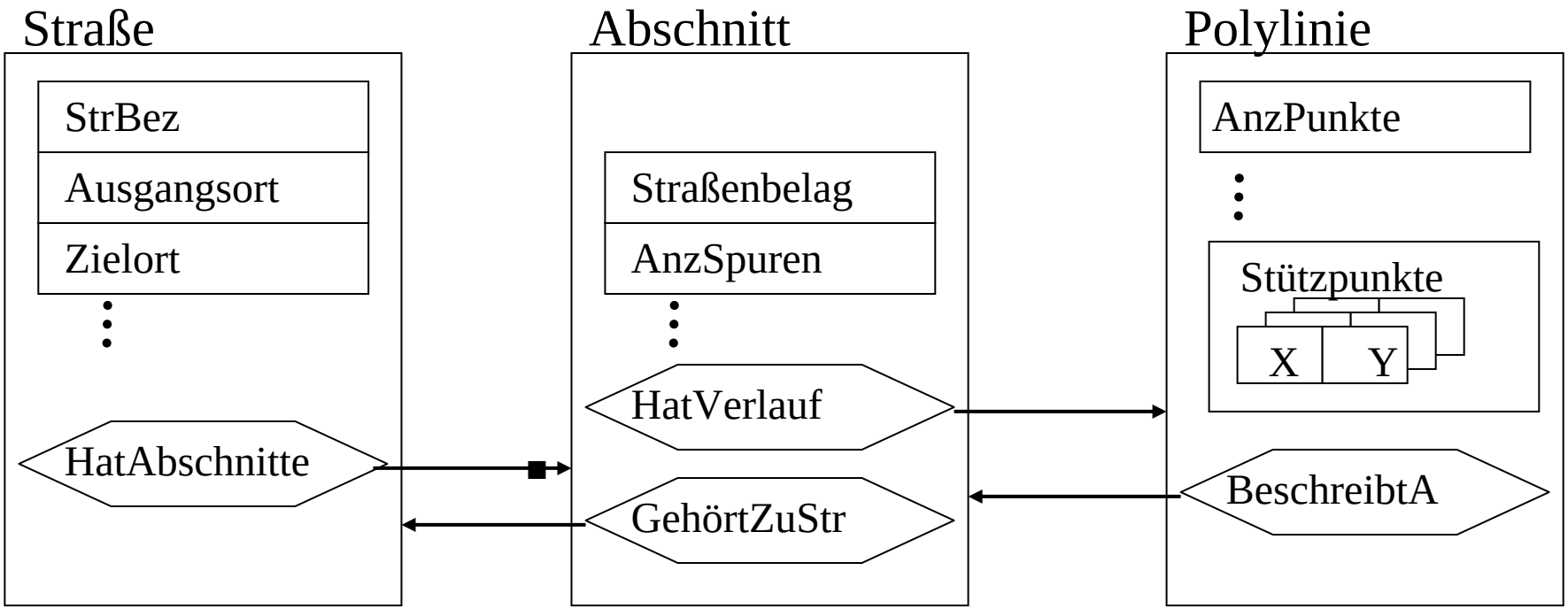
```
class Straße { extent Straßen;
        key StrBez;
        attribute String StrBez;
        attribute String Ausgangsort;
        attribute String Zielort;
        ...
        relationship Set<Abschnitt> HatAbschnitte;
        }
class Abschnitt { extent Abschnitte;
        key (GehörtZuStr.StrBez, ANr);
        attribute Integer ANr;
        attribute String Straßenbelag;
        attribute Integer AnzSpuren;
        attribute Integer Tempolimit;
        ..
        relationship Polylinie HatVerlauf;
        relationship Straße GehörtZuStr;
        }
```

Beispiel – Alternative 2



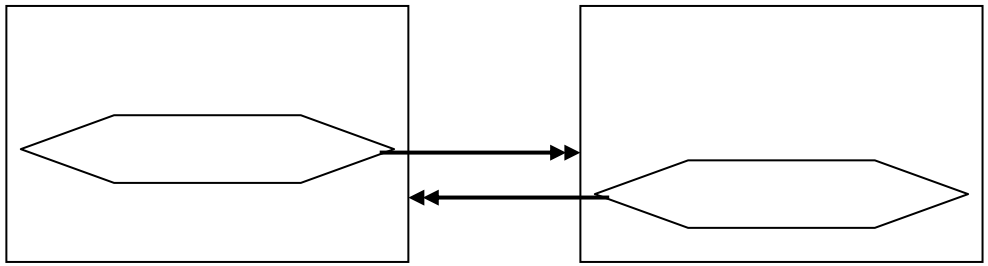

```
class Straße {      ...
        relationship List<Abschnitt> HatAbschnitte;
        }
...
class Polylinie {   ...
        relationship List<Punkt> HatStützpunkte;
        }
...
```

Beispiel – Alternative 3

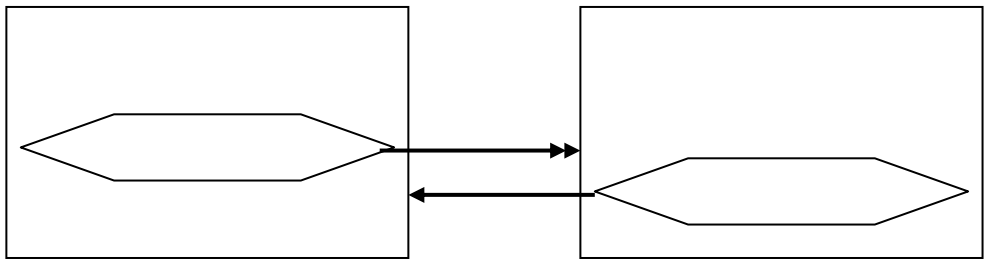



```
...
class Polylinie {   ...
        attribute Integer AnzPunkte;
        attribute List<Struct<X: Real; Y: Real>> Stützpunkte;
        }
```

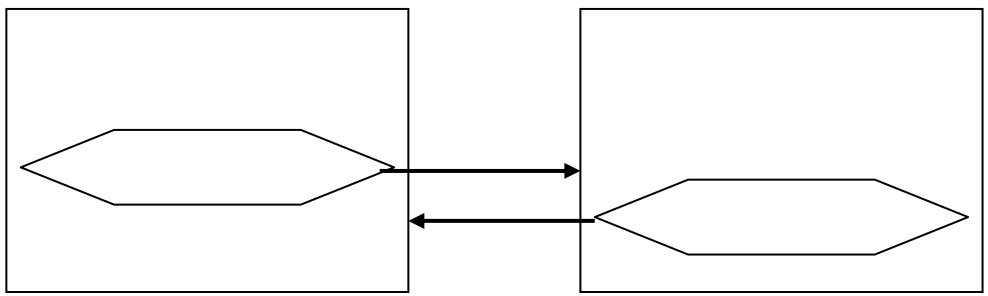
Verschiedene Arten von Relationships



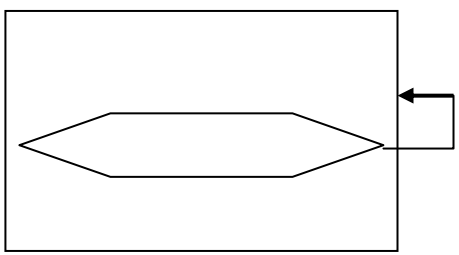
M:N-Relationship



1:N-Relationship
N:1-Relationship



1:1-Relationship



Relationship zwischen Objekten derselben Klasse

Integritätsbedingungen für Relationships

Sei $R \subseteq A \times B$.

Dies entspricht: $R_A: A \rightarrow 2^B$ und $R_B: B \rightarrow 2^A$.

Die folgende Invariante muß gelten:

$$\forall x \in A: \; x \in \bigcup_{y \in R_A(x)} R_B(y) \quad \text{und} \quad \forall y \in B: \; y \in \bigcup_{x \in R_B(y)} R_A(x)$$

bzw.: $R_A(x) = \{y \in B \mid (x,y) \in R\}$ und $R_B(y) = \{x \in A \mid (x,y) \in R\}$

Beispiele:

1) class Männer {...
   relationship Frauen Ehefrau inverse Frauen::Ehemann; }
   class Frauen { ...
   relationship Männer Ehemann inverse Männer::Ehefrau; }
2) class Straße { ...
   relationship Set<Abschnitt> HatAbschnitte inverse Abschnitt::GehörtZuStr; }
   class Abschnitt { ...
   relationship Straße GehörtZuStr inverse Straße::HatAbschnitte; ... }

Objekt-Identität und Objekt-Sharing

Es kann mehrere Objekten geben, die in allen Attributwerten übereinstimmen, aber verschiedene *Objekt-Ids (OIDs)* haben.

Ein Objekt kann von mehreren Objekten referenziert werden. Damit sind Änderungen auf einem Subobjekt eines Objekts für andere Objekte mit denselben Subobjekt unmittelbar sichtbar („*Objekt-Sharing*“).

Beispiel:

```
class Abschnitt { ...
    relationship Set<Straße> GehörtZuStr inverse Straße::HatAbschnitte; ... }
```

```
Straße [StrBez="Mainzer Str."].HatAbschnitte.first().AnzSpuren = 4
```

wirkt sich sofort auf Abschnitt der B51 aus.

10.2.2 Objektmethoden und Kapselung

Zu jeder Klasse kann eine Menge von *Methoden* definiert werden:
Funktionen mit Signatur $T1 \times T2 \times ... \times Tn \rightarrow T(n+1)$ und
Prozeduren mit Seiteneffekten auf den Objektzustand.
Ein Methodenaufruf x.m(...) hat das Objekt x als impliziten Parameter.

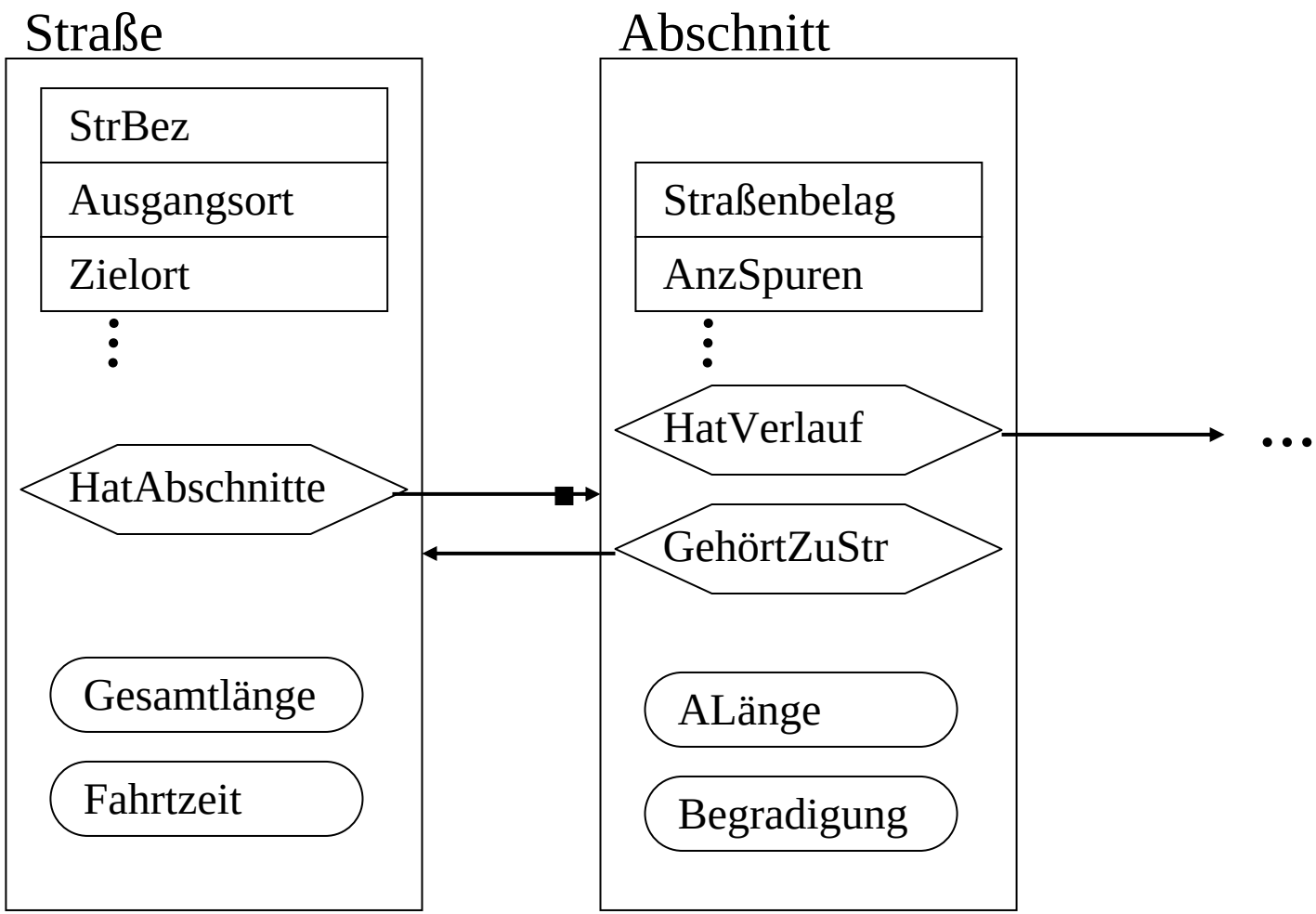
Die Gesamtheit der zu einer Klasse definierten – öffentlich sichtbaren – Methoden bildet die *Schnittstelle eines Abstrakten Datentyps (ADT)*, und jedes Objekt der Klasse ist dann eine Instanz dieses ADTs.
Attribute (d.h. der konkrete Objektzustand) können als privat definiert werden und sind dann nicht mehr öffentlich sichtbar, sondern „gekapselt“.

Vorteil der Kapselung:
Die Implementierung einer Klasse und seiner Methoden kann geändert werden, ohne dass sich die Schnittstelle der Klasse ändert.

Beispiel: Objektmethoden

```
class Straße { ...      private attribute Integer Durchschnittstempo;
                        Real Gesamtlänge ( );
                         Integer Fahrtzeit ( ); }
class Abschnitt { ... Real ALänge ( );
                         Boolean Begradigung (in Punkt, in Punkt); }

Straße:: Gesamtlänge () {
        float l = 0.0;
        Set<Ref<Abschnitt>> MeineAbschnitte = this->HatAbschnitte;
        Iterator<Abschnitt> it = MeineAbschnitte->create_iterator();
        Ref<Abschnitt> a;
        while (a=it.next())  { l += a->ALänge(); } return l };
    Abschnitt:: ALänge () {
        float l = 0.0;
        List<Ref<Punkt>> MeinePunkte = this->HatVerlauf->HatStützpunkte;
        Iterator<Punkt> it = MeinePunkte->create_iterator();
        Ref<Punkt> p, q;
        p = it->next();
        while (q=it->next()) {
            l += sqrt ((p->X - q->X) * (p->X - q->X) + (p->Y - q->Y) * (p->Y - q->Y));
            p = q; };
        return l };
```

Beispiel: Objektmethoden (Diagramm)

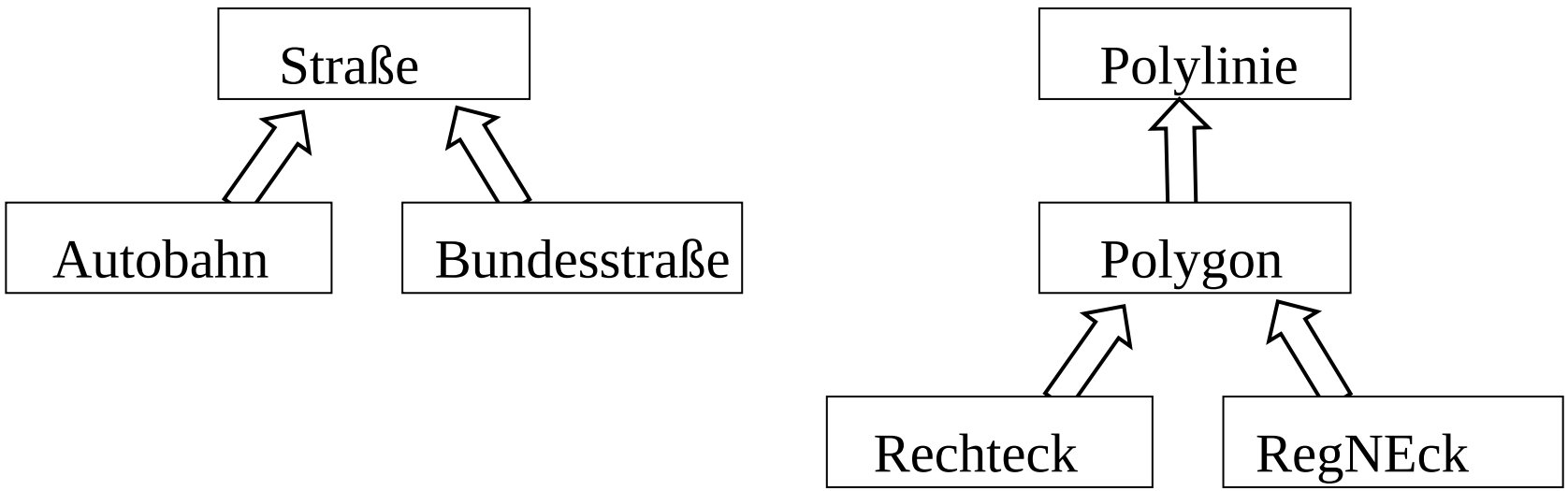


10.2.3 Vererbung

Klasse B heißt Subklasse von Klasse A (A Superklasse von B), wenn
- Attribute und Methoden von B eine Obermenge der von A bilden und
- die Objektmenge von B eine Teilmenge der von A ist.

B heißt Spezialisierung von A, und A heißt Generalisierung von B.

Beispiele:



Beispiele: Vererbung

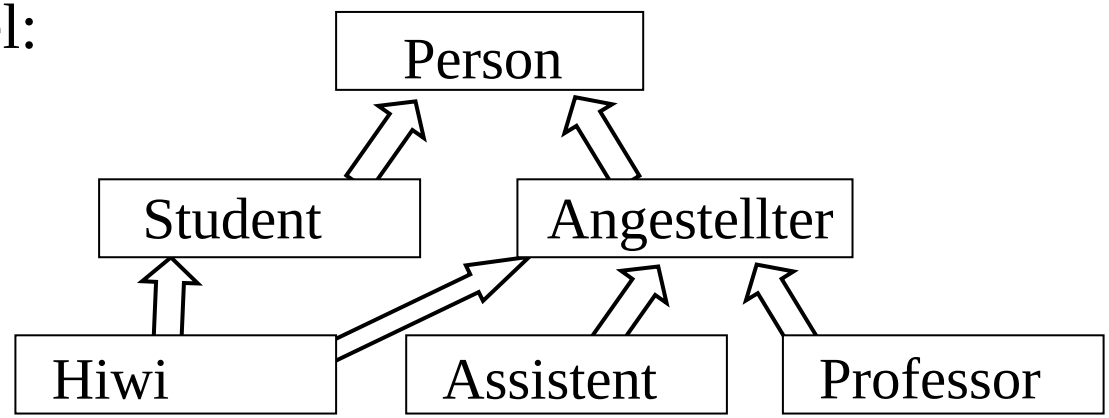
```
class Bundesstraße: Straße { extent Bundesstraßen;
        attribute Integer Verkehrsdichte; }
class Autobahn: Straße { extent Autobahnen;
        attribute Integer Mindesttempo;
        attribute Array<Integer> VerkehrsdichteProTag;
        relationship Set<Punkt> Auffahrten; }

class Polygon: Polylinie { extent Polygone;
        relationship Punkt Zentrum;
        relationship Stadt BeschreibtS inverse Stadt::Stadtgrenze;
        relationship Land BeschreibtL inverse Land::Landesgrenze;
        Real Fläche ( );  }
class Rechteck: Polygon { extent Rechtecke;
        attribute Real Diagonallänge; }
class RegNEck: Polygon { extent RegNEcke;
        attribute Integer AnzEcken;
        Real Inkreisradius ();
        Real Umkreisradius (); }
```

Umgang mit geerbten Methoden

Überschreiben der Implementierung einer Methode
wegen speziellerer Semantik oder Effizienz
Beispiele:

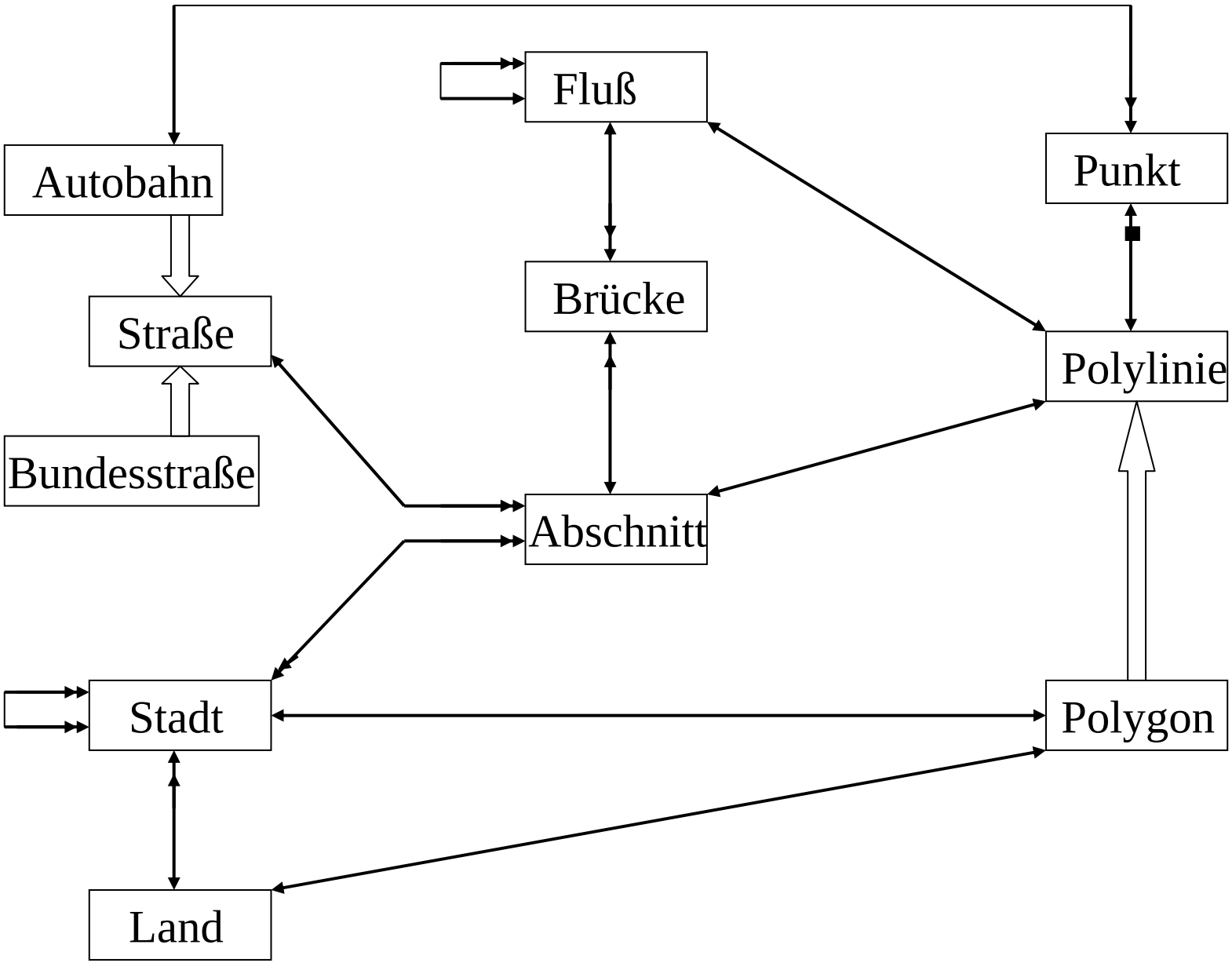
1) Fläche ( ) der Subklasse „Polygone mit Löchern“
2) Fläche ( ) der Subklassen „Rechtecke“ und „RegNEcke“
3) SchneidetBox (in Rechteck) der Subklasse „Polygone“

Mehrfachvererbung erfordert ggf. Umbenennung
Beispiel:



Bei der Verarbeitung **polymorpher Objektmengen** (z.B. Polygone)
wird die jeweils passende Methodenimplementierung
zur Laufzeit bestimmt („late binding“)

10.3 ODMG-Standard: ODL



Beispiel: ODL-Schema (1)

```
class Straße { extent Straßen; key StrBez;
        attribute String StrBez;
        attribute String Ausgangsort; attribute String Zielort;
        relationship Set<Abschnitt> HatAbschnitte inverse Abschnitt::GehörtZuStr;
        Real Gesamtlänge ( ); Integer Fahrtzeit ( );  }
class Bundesstraße: Straße { extent Bundesstraßen;
        attribute Integer Verkehrsdichte; }
class Autobahn: Straße { extent Autobahnen;
        attribute Integer Mindesttempo;
        attribute Array<Integer> VerkehrsdichteProTag;
        relationship Set<Punkt> Auffahrten; }
class Abschnitt { extent Abschnitte; key (GehörtZuStr.StrBez, ANr);
        attribute Integer ANr; attribute String Straßenbelag;
        attribute Integer AnzSpuren; attribute Integer Tempolimit;
        relationship Polylinie HatVerlauf inverse Polylinie::BeschreibtA;
        relationship Straße GehörtZuStr inverse Straße::HatAbschnitte;
        relationship Set<Brücke> HatBrücken inverse Brücke::GehörtZuA;
        relationship Set<Stadt> FührtDurch inverse Stadt::LiegtAn;
        Real ALänge ( ); }
```

Beispiel: ODL-Schema (2)

```
class Polylinie { extent Polylinien;
        attribute Integer AnzPunkte;
        relationship List<Punkt> HatStützpunkte inverse Punkt::GehörtZuLinie;
        relationship BeschreibtA inverse Abschnitt::HatVerlauf;
        relationship BeschreibtF inverse Fluß::HatVerlauf;
        Real Länge ( );
        void Begradigung (in Punkt, in Punkt) Raises (Zu_große_Anschlußkrümmung);
        Boolean Schneidet (in Polylinie);
        Boolean SchneidetBox (in Rechteck);
        Boolean LiegtInBox (in Rechteck); }
class Punkt { extent Punkte;
        attribute Real X; attribute Real Y;
        relationship Polylinie GehörtZuLinie inverse Polylinie::HatStützpunkte
        Boolean LiegtInBox (in Rechteck);
        Boolean LiegtInRegion (in Polygon); }
class Polygon: Polylinie { extent Polygone;
        relationship Punkt Zentrum;
        relationship Stadt BeschreibtS inverse Stadt::Stadtgrenze;
        relationship Land BeschreibtL inverse Land::Landesgrenze;
        Real Fläche ( ); }
```

Beispiel: ODL-Schema (3)

```
class Fluß { extent Flüsse; key Flußbez;
        attribute String Flußbez; attribute Array<Real> Verschmutzungsgrad;
        relationship Punkt Quelle; relationship Punkt Mündung;
        relationship Fluß MündetIn inverse Fluß::HatZuflüsse;
        relationship Set<Fluß> HatZuflüsse inverse Fluß::MündetIn;
        relationship Polylinie HatVerlauf inverse Polylinie::BeschreibtF;
        relationship Set<Brücke> FließtUnter inverse Brücke::ÜberquertF; }
class Brücke { extent Brücken;
        attribute Real Höhe; attribute Real Spannweite;
        relationship Punkt Anfang; relationship Punkt Ende;
        relationship Fluß ÜberquertF inverse Fluß::FließtUnter;
        relationship Abschnitt GehörtZuA inverse Abschnitt::HatBrücken; }
class Stadt { extent Städte;
        attribute String Stadtname; attribute Integer Einwohnerzahl;
        relationship Person Bürgermeister;
        relationship Set<Stadt> Partnerstädte inverse Stadt::Partnerstädte;
        relationship Land GehörtZuL inverse Land::HatStädte;
        relationship Polygon Stadtgrenze inverse Polygon::BeschreibtS;
        relationship Set<Abschnitt> LiegtAn inverse Abschnitt::FührtDurch;
        Real Einwohnerdichte ( ); void Eingemeindung (Inout Stadt); }
class Land { extent Länder; key Landesbez;
        attribute String Landesbez; relationship Person Staatsoberhaupt;
        relationship Set<Stadt> HatStädte inverse Stadt::GehörtZuL;
        relationship Polygon Landesgrenze inverse Polygon::BeschreibtL;
        Integer Bevölkerung ( ); }
```

Generische Typen (“Templates”) in ODL

Collection<T>:

attribute Integer cardinality; attribute Boolean empty?;
Collection<T> create (); void delete ();
void insert_element (in T); void remove_element (in T);
Collection<T> select (in String);
Boolean exists? (in String); Boolean contains_element? (in T);
Iterator create_iterator ();
...

Set<T>:

Set<T> union (in Set<T>); Set<T> intersection (in Set<T>); Set<T> difference (in Set<T>);
Boolean is_subset? (in Set<T>); Boolean is_proper_subset? (in Set<T>);
Boolean is_superset? (in Set<T>); Boolean is_proper_superset? (in Set<T>);
...

List<T>:

T retrieve_first_element (); T retrieve_last_element ();
T retrieve_element_at (in Integer);
void insert_first_element (in T); void insert_last_element (in T);
void insert_element_after (in T, in Integer); void insert_element_before (in T, in Integer);
...

10.4 ODMG-Standard: OQL

- angelehnt an SQL, aber mit signifikanten Abweichungen und OO-Erweiterungen

- Pfadausdrücke zur Traversierung von Relationships („implizite Joins“):

  C0.R1.R2. ... .Rn.A in Select-Klausel entspricht
  $\{tn.A, ... \mid ... \wedge \exists\, t0\, \exists\, t1 ... \exists\, t(n-1)$
  $(t0.R1 = t1.OID \wedge t1.R2 = t2.OID \wedge ... \wedge t(n-1).Rn = tn.OID)\}$

  C0.R1.R2. ... .Rn.A θ wert in Where-Klausel entspricht
  $\exists\, t0\, \exists\, t1 ... \exists\, t(n-1)\ (t0.R1 = t1.OID \wedge ... \wedge t(n-1).Rn = tn.OID \wedge tn.A\ \theta$ wert)

- Subqueries auch in Select- und From-Klausel: konstruieren geschachtelte Ergebnisse (z.B. vom Typ Set<Set<T>>)
- Methodenaufrufe: wie Attribute einer Klasse, z.B. C.F(x,y) oder C.G( )
- Automatische Berücksichtigung von Subklassen
- Erweiterte Aggregationen und Gruppierung

Beispiele: Pfadausdrücke in OQL (1)

1) Welche Flüsse überquert die B51?

```
SELECT FLATTEN (S.HatAbschnitte.HatBrücken.ÜberquertF)
FROM Straßen S
WHERE S.StrBez="B51"
```

oder:

```
SELECT B.ÜberquertF
FROM ( SELECT FLATTEN(A.HatBrücken)
       FROM ( SELECT FLATTEN (S.HatAbschnitte)
              FROM S IN Straßen
              WHERE S.StrBez="B51" ) AS A
     ) AS B
```

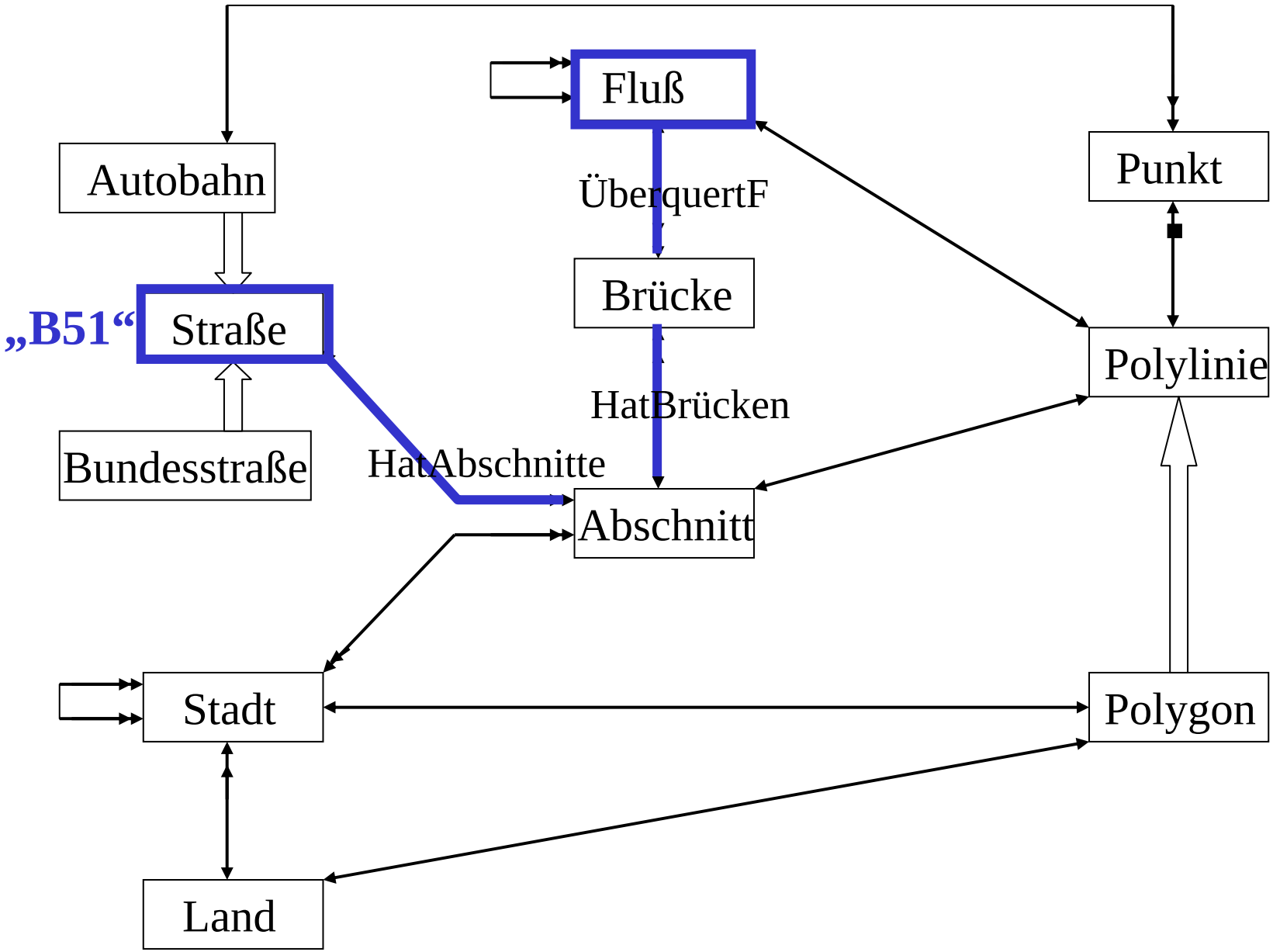
2) Welche Straßen überqueren die Mosel?

```
SELECT S
FROM Straßen S
WHERE S.HatAbschnitte.HatBrücken.ÜberquertF.Flußbez="Mosel"
```

3) Welche Partnerstädte haben die Städte der Elfenbeinküste?

```
SELECT FLATTEN(L.HatStädte.Partnerstädte)
FROM Länder L
WHERE L.Landesbez="Elfenbeinküste"
```

Pfadausdrücke in OQL – Beispiel 1



Beispiele: Pfadausdrücke in OQL (2)

4) In welchen Ländern haben Städte der Elfenbeinküste Partnerstädte?

```
SELECT DISTINCT FLATTEN ( L.HatStädte.Partnerstädte.GehörtZuL )
FROM Länder L
WHERE L.Landesbez="Elfenbeinküste"
```

5) Ausgabe des Verlaufs der B51:

```
SELECT S.HatAbschnitte.HatVerlauf.HatStützpunkte
FROM Straßen S
WHERE S.StrBez="B51"
```

oder:

```
SELECT A.HatVerlauf.HatStützpunkte
FROM  ( SELECT S.HatAbschnitte
          FROM Straßen S
          WHERE S.StrBez="B51"
          SORT A IN S.HatAbschnitte BY A.ANr ) AS A
```

Beispiele: Methodenaufrufe in OQL (1)

1) Wieviele Kilometer hat das gesamte Straßennetz?

```
SELECT SUM ( S.Gesamtlänge() ) FROM Straßen S
```

2) Welche Städte haben eine Fläche von mehr als 30 Quadratkilometern?

```
SELECT S.Stadtname FROM Städte S
WHERE S.Stadtgrenze.Fläche() > 30
```

3) In welchen Städten mit einer Fläche von mehr als 30 km^2 haben alle durch die Stadt führenden Straßenabschnitte ein Tempolimit von max. 80 km/h?

```
SELECT S.Stadtname FROM Städte S
WHERE S.Stadtgrenze.Fläche() > 30
AND ( FOR ALL A IN S.LiegtAn : A.Tempolimit <= 80 )
```

4) In welchen Ländern mit mehr als 100 Millionen Einwohnern haben Städte der Elfenbeinküste Partnerstädte?

```
SELECT DISTINCT PL
FROM ( SELECT FLATTEN (L.HatStädte.Partnerstädte.GehörtZuL)
          FROM L IN Länder
          WHERE L.Landesbez="Elfenbeinküste" ) AS PL
WHERE PL.Bevölkerung() > 100000000
```

Beispiele: Methodenaufrufe in OQL (2)

5) Bestimmung aller Straßen mit einem Stützpunkt innerhalb des Rechtecks mit der linken unteren Ecke (1,2) und der rechten oberen Ecke (5,8)

```
SELECT S.StrBez
FROM Straßen S
WHERE  S.HatAbschnitte.Verlauf.HatStützpunkte.LiegtInBox (
                STRUCT(lu:STRUCT(x:1,y:2), ro:STRUCT(x:5,y:8)) )
```

6) Bestimmung aller Straßen, die das Rechteck mit der linken unteren Ecke (1,2) und der rechten oberen Ecke (5,8) schneiden

```
SELECT S.StrBez
FROM Straßen S
WHERE  S.HatAbschnitte.Verlauf.SchneidetBox (
                STRUCT(lu:STRUCT(x:1,y:2), ro:STRUCT(x:5,y:8)) )
```

Beispiele: Generalisierungshierarchie in OQL

1) Welche Polylinien, inkl. Polygonen, schneiden das Rechteck ... ?

```
SELECT P FROM Polylinien P
WHERE P.SchneidetBox ( STRUCT(lu:STRUCT(x:1,y:2), ro:STRUCT(x:5,y:8)) )
```

2) Das niedrigste Mindesttempo auf einer Straße von mehr als 1000 km Gesamtlänge?

```
SELECT MIN ( ((Autobahn) S).Mindesttempo ) FROM Straßen S
WHERE S.Gesamtlänge() > 1000
```

3) Autobahnen über die Mosel mit einem Abschnitt mit Tempolimit 60 km/h ?

```
SELECT A.GehörtZuStr.StrBez
FROM ( SELECT F.FließtUnter.GehörtZuA FROM Flüsse F
          WHERE F.Flußbez = “Mosel" ) AS A
WHERE ( EXISTS B IN Autobahnen: B.StrBez = A.GehörtZuStr.StrBez )
AND A.Tempolimit = 60
```

oder:

```
SELECT ((Autobahn) A.GehörtZuStr).StrBez
FROM ( SELECT F.FließtUnter.GehörtZuA FROM Flüsse F
          WHERE F.Flußbez = “Mosel" ) AS A
WHERE A.Tempolimit = 60
```

Erweiterte Gruppierung in OQL

Resultat von ... GROUP BY G1: Gruppierungsattribut1, ...
hat den Typ set<struct(G1: type_of(Gruppierungsattribut1), ...),
partition: bag<type_of(Nichtgruppierungsattribute)>>

Beispiel: Gesamteinwohnerzahl aller Städte an der E12 pro Land

```
SELECT Land,
   Gesamteinwohner: SUM(SELECT R.Einwohnerzahl FROM partition),
   Stadteinwohner: (SELECT R.Stadtname, R. Einwohnerzahl FROM partition)
FROM ( SELECT S FROM Städte S
          WHERE S.LiegtAn.GehörtZu.StrBez = “E12" ) AS R
GROUP BY Land: R.GehörtZuL
```

Land	*Gesamteinwohner*	*Stadteinwohner*	
Deutschland	450 000	*Stadtname*	*Einwohnerzahl*
		Kaiserslautern	200 000
		Saarbrücken	250 000
Frankreich	5 350 000	*Stadtname*	*Einwohnerzahl*
		Metz	200 000
		Reims	150 000
		Paris	5 000 000

Erweiterte Aggregation in OQL

Anwendungsspezifische Aggregationsfunktionen

Beispiel:

```
class Zahlen { attribute bag<Zahl: real> Zahlenmultimenge;
    Real mean(); Real median(); Real stddev(); }
```

a) Mittelwert und Standardabweichung der Bestellungssummen

```
SELECT Z.mean(), Z.stddev()
FROM ( SELECT B.Summe FROM Bestellungen B ) AS Z
```

b) Mittelwert und Standardabweichung der Bestellungssummen für jedes einzelne Produkt

```
SELECT G.Prod, G.Best.mean(), G.Best.stddev()
FROM ( SELECT Prod, (SELECT B.Summe FROM partition)
                                    AS Best
        FROM Bestellungen B
        GROUP BY Prod: B.PNr ) AS G
```

8.5.1 Komplexe Datentypen in SQL-99

Typkonstruktoren ROW, SET, LIST, MULTISET
Subklassen UNDER Superklasse

Beispiele:

```
CREATE ROW TYPE Punkt (X FLOAT, Y FLOAT);
CREATE ROW TYPE Polylinie
   (AnzPunkte INTEGER, HatStützpunkte LIST OF Punkt);
CREATE ROW TYPE Straße
   (StrBez VARCHAR(10), Ausgangsort VARCHAR(30), Zielort VARCHAR(30),
   HatVerlauf Polylinie);
CREATE TABLE Straßen OF TYPE Straße;
CREATE TYPE Autobahn UNDER Straße
  (Mindesttempo FLOAT);
CREATE TABLE Autobahnen OF Autobahn;

SELECT * FROM Autobahnen
   liefert alle Autobahnen
SELECT * FROM Straßen
   liefert alle Straßen inklusive der Autobahnen
```

Komplexe Datentypen in Oracle8i

Typkonstruktoren OBJECT, REF, VARRAY (ohne Schachtelung)
TABLE (mit 1 Stufe von NESTED TABLES)

Beispiele:

```
CREATE TYPE Kenntnissetyp AS VARRAY(5) OF VARCHAR(30);
CREATE TABLE Angestellte
    (AngNr NUMBER, Name VARCHAR(50), Kenntnisse Kenntnissetyp);
INSERT INTO Angestellte VALUES (0815, 'Heinz Becker',
                                Kenntnissetyp ('Oracle', 'Java', 'Urpils') );
```

```
CREATE TYPE Pruefungstyp AS OBJECT
    (Fach VARCHAR(30), Datum DATE, Note NUMBER);
CREATE TYPE Pruefungstabellentyp AS TABLE OF Pruefungstyp;
CREATE TABLE Studenten
    (MatrNr NUMBER, Name VARCHAR(30),
     AbgelegtePruefungen Pruefungstabellentyp)
    NESTED TABLE AbgelegtePruefungen STORE AS Pruefungstabelle;
INSERT INTO Studenten VALUES (4711, 'Jacques Bistro',
          Pruefungstabellentyp (Pruefungstyp ('Praktische Informatik', 11.11.1998, 1.3),
                                Pruefungstyp ('Nebenfach', 24.12.1998, 1.7) ) );
```

Mehrfache Schachtelung in Oracle8i

Beispiel:

```
CREATE TYPE Frageantworttyp AS OBJECT
    (LfdNr NUMBER, Frage VARCHAR(200), Antwort VARCHAR(2000));
CREATE TYPE Protokolltyp AS TABLE OF Frageantworttyp;
CREATE TYPE Pruefungstyp AS OBJECT
    (Fach VARCHAR(30), Datum DATE, Note NUMBER, Protokoll Protokolltyp);
CREATE TYPE Pruefungstabellentyp AS TABLE OF Pruefungstyp;
CREATE TABLE Studenten
     (MatrNr NUMBER, Name VARCHAR(30),
      AbgelegtePruefungen Pruefungstabellentyp)
     NESTED TABLE AbgelegtePruefungen STORE AS Pruefungstabelle;
INSERT INTO Studenten VALUES (4711, 'Jacques Bistro',
            Pruefungstabellentyp
               (Pruefungstyp ('Praktische Informatik', 11.11.1998, 1.3,
                              Protokolltyp (
                                  Frageantworttyp (1, 'Wie baut man ... ?', 'Das macht ... '),
                                  Frageantworttyp (2, 'Wieso ist ...? ', 'Der Grund ist ... '))),
                Pruefungstyp ('Nebenfach', 24.12.1998, 1.7,
                              Protokolltyp (
                                  Frageantworttyp (1, 'Was ist ... ? ', 'Das ist ... '),
                                  Frageantworttyp (2, 'Wie funktioniert ...? ', 'Das ... '))) ));
```

Oracle8i: Anfragen auf Nested Tables (1)

Beispiele:

```
CREATE TYPE Namenstyp AS OBJECT
    (Vorname VARCHAR(30), Nachname VARCHAR(30), Spitzname VARCHAR(30))
CREATE TYPE Pruefungstyp AS OBJECT
    (Fach VARCHAR(30), Datum DATE, Note NUMBER);
CREATE TYPE Pruefungstabellentyp AS TABLE OF Pruefungstyp;
CREATE TABLE Studenten
    (MatrNr NUMBER, Name Namenstyp,
    AbgelegtePruefungen Pruefungstabellentyp)
    NESTED TABLE AbgelegtePruefungen STORE AS Pruefungstabelle;
```

Oracle8i: Anfragen auf Nested Tables (2)

1) Vollständige Namen von Studenten mit Spitznamen „Schlumpf“

```
SELECT S.Name.Vorname, S.Name.Nachname FROM Studenten S
WHERE S.Name.Spitzname = 'Schlumpf'
```

2) Gib alle Prüfungen des Studenten mit Matrikelnummer 1234 aus

```
SELECT S.AbgelegtePruefungen FROM Studenten S WHERE S.MatrNr = 1234
```

bzw. besser:

```
SELECT * FROM TABLE ( SELECT S.AbgelegtePruefungen
                      FROM Studenten S WHERE S.MatrNr = 1234 )
```

3) Note des Studenten mit Matrikelnr. 1234 im Fach Informationssysteme

```
SELECT P.Note
FROM TABLE ( SELECT S.AbgelegtePruefungen FROM Studenten S
             WHERE S.MatrNr = 123456 ) P
WHERE P.Fach = 'Informationssysteme';
```

Oracle8i: Anfragen auf Nested Tables (3)

4) Gib die Noten aller Studenten in allen Fächern aus.

```
SELECT S.MatrNr, P.Fach, P.Note
FROM Studenten S, TABLE (S.AbgelegtePruefungen) P
```

5) Gib die Noten aller Studenten im Fach Informationssysteme aus.

```
SELECT S.MatrNr, P.Note
FROM Studenten S,  TABLE (S.AbgelegtePruefungen) P
WHERE P.Fach = 'Informationssysteme';
```

6) Gib für jeden Studenten das letzte Prüfungsdatum aus
bzw. einen Nullwert für Studenten ohne abgelegte Prüfungen

```
SELECT S.MatrNr, Max(P.Datum)
FROM Studenten S, TABLE (S.AbgelegtePruefungen) (+) P
```

oder:

```
SELECT S.MatrNr,
        (SELECT Max(P.Datum) FROM TABLE(S.AbgelegtePruefungen))
FROM Studenten S
```

Funktionen und ADTs in SQL-99

Beispiele:

```
CREATE TYPE Rechteck (lu Punkt, ro Punkt);
CREATE FUNCTION  SchneidetBox (p Polylinie, r Rechteck) RETURNS BOOLEAN
    BEGIN ... END;
CREATE FUNCTION Gesamtlänge (p Polylinie) RETURNS FLOAT
    BEGIN ... END;

SELECT S.StrBez, Gesamtlänge(S.HatVerlauf) FROM Straßen S
WHERE S.Ausgangsort='Saarbrücken'
AND SchneidetBox (S.HatVerlauf, <lu: <X:5.71, Y:6.24>, ro: <X:17.5, Y:22>>);
```

```
CREATE TYPE Straße
  (StrBez VARCHAR(10), Ausgangsort VARCHAR(30), Zielort VARCHAR(30),
  PRIVATE HatVerlauf Polylinie,
  PUBLIC FUNCTION Gesamtlänge (s Straße) RETURNS FLOAT
       ...
  END FUNCTION );
CREATE TABLE Straßen OF TYPE Straße;
```

Funktionen und ADTs in Oracle8i

Beispiele:

```
CREATE TYPE Kontotyp AS OBJECT
   (Kontonr NUMBER, Inhaber VARCHAR(30), Kontostand MONEY,
   MEMBER FUNCTION Tageszinsen (Zinssatz IN NUMBER) RETURN MONEY,
   MEMBER PROCEDURE Einzahlung (Betrag IN MONEY),
   MEMBER PROCEDURE Auszahlung (Betrag IN MONEY) RETURN Fehlertyp );
CREATE OR REPLACE TYPE BODY Kontotyp AS
   MEMBER FUNCTION Tageszinsen (Zinssatz IN NUMBER) RETURN MONEY
       IS BEGIN ... RETURN Kontostand*Zinssatz/100; END;
   MEMBER PROCEDURE Einzahlung (...) IS BEGIN ... END; ...
   END;
CREATE TABLE Konto OF Kontotyp;

SELECT KontoNr, Tageszinsen (0.05) FROM Konto WHERE Kontostand > 100000;
SELECT KontoNr FROM Konto WHERE Tageszinsen (0.05) > 1000;

UPDATE Konto K SET K = K.Einzahlung (100) WHERE KontoNr = 1234;
```