In T. Härder, editor, Proc. GI/SI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft", BTW'89, pages 43-57, Berlin, 1989. Springer-Verlag.

Entwurf eines Datenbank-Prototyps für geowissenschaftliche Anwendungen

Friedrich Lohmann, Karl Neumann, Hans-Dieter Ehrich

Informatik, Abt. Datenbanken
Technische Universität Braunschweig
Postfach 3329, D-3300 Braunschweig

Kurzfassung: In dieser Arbeit wird die Konzeption eines Nichtstandard-Datenbanksystems für geowissenschaftliche Anwendungen vorgestellt. Das System bietet eine objektorientierte Datenbanksprache mit einer erweiterbaren Menge von geometrischen Datentypen; in Anwendungsprogrammen ist die Datenbanksprache als eingebettete Datenteilsprache verfügbar, wobei von der Datenbank gelesene Objektmengen nach einem Abstrakten-Datentyp-Ansatz manipuliert werden können. Ein spezieller NF2-Datenbankkern ist als Grundlage für eine effiziente Implementierung vorgesehen.

Abstract: This paper presents the design of a non-standard database system for geoscientific applications. The system offers an object-oriented database language with an extensible set of geometric data types; in application programs, the database language is available as an embedded data sublanguage, and sets of objects read from the database can be manipulated by operations offered in abstract data type modules. A special NF2 database kernel is to provide the basis for an efficient implementation.

1. Einleitung

Gegenwärtig existieren im Bereich der Nichtstandard-Datenbanksysteme zahlreiche Forschungsprojekte, deren Aktivitäten zwei Schwerpunkten zugeordnet werden können. Diese Schwerpunkte sind zum einen die Entwicklung neuer, an die jeweilige Anwendung angepaßter Daten- oder Objektmodelle [NR86, ScS86, HNSE87] mit zugehörigen Sprachen und deren theoretischer Fundierung [BK86, PA86, SSE87, HG88] und zum anderen die Bereitstellung von neuartigen Datenbankarchitekturen, die diese Anwendungen effizient unterstützen [CDWF86, Me87, Sc87, ALPS88]. Auf der Ebene der Datenbanksprachen setzen sich dabei Konzepte wie *Objektorientiertheit* und *Erweiterbarkeit* durch [BM86, DD86, LKDP87, WSSH88]. Bei den Implementierungsaspekten rückt das *NF2-Relationenmodell* [SP82, ScS83] als Implementierungs-Datenmodell immer mehr in den Vordergrund [KTW85, DDKL87, KW87].

Zum Entwurf eines geowissenschaftlichen Datenbanksystems als zentrale Komponente innerhalb des DFG-Schwerpunktprogrammes „Digitale geowissenschaftliche Kartenwerke“ [Vi85, Vi88] greifen wir diese aktuellen Forschungsergebnisse auf. Es geht dabei zunächst um die Bereitstellung einer Benutzerschnittstelle, die es ermöglicht, die sehr verschiedenen Daten der ca. 25 am Schwerpunktprogramm beteiligten Gruppen zu vereinheitlichen und so untereinander

Diese Arbeit wurde von der Deutschen Forschungsgemeinschaft (DFG) gefördert unter Az. Eh 75/3-4.

austauschbar zu machen. Des weiteren soll das Datenbanksystem auch Sprachmittel bieten, die kartographische Anwendungen unterstützen.

Aus diesen Anforderungen wurde eine geowissenschaftliche *Datenbanksprache* abgeleitet, die im nächsten Abschnitt vorgestellt wird. Sie zeichnet sich aus durch eine Zweiteilung des Datenbankschemas in fest vorgegebene und frei definierbare Objekttypen mit Generalisierungshierarchien und Beziehungsklassen sowie durch ihre semantische Fundierung, die sich auf das Konzept der Trennung zwischen Objekt- und Datenschicht [Eh85] stützt. In Abschnitt 3 beschreiben wir die *Anwendungsprogrammschnittstelle* unseres Datenbanksystems. Es wurde eine Einbettungsstrategie mit Vorübersetzung gewählt, wobei Mengen von komplexen geowissenschaftlichen Objekten im Anwendungsprogramm mittels automatisch generierter Operationen bequem manipuliert werden können. Zur Vorbereitung von Implementierungsaspekten charakterisieren wir in Abschnitt 4 zunächst die unterste Systemschicht, den sogenannten *Geo-Kern*, den wir von einer anderen Forschungsgruppe übernehmen [SW86, Sc87, HSWW88]. Darauf aufbauend wird in Abschnitt 5 die schrittweise *Übersetzung von Anfragen* unserer deskriptiven Datenbanksprache auf die prozedurale Schnittstelle dieses Kerns diskutiert. Abschnitt 6 schließlich stellt die geplante Gesamtkonzeption unseres Datenbanksystems dar. Des weiteren finden sich hier eine Zusammenfassung der Erfahrungen, die wir mit einem ersten Prototyp bereits sammeln konnten.

2. Datenbanksprache

In diesem Abschnitt sollen die geowissenschaftliche Datenbanksprache und das zugrundeliegende Objektmodell soweit beschrieben werden, wie es für das Verständnis dieser Arbeit erforderlich ist. Die Darstellung ist bewußt sehr knapp und überblicksartig gehalten, da Sprache und Objektmodell bereits an anderer Stelle ausführlich vorgestellt wurden. Für eine detailliertere Beschreibung muß auf [LN87, ELNR88, Ne88] verwiesen werden.

Der geowissenschaftlichen Datenbanksprache liegt ein spezielles *Geo-Objektmodell* zugrunde, das durch Erweiterung des bekannten ER-Modells [Ch76] entwickelt wurde. Als Modellierungskonzepte bietet das Geo-Objektmodell – ähnlich wie das ER-Modell – Objektklassen und Beziehungsklassen. Jedes Objekt wird in der Datenbank repräsentiert durch seine Attributwerte. Dabei sind nicht nur atomare Objektklassen zugelassen, sondern auch komplexe Objektklassen, d. h. ein Objekt kann zusammengesetzt sein aus einer Menge oder Liste von Unterobjekten gleichen Typs oder aus einzelnen Unterobjekten verschiedenen Typs (Aggregation). Darüber hinaus ist es möglich, unterschiedliche Objektklassen zu allgemeineren Klassen zu generalisieren.

Das Geo-Objektmodell unterstützt eine grundsätzliche konzeptionelle Trennung zwischen den Informationen über geowissenschaftliche Gegebenheiten *(Geoobjekte)* und deren grafischen Darstellungen in Landkarten *(Kartenobjekte)*. Dabei können Geoobjektklassen, entsprechend den Erfordernissen des jeweiligen Anwendungsfalles, vom Benutzer frei definiert werden, während das Attributschema der Karten und Kartenobjekte fest vordefiniert ist. Spezielle Konstrukte der Datenbanksprache ermöglichen es, aus Geoobjekten, die in einer Landkarte grafisch dargestellt werden sollen, entsprechende Kartenobjekte abzuleiten.

Eine charakteristische Eigenschaft geowissenschaftlicher Objekte liegt darin, daß sie in der Regel eine räumliche Ausdehnung besitzen. Zur Unterstützung dieser wichtigen Eigenschaft bietet das Geo-Objektmodell zusätzlich zu den herkömmlichen Datentypen spezielle *geometrische*

Wir wollen nun diese sehr kurze Einführung in die Datenbanksprache abschließen und uns im nächsten Abschnitt den Fragen der Anwendungsprogramm-Schnittstelle zuwenden.

3. Anwendungsprogrammierung

Um einen möglichst großen Bereich von Anwendungen zu unterstützen, sollten Datenbanksysteme außer einer interaktiven Schnittstelle auch die Möglichkeit bieten, von Anwendungsprogrammen aus direkt auf die Datenbank zuzugreifen. Eine solche Anwendungsprogramm-Schnittstelle erlaubt es, komplexere Auswertungen auf in der Datenbank gespeicherten Daten durchzuführen (z. B. Erosions- und Akkumulations-Berechnungen auf der Grundlage von statistischen Modellen [BH88]); ferner erlaubt sie die Entwicklung von einfach zu benutzenden Dialog-Schnittstellen für spezielle, immer wiederkehrende Datenbankoperationen.

Für die Anwendungsprogramm-Schnittstelle des Geo-Datenbanksystems wurde eine *Einbettungs-Strategie* gewählt, d. h. Anwendungsprogramme bestehen aus einer Mischung von Datenbankanweisungen und Anweisungen der Programmiersprache. Gegenüber anderen Ansätzen (prozedurale Schnittstellen, integrierte Datenbank-Programmiersprachen, vgl. z. B. [LP83, ESW88]) bietet die Einbettung zwei wichtige Vorteile: Zum einen kann in Anwendungsprogrammen dieselbe Datenbanksprache verwendet werden wie an der interaktiven Schnittstelle, und zum anderen ist keine Änderung der Programmiersprache und damit des Programmiersprachen-Übersetzers erforderlich, denn die Datenbankanweisungen können von einem Vorübersetzer (im folgenden *Datenbanksprach-Übersetzer* genannt) in Aufrufe entsprechender Module des Datenbankverwaltungssystems übersetzt werden. In Anwendungsprogrammen dürfen die Datenbankanweisungen parametrisiert werden, d. h. anstelle von Datenwerten können Programmvariable entsprechenden Typs angegeben werden.

Die in Abschnitt 2 als Beispiel angegebene RETRIEVE-Anweisung könnte – zusammen mit den zugrundeliegenden Range-Deklarationen – also auch in einem Anwendungsprogramm stehen. Anstelle der konstanten Bedingung „a.Jahr >= 1986“ im Qualifikationsteil könnte flexibler „a.Jahr >= :jahr“ geschrieben werden; dabei ist „jahr“ eine Programmvariable, an die vorher die jeweils gewünschte Jahreszahl zugewiesen wurde.

Da das Geo-Datenbanksystem die Handhabung beliebig geschachtelter komplexer Objekte ermöglicht, ist auch für Anwendungsprogramme ein objektorientierter Zugriff auf die Datenbank wünschenswert. Während [ESW88] eine solche Objektorientierung durch Definition entsprechend geschachtelter Record- und Arraytypen in Pascal erreicht, verfolgen wir beim Geo-Datenbanksystem einen auf den Ideen der Datenabstraktion [LG86] beruhenden Ansatz: Alle erforderlichen Objekttypen, Objektmengentypen, Objektlistentypen und geometrischen Datentypen werden dem Anwendungsprogramm als Abstrakte-Datentyp-Module (ADT-Module) mit den benötigten Operationen zur Verfügung gestellt. Als Programmiersprache wurde aufgrund dieses Ansatzes Modula-2 [Wi85] gewählt, da diese Sprache die Datenabstraktion durch ihr Konzept der Definitions- und Implementationsmodule gut unterstützt.

Als Beispiel wollen wir annehmen, daß die in Abschnitt 2 berechnete temporäre Objektklasse „Schnittfläche“ in einem Anwendungsprogramm gelesen und weiterverarbeitet werden soll. Zu diesem Zweck müssen zunächst ADT-Module für den Objekttyp „Schnittfläche“ und den Objektmengentyp „SET_OF_Schnittfläche“, ferner für den Unterobjekttyp „Anbaujahr“ und den entsprechenden Mengentyp „SET_OF_Anbaujahr“ generiert werden. Diese Generierung erfolgt weitgehend automatisch mit Hilfe von zwei Werkzeugen, dem *Objekttyp-Generator* und dem

Objekttyp-Übersetzer. Der Objekttyp-Generator erhält als Eingabe die RETRIEVE-Anweisung aus Abschnitt 2 zusammen mit den zugrundeliegenden Range-Deklarationen. Unter Zuhilfenahme des Datenbankkatalogs, in dem die Objektschemata der referenzierten Klassen „Bodenareal" und „Parzelle" beschrieben sind, erzeugt er hieraus ebenenweise die folgende Pseudocode-Spezifikation der beteiligten Objekt- und Objektmengentypen:

```
OBJECTTYPES
   SET_OF_Schnittfläche = SET_OF (Schnittfläche);
   Schnittfläche =
   OBJECT
      Bezeichnung      STRING (10),
      Geometrie        POLYGONS,
      Bodenart         STRING (20),
      Anbau            SET_OF_Anbaujahr
   END;
   SET_OF_Anbaujahr = SET_OF (Anbaujahr);
   Anbaujahr =
   OBJECT
      Jahr             INTEGER,
      Fruchtart        STRING (20)
   END
END_OBJECTTYPES
```

Aus dieser Pseudocode-Spezifikation – die der Benutzer auch „von Hand" hätte schreiben können – erzeugt der Objekttyp-Übersetzer Definitions- und Implementationsmodule für die vier spezifizierten Typen. Die ADT-Module für die Objekttypen „Schnittfläche" und „Anbaujahr" enthalten dabei u. a. Operationen zum Erzeugen von Objekten sowie zum Lesen und zum Ändern einzelner Attributwerte; die ADT-Module für die Objektmengentypen „SET_OF_Schnittfläche" und „SET_OF_Anbaujahr" beinhalten Operationen zum Erzeugen einer leeren Menge, zum Einfügen und Entfernen von Objekten, zur Prüfung, ob ein Objekt in der Menge vorkommt, zur Berechnung der Anzahl der aktuell enthaltenen Elemente usw.

Auf ähnliche Weise erzeugt eine weitere Komponente des Systems, der *Datentyp-Übersetzer*, die ADT-Module für vom Benutzer spezifizierte geometrische Datentypen. Der Typ POLYGONS beispielsweise könnte wie folgt spezifiziert werden:

```
DATATYPES
   DB_TYPE POLYGONS            REPR SET (SINGLE_POLYGON);
   NON_DB_TYPE SINGLE_POLYGON REPR LIST (SINGLE_POINT) (MIN = 3);
   NON_DB_TYPE SINGLE_POINT   REPR TUPLE (X: CARDINAL, Y: CARDINAL);
END_DATATYPES
```

POLYGONS wird in dieser Spezifikation in mehreren Stufen auf den Grunddatentyp CARDINAL zurückgeführt: Polygonmengen werden als Mengen von Einzelpolygonen erklärt, Einzelpolygone als Listen ihrer Stützpunkte (Einzelpunkte, mind. 3) und Einzelpunkte wiederum als Paare kartesischer Koordinaten (CARDINAL-Werte).

Die vom Datentyp-Übersetzer erzeugten ADT-Module für die Typen POLYGONS, SINGLE_POLYGON und SINGLE_POINT enthalten alle Operationen, die für das Zerlegen und Aufbauen von Werten dieser Typen erforderlich sind. Weitere geometrische Operationen, wie etwa Flächenverschneidungen, Flächeninhaltsberechnungen etc., müssen vom Benutzer selbst programmiert und als Modul dem Datentyp-Übersetzer zur Verfügung gestellt werden.

Der Datentyp-Übersetzer macht die mit DB_TYPE gekennzeichneten Typen sowie die zugehörigen vom Benutzer geschriebenen geometrischen Operationen auch dem Datenbanksprach-Übersetzer bekannt, so daß diese auch in Datenbankanweisungen verwendet werden können. Die mit NON_DB_TYPE gekennzeichneten Typen (im Beispiel SINGLE_POLYGON und SINGLE_POINT) dienen hingegen lediglich als „Hilfstypen", um die Bearbeitung komplexerer Geometrien (im Beispiel von POLYGONS-Werten, also Polygonmengen) in Anwendungsprogrammen zu ermöglichen; sie können in Datenbankanweisungen nicht verwendet werden.

Durch die Integration aller benötigten Typen als ADT-Module wird für den Anwendungsprogrammierer das Lesen und Verarbeiten von Daten aus der Datenbank sehr einfach. Als Beispiel wollen wir das folgende Programmstück betrachten, das auf den Beispielen aus Abschnitt 2 beruht; es wird berechnet, in wieviel Prozent der Gesamtfläche des Untersuchungsgebietes Lehmboden vorliegt:

```
/* Deklaration von Programmvariablen */

VAR schnitt_menge: SET_OF_Schnittfläche;
    einzel_fläche, gesamt_fläche, lehm_fläche, lehm_prozente: REAL;

/* Berechnung der temporären Objektklasse "Schnittfläche"
   (Datenbank-Anweisungen) */

...  wie in Abschnitt 2  ...

/* Kopieren der temporären Objektklasse in die
   Programmvariable "schnitt_menge" (Datenbank-Anweisungen) */

RANGE_OF s IS Schnittfläche;
WRITE_GEOOBJ s INTO :schnitt_menge;

/* Berechnung der Gesamtfläche des Untersuchungsgebietes sowie
   der Summe aller Lehmflächen im Untersuchungsgebiet */

gesamt_fläche := 0.0;
lehm_fläche   := 0.0;

FOR schnitt: Schnittfläche  IN  schnitt_menge  DO
    /* Bearbeitung der einzelnen Objekte */
    einzel_fläche := AREA (Schnittfläche.VALUE_Geometrie (schnitt));
    gesamt_fläche := gesamt_fläche + einzel_fläche;
    IF Schnittfläche.VALUE_Bodenart (schnitt) = "Lehm"
    THEN lehm_fläche := lehm_fläche + einzel_fläche
END;
```

```
/* Berechnung des Prozentsatzes Lehmfläche zu Gesamtfläche */
lehm_prozente := lehm_fläche * 100 / gesamt_fläche;
...
```

Mittels der WRITE-Anweisung wird die Geoobjektklasse „Schnittfläche", die mit der RETRIEVE-Anweisung aus Abschnitt 2 berechnet wurde, in die Programmvariable „schnitt_menge" gelesen, die gerade vom entsprechenden Typ SET_OF_Schnittfläche ist. Das FOR-Konstrukt erlaubt es, in einer vom System bestimmten Reihenfolge alle Elemente der Objektmenge zu bearbeiten; die Einzelobjekte werden nacheinander an die Laufvariable „schnitt" zugewiesen. Es handelt sich hierbei um ein Sprachkonstrukt, das die Ausdrucksfähigkeit von Modula-2 erweitert; es wird daher vor der Bearbeitung des Programms durch den Datenbanksprach-Übersetzer von einem weiteren Vorübersetzer, dem *Sprachkonstrukt-Übersetzer*, durch den Aufruf entsprechender Prozeduren ersetzt, die in den ADT-Modulen für Objektmengentypen enthalten sind. Die Operationen „Schnittfläche.VALUE_Geometrie" und „Schnittfläche.VALUE_Bodenart" sind enthalten in dem vom Objekttyp-Übersetzer erzeugten ADT-Modul für den Objekttyp „Schnittfläche"; der Aufruf dieser Operationen liefert den Geometrie- bzw. Bodenartwert des jeweils durch die Variable „jahr" bezeichneten Objektes.

Wir wollen uns mit dieser kurzen Übersicht über die Konzeption der Anwendungsprogramm-Schnittstelle begnügen; eine genauere Beschreibung findet sich in [Lo88]. Gleichzeitig beenden wir hiermit die Beschreibung der Anwendersicht auf das System und diskutieren im folgenden die Grundlagen der Implementierung.

4. Benutzerschnittstelle des Geo-Kerns

Der für die Verwaltung der Datenbasis gewählte Geo-Kern realisiert das *NF2-Relationenmodell* mit zusätzlichen *geometrischen Datentypen* und deren Unterstützung durch *räumliche Indexe* [SW86, Sc87, WHSW88]. Als Benutzerschnittstelle wird eine Sammlung von Prozeduren angeboten, die jeweils entsprechende Funktionen realisieren. So gibt es beispielsweise Prozeduren zum Definieren und Löschen von Relationenschemata; andere Prozeduren ermöglichen die Zusammenstellung von Qualifikationsformeln und Projektionslisten. Zur Bildung der Qualifikationsformeln sind Teilformeln der Art „Attribut Vergleichsoperator Konstante" möglich. Bei geometrischen Attributen werden die Vergleiche IN_WINDOW und INTERSECTS_WINDOW direkt vom Kern ausgewertet [HSWW88].

Zur Erhöhung der Effizienz wurde auf allen Implementierungsschichten des Geo-Kerns eine mengenorientierte Behandlung der NF2-Tupel realisiert. An der Benutzerschnittstelle spiegelt sich diese Mengenorientierung in dem Konzept der Übergabebereiche wider, die als Zwischenbehälter für Mengen von NF2-Tupeln fungieren. Eine Anfrage wird gestellt, indem man dem Geo-Kern eine Projektionsliste übergibt, eventuell mit zugehörigen Filterprädikaten. Das System stellt daraufhin die Antwortmenge zusammen, die immer aus einer Teilstruktur einer gespeicherten NF2-Relation besteht, und füllt einen Ergebnisübergabebereich mit dieser Tupelmenge. Bei der Eröffnung eines Übergabebereiches wird vom System automatisch ein zugehöriger (impliziter) Zeiger generiert, mit dessen Hilfe Attribute von Tupeln oder Subtupeln gelesen werden können; dabei wird der Zeiger mittels der Navigationskommandos UP, DOWN, NEXT, FIRST, PRIOR auf die entsprechenden Attribute positioniert.

Das folgende Programmstück illustriert die Anwendung dieser Prozeduren des Geo-Kerns. Zunächst wird die Struktur einer NF2-Relation „Parzelle" angegeben, auf die sich die folgende

Anfrage bezieht: Es sollen alle Namen von Parzellen, die in einem bestimmten Weltausschnitt liegen, zusammen mit ihren Anbaufrüchten von 1986 an am Bildschirm angezeigt werden.

```
RELATION Parzelle (Name          STRING,
                   Geometrie     POLYGON_GEO,
                   SUBRELATION   Anbaufrüchte
                                 (Jahr        INTEGER,
                                  Fruchtart   STRING));

VAR Parzellen_Name, Fruchtart_Bezeichnung: STRING;

RETRIEVE (RELATION:    Parzelle,
          ATTRIBUTES:  Parzelle(Name,Anbaufrüchte(Fruchtart)),
          PREDICATES:  Parzelle(Geometrie IN_WINDOW (...) AND
                                Anbaufrüchte(Jahr GE 1986)),
          RETURN:      Result_Buffer);

OPEN (Result_Buffer, RETURN: No_of_Tuples);
FOR i = 1 TO No_of_Tuples DO
BEGIN
    NEXT (Result_Buffer);
    READ (Result_Buffer, Parzellen_Name);
    DISPLAY (Parzellen_Name);
    DOWN (Result_Buffer, RETURN: No_of_Subtuples);
    FOR j = 1 TO No_of_Subtuples DO
    BEGIN
        NEXT (Result_Buffer);
        READ (Result_Buffer, Fruchtart_Bezeichnung);
        DISPLAY (Fruchtart_Bezeichnung)
    END;
    UP (Result_Buffer)
END;
```

Zunächst wird mit der Kern-Prozedur RETRIEVE der Suchvorgang angestoßen. Man erkennt die Projektionsliste (ATTRIBUTES: ...) und das Qualifikationsprädikat (PREDICATES: ...), das hier auch den geometrischen Vergleichsoperator IN_WINDOW enthält. Im Übergabebereich „Result_Buffer“ werden die Ergebnistupel abgelegt. Danach erfolgt das Durchmustern des Übergabebereiches mittels der erwähnten Positionier-Prozeduren NEXT, DOWN, UP. Als weitere Kern-Prozeduren treten noch die Routinen OPEN zum Eröffnen von Übergabebereichen und READ zum Transferieren von Attributinhalten aus dem Übergabebereich in Variable der Wirtssprache auf.

Neben den in diesem Beispiel aufgeführten gibt es noch eine Reihe weiterer Prozeduren, etwa zum Einfügen und Löschen von Tupeln in Übergabebereiche, zum Anlegen von (räumlichen) Indexen, zum Transferieren von Tupelmengen aus Übergabebereichen in NF2-Relationen etc. Zur Erläuterung dieser Funktionen sei auf [SW86, HSWW88, WHSW88] verwiesen.

Geoobjektklassen können direkt durch Kernrelationen implementiert werden, da lediglich Hierarchien auf Hierarchien abgebildet werden müssen. Außerdem stellt der Geo-Kern Mengen- sowie Tupelkonstrukte bereit, so daß auf der Ebene der *Datendefinition* nur wenig Arbeit zu leisten ist. Die Transformation von *Anfragen* der Geo-Datenbanksprache auf die Kernschnittstelle gestaltet sich komplexer. Dies wird anhand eines Beispiels im nächsten Abschnitt dargestellt.

5. Übersetzung von Anfragen

Bei der Übersetzung von Anfragen unserer Datenbanksprache gehen wir in zwei Stufen vor: Im ersten Schritt werden die Datenbankanweisungen syntaktisch/semantisch analysiert und dann auf Mengenausdrücke abgebildet, die denen aus den bekannten Tupel- oder Bereichskalkülen [Ul82, Ma83] ähnlich sind; im zweiten Schritt werden die Mengenausdrücke in Programmstücke mit eingebetteten Aufrufen des Geo-Kerns übersetzt. Das Ausführen dieser Programme schließlich bewirkt die Interpretation der Mengenausdrücke über dem aktuellen Datenbankzustand.

Diese Vorgehensweise wollen wir an der in Abschnitt 2 diskutierten Anfrage illustrieren, bei der die komplexe Objektklasse „Schnittfläche“ aus Exemplaren von „Bodenarealen“ und „Parzellen“ berechnet wird.

```
RANGE_OF b IS Bodenareal IN_WINDOW(4404000,5768000,4406000,5770000);
RANGE_OF p IS Parzelle IN_WINDOW(4404000,5768000,4406000,5770000);
RANGE_OF a IS p.Anbaujahr;
RETRIEVE_GEOOBJ_INTO Schnittfläche
   (Bezeichnung = p.Bezeichnung CAT b.Bezeichnung,
    Geometrie   = INTERSECTION (p.Geometrie, b.Geometrie),
    Bodenart    = b.Bodenart,
    Anbau       = SET_OF GEOOBJ Anbaujahr
                  (Jahr      = a.Jahr,
                   Fruchtart = a.Fruchtart))
WHERE CUT (p.Geometrie, b.Geometrie)
  AND a.Jahr >= 1986;
```

Der zu dieser Anfrage äquivalente Mengenausdruck enthält neben den wie üblich gebildeten Termen und Formeln als wesentliche Neuerung ein *Mengengleichheits-Prädikat.* Dieses drückt die Mengenbildung von Subobjekten aus.

```
{ s' | ∃ p ∈ Parzelle ∃ b ∈ Bodenareal
    (cut (Geometrie(p),Geometrie(b)) ∧
     within (Geometrie(p),(4404000,5768000,4406000,5770000)) ∧
     within (Geometrie(b),(4404000,5768000,4406000,5770000)) ∧
     Bezeichnung(s') = cat (Bezeichnung(p),Bezeichnung(b)) ∧
     Geometrie(s') = intersection (Geometrie(p),Geometrie(b)) ∧
     Bodenart(s') = Bodenart(b) ∧
     Anbau(s') = { a' | ∃ a ∈ Anbaujahr(p)
                   (Jahr(a) ≥ 1986 ∧
                    Jahr(a') = Jahr(a) ∧
                    Fruchtart(a') = Fruchtart(a) )} )}
```

Zur Umsetzung solcher geschachtelter Mengenausdrücke auf die Schnittstelle des Geo-Kerns werden zunächst alle mit logischem Und verknüpften Teilformeln, die vom Kern direkt ausgewertet werden können, aus den Qualifikationsprädikaten entfernt und als Filterprädikate für die Kernprozedur RETRIEVE vorgemerkt. Dann ist für jede Objektvaribale o, die im Kontext „$\exists$ o $\in$ *Geoobjektklasse*" vorkommt, ein Aufruf der RETRIEVE-Prozedur zu generieren. Dabei muß die jeweilige Projektionsliste alle Attribute enthalten, die im zu übersetzenden Mengenausdruck vorkommen. In unserem Beispiel sind deshalb die folgenden beiden RETRIEVE-Aufrufe zu generieren:

```
RETRIEVE (RELATION:   Parzelle,
          ATTRIBUTES: Parzelle(Bezeichnung, Geometrie,
                              Anbaufrüchte(Jahr, Fruchtart)),
          PREDICATES: Parzelle(Geometrie IN_WINDOW
                                 (4404000,5768000,4406000,5770000)
                              AND
                              Anbaufrüchte(Geometrie IN_WINDOW
                                    (4404000,5768000,4406000,5770000)
                                       AND Jahr GE 1986)),
          RETURN:     Parzelle_Buffer);

RETRIEVE (RELATION:   Bodenareal,
          ATTRIBUTES: Bodenareal(Bezeichnung, Geometrie, Bodenart),
          PREDICATES: -
          RETURN:     Bodenareal_Buffer);
```

Die Auswertung der verbleibenden Prädikate des WHERE-Teils und der Wertzuweisungen aus der ursprünglichen Zielliste wird von geschachtelten DO-Schleifen durchgeführt, deren Schachtelungsstruktur durch die Reihenfolge der Existenzquantoren im zu übersetzenden Mengenausdruck gegeben ist. Neben diesen Schleifen werden als weitere Konstrukte der Wirtssprache noch Variablen passenden Typs benötigt, die als Parameter der Übergabebereichsfunktionen READ und WRITE zum Transfer von Attributwerten dienen. In unserem Beispiel werden zu diesem Zweck folgende Variablen deklariert:

```
VAR  b_Bezeichnung : STRING,    p_Bezeichnung : STRING,
     b_Bodenart    : STRING,    a_Fruchtart   : STRING,
     a_Jahr        : INTEGER,
     b_Geometrie   : POLYGONS,  p_Geometrie   : POLYGONS;
```

Weitere Aktionen bestehen im Einrichten einer NF2-Relation mit passendem Schema, die die Ergebnismenge der zu übersetzenden Anfrage aufnehmen kann, sowie eines entsprechenden Übergabebereichs. Wir gehen davon aus, daß die Relation „Schnittfläche" und der Übergabebereich „Schnittfläche_Buffer" bereits eingerichtet sind. Damit ergibt sich folgendes Programmstück, das zusammen mit den oben angegebenen RETRIEVE-Aufrufen den übersetzten Mengenausdruck darstellt:

```
 1  OPEN (Parzelle_Buffer, RETURN: No_of_Parzelle_Tuples);
 2  FOR p=1 TO No_of_Parzelle_Tuples DO
 3  BEGIN
 4    NEXT (Parzelle_Buffer);
 5    READ (Parzelle_Buffer, p_Bezeichnung);
 6    READ (Parzelle_Buffer, p_Geometrie);
 7    OPEN (Bodenareal_Buffer, RETURN: No_of_Bodenareal_Tuples);
 8    FOR b=1 TO No_of_Bodenareal_Tuples DO
 9    BEGIN
10      NEXT (Bodenareal_Buffer);
11      READ (Bodenareal_Buffer, b_Bezeichnung);
12      READ (Bodenareal_Buffer, b_Geometrie);
13      READ (Bodenareal_Buffer, b_Bodenart);
14      IF cut(p_Geometrie, b_Geometrie)
15      THEN NEXT (Schnittfläche_Buffer);
16           WRITE (Schnittfläche_Buffer,
17                   cat(p_Bezeichnung, b_Bezeichnung);
18           WRITE (Schnittfläche_Buffer,
19                   intersection(p_Geometrie, b_Geometrie);
20           WRITE (Schnittfläche_Buffer, b_Bodenart);
21           DOWN (Schnittfläche_Buffer);
22           DOWN (Parzelle_Buffer, RETURN: No_of_Anbaujahr_Subtuples);
23           FOR a=1 TO No_of_Anbaujahr_Subtuples DO
24           BEGIN
25             NEXT (Parzelle_Buffer);
26             READ (Parzelle_Buffer, a_Jahr);
27             READ (Parzelle_Buffer, a_Fruchtart);
28             NEXT (Schnittfläche_Buffer);
29             WRITE (Schnittfläche_Buffer, a_Jahr);
30             WRITE (Schnittfläche_Buffer, a_Fruchtart);
31           END; /* Anbaujahr_Subtuples */
32           UP (Parzelle_Buffer);
33           UP (Schnittfläche_Buffer)
34      FI /* cut (p_Geometrie, b_Geometrie) */
35    END; /* Bodenareal_Tuples */
36    CLOSE (Bodenareal_Buffer)
37  END; /* Parzelle_Tuples */
38  CLOSE (Parzelle_Buffer);
39  INSERT (RELATION: Schnittfläche, BUFFER: Schnittfläche_Buffer);
```

Man erkennt die systematische Umsetzung des Mengenausdrucks; so entsprechen beispielsweise die Zeilen 1–4 dem Konstrukt „∃ p ∈ Parzelle“, die Zeilen 7–10 entsprechen „∃ b ∈ Bodenareal“. Da die beiden IN_WINDOW-Prädikate bereits vom Kern überprüft worden sind, braucht in Zeile 14 nur noch die Bedingung „cut(Geometrie(p),Geometrie(b))“ ausgewertet werden, bevor in den Zeilen 16–20 die Wertzuweisungen an die Attribute der Ergebnisrelation durchgeführt werden. Die Zeilen 23–31 beinhalten die Berechnung der Werte der Subtupel-Attribute. Außerdem treten noch zahlreiche Aufrufe der in Abschnitt 4 beschriebenen Positionierprozeduren auf, z. B. in den Zeilen 21, 22, 32, 33. Nachdem im Ergebnisübergabebereich alle Tupel mit ihren Subtupeln aufgebaut worden sind, werden sie als letzte Aktion, Zeile 39, in die Resultatsrelation „Schnittfläche“ geschrieben.

Aus diesem Programmstück und den weiter oben angegebenen RETRIEVE-Aufrufen sowie den Variablen-Deklarationen wird ein Modul gebildet, das zur Laufzeit des Anwendungsprogramms, in das die Anfrage eingebettet ist, aufgerufen wird. Die ursprüngliche Datenbankanfrage wird dabei im Anwendungsprogramm durch den Aufruf dieses Moduls ersetzt.

Die Übersetzung anderer Datenbankanweisungen, etwa DELETE oder UPDATE, ähnelt zwar dem Vorgehen bei Anfragen, setzt sich jedoch aus mehreren Teilschritten zusammen. Zur Erläuterung dieser Sachverhalte verweisen wir aus Platzgründen auf [Ne88]. Dort wird auch die Umsetzung der Karten- und Kartenobjekt-Konstrukte diskutiert.

6. Gesamtkonzeption und Ausblick

Verglichen mit der inzwischen klassischen Vorgehensweise bei der Implementierung von relationalen Datenbanksystemen [Hä87] weist unser System zwei wesentliche Unterschiede auf: zum einen die Werkzeuge, die die Einbettung in die Programmiersprache und die Erweiterbarkeit der Datenbanksprache um neue Datentypen realisieren (siehe Abschnitt 3), zum anderen die Tatsache, daß als Impementierungsschicht der in Abschnitt 4 vorgestellte Geo-Kern benutzt wird. Die tieferen Systemschichten, wie Speicherungs- und Zugriffssysteme, wurden deshalb von uns hier nicht betrachtet.

Die daraus resultierende Gesamtkonzeption des Datenbanksystems stellt sich wie folgt dar: Aus den für die jeweilige Anwendung relevanten Datenbankanweisungen erzeugen der *Objekttyp-Generator* und der *Objekttyp-Übersetzer* die ADT-Module, die benötigt werden, um die von der Datenbank gelesenen Objektmengen im Anwendungsprogramm handhaben zu können. Analog generiert der *Datentyp-Übersetzer* aus Datentyp-Spezifikationen die ADT-Module für die Handhabung benutzerdefinierter Geometrietypen; darüber hinaus macht er die neu definierten Datentypen den übrigen Systemkomponenten bekannt, so daß diese Typen auch in den Datendefinitions- und Datenmanipulations-Anweisungen der Datenbanksprache verwendet werden können. Das eigentliche Anwendungsprogramm wird nacheinander von zwei Vorübersetzern bearbeitet: Zunächst ersetzt der *Sprachkonstrukt-Übersetzer* jedes Auftreten des neu eingeführten Sprachkonstruktes für die Iteration über Objektmengen durch entsprechende Prozeduraufrufe. Hierauf folgt die Bearbeitung durch den *Datenbanksprach-Übersetzer*: Dieser ersetzt die in das Anwendungsprogramm eingestreuten Datenbankanweisungen, wie in Abschnitt 5 ausgeführt, in mehreren Schritten durch entsprechende Module. Zum Schluß können das Anwendungsprogramm und alle neu generierten Module durch einen Programmiersprachen-Übersetzer in ausführbaren Objektcode umgesetzt werden.

Eine Untermenge der in Abschnitt 2 kurz vorgestellten Datenbanksprache wurde inzwischen durch einen ersten experimentellen Prototyp implementiert [JN88]. Dieser stützt sich auf eine relationale Datenbankmaschine (IDM 500 [Br84]) und bietet zunächst nur atomare Geoobjektklassen, aber bereits die wichtigsten geometrischen Datentypen mit zugehörigen Operationen. Des weiteren wurden die Konzepte der Karten und Kartenobjekte vollständig implementiert. Bei ersten realistischen Anwendungen, die im Rahmen des erwähnten Schwerpunktprogrammes durchgeführt wurden [Os87, He88, Ti88], zeigte sich neben der Adäquatheit der entworfenen Datenbanksprache auch die Problematik der langen Antwortzeiten eines Nichtstandard-Datenbanksystems, das auf der Basis eines konventionellen Datenbanksystems implementiert ist. Ähnliche Erfahrungen sind bereits aus anderen Projekten bekannt (etwa [HHLM87] und [AL88]). Durch den im zweiten Prototyp zur Implementierung vorgesehenen Geo-Kern hoffen wir jedoch, die Antwortzeiten erheblich reduzieren zu können.

Literatur

[AL88] Appelrath, H.J.; Lorek, H.: Der Einsatz von Prolog-Werkzeugen für Geo-Datenbanken. Proc. Non-Standard-Datenbanken für Anwendungen der Graphischen Datenverarbeitung, Lutterbach, H. (Hrsg.), Dortmund 1988, 147–165.

[ALPS88] Andersen, F.; Linnemann, V.; Pistor, P.; Südkamp, N.: AIM-P, User Manual for the Online Interface of the Heidelberg Data Base Language (HDBL) Prototype Implementation. TN 86.01, Heidelberg 1988.

[BH88] Bork, H.-R.; Hensel, H.: Computer-Aided Construction of Erosion Maps. In [NLfB88].

[BK86] Bancilhon, F.; Khoshafian, S.: A Calculus for Complex Objects. Proc. 5th Symp. on Principles of Database Systems, 1986, 53–59.

[BM86] Batory, D.S.; Mannino, M.: Panel on Extensible Database Systems. Proc. SIGMOD'86, Zaniolo, C. (Hrsg.), 1986, 187–190.

[Br84] Britton Lee Inc.: IDM Software Reference Manual, Version 1.7. Los Gatos (CA), 1984.

[CDWF86] Carey, M.J.; DeWitt, D.J.; Frank, D.; Graefe, G.; Muralikrishna, M.; Richardson, J.E.; Shekita, E.J.: The Architecture of the EXODUS Extensible DBMS. In [DD86], 52–65.

[Ch76] Chen, P.P.: The Entity-Relationship Model – Toward a Unified View of Data. ACM Transactions on Database Systems, Vol. 1, No. 1, 1976, 9–36.

[DD86] Dittrich, K.; Dayal, U. (Hrsg.): Proc. Int. Workshop on Object-Oriented Database Systems. Pacific Grove 1986.

[DDKL87] Dadam, P.; Dillmann, R.; Kemper, A.; Lockemann, P.C.: Objektorientierte Datenhaltung für die Roboterprogrammierung. Informatik Forschung und Entwicklung, Band 2, Nr. 2, 1987, 151–170.

[Eh85] Ehrich, H.-D.: Spezifikation konzeptioneller Schemata mit abstrakten Datentypen und Versionen. Proc. GI-Fachgespräch „Entwurf von Informationssystemen – Methoden und Modelle". Mayr, H.C.; Meyer, B.E. (Hrsg.). Tutzing 1985, 1–19.

[ELNR88] Ehrich, H.-D.; Lohmann, F.; Neumann, K.; Ramm, I.: A Database Language for Scientific Map Data. In [NLfB88].

[ESW88] Erbe, R.; Südkamp, N.; Walch, G.: An Application Program Interface for a Complex Object Database. Proc. 3rd Int. Conf. on Data and Knowledge Bases, Jerusalem 1988.

[Hä87] Härder, T.: Realisierung von operationalen Schnittstellen. In: Datenbank-Handbuch. Lockemann, P.C.; Schmidt, J.W. (Hrsg.). Springer: Heidelberg 1987, 163–335.

[He88] Heitland, M.: Der Einsatz eines Geo-Datenbanksystems auf dem Gebiet der Bodenerosion. Studienarbeit, TU Braunschweig 1988.

[HG88] Hohenstein, U; Gogolla, M.: A Calculus for an Extended Entity-Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions. Proc. 7th Int. Conf. on Entity-Relationship Approach, Batini, C. (Hrsg.), North Holland, Amsterdam, erscheint 1988.

[HHLM87] Härder, T.; Hübel, C.; Langenfeld, S.; Mitschang, B.: KUNICAD – ein datenbankgestütztes geometrisches Modellierungssystem für Werkstücke. Informatik Forschung und Entwicklung, Band 2, Nr. 1, 1987, 1–18.

[HMMS87] Härder, T.; Meyer-Wegener, K.; Mitschang, B.; Sikeler, A.: PRIMA - a DBMS Prototype Supporting Engineering Applications. In [SKH87], 433–442.

[HNSE87] Hohenstein, U.; Neugebauer, L.; Saake, G.; Ehrich, H.-D.: Three-Level-Specification of Databases Using an Extended Entity-Relationship Model. Proc. Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen, Linz 1987, 58–88.

[HS86] Hommel, G.; Schindler, S. (Hrsg.): GI – 16. Jahrestagung, Proceedings I. Berlin 1986.

[HSWW88] Horn, D.; Schek, H.-J.; Waterfeld, W.; Wolf, A.: Spatial Access Paths and Physical Clustering in a Low-Level Geo-Database System. In [NLfB88].

[JN88] Jungclaus, R.; Neumann, K.: Benutzerhandbuch zum ersten Prototypen des Braunschweiger Geo-Datenbanksystems. Informatik-Bericht Nr. 88-01, TU Braunschweig 1988.

[KTW85] Kappel, G.; Tjoa, A.M.; Wagner, R.R.: Form Flow Systems Based on NF2-Relations. Proc. Datenbanksysteme für Büro, Technik und Wissenschaft. Blaser, A.; Pistor, P. (Hrsg.). Karlsruhe 1985, 234–252.

[KW87] Kemper, A.; Wallrath, M.: Konzepte zur Integration abstrakter Datentypen in R2D2. Proc. Datenbanksysteme für Büro, Technik und Wissenschaft. Schek, H.J.; Schlageter, G. (Hrsg.). Darmstadt 1987, 344–359.

[LG86] Liskov, G.; Guttag, J.: Abstraction and Specification in Program Development. McGraw-Hill: 1986.

[LKDP87] Linnemann, V.; Küspert, K.; Dadam, P.; Erbe, R.; Kemper, A.; Südkamp, N.; Walch, G.; Wallrath, M.: Design and Implementation of an Extensible Database Management System Supporting User Defined Types and Functions. TR 87.12.011, Heidelberg 1987.

[LN87] Lipeck, U.W.; Neumann, K.: Modelling and Manipulating Objects in Geoscientific Databases. In: Entity-Relationship Approach: Ten Years of Experience in Information Modelling (Proc. Int. Conf.), Spaccapietra, S. (Hrsg.). North-Holland, Amsterdam 1987, 67–86.

[Lo88] Lohmann, F.: Processing Non-Standard Database Objects in a Higher Level Programming Language - An Abstract Data Type Approach. Proc. Int. Workshop on Software Engineering and its Applications, Toulouse, erscheint 1988.

[LP83] Lacroix, M.; Pirotte, A.: Comparison of Database Interfaces for Application Programming. Information Systems, Vol. 8, No. 3, 1983, 217–229.

[Ma83] Maier, D.: The Theory of Relational Databases. Pitman, 1983.

[Me87] Meier, A.: Erweiterung relationaler Datenbanksysteme für technische Anwendungen. Springer: Heidelberg 1987.

[Ne88] Neumann, K.: Eine geowissenschaftliche Datenbanksprache mit benutzerdefinierbaren geometrischen Datentypen. Dissertation, TU Braunschweig 1988.

[NR86] Noltemeier, H.; Ruland, D.: Datenmodellierung in Geo-Datenbanken. In [HS86], 470–482.

[NLfB88] Niedersächsisches Landesamt für Bodenforschung (Hrsg.): Construction and Display of Geoscientific Maps Derived from Databases (Proc. Int. Coll.). Geologisches Jahrbuch, Sonderband, Hannover, erscheint 1988.

[Os87] Osterhold, A.: Der Einsatz eines geowissenschaftlichen Datenbanksystems im Bereich der Ökologie. Studienarbeit, TU Braunschweig 1987.

[PA86] Pistor, P.; Andersen, F.: Designing a Generalized NF2 Data Model with an SQL-Type Language Interface. Proc. 12th VLDB 1986, Kambayashi, Y. (Hrsg.), 1986, 278–288.

[RKB87] Roth, A. M.; Korth, H. F.; Batory, D. S.: SQL/NF: A Query Language for ¬1NF Relational Databases. Information Systems 12 (1987), 99–144.

[Sc86] Schek, H.-J.: Datenbanksysteme für die Verwaltung geometrischer Objekte. In [HS86], 483–497.

[Sc87] Schek, H.-J.: Ein Datenbank-Kernsystem für anwendungsspezifische Schichten – Architektur der DASDBS-Familie. Informationstechnik 3 (1987), 153–164.

[ScS83] Schek, H.-J.; Scholl, M.H.: Die NF2-Relationenalgebra zur einheitlichen Manipulation externer, konzeptueller und interner Datenstrukturen. Proc. Sprachen für Datenbanken, 1983, 113–133.

[ScS86] Schek, H.-J.; Scholl, M.H.: The Relational Model with Relational-Valued Attributes. Information Systems 11 (1986), 137–147.

[SKH87] Stocker, P. M.; Kent, W.; Hammersley, P. (Hrsg.): Proc. of the 13th Int. Conf. on Very Large Data Bases. Brighton 1987.

[SP82] Schek, H.-J.; Pistor, P.: Data Structures for an Integrated Data Base Management and Information Retrieval System. Proc. 8th VLDB, 1982, 197–207.

[SSE87] Sernadas, A.; Sernadas, C.; Ehrich, H.-D.: Object-Oriented Specification of Databases: An Algebraic Approach. In [SKH87], 107–116.

[SW86] Schek, H.-J.; Waterfeld, W.: A Database Kernel System for Geoscientific Applications. Proc. 2nd Int. Symposium on Spatial Data Handling, Seattle 1986, 273–288.

[SWKH76] Stonebraker, M.; Wong, E.; Kreps, P.; Held, G.: The Design and Implementation of INGRES. ACM Transactions on Database Systems, Vol. 1, No. 3, 1976, 189–222.

[Ti88] Tietjen, S.: Einrichtung von virtuellen Basiskarten auf einem geowissenschaftlichen Datenbanksystem am Beispiel von Island. Studienarbeit, TU Braunschweig 1988.

[Ul82] Ullman, J.D.: Principles of Database Systems. 2nd ed., Computer Science Press, Rockville (Md.) 1982.

[Vi85] Vinken, R.: Digitale geowissenschaftliche Kartenwerke – ein neues Schwerpunktprogramm der Deutschen Forschungsgemeinschaft. Nachrichten aus dem Karten- und Vermessungswesen, Reihe I, Heft 95 (1985), 163–173.

[Vi88] Vinken, R.: Digital geoscientific Maps: A Dream or a Chance? In [NLfB88].

[WHSW88] Waterfeld, W.; Horn, D.; Schek, H.-J.; Wolf, A.: How to Make Spatial Access Methods Extensible? Proc. 3rd Int. Symp. on Spatial Data Handling, Sydney 1988.

[Wi85] Wirth, N.: Programming in Modula 2, Third corrected Edition. Springer: Berlin, Heidelberg, New York, Tokyo 1985.

[WSSH88] Wilms, P.F.; Schwarz, P.M.; Schek, H.-J.; Haas, L.M.: Incorporating Data Types in an Extensible Database Architecture. Proc. 3rd Int. Conf. on Data and Knowledge Bases, Jerusalem 1988.