Auf dem Weg zu einer integrierten Datenbankentwurfsumgebung

G. Engels, U. Hohenstein, G. Saake, H.-D. Ehrich
TU Braunschweig, Informatik, Abt. Datenbanken
Postfach 33 29, 3300 Braunschweig

Zusammenfassung

Die hier vorgestellte Datenbank-Entwurfsumgebung hat zum Ziel, Software-Werkzeuge zur Unterstützung des Erstellens, Änderns, Analysierens und Testens eines konzeptionellen Datenbankschemas unter einer einheitlichen Benutzerschnittstelle zu integrieren. Das Vorhaben basiert auf umfangreichen theoretischen Vorüberlegungen zur Festlegung eines semantisch wohldefinierten konzeptionellen Datenmodells. Die zugehörigen Schemata bestehen aus einer 4-Schichten-Spezifikation. In den ersten beiden Schichten werden Objekte und Beziehungen zwischen diesen durch eine erweiterte Form von Entity-Relationship-Diagrammen und anwendungsspezifische Attributwertebereiche durch algebraische Spezifikationen modelliert. In der dritten Schicht werden durch dynamische Integritätsbedingungen in der Form temporaler Formeln zulässige Zustandsfolgen der entworfenen Datenbank aktionsunabhängig festgelegt, während in der vierten Schicht schließlich die erlaubten Aktionen auf einer Datenbank festgelegt werden.
Zu den zu entwickelnden Werkzeugen gehört neben komfortablen, syntaxgestützten Text- bzw. Graphikeditoren zur inkrementellen Erstellung und Änderung eines konzeptionellen Datenbankschemas insbesondere ein Ausführungswerkzeug, durch das das entworfene Datenbankschema in ein relationales Schema transformiert und eine entsprechende Testdatenbank mit künstlichen Testdaten installiert wird, auf der interaktiv vom Datenbankentwerfer Anfragen und Aktionen ausgeführt werden können. Ein Integritätsmonitor überwacht dabei die Einhaltung der spezifizierten Integritätsbedingungen.

Gliederung

(IN: Proc. GI Workshop 'Sprachspezifische Programmierumgebungen' (G. Snelting, ed), TH Darmstadt 1988, S. 38-51)

1 Einleitung

Auf dem Gebiet der **Software-Technologie** existieren zahlreiche Bemühungen, die Probleme bei der Entwicklung umfangreicher Softwaresysteme in den Griff zu bekommen. Drei wesentliche Ansätze sind dabei die Strukturierung des Software-Entwicklungsprozesses durch ein geeignetes Softwarelebenszyklusmodell, die Bereitstellung geeigneter Spezifikations- und Implementierungssprachen und zugehöriger Methoden für die einzelnen Phasen des Softwarelebenszyklus und schließlich die Unterstützung der Tätigkeiten eines Software-Entwicklers durch Software-Werkzeuge innerhalb einer Software-Entwicklungsumgebung (vgl. z.B. /Ba 82/).

Analoge Ansätze können auch auf dem Gebiet der **Datenbank-Technologie** beobachtet werden, um die Probleme beim Entwurf einer Datenbank in den Griff zu bekommen. Zur Strukturierung des Entwurfsvorgangs unterscheidet man hier zwischen den folgenden Schritten (vgl. /MDL 87/):

(a) Informationsbedarfsanalyse
(b) konzeptioneller Entwurf
(c) logischer Entwurf
(d) physischer Entwurf

Die Ergebnisse der einzelnen Entwurfsschritte werden in Dokumenten festgehalten, die bei den Schritten (b) bis (d) **Schema** genannt werden.
Eine immer größere Bedeutung hat hierbei der Schritt des konzeptionellen Entwurfs bekommen, in dem noch unabhängig von Realisierungsdetails die Struktur einer Datenbank und zugehörige Aktionen durch ein konzeptionelles Schema modelliert werden. Ausgehend von der Erkenntnis, daß bei Nicht-Standard-Anwendungen die klassischen Datenmodelle des hierarchischen, Netzwerk- oder auch relationalen Datenmodells keine angemessenen Modellierungshilfsmittel zur Verfügung stellen, wurden geeignetere konzeptionelle (sogenannte semantische) Datenmodelle entwickelt. Bekannte Ansätze sind hierbei Erweiterungen des relationalen Datenmodells (/LK 84/, /SS 86/, /Da 86/) oder Erweiterungen des Entity-Relationship-Modells (/Ch 76/, /EWH 85/, /HNSE 87/).

Schließlich wurden analog zu Software-Entwicklungsumgebungen auch im Datenbankbereich eine Reihe von Umgebungen entwickelt, deren Ziel es ist, die Tätigkeiten beim Entwerfen und Implementieren einer Datenbank durch Software-Werkzeuge geeignet zu unterstützen. /LM 86/ gibt hierzu einen umfassenden Überblick. Auch hier wurden schwerpunktmäßig Werkzeuge zur Unterstützung des konzeptionellen Entwurfs entwickelt. Übliche Werkzeuge sind Editoren (insbesondere auch graphische) zum Erstellen und Ändern eines konzeptionellen Schemas (z.B. /BNTT 85/) und auch Ausführungswerkzeuge, die ein schnelles Testen des entworfenen Schemas im Rahmen eines "rapid prototyping" ermöglichen (z.B. /OSLS 86/).

Auch die von uns geplante, hier vorgestellte **Datenbank-Entwurfsumgebung** (DBEU) hat zum Ziel, Werkzeuge zum Erstellen, Ändern, Analysieren und Testen eines konzeptionellen Datenbankschemas für Nicht-Standard-Anwendungen einem Datenbankentwickler zur Verfügung zu stellen. Hierbei wird auf die folgenden Charakteristika besonderer Wert gelegt:

Grundlage für alle Werkzeuge ist ein **semantisch wohldefiniertes konzeptionelles Datenmodell** (/HNSE 87/, /HG 88/). In einem konzeptionellen Datenbankschema wird dabei neben der statischen Struktur einer Datenbank (auf der Grundlage eines erweiterten Entity-Relationship-Modells) auch das dynamische Verhalten der Datenbank durch die Angabe dynamischer Integritätsbedingungen (/LS 87/) und anwendungsabhängiger Aktionen spezifiziert. Wir erläutern dies ausführlicher an einem Beispiel im nächsten Kapitel.

Die Datenbank-Entwurfsumgebung ist von Anfang an als **integrierte Umgebung** konzipiert. Das betrifft sowohl das externe Verhalten der Werkzeuge im Dialog mit dem Entwerfer als auch die interne Realisierung der DBEU. Hierzu gehört insbesondere eine **einheitliche Benutzerschnittstelle** für alle Werkzeuge. Das heißt, daß der Bildschirmaufbau, die Kommandosprache und das Dialogverhalten einheitlich für alle Werkzeuge sind. Weiterhin bedeutet dies, daß alle Werkzeuge eine **inkrementelle Arbeitsweise** unterstützen. Dies ermöglicht dem Entwickler, ohne Aufwand zwischen den einzelnen Werkzeugen zu wechseln und dadurch ein Datenbankschema schrittweise zu erstellen und zu testen.

Schließlich ist die DBEU als **offene** und **anpaßbare** Umgebung konzipiert. Das bedeutet, daß eine eventuelle Erweiterung des Kommando- oder Werkzeugsatzes oder gar die Unterstützung weiterer Aufgabenbereiche von Anfang an berücksichtigt wird. Hierbei ist insbesondere daran gedacht, die DBEU um Werkzeuge zur Unterstützung aller Schritte des Entwurfs, also auch des logischen und physischen Entwurfs einer Datenbank zu erweitern und somit ausgehend von der bisher vorgesehenen Prototypimplementierung auch eine effiziente Implementierung einer Datenbank zu unterstützen. Denkbar ist dann auch die Einbettung dieser DBEU in eine Software-Entwicklungsumgebung, in der alle Tätigkeiten bei der Software-Entwicklung unterstützt werden.

Um unabhängig von der jeweils zur Verfügung stehenden Hardware zu sein, wird die **Portabilität** der DBEU durch die Benutzung von verbreiteten, weitgehend normierten Schnittstellen wie UNIX, C und X-Windows gewährleistet.
Auf diese Aspekte einer zugehörigen Software-Architektur und auf die Funktionalität und Realisierung der einzelnen Werkzeuge der geplanten Datenbank-Entwurfsumgebung gehen wir ausführlich im dritten Kapitel ein.

2 Semantisches Datenmodell

Grundlage für alle Werkzeuge zum Entwickeln eines konzeptionellen Datenbankschemas ist ein wohldefiniertes semantisches Datenmodell. Bei diesem Datenmodell, das im Rahmen umfangreicher theoretischer Arbeiten entwickelt wurde (/HNSE 87/, /HG 88/), wurden insbesondere Anforderungen berücksichtigt, die bei der Modellierung von Nicht-Standard-Anwendungen auftreten. Die daraus resultierende Spezifikationsmethodik wird in diesem Kapitel dargestellt. Insgesamt liegt diesem Ansatz folgende semantische Modellbildung zugrunde (vgl. /SNHE 87/):

Ein einzelner Datenbankzustand entspricht einem Modell zu einer prädikatenlogischen Spezifikation. Das dynamische Verhalten wird durch eine Folge von Datenbankzuständen charakterisiert, die durch die ausgeführten Aktionen induziert wird. Ein konzeptionelles Datenbankschema beschreibt somit

- die statische Struktur der Datenbankzustände,
- die erlaubten Folgen von Datenbankzuständen und
- die zur Verfügung gestellten Aktionen.

Die Beschreibung der statischen Struktur wird dabei noch einmal unterteilt in die Beschreibung einer Daten- und einer Objektschicht, so daß die gesamte Beschreibung aus einer vierschichtigen Spezifikation besteht.

In der **Datenschicht** werden die anwendungsspezifischen Attributwertebereiche in Form von Datentypspezifikationen festgelegt. Hier können über die üblichen Standarddatentypen (z.B. INTEGER, STRING) hinaus weitere, insbesondere bei Nicht-Standard-Anwendungen benötigte, anwendungsspezifische Datentypen definiert werden. Dies kann z.B. durch Gleichungsspezifikationen in Form algebraischer Spezifikationen geschehen. Wir geben hierzu das folgende Beispiel an, in dem die Spezifikation der beiden Datentypen POINT und LINES skizziert ist.

```
data type  POINT;
   imports  REAL;
   sorts  point = record
                     x, y : real
                  end;
   opns  Abstand : point × point → real;
   eqns  Abstand ( p1, p2) = Sqrt ( (p1.x - p2.x ) ↑ 2 + (p1.y - p2.y) ↑ 2 )
end  POINT;

data type  LINES;
   imports  POINT;
   sorts  lines = list of  point;
   ...
end LINES;
```

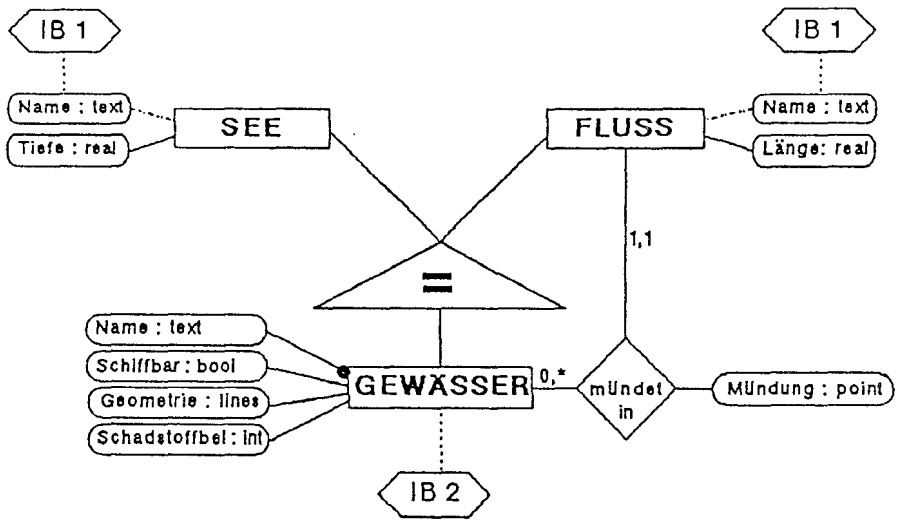
Abb. 2.1: Beispiele für Datentypspezifikationen

Die Erstellung der Gesamtspezifikation wird dabei unterstützt durch die Bereitstellung einiger vordefinierter Sortenkonstruktoren wie z.B. Verbund- oder Listenbildung (im Sinne parametrisierter abstrakter Datentypen) und durch die Möglichkeit, bereits existierende Spezifikationen durch die Benutzung einer *imports*-Klausel zu erweitern.

Die **Objektschicht** enthält die Spezifikation der Objekttypen, d.h. der Objektsorten mit ihren Attributen, und Beziehungen zwischen ihnen. Die Grundlage dazu bildet ein erweitertes Entity-Relationship-Modell (kurz: EER-Modell), welches das ursprüngliche ER-Modell (/Ch 76/) um die Konzepte

Typkonstruktion (Generalisierung/ Spezialisierung nach /SS 77/), komplexe Objekte (/RNLE 85/ /LN 87/), abgeleitete Informationen, ein erweitertes Schlüsselkonzept (/EDG 86/) und die Möglichkeit zur Spezifikation statischer Integritätsbedingungen ergänzt. Die oben erläuterten Datentypen werden dabei als Wertebereiche der Attribute verwendet.

Die Elemente der Objektschicht werden in der Regel graphisch in der Form eines EER-Diagramms dargestellt. Als Beispiel erläutern wir hier die Spezifikation eines Objekttyps GEWÄSSER, der aus der Schlüsselattribut "Name" und drei weiteren Attributen besteht. Durch die Typkonstruktion der totalen Partition werden zwei weitere Objekttypen spezifiziert, nämlich SEE und FLUSS. Dies besagt, daß jedes Objekt vom Typ GEWÄSSER gleichzeitig auch ein Objekt vom Typ SEE oder FLUSS ist. Diese Objekte werden dadurch in einem neuen Kontext gesehen und können weitere Attribute oder auch vererbte, berechnete Attribute besitzen. Im Beispiel wird durch die Berechnungsvorschrift IB1 das Attribut "Name" vererbt. Zwischen zwei Objekttypen können Beziehungen existieren, deren Kardinalität eingeschränkt sein kann. Im Beispiel existiert die Beziehung "mündet-in" mit der durch die Angabe "1,1" geforderten Einschränkung, daß jedes Objekt vom Typ FLUSS genau einmal in Beziehung steht zu einem Objekt vom Typ GEWÄSSER. Entsprechend dürfen auf Grund der Angabe "0,*" beliebig viele Flüsse in ein bestimmtes Gewässer münden. Das Beispiel demonstriert weiterhin die Benutzung von getrennt spezifizierten Datentypen (lines, point) als Attributwertebereiche. Schließlich kann durch die Angabe statischer Integritätsbedingungen die Struktur eines Datenbankzustands eingeschränkt werden. Die angegebene Integritätsbedingung IB2 besagt, daß für alle Objekte vom Typ GEWÄSSER das Attribut "Schiffbar" den Wert FALSE hat, falls das Attribut "Schadstoffbel(astung)" einen Wert hat, der größer ist als 100.



IB1: Name = GEWÄSSER.Name

IB2: ***forall*** (X : GEWÄSSER)
(Schadstoffbel (X) > 100 ⇒ ***not*** Schiffbar (X))

Abb. 2.2: Beispiel für ein EER-Diagramm

In der Daten- und Objektschicht wird somit die statische Struktur der zulässigen Zustände einer Datenbank beschrieben. Die Beschreibung des dynamischen Verhaltens der Datenbank erfolgt in den nächsten beiden Schichten, der Entwicklungs- und der Aktionsschicht:

In der **Entwicklungsschicht** werden die zulässigen Entwicklungen, d.h. die erlaubten zeitlichen Abfolgen von Zuständen einer Datenbank, auf deskriptive Art und Weise durch dynamische Integritätsbedingungen spezifiziert. Dies geschieht durch die Angabe von Formeln in temporaler Logik, einer Erweiterung des Prädikatenkalküls 1. Stufe (/LS 87/, /Sa 88/). Als Beispiel betrachte man die in Abb. 2.3 stehende temporale Formel, durch die beschrieben wird, daß ein Gewässer, das zu einem bestimmten Zeitpunkt eine Schadstoffbelastung hatte, die höher als 100 war, erst dann wieder schiffbar sein darf, wenn die Schadstoffbelastung unter den Wert 10 gesunken ist.

forall (X : Gewässer)
  always ((Schadstoffbel (X) > 100) ⇒
    (*not* Schiffbar (X) *until* Schadstoffbel (X) < 10))

Abb. 2.3: Beispiel für eine dynamische Integritätsbedingung

In einem zugehörigen semantischen Modell müssen somit Folgen von Datenbankzuständen berücksichtigt werden. Das bedeutet, daß semantische Modelle für Datenbankschemata mit dynamischen Integritätsbedingungen somit Abbildungen von mathematischen Zeitmodellen (z.B. der linearen diskreten Zeit mit Anfangszeitpunkt) in Mengen von Modellen sind, wobei jedes Modell einen einzelnen Datenbankzustand repräsentiert.

In der **Aktionsschicht** werden die Aktionen festgelegt, mit denen schreibend oder lesend auf einen aktuellen Datenbankzustand zugegriffen werden darf. Für lesende Zugriffe, d.h. Anfragen an eine Datenbank, steht eine komfortable Anfragesprache zur Verfügung, deren Semantik durch die Definition eines EER-Objektkalküls abgesichert ist (/HG 88/). Das heißt insbesondere, daß Sprachmittel zur Verfügung stehen, um in Anfragen die in der Objekt- und Datenschicht festgelegten strukturellen Zusammenhänge entsprechend zu berücksichtigen. Wir erläutern dies an zwei Anfragen zu dem bisher beschriebenen Beispielschema, die im EER-Objektkalkül formalisiert sind.

Die erste Anfrage liefert den prozentualen Anteil von Objekten vom Typ FLUSS unter allen Objekten vom Typ GEWÄSSER.

-[(100 * CNT -[f | (f : FLUSS)]-) / CNT -[g | (g : GEWÄSSER)]- |]-

Abb. 2.4: Beispielanfrage 1

Die Anfragesprache enthält eine Reihe vordefinierter aggregierender Funktionen. Im obigen Beispiel wird die Funktion CNT benutzt, die angewandt auf eine Resultatmenge die Anzahl der Elemente liefert.

Die zweite Anfrage liefert die Namen aller Flüsse, die in einen See münden, der eine hohe Schadstoffbelastung (> 50) hat. In der Anfrage enthalten ist ein vordefiniertes IS-Prädikat, das bei Typkonstruktionen entscheidet, ob ein Objekt eines Eingangstyps (hier GEWÄSSER) auch ein Objekt eines Ausgangstyps (hier SEE) ist. Weiterhin ist zu erwähnen, daß das Ergebnis einer Anfrage Multimengen sind, so daß in dem Beispiel bei Namensgleichheit zweier einmündender Flüsse der Name beider Flüsse im Anfrageergebnis steht.

$$\dashv\{\ \mathrm{Name}(f)\ |\ (f : \mathrm{FLUSS}) \wedge \exists\ (s : \mathrm{SEE})\ \exists\ (g : \mathrm{GEWÄSSER})$$
$$(\ g\ \mathrm{IS}\ s\ \wedge\ \text{mündet-in}(\ f, g\ ) \wedge\ \mathrm{Schadstoffbel}(\ g\ ) > 50)\ \}\vdash$$

Abb. 2.5: Beispielanfrage 2

Die schreibenden Zugriffe auf eine Datenbank, d.h. Aktionen, werden zusammengesetzt aus **elementaren Aktionen**, die automatisch aus der in der Objekt- und Datenschicht erstellten Beschreibung der Struktur eines Datenbankzustands abgeleitet werden. Dadurch ist sichergestellt, daß diese elementaren Aktionen alle durch das Schema festgelegten strukturellen Einschränkungen, also bereits alle modellinhärenten Integritätsbedingungen, berücksichtigen. So ist zum Beispiel beim Einfügen eines Objekts vom Typ FLUSS zu berücksichtigen, daß dieser Objekttyp durch Partition aus dem Objekttyp GEWÄSSER entstanden ist. Das bedeutet, daß gleichzeitig auch ein Objekt vom Typ GEWÄSSER mit allen Attributen einzufügen ist. Weiterhin muß jedes Objekt vom Typ FLUSS an der Beziehung "mündet-in" teilnehmen, die das neue Objekt vom Typ FLUSS mit einem Objekt vom Typ GEWÄSSER verbindet, dessen Existenz bei der Ausführung dieser Einfügeoperation vorausgesetzt wird. Die gesamte elementare Aktion zum Einfügen eines neuen Objekt vom Typ FLUSS enthält demnach eine Reihe von Parametern, um nach Ausführung wieder einen Datenbankzustand zu erhalten, der alle modellinhärenten Integritätsbedingungen erfüllt.

```
insert-FLUSS ( fName : string; fLänge : real;                          (i)
               gSchiffbar : bool; gGeometrie : lines; gSchadstoffbel : int;   (ii)
               mMündung : point; mgName : string )                     (iii)
```

Abb. 2.6: Beispiel für eine elementare Aktion

Hierbei werden durch die Parameter in Zeile (i) die Attribute des neuen Objekts vom Typ FLUSS und durch die Parameter in Zeile (ii) die Attribute des zugehörigen Objekts vom Typ GEWÄSSER festgelegt. Mit Hilfe der Parameter in Zeile (iii) wird das neue Objekt vom Typ FLUSS in die Relation mündet-in eingetragen, wobei der Parameter mgName das entsprechende Objekt vom Typ GEWÄSSER bezeichnet, in das das neue Objekt vom Typ FLUSS mündet.

Elementare Aktionen sind die Bausteine für (komplexe) **Aktionen**, durch deren Ausführung konsistente Datenbankzustände ineinander überführt werden sollen, d.h. Datenbankzustände, die sowohl die statischen als auch dynamischen Integritätsbedingungen berücksichtigen. Im Gegensatz zu den oben erläuterten elementaren Aktionen ist die Konsistenzeigenschaft bei der Ausführung einer Aktion noch nicht automatisch gewährleistet. Nach Beendigung der Ausführung einer Aktion muß überprüft werden, ob eine der statischen oder dynamischen Integritätsbedingungen verletzt wurde.

3. Entwurfswerkzeuge

Auf der Grundlage des formalen Spezifikationsansatzes aus Kapitel 2 lassen sich (automatische) Entwurfshilfen konstruieren, die dazu dienen, den Entwurfsvorgang effizient zu gestalten und Entwurfsfehler zu vermeiden oder frühzeitig zu entdecken. Zur Unterstützung des Entwurfs eines konzeptionellen Datenbankschemas unterscheiden wir drei Werkzeuggruppen in der DBEU:

1. **Erstellung** eines Datenbankschemas im Sinne der oben erläuterten vierschichtigen Spezifikation,
2. **Analyse** der erstellten Spezifikation und
3. Durchführung eines **Prototyping**.

Abb. 3.1 gibt eine Übersicht über das Zusammenspiel der Werkzeuggruppen in Form eines Datenflußdiagramms:

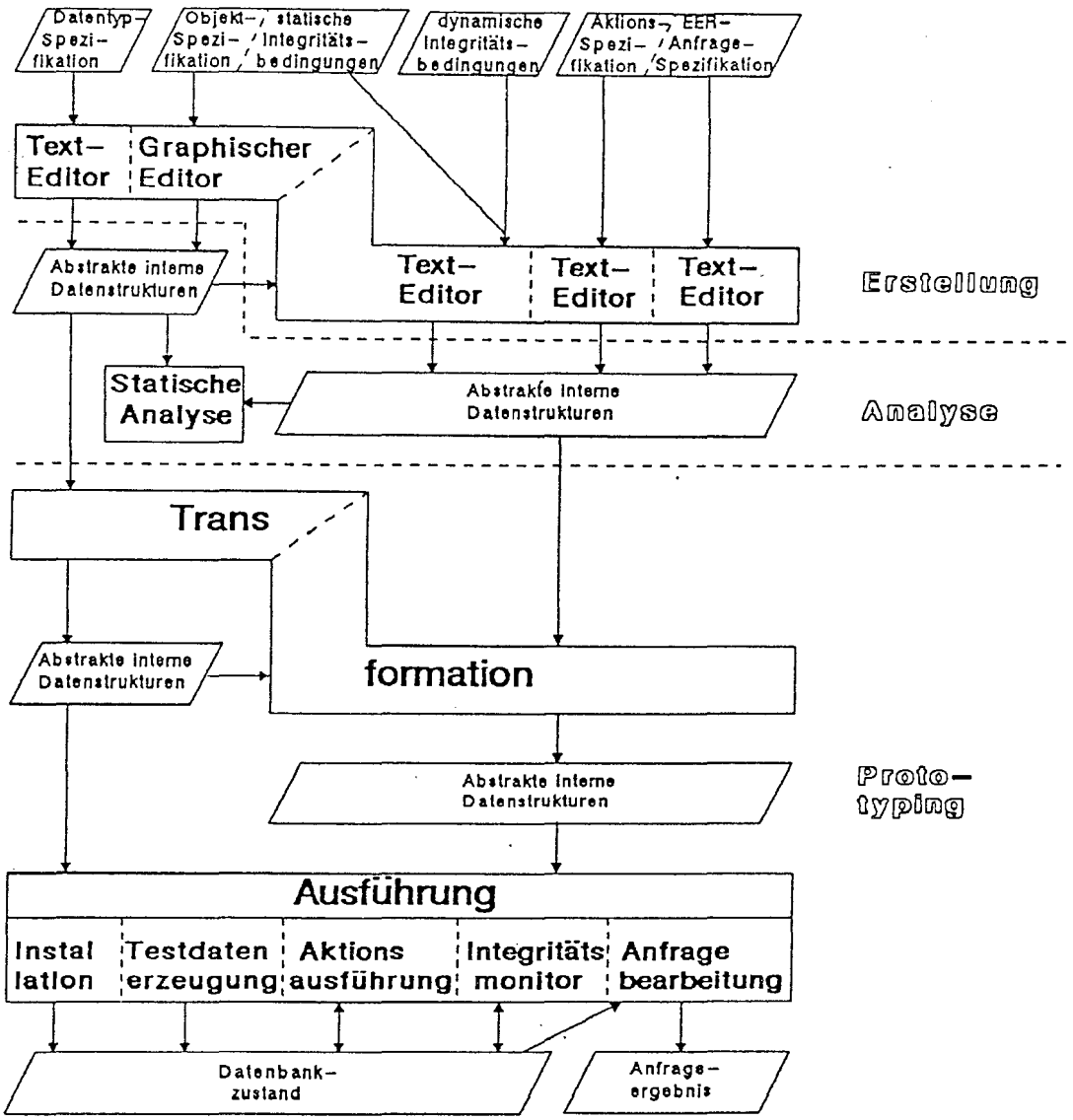


Abb. 3.1: Datenfluß in der DBEU

Der Datenfluß folgt im wesentlichen der Reihenfolge der drei Gruppen:

1. Erstellung

Zur Eingabe, Änderung und Dokumentation eines konzeptionellen Datenbankschemas steht eine **Familie syntaxgestützter Editoren** zur Verfügung:

- auf der Datenschicht ein syntaxgestützter Editor zur textuellen Eingabe und Änderung von algebraischen Spezifikationen zur Beschreibung von benutzerdefinierten Datentypen,
- auf der Objektschicht ein strukturorientierter, graphischer Editor zur Eingabe und Änderung erweiterter Entity-Relationship-Diagramme,
- auf der Entwicklungschicht ein syntaxgestützter Editor zur textuellen Eingabe und Änderung von (statischen und dynamischen) Integritätsbedingungen, und
- auf der Aktionsschicht jeweils ein syntaxgestützter Editor zur textuellen Eingabe und Änderung der Beschreibung von erlaubten Aktionen, insbesondere Änderungsoperationen, und zur Formulierung von Anfragen auf den im Schema definierten Objekten und Beziehungen.

Wir verlassen hier die strikte Trennung zwischen der Objekt- und der Entwicklungsschicht, indem wir nur eine kombinierte Spezifikationssprache für statische und dynamische Integritätsbedingungen und folglich auch nur einen syntaxgestützten Editor zur Spezifikation von Integritätsbedingungen verwenden.
Jeder der obigen Editoren verwaltet die bearbeiteten Dokumente in (abstrakten) internen Datenstrukturen, die zur weiteren (evtl. wiederholten) Verarbeitung der Spezifikation in einer *Projekt-Datenbank* abgelegt werden. Die zur Datentyp- und Objektspezifikation korrespondierenden Datenstrukturen fließen aus Gründen der inneren Konsistenz als Eingabe in die Editoren der Integritätsbedingungen, Aktionen und Anfragen ein, da diese nur bzgl. einer bereits spezifizierten Schemastruktur (Daten- und Objektschicht) formuliert werden können.

2. Analyse

Die oben genannten Editoren ermöglichen einen inkrementellen Schemaentwurf über alle vier Schichten der Spezifikation, indem sie verschiedene Konsistenzprüfungen durchführen und dadurch weitgehende kontextfreie und kontextsensitive Korrektheit des gesamten Schemas garantieren. Weitergehende Zusammenhänge innerhalb des Schemas können durch das Werkzeug der **statischen Analyse** vom Entwickler ermittelt werden. Hierzu gehört z.B. eine Analyse der Datentypen-Spezifikationen durch ein Termersetzungssystem. Ein Schwerpunkt der statischen Analyse der konzeptionellen Schemata ist die Untersuchung der Erfüllbarkeit der spezifizierten Integritätsbedingungen. Modellinhärente und durch den Entwerfer definierte explizite statische Integritätsbedingungen werden mit Methoden der logischen Programmierung auf Erfüllbarkeit untersucht.
Dynamische Integritätsbedingungen, d.h. Formeln einer temporal erweiterten Prädikatenlogik, werden mit den in /LS 87/ und /Sa 87/ vorgestellten Methoden untersucht. Kern dieser Analyse ist die Konstruktion von Transitionsgraphen aus temporalen Formeln, die zusätzlich zur Konsistenzprüfung eine graphische Darstellung der spezifizierten Objektlebensläufe ermöglichen.

3. Prototyping

Für einen Datenbankentwerfer ist es hilfreich und informativ, bereits in einer sehr frühen Phas spezifizierte Aktionen der entworfenen Datenbank im Sinne eines "rapid prototyping" ausführen z lassen. Hierzu werden entsprechende **Prototyping-Werkzeuge** zur Verfügung gestellt, die sich gro in zwei Klassen unterteilen lassen,

- in Werkzeuge zur Transformation des konzeptionellen Datenbankschemas in ein logisches Schem des Relationenmodells und
- Werkzeuge zur Ausführung von Aktionen und Anfragen auf einem dazu erzeugten relationale Datenbestand.

Die **Transformation** erfolgt zweistufig:

- Im ersten Schritt wird die Struktur des entworfenen Datenbankschemas, gegeben durch Daten- und Objektschicht, in ein relationales Datenbankschema übersetzt, d.h. bei der Abbildung der Objekt-, Beziehungs- und Typkonstruktionsschemata auf Relationenschemata muß die Abbildung der Datentypen auf Standarddatentypen evtl. unter Zuhilfenahme weiterer Relationenschemata erfolgen.
- Der zweite Schritt beinhaltet dann die Übersetzung des Verhaltens. So werden die aus Elementaraktionen zusammengesetzten Aktionen wie auch die Anfragesprache in Programme mit eingebetteten relationalen Operationen bzw. in eine relationalen Anfragesprache wie z.B. SQL übersetzt. Das kann nur in Abhängigkeit vom Ergebnis des ersten Schrittes erfolgen, da sowohl die Spezifikation im EER-Modell wie auch die korrespondierenden Relationenschemata zur Übersetzung herangezogen werden müssen. Statische Integritätsbedingungen werden ebenfalls in relationale Anfragen (die einen Wahrheitswert liefern) abgebildet. Die dynamischen Integritätsbedingungen werden dagegen gesondert behandelt: Aus ihnen werden "gewisse" Anfragen extrahiert (siehe unten) und mit den spezifizierten Anfragen zusammen übersetzt.

Dieser Transformationsschritt wurde bisher für die Umsetzung in ein relationales Datenbankschema untersucht (/HNS 86/). Die Transformation geschieht größtenteils automatisch, wobei dem Entwerfer zusätzlich die Möglichkeit der Einflußnahme gegeben ist. Derzeit wird untersucht, inwieweit andere Datenbanksysteme mit komplexeren Datenmodellen (z.B. NF^2-Datenmodell (/SS 86/)) für die Prototyp-Generierung geeigneter sind.

Die **Ausführung** des Datenbankschemas kann anschließend mittels der folgenden Werkzeuge erfolgen:

- Bei der **Installation** der *Prototyp-Datenbank* werden die aus der Strukturtransformation resultierenden Relationenschemata auf einem konkreten, existierenden (relationalen) Datenbanksystem eingerichtet.
- Die **Testdatenerzeugung** (/NN 85/) füllt die entsprechenden Relationen mit künstlichen Testdaten, wobei die inhärenten Eigenschaften des zugrundeliegenden EER-Entwurfs berücksichtigt werden.
- Die mit Testdaten gefüllte *Prototyp-Datenbank* kann nun dazu benutzt werden, die aus der Transformation entstandenen (relationalen) **Aktionen** und **Anfragen auszuführen**. Bei der Ausführung von Anfragen ist insbesondere darauf zu achten, daß das Ergebnis (gemäß dem Relationenmodell) in das EER-Modell zurücktransformiert wird. Insbesondere müssen Ergebnisse zu Nicht-Stan-

darddatentypen wie POINT, die im Relationenmodell "versteckt" sind, entsprechend interpretiert und zur Ausgabe aufbereitet werden.

- Während der Ausführung von Aktionen auf einer Testdatenbank steht ein dynamisches Analysewerkzeug, der **Integritätsmonitor** (/Hü 88/, /LS 87/), zur Verfügung. Dieser konstruiert vor Beginn der Ausführung für jede der zu überwachenden dynamischen Integritätsbedingungen ein optimiertes Überprüfungsschema bestehend aus sogenannten Transitionsgraphen. Die Knoten jedes dieser Graphen repräsentieren die zur Überwachung einer dynamischen Integritätsbedingung relevanten Datenbankzustände. Die Kantenbeschriftungen sind Anfragen an die Datenbank, in deren Abhängigkeit die Zustände fortgeschaltet werden. (Zur Ermittlung der Ergebnisse der Anfragen werden diese bei der Transformation aus den Transitionsgraphen extrahiert und in relationale Anfragen übersetzt.) Zur Ausführungszeit wird die Einhaltung der statischen und dynamischen Integrität nach jeder Aktion anhand dieser Überprüfungsschemata kontrolliert und die zusätzlich in der Datenbank abgelegte historische Information über die betroffenen Objekte aktualisiert. Der Integritätsmonitor erkennt alle Aktionsausführungen, die die Integritätsbedingungen verletzen.

Der hier erläuterte streng sequentielle Datenfluß über die drei Werkzeuggruppen entspricht in der Regel nicht dem Vorgehen eines Entwicklers. Aus diesem Grunde soll durch die Werkzeuge insbesondere auch eine schrittweise, zyklische und stark verzahnte Arbeitsweise unterstützt werden. Dies bedeutet zum Beispiel, daß parallel zu einer schrittweisen Entwicklung eines konzeptionellen Datenbankschemas auch eine schrittweise Erweiterung der mit Testdaten gefüllten Prototyping-Datenbank unterstützt werden soll. Dadurch wird es für den Entwickler möglich, sehr frühzeitig durch die Ausführung von Aktionen und Anfragen auf der Prototyping-Datenbank das bis dahin entwickelte Schema zu untersuchen.

Abb. 3.2 zeigt die drei Werkzeuggruppen noch einmal aus der Sicht der DBEU-Architektur. Die einzelnen Kästchen stellen hier Teilsysteme bzw. Moduln dar, die sich gemäß der eingezeichneten Kanten gegenseitig benutzen.
Die gesamte Architektur ruht auf den drei Basiskomponenten *Ein-/Ausgabesystem*, *Projekt-Datenbank* und *Prototyping-Datenbanksystem.* Das *Ein-/Ausgabesystem* dient als einheitliche Schnittstelle zum Benutzer bei der Erstellung des Datenbankschemas, für Ausgaben der Analysewerkzeuge und während des Prototypings. Wesentlicher Bestandteil dieses Systems ist ein Fenstersystem, das aus Portabilitätsgründen auf der Basis von X-Windows realisiert wird. In der *Projekt-Datenbank* werden alle von den Werkzeugen verwalteten Informationen wie z.B. die erstellten Datenbankschemata und die Ergebnisse der Transformation abgespeichert. Alle diese Dokumente werden in der Form abstrakter Syntaxgraphen abgelegt (vgl. /En 86/). In der aktuellen Realisierung ist die *Projekt-Datenbank* als Hauptspeicherdatenstruktur realisiert, die auf der Implementierung eines Graphenlabors (/Eb 87/) basiert. Das *Prototyping-Datenbanksystem* verwaltet die mit Testdaten gefüllte Prototyp-Datenbank und ermöglicht die Ausführung von Aktionen und Anfragen.

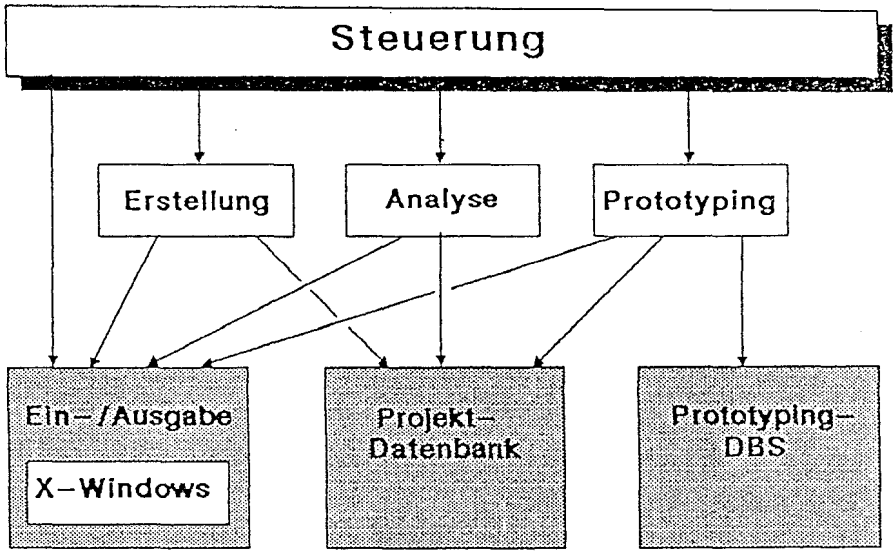


Abb. 3.2: Architektur der DBEU

Im Rahmen von Diplom- und Promotionsarbeiten werden diese Werkzeuge zur Entwicklung eines konzeptionellen Schemas innerhalb einer Prototypimplementierung realisiert, wobei Erfahrungen aus der Implementierung einer allgemeinen Software-Entwicklungsumgebung einfließen (/ENS 87/). Die Implementierung erfolgt in der Programmiersprache C auf 68020-Workstations unter dem Betriebssystem UNIX.

Danksagung

Für viele fruchtbare Diskussionen über die hier vorgestellten Ergebnisse danken wir Frau L. Neugebauer. Für Bemerkungen und Anregungen zu diesem Papier danken wir unserer Kollegin Frau P. Löhr und den Kollegen P. Herr und K. Hülsmann. Prof. J. Ebert (EWH Koblenz) danken wir für die Bereitstellung des "EMS-Graphenlabors".

Literatur

/Ba 82/ Balzert, H.: *Die Entwicklung von Software-Systemen: Prinzipien, Methoden, Sprachen, Werkzeuge.* Mannheim: Bibliographisches Institut 1982

/BNTT 85/ Batini, C./ Nardelli, E./ Talamo, M./ Tamassia, R.: *GINCOD: A Graphical Tool for Conceptual Design of Data Base Applications.* In Albano, A./ DeAntonellis, V./ DiLeva, A. (eds.): Computer-Aided Database Design: The DATAID Project. North-Holland, Amsterdam 1985

/Ch 76/ Chen, P.P.: *The Entity-Relationship Model: Towards a Unified View of Data.* ACM Transactions onDatabase Systems, Vol. 1, No. 1, 1976

/Da 86/ Dadam, P.P. et al: *A DBMS Prototype to Support Extended NF^2-Relations.* ACM Proc. SIGMOD, 1986

/Eb 87/ Ebert, J.: *A Versatile Data Structure for Edge-Oriented Graph Algorithms.* Communication of the ACM, Vol. 30, No. 6, 1987, 513-519

/EDG 86/ Ehrich, H.-D./ Drosten, K./ Gogolla, M.: *Towards an Algebraic Semantics for Database Specification.* Proc. IFIP WG 2.6 Working Conf. "Knowledge and Data" (DS-2), R.Meersman, A.Sernadas (eds.), North-Holland (to appear)

/En 86/ Engels, G.: *Graphen als zentrale Datenstrukturen in einer Software-Entwicklungsumgebung.* VDI-Fortschritt-Berichte Nr. 62, Düsseldorf: VDI-Verlag 1986

/ENS 87/ Engels, G./ Nagl, M./ Schäfer, W.: *On the Structure of Structure-Oriented Editors for Different Applications.* In Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Palo Alto, Dec. 1986, ACM SIGPLAN Notices, Vol. 22 No. 1, January 1987, 190-198

/EWH 85/ Elmasri, R./ Weeldreyer, J./ Hevner, A.: *The Category Concept: An Extension to the Entity-Relationship Model.* Data & Knowledge Engineering, Vol. 1, 1985, 75-116

/HG 88/ Hohenstein, U./ Gogolla, M.: *Towards a Semantic View of an Extended Entity-Relationship Model.* Informatik-Bericht Nr. 88-02, TU Braunschweig, 1988

/HNS 86/ Hohenstein, U./ Neugebauer, L./ Saake, G.: *An Extended Entity-Relationship Model for Non-Standard Databases.* Proc. Workshop "Relationale Datenbanken" in Lessach, Bericht Nr. 3-86, Inst. für Informatik, TU Clausthal-Zellerfeld 1986

/HNSE 87/ Hohenstein, U./ Neugebauer, L./ Saake, G./ Ehrich, H.-D.: *Three-Level Specification of Databases Using an Extended Entity-Relationship Model.* Proc. GI-Fachtagung "Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen", R.R.Wagner, R.Traunmüller, H.C.Mayr (eds.), Informatik-Fachbericht Bd. 143, Springer-Verlag, Berlin 1987, 58-88

/Hü 88/ Hülsmann, K.: *Entwurf eines Systems zur Überwachung dynamischer Integritätsbedingungen.* Diplomarbeit, Informatik, TU Braunschweig, Januar 1988

/LK 84/ Lorie, R.A./ Kim, W.: *Supporting Complex Objects in a Relational System for Engineering Databases.* IBM Research Report, San Jose 1984

/LM 86/ Lockemann, P.C./ Mayr, H.C.: *Information System Design: Techniques and Software Support.* In H.-J.Kugler (ed.): Information Processing 1986, Elsevier Science Publ. 1986

/LN 87/ Lipeck, U.W./ Neumann, K.: *Modelling and Manipulating Objects in Geoscientific Databases.* In S. Spaccapietra (ed.): Proc. 5th Int. Conf. on the Entity-Relationship Approach. North-Holland, Amsterdam 1987, 67-86

/LS 87/ Lipeck, U.W./ Saake, G.: *Monitoring Dynamic Integrity Constraints Based on Temporal Logic.* Information Systems, Vol. 12 (1987), 255-269

/MDL 87/ Mayr, H.C./ Dittrich, K.R./ Lockemann, P.C.: *Datenbankentwurf.*
In Lockemann, P.C./ Schmidt, J.W. (eds.): Datenbank-Handbuch. Springer-Verlag, Berlin 1987, 489-562

/NN 85/ Neugebauer, L./ Neumann, K.: *Schemagesteuerte Testdatengenerierung für relationale Datenbanken.* Informatik-Bericht Nr. 85-02, TU Braunschweig 1985

/OSLS 86/ Oberweis, A./ Schönthaler, F./ Lausen, G./ Stucky, W.: *Net based conceptual modelling and rapid prototyping with INCOME.*
In Proc. of the 3rd Conference on Software Engineering, AFCET, Paris 1986, 165-176

/RNLE 85/ Ramm, I./Neumann, K./Lipeck, U.W./Ehrich, H.-D.: *Eine Benutzerschnittstelle für geowissenschaftliche Datenbanken.* Informatik-Bericht Nr. 85-08, TU Braunschweig, 1985

/Sa 88/ Saake, G.: *Spezifikation, Semantik und Überwachung von Objektlebensläufen in Datenbanken.* Dissertation, Informatik, TU Braunschweig

/SNHE 87/ Saake, G./ Neugebauer, L./ Hohenstein, U./ Ehrich, H.-D.: *Konzepte und Werkzeuge für eine Datenbank-Entwurfsumgebung.*
Informatik-Bericht Nr. 87-05, TU Braunschweig 1987

/SS 77/ Smith, J.M./ Smith, D.C.P.: *Database Abstractions: Aggregation and Generalization.* ACM Transactions on Database Systems Vol. 2, 1977, 105-133

/SS 86/ Schek, H.-J./ Scholl, M.H.: *The Relational Model with Relation-Valued Attributes.* Information Systems, Vol. 11, No. 2, 1986