CADDY-O: Syntaxgestütztes, graphisches Entwerfen konzeptioneller Datenbankschemata

Gregor Engels
TU Braunschweig, Informatik
Postfach 3329, D-3300 Braunschweig

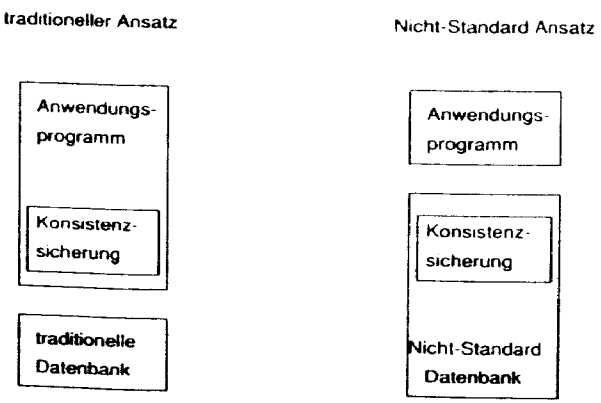
Kurzfassung

In diesem Beitrag werden die komfortable, interaktive Benutzerschnittstelle und die Realisierung des syntaxgestützten graphischen Editors CADDY-O zur Bearbeitung erweiterter Entitiy-Relationship-Diagramme beschrieben. Dieser Editor ist Bestandteil der integrierten Datenbankentwurfsumgebung CADDY (Computer-aided Conceptual Design of Non-standard Databases), die Werkzeuge zum Edieren und Testen ("rapid prototyping") konzeptioneller Datenbankschemata enthält.
CADDY wird auf 68.0x0-Workstations implementiert. Ein erster Prototyp, der insbesondere den hier beschriebenen Editor CADDY-O enthält, ist fertiggestellt und vorführbar.

1. Einführung

Aufgrund der ständig fallenden Hardware- und Softwarepreise wird der Einsatz leistungsfähiger Arbeitsplatzrechner in immer mehr Arbeitsbereichen ermöglicht. Wesentliche Charakteristika derartiger Arbeitsplatzrechner sind die Verfügbarkeit komfortabler Ein-/Ausgabemedien wie hochauflösender Bildschirm, Maus, graphisches Tablett u.ä.. Diese ermöglichen die Realisierung komfortabler, leicht zu erlernender und zu bedienender interaktiver Benutzerschnittstellen und erhöhen dadurch die Akzeptanz des Einsatzes von Rechnern in den jeweiligen Arbeitsbereichen. Für den Bereich der Informationssysteme gilt dies sowohl für die Gestaltung der **Endbenutzerschnittstelle** eines Informationssystems als auch für die Gestaltung der Benutzerschnittstelle von Softwarewerkzeugen, die einen **Software-Entwickler** beim Entwerfen und Realisieren eines Informationssystems unterstützen.

Es ist das Ziel des Projekts **CADDY** (Computer-Aided Conceptual Design of Non-standard Databases) einen derartigen integrierten Werkzeugkasten für den konzeptionellen Entwurf von Informationssystemen, d.h. insbesondere der in ihnen enthaltenen Datenbanken zu konzipieren und zu realisieren (/EHHLE 89/). Hierbei soll insbesondere der Entwurf von Nicht-Standard-Datenbanken unterstützt werden. Derartige **Nicht-Standard-Datenbanken** unterscheiden sich von traditionellen Datenbanken dadurch, daß die für einen speziellen Anwendungsbereich adäquate Modellierung und Verwaltung von Datenbankobjekten nicht im darüberliegenden Anwendungsprogramm, sondern in der Datenbank und in dem zugehörigen Datenbankverwaltungssystem verkapselt sind. Figur 1 veranschaulicht diesen Unterschied. Bei der Benutzung eines traditionellen, z.B. relationalen Datenbanksystems, ist die Modellierung einer komplexen Datenbankobjektstruktur und die damit verbundene Konsistenzsicherung wesentlicher Bestandteil des Anwendungsprogramms. Beim Einsatz von Nicht-Standard-Datenbanken wird die Modellierung der komplexen Objektstruktur vom eigentlichen Anwendungsprogramm getrennt. Diese Vorgehensweise ermöglicht, die Struktur der Datenbank (und auch die zugehörigen Datenbankoperationen) in einer Vorphase zu entwerfen und zu testen. Für die Konsistenz der komplex strukturierten Datenbankobjekte ist dann während der Ausführung des Anwendungsprogramms das zugehörige Verwaltungssystem der Nicht-Standard- Datenbank zuständig.



Figur 1: Vergleich zwischen traditionellem und Nicht-Standard-Ansatz

Grundlage für den Entwurf einer Nicht-Standard-Datenbank ist ein **Datenmodell**, durch das die Beschreibungsmöglichkeiten für die Struktur und das Verhalten einer Datenbank festgelegt sind. Ein derartiges semantisch wohldefiniertes Datenmodell wurde in den letzten Jahren an der TU Braunschweig entwickelt (/HNSE 87/). Es soll an dieser Stelle nur kurz skizziert werden. Ein zugehöriges **Datenbankschema** $S = (O, D, E, A)$ besteht

dabei aus den folgenden 4 Komponenten:

O : Zur Modellierung der Struktur eines Datenbankzustands, d.h. der möglichen **Datenbankobjekte** und ihrer Beziehungen untereinander wird auf ein erweitertes Entity-Relationship-Modell (EER-Modell) zurückgegriffen.

D : Einzelne Datenbankobjekte besitzen Attribute, deren Werte aus einem vordefinierten oder aus einem selbstdefinierten Attributwertebereich stammen können. Derartige, insbesondere für Nicht-Standard-Anwendungen benötigte selbstdefinierte **Datentypen** für Attributwertebereiche werden in der zweiten Komponente in der Form algebraischer Spezifikationen spezifiziert.

E : Datenbankobjekte werden kreiert, verändert und wieder gelöscht. Dadurch entstehen Folgen von Datenbankzuständen. Durch die Angabe dynamischer Integritätsbedingungen durch Ausdrücke einer temporalen Logik wird auf deskriptive Art die Menge aller möglichen Datenbankzustandsfolgen auf die zulässigen **Entwicklungen** von Datenbankzuständen eingeschränkt.

A : Auch die Festlegung der auf einer Datenbank erlaubten **Aktionen** gehört zu einem Datenbankschema. Diese Aktionen werden in prozeduraler Weise angegeben. Sie setzen sich zusammen aus Kontrollstrukturen, Anfrageaktionen und elementaren Aktionen. Letztere werden zu der in der 1. Komponente festgelegten Objektstruktur automatisch erzeugt. Sie gewährleisten bei Ausführung die Einhaltung aller modellinhärenten Integritätsbedingungen. Anfrageaktionen werden in einer auf einem EER-Kalkül basierenden Anfragesprache formuliert (/HG 88/).

Ziel der CADDY-Umgebung ist nun, den Entwurf und den frühzeitigen Test derartiger Datenbankschemata durch geeignete Werkzeuge zu unterstützen. Wesentliche Ziele hierbei sind, daß alle Werkzeuge miteinander **integriert** sind, so daß der Entwerfer ohne Aufwand zwischen den einzelnen Werkzeugen wechseln kann und so **inkrementell**, d.h. schrittweise ein Datenbankschema entwickeln kann.

Die Werkzeuge von CADDY können in zwei Gruppen aufgeteilt werden: Für den Entwurf, d.h. die Eingabe und die Veränderung eines Datenbankschemas stehen für jede Komponente eines Schemas ein **syntaxgestützter Editor** zur Verfügung, der je nach Dokumentart ein Graphik- oder Texteditor ist. Für den Test des spezifizierten Schemas, auch als "rapid prototyping" oder "schema animation" bezeichnet, stehen **Interpreter** für die Ausführung von Anfrage- oder Änderungsaktionen zur Verfügung. Ein objektbezogenes Navigieren durch die Testdatenbank wird durch einen **Datenbankbrowser** ermöglicht.

Einige dieser Werkzeuge sind bereits realisiert, andere befinden sich zur Zeit in der Entwicklung (vgl. Kapitel 4). Im Rahmen dieses Papiers wollen wir uns auf den bereits implementierten syntaxgestützten, graphischen Editor **CADDY-O** für die Bearbeitung von

EER-Diagrammen konzentrieren. An ihm sollen im Kapitel 2 einige Charakteristika der komfortablen Benutzerschnittstelle von CADDY erläutert werden und im Kapitel 3 und 4 gezeigt werden, wie eine derartige interaktive Benutzerschnittstelle auf einem Arbeitsplatzrechner realisiert wurde.

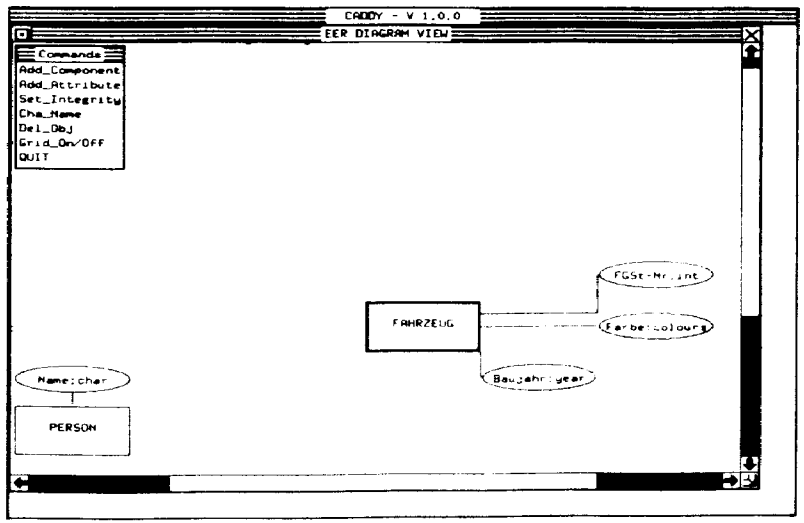
Die Entwicklung von Prototyping-Werkzeugen ist seit mehreren Jahren ein aktuelles Thema in der Forschung, deren erste Ergebnisse mittlerweile auch in industrielle Produkte eingeflossen sind (z.B. *Software through Pictures* /WP 87/, *ISAC* /Lu 82/). /Sch 89/ gibt einen Überblick über aktuelle Ansätze. Diese unterscheiden sich von den Zielen von CADDY z.T. dadurch, daß sie andere Konzepte zur Modellierung eines konzeptionellen Schemas zugrundelegen (/Sch 89/, /OSLS 86/) oder bei der Realisierung von Werkzeugen weniger Gewicht auf deren Integration in eine geschlossene Umgebung legen (z.B. /WP 87/).

2. Benutzerschnittstelle

Bevor wir die Benutzerschnittstelle des graphischen Editors CADDY-O zur Bearbeitung eines EER-Diagramms detaillierter beschreiben, sollen einige Charakteristika und Begriffe erläutert werden, die für CADDY-O und auch alle anderen Werkzeuge von CADDY relevant sind.

Ein wesentliches Merkmal der komfortablen Benutzerschnittstelle von CADDY ist, den Datenbankentwerfer so weit wie möglich durch das System zu unterstützen, ohne ihn in seiner Arbeit zu behindern. Eine wesentliche Grundlage hierfür ist der Einsatz eines **Fenstersystems**, das es ermöglicht, sämtliche für den Datenbankentwerfer zum aktuellen Zeitpunkt relevanten Informationen in mehreren, u.U. überlappenden Fenstern anzuzeigen. Der Entwerfer hat die Möglichkeit, das Fensterlayout auf dem Bildschirm z.B. durch Vergrößern, Verkleinern, Verschieben von Fenstern seinen persönlichen Vorstellungen anzupassen. Jedem Werkzeug ist in der Regel ein **Dokumentenfenster** zugeordnet, in dem ein Ausschnitt des aktuell bearbeiteten Dokuments angezeigt wird. Dies ist z.B. bei CADDY-O ein Ausschnitt aus einem EER-Diagramm. In jedem Dokument existiert eine aktuelle Bearbeitungseinheit, **aktuelles Inkrement** genannt, auf das sich alle Aktivitäten eines Entwerfers beziehen. Derartige Aktivitäten werden vom Entwerfer durch Eingabe von Kommandos angestoßen. Dies geschieht in der Regel durch Auswahl in einem Kommandomenü, in dem alle zum aktuellen Inkrement erlaubten Kommandos aufgelistet sind. Diese **kommandogesteuerte, syntaxgestützte** Vorgehensweise garantiert, daß das bearbeitete Dokument stets syntaktisch korrekt ist. Hierbei wird bei CADDY-O neben der **kontextfreien** auch die **kontextsensitive** Korrektheit gewährleistet. Das heißt z.B., daß bei Eingabe

eines Bezeichners für eine Objektklasse sofort überprüft wird, ob dieser Bezeichner bereits vergeben wurde. Die Auswahl einer Alternative in einem Kommandomenü, das Bestätigen von Systemmeldungen in einem Nachrichtenfenster und insbesondere das Auswählen eines neuen aktuellen Inkrements geschieht durch **Anklicken** mit einer **Maus**. Letzteres ermöglicht dem Entwerfer, das aktuelle Inkrement im Dokument zu wechseln und so **inkrementell** das Dokument, also z.B. ein EER-Diagramm, weiterzuentwickeln. Dies wird auch durch die **Integration** aller Werkzeuge unterstützt. So ist ein beliebiger Wechsel zwischen den einzelnen Editoren zur Erstellung eines Datenbankschemas möglich, wobei dann vom System die Konsistenz der Abhängigkeiten zwischen den einzelnen Teildokumenten sichergestellt wird.

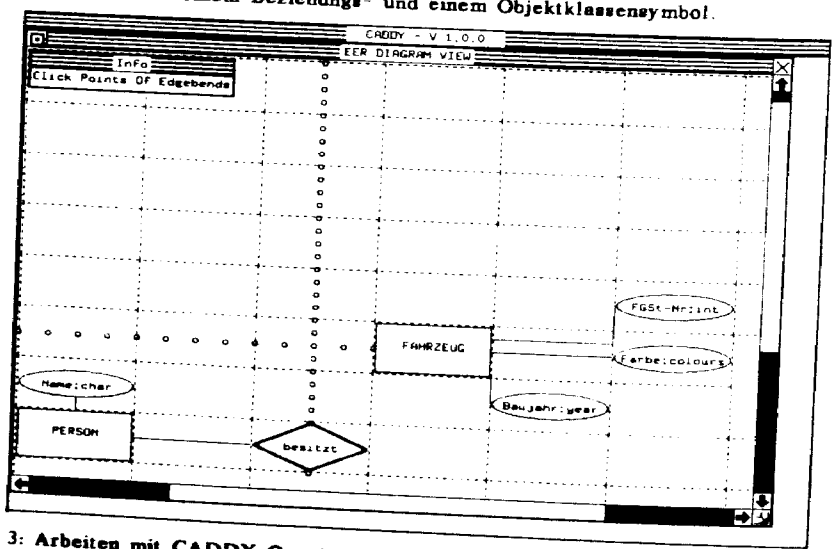


Figur 2: Arbeiten mit CADDY-O - Ausschnitt 1

Der graphische Editor CADDY-O unterstützt die Bearbeitung von Diagrammen eines erweiterten ER-Modells. Figur 2 zeigt einen Ausschnitt aus der Arbeit mit diesem Editor. Der Benutzer hat zwei Objektklassen mit zugehörigen Attributen definiert. Die in dem Kommandomenü angezeigten Alternativen beziehen sich auf das aktuelle Inkrement, die Objektklasse FAHRZEUG, und erlauben, den Namen zu ändern (*Cha-Name*), einfache Attribute hinzuzufügen (*Add-Attribute*), objektwertige Attribute hinzuzufügen (*Add-Component*), eine statische Integritätsbedingung anzugeben (*Set-Integrity*) oder die gesamte Objektklasse zu löschen (*Del-Obj*).

Für die Gestaltung des geometrischen Layouts eines EER-Diagramms ist der Entwerfer bei Benutzung von CADDY-O selbst verantwortlich. Hierbei wird er vom System auf

verschiedene Weisen unterstützt (vgl. hierzu /BNTT 85/). Zunächst einmal ist das gesamte Dokument mit einem Gitter unterlegt, das der Benutzer sichtbar bzw. unsichtbar machen kann (*Grid-On/Off*). Jedes Symbol eines EER-Diagramms liegt vollständig in einem Rasterfeld dieses Gitters. Bei Ergänzung des EER-Diagramms um ein neues Symbol klickt der Entwerfer die von ihm gewünschte Position auf dem Bildschirm an. Das System bestimmt dann automatisch das zugehörige Rasterfeld im Gitter und positioniert das entsprechende graphische Symbol. Durch diese Vorgehensweise wird ein gut strukturiertes, übersichtliches Layout eines EER-Diagramms ermöglicht. Auch bei der Eingabe von Verbindungskanten wird der Entwerfer vom System unterstützt: Hierzu werden vom System horizontale und vertikale Punktlinien angezeigt, mit denen der Entwerfer durch Anklicken der Knick- bzw. des Endpunktes eine Verbindungskante stückweise zusammensetzen kann. Figur 3 zeigt einen Ausschnitt aus einer derartigen Bearbeitungssituation beim Eintragen einer Verbindungskante zwischen einem Beziehungs- und einem Objektklassensymbol.



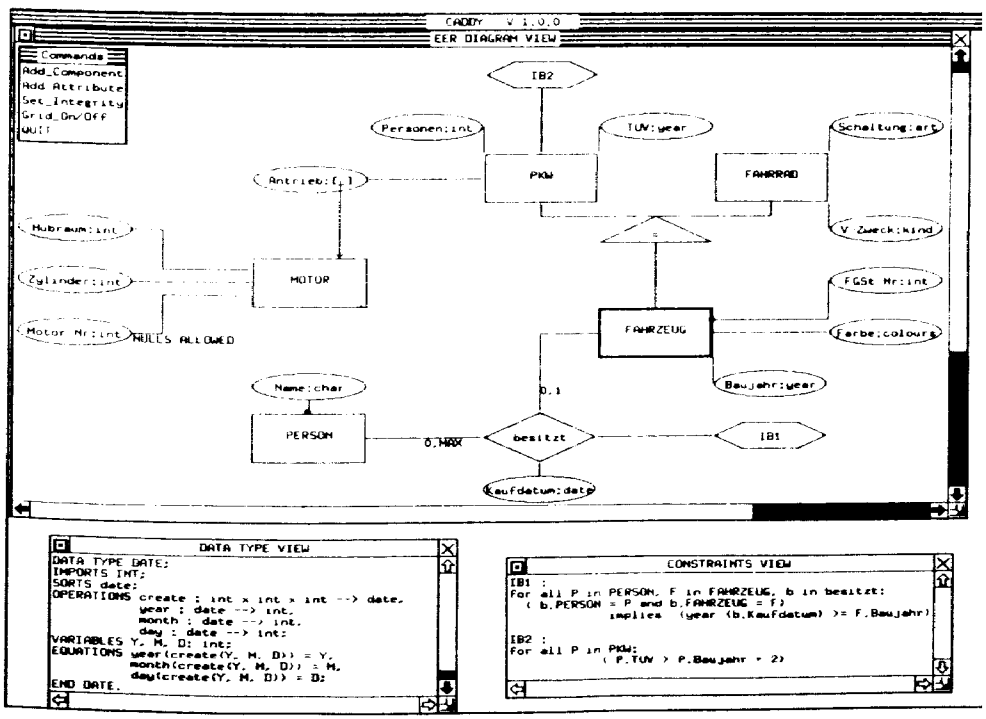
Figur 3: Arbeiten mit CADDY-O - Ausschnitt 2

CADDY-O liegt ein **erweitertes** ER-Modell zugrunde, das umfassend in /HNSE 87/ beschrieben ist. Wir wollen hier nur einige dieser Erweiterungen an Beispielen erläutern (vgl. hierzu Figur 4).

- **Typkonstrukteren** erlauben die Definition von Spezialisierungen bzw. Generalisierungen von Objektklassen. Im Beispiel ist die Objektklasse FAHRZEUG spezialisiert zu den beiden Objektklassen FAHRRAD bzw. PKW. Hierbei werden alle Attribute von FAHRZEUG auf die spezialisierten Objekte vererbt. Im Gegensatz hierzu können bei einer Generalisierung verschiedene Objektklassen zu einer Objektklasse zusammengefaßt

werden. Bei beiden Arten der Typkonstruktion werden keine neuen Objekte erzeugt; jedes Objekt einer konstruierten Objektklasse ist Objekt einer Eingangsklasse der Typkonstruktion.

- Geschachtelte Objekte, d.h. Objekte, die als Bestandteil andere Objekte enthalten, werden durch **objektwertige Attribute** (auch **Komponenten** genannt) modelliert. Im Beispiel gilt dies für die Objektklasse MOTOR. Jeder Motor ist in genau einem PKW enthalten.
- **Optionale Attribute** werden durch den Zusatz *Nulls allowed* an der Verbindungskante dargestellt (z.B. die Motor-Nr. eines Motors)
- **Schlüsselattribute** werden durch einen Punkt auf der Verbindungskante gekennzeichnet (z.B. Name bei PERSON).
- Beziehungsklassen können mit **Kardinalitäten** versehen werden. Im Beispiel bedeutet die Angabe (0,MAX), daß jede Person beliebig viele Fahrzeuge besitzen kann (evtl. auch keins). Andererseits wird durch (0,1) ausgedrückt, daß jedes Fahrzeug höchstens einen Besitzer hat.



Figur 4: Arbeiten mit CADDY - Ausschnitt 3

- Die Ausprägung des spezifizierten Schemas kann durch zusätzliche **statische Integritätsbedingungen** eingeschränkt werden. Diese werden im EER-Diagramm durch ein Sechseck dargestellt. Um die Übersichtlichkeit nicht zu verlieren, wird die eigentliche Integritätsbedingung in einem weiteren Fenster "CONSTRAINTS VIEW" dargestellt. Dort kann sie auch mit einem zugehörigen Texteditor CADDY-I bearbeitet werden.
- Das gleiche gilt für selbstdefinierte Datentypen, die in einem weiteren Dokumentenfenster "DATA TYPE VIEW" in Form algebraischer Spezifikationen mit Hilfe des syntaxgestützten Texteditors CADDY-D spezifiziert werden können (/Os 88/).

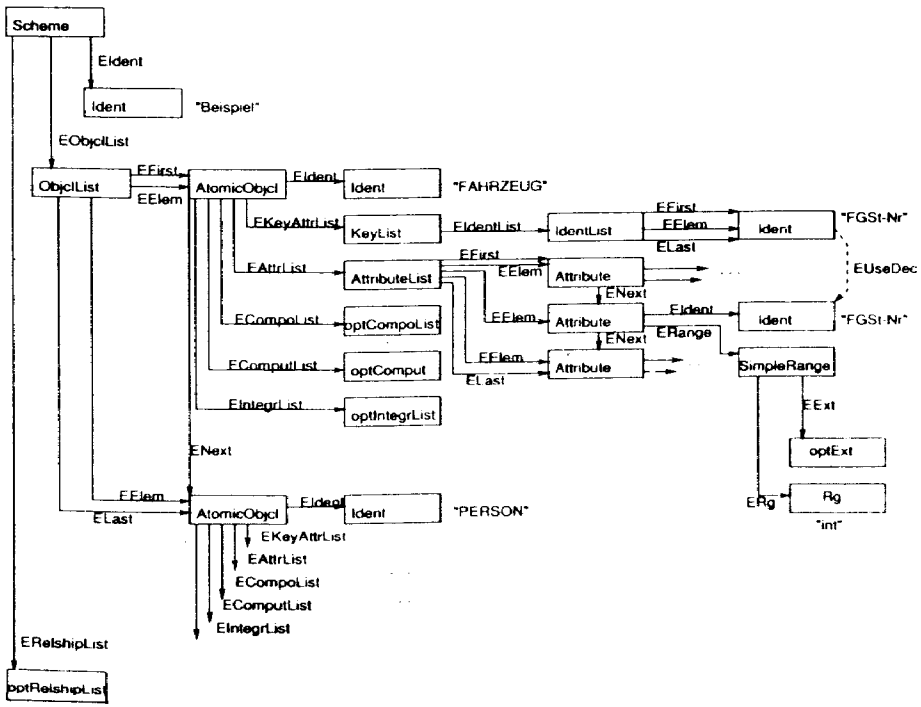
Die auf dem Bildschirm in verschiedenen Fenstern arbeitenden Editoren sind intern miteinander integriert. Das heißt, daß z.B. bei der Eingabe oder Veränderung einer Integritätsbedingung durch CADDY-I automatisch überprüft wird, ob alle benutzten Attributnamen bereits im EER-Schema definiert wurden.

3. Entwurfsüberlegungen

Um die oben skizzierte Funktionalität des Editors CADDY-O und die damit verbundene Integration aller Editoren realisieren zu können, werden intern Datenstrukturen zur Darstellung der bearbeiteten Dokumente (z.B. EER-Diagramm, Integritätsbedingungen usw.) benötigt, in der die kontextfreie und auch kontextsensitive Struktur eines Dokuments unmittelbar ausgedrückt wird. Nur so ist es effizient realisierbar, in der aktuellen Situation in dem Kommandomenü nur die Kommandos anzuzeigen, die für das aktuelle Inkrement erlaubt sind, d.h. bei Ausführung die kontextfreie Korrektheit des Dokuments nicht verletzen. Es ist aus diesem Grunde mittlerweile Standard bei der Realisierung syntaxgestützter Editoren, **abstrakte Syntaxbäume** als interne Datenstruktur zur Verwaltung der bearbeiteten Dokumente einzusetzen (/He 89/). Während in diesen Syntaxbäumen nur die kontextfreie Struktur eines Dokuments ausgedrückt wird, gehen wir hier noch einen Schritt weiter und benutzen **abstrakte Syntaxgraphen**, in denen zusätzlich durch entsprechende Kanten auch kontextsensitive Zusammenhänge unmittelbar dargestellt werden (/En 86/, /ENS 87/). Figur 5 zeigt einen Ausschnitt aus einem derartigen abstrakten Syntaxgraphen, hier **Schemagraph** genannt, der das in Figur 4 dargestellte EER-Diagramm darstellt.

Man erkennt, daß z.B. für die Objektklasse "FAHRZEUG" ein Knoten mit der Markierung "AtomicObjcl" existiert, an dem über entsprechend markierte Kanten als Söhne der Bezeichner ("Ident"), die Liste der Bezeichner der Schlüsselattribute ("KeyList"), die Liste der Attribute ("AttributeList") und weitere hier nicht benötigte, optionale Bestandteile einer Objektklasse hängen. Durch entsprechend markierte Kanten und Knoten kann somit die gesamte kontextfreie Struktur eines EER-Diagramms dargestellt werden (/HS 89/).

Konkrete Bezeichner stehen dabei in Knotenattributen an den entsprechenden Knoten (z.B. "FGSt.-Nr."). Kontextsensitive Zusammenhänge zwischen Bezeichnerknoten werden durch zusätzliche Kanten (z.B. "EUseDec") unmittelbar dargestellt. Im Beispiel ist eine mit "EUseDec" markierte Kante von dem mit "FGSt-Nr." attributierten Bezeichnerknoten in der separat geführten Bezeichnerliste für Schlüsselattribute zur zugehörigen Deklaration dieses Attributs gezogen worden. Aufgrund dieses kontextsensitiven Zusamenhangs wird dem Datenbankentwerfer bei der Benutzung von CADDY-O ein Löschen dieses Attributs im EER-Schema erst ermöglicht, nachdem von ihm die Schlüsseleigenschaft dieses Attributs (durch ein anderes Kommando) zurückgenommen wurde.

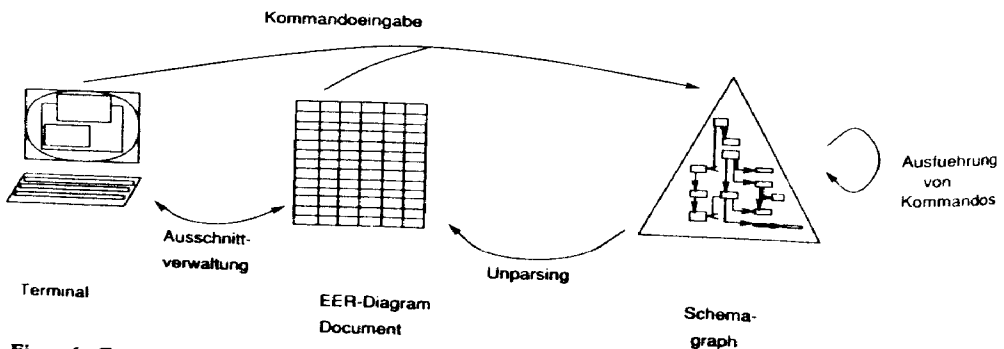


Figur 5: Ausschnitt aus einem Schemagraph

Diese Darstellung der kontextfreien und auch kontextsensitiven Struktur in der internen Datenstruktur ist darüberhinaus auch hilfreich für die Realisierung weiterer Werkzeuge in CADDY. Denn alle Werkzeuge, die Informationen über das aktuell bearbeitete Schema benötigen, greifen auf diese zentrale Datenstruktur Schemagraph zu. Die während des

Edierens ermittelte und im Schemagraphen abgelegte Information (z.B. kontextsensitive Zusammenhänge) kann dann unmittelbar von anderen Werkzeugen gelesen und benutzt werden.

Wir hatten im Kapitel 2 erläutert, daß der Datenbankentwerfer für das geometrische Layout selbst verantwortlich ist. Das bedeutet, daß neben der oben erläuterten strukturellen Information auch **geometrische Informationen** in der internen Datenstruktur verwaltet werden müssen, damit dem Entwerfer das EER-Diagramm stets in dem von ihm bestimmten Layout angezeigt werden kann. Da die Verwaltung einer weiteren Datenstruktur für die geometrische Darstellung Konsistenzprobleme mit sich bringen würde, werden bei der Realisierung von CADDY-O die geometrischen Informationen in **zusätzlichen Knotenattributen** im Schemagraphen abgelegt. Hierbei ist die getroffene Entwurfsentscheidung hilfreich, das gesamte EER-Diagramm mit einem Gitter zu unterlegen. Das bedeutet, daß für die einzelnen Bestandteile eines EER-Diagramms (z.B. Objektklassen, Attribute usw.) nur das zugehörige Rasterfeld im Gitter gespeichert werden muß. Damit ist es dann möglich, bei einer Ausgabe des EER-Diagramms durch einen sogenannten "Unparsing"- oder "Pretty-printing"-Vorgang aus der Schemagraph-Darstellung das geometrische Layout zu rekonstruieren. Figur 6 verdeutlicht den Zusammenhang.

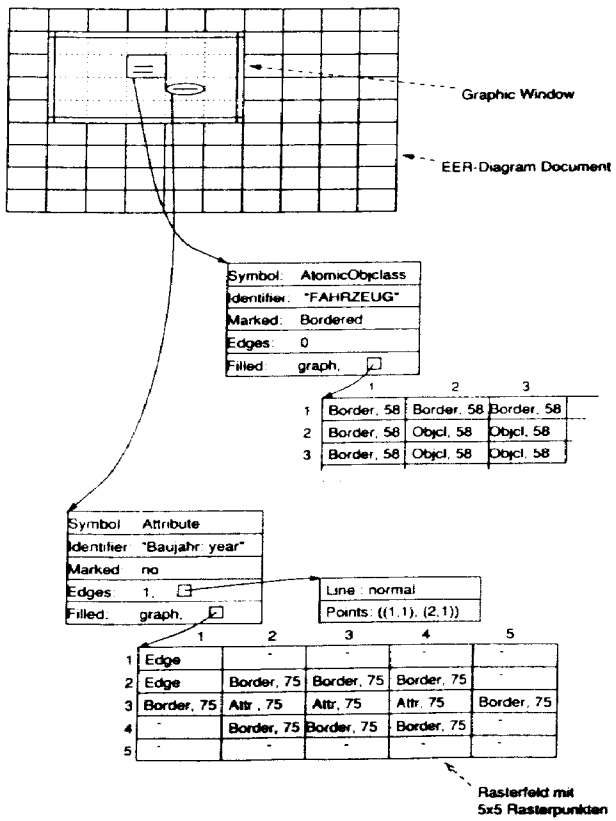


Figur 6: Zusammenhang zwischen externer und interner Darstellung eines EER-Diagramms

Der Baustein "EER-Diagram Document" verkapselt die graphische Darstellung des EER-Diagramms mit dem zugrundeliegenden Gitter. Für jedes Rasterfeld des Gitters wird gespeichert, ob und welches graphische Symbol mit welcher Beschriftung in dem Rasterfeld stehen und ob und wie sie markiert sind (z.B. "fetter Rand", "grau unterlegt"). Weiterhin wird zu jedem Rasterfeld angegeben, ob und wo Linien bzw. Pfeile in dem Rasterfeld verlaufen. Zusätzlich wird jedes Rasterfeld noch einmal in 5 x 5 Rasterpunkte unterteilt und für jeden Rasterpunkt gespeichert, ob und welches graphisches Symbol dort dargestellt ist. Figur 7 zeigt einen Ausschnitt aus dieser Datenstruktur.

Ein Ausschnitt aus dem EER-Diagram Document wird dann auf dem Bildschirm in einem entsprechenden Graphikfenster (Graphic Window) dargestellt. Eine Ausschnittsverwaltung ermöglicht dem Datenbankentwerfer, den aktuellen Ausschnitt auf dem zugrundeliegenden Dokument zu verschieben (d.h. zu "scrollen").

Bei der Auswahl eines neuen aktuellen Inkrements klickt der Datenbankentwerfer mit der Maus eine Position auf dem Bildschirm an. Dies bedeutet, daß auch die Zuordnung Fensterkoordinate zur entsprechenden syntaktischen Einheit im Schemagraphen bekannt sein muß. Aus diesem Grunde wird zu jedem Rasterpunkt zusätzlich die Identifikation des zugehörigen Knotens im Schemagraphen (in Figur 7 die Knotennummern 58 bzw. 75) im EER-Diagram Document abgelegt. Nach dem Anklicken einer Position auf dem Graphic Window kann somit der zugehörige Knoten im Schemagraphen bestimmt werden.



Figur 7: Ausschnitt aus der Datenstruktur EER-Diagram Document

4. Entwurf und Implementierung

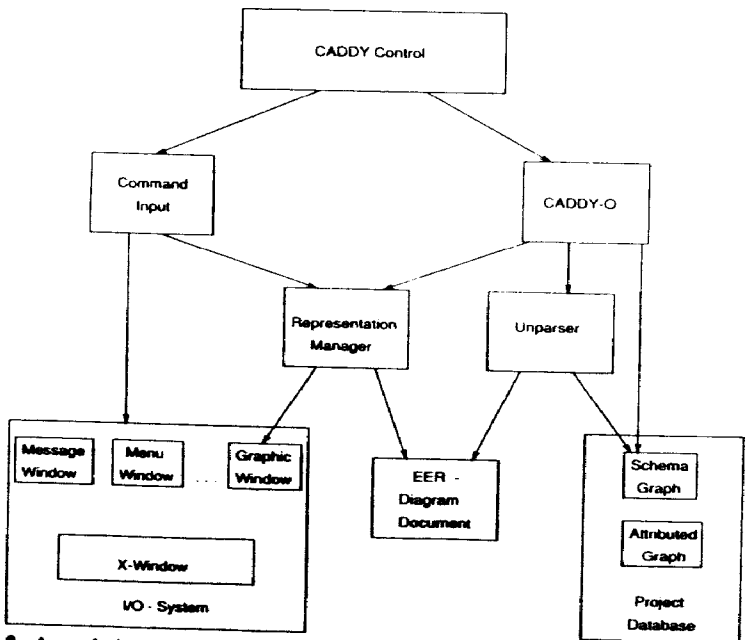
Aufgrund der obigen Entwurfsüberlegungen basiert die Realisierung von CADDY-O also auf drei Basiskomponenten (vgl. Figur 8):

(i) einem **Fenstersystem** für die einheitliche Gestaltung der Ein-/Ausgabe auf dem Bildschirm,

(ii) einer temporären Datenstruktur **EER-Diagram Document**, in der das gesamte aktuell bearbeitete EER-Diagramm in externer, also graphischer Darstellung verwaltet wird und

(iii) einer **Projektdatenbank** zur persistenten Speicherung abstrakter Syntaxgraphen, der internen Darstellung der bearbeiteten Dokumente.

Um eine hohe **Portabilität** für die Entwurfsumgebung CADDY zu erzielen, wird bei der Implementierung so weit wie möglich auf Standards zurückgegriffen. Das heißt zur Zeit, daß CADDY auf SUN 68.0x0-Workstations unter einem UNIX-artigen Betriebssystem (SUN OS) in der Programmiersprache C realisiert wird. Für die Ein-/Ausgabe auf dem Bildschirm wird auf X-Window zurückgegriffen (/SG 86/). Um eine einheitliche Benutzerschnittstelle für alle Werkzeuge von CADDY sicherzustellen, wurde oberhalb von X-Windows ein **Fenstersystem** realisiert, durch das verschiedene Fenstertypen zur Verfügung gestellt



Figur 8: Ausschnitt aus der Software-Architektur von CADDY

werden. Hierzu gehören z.B. Menüfenster ("Menu Window") für die Ausgabe von Kommandomenüs, Meldungsfenster ("Message Window") für die Ausgabe von Systemmeldungen und Graphikfenster ("Graphic Window") für die Ausgabe von graphischen Darstellungen. Typische Zugriffsoperationen des Moduls Graphic Window sind Operationen zur Ausgabe einzelner graphischer Symbole wie Rechteck, Oval, Raute, Sechseck, Linie, Text usw. und zur Eingabe einer Position im Fenster durch einen Mausklick.

Die verschiedenen Fenstertypen haben keine Kenntnis über Struktur und Inhalt der in ihnen dargestellten Informationen. Diese Information ist z.B. für das EER-Diagramm nur in der Zwischendatenstruktur **EER-Diagram Document** enthalten. Ein Funktionsmodul zur Ausschnittsverwaltung ("Representation Manager") sorgt dafür, daß ein Ausschnitt des gesamten EER-Diagramms in einem Graphic Window dargestellt wird. Hierzu benutzt es lesende Zugriffsoperationen auf das Modul EER-Diagram Document. Die dort verkapselte Datenstruktur ist in der im vorherigen Kapitel beschriebenen Form als verzeigerte Hauptspeicherdatenstruktur realisiert.

Während diese Zwischendatenstruktur nur temporär während des Arbeitens mit CADDY-O existiert, muß das bearbeitete EER-Diagramm langfristig gespeichert werden, damit der Entwerfer in mehreren aufeinanderfolgenden Sitzungen ein Datenbankschema, und damit insbesondere das in ihm enthaltene EER-Diagramm schrittweise eingeben, testen und verändern kann. Langfristig ist deshalb hier der Einsatz eines Nicht-Standard-Datenbankkerns zur Verwaltung attributierter Graphen als Projektdatenbank geplant (/LS 88/). Zur Zeit ist diese Projektdatenbank als Hauptspeicherdatenstruktur realisiert, die am Ende einer CADDY-O Sitzung in eine Datei ausgelagert wird. Diese Datenstruktur "Attributed Graph" ermöglicht eine effiziente Verwaltung attributierter Graphen. Oberhalb dieses Moduls ist die Datenstruktur "Schema Graph" in einem eigenen Modul verkapselt. Während Attributed Graph anwendungsunabhängig Zugriffsoperationen exportiert (z.B. "Liefere den Zielknoten einer auslaufenden Kante mit einer bestimmten Markierung"), sind die Operationen von Schema Graph auf die Darstellung von EER-Diagrammen bezogen (z.B. "Liefere die Liste der Attribute zu einer Objektklasse").

Diese graphartige, abstrakte Darstellung eines EER-Diagramms wird mit Hilfe eines **Unparsers** in eine konkrete Darstellung im EER-Diagram Document übersetzt. Hierbei werden insbesondere die Koordinatenangaben benutzt, die bei der Eingabe eines Symbols als zusätzliches Attribut an dem entsprechenden Knoten im Schemagraphen abgelegt wurden.

Oberhalb dieser Basiskomponenten und der sie verbindenden Transformatoren "Representation Manager" und "Unparser" liegen die beiden Moduln zur Verwaltung der Kommandoeingabe

("Command Input") und zur Steuerung des graphischen Editors CADDY-O. Die gesamte Koordination der Entwurfsumgebung CADDY, d.h. aller anderen, hier nicht erläuterten Werkzeuge ist im Funktionsmodul "CADDY Control" verkapselt.

5. Schlußbemerkungen

Das in diesem Bericht beschriebene Werkzeug CADDY-O zur syntaxgestützten Bearbeitung von EER-Diagrammen ist auf SUN-Workstations (3-er Reihe) realisiert und vorführbar. Die gesamte Implementierung von CADDY umfaßt bisher ca. 20.000 Zeilen C. Zur Zeit werden erste Erfahrungen mit dem Arbeiten mit CADDY-O ausgewertet. Geplant ist eine Erweiterung der Funktionalität (z.B. Verschieben von Diagrammteilen), um den Benutzerkomfort und damit die Akzeptanz noch weiter zu erhöhen.

Außerdem ist CADDY-O nur ein Teil der integrierten CADDY-Umgebung. Zur Zeit laufen im Rahmen von Studien- und Diplomarbeiten zahlreiche Implementierungsarbeiten, um weitere Werkzeuge zur Bearbeitung und zum Test eines konzeptionellen Datenbankschemas zu realisieren. Hierzu gehört z.B. ein Transformator, der das im Schemagraph abgelegte EER-Diagramm automatisch in ein relationales Schema übersetzt und in einem INGRES-Datenbanksystem eine zugehörige Prototyping-Datenbank zum Testen des spezifizierten Schemas installiert. Eine ausführliche Beschreibung dieser Werkzeuge findet man u.a. in /EHHLE 89/.

Danksagung
Ganz besonders möchte ich mich bei Christiane Beer bedanken, die im Rahmen Ihrer Tätigkeit als technische Mitarbeiterin an CADDY mitarbeitet. Sie hat wesentlich dazu beigetragen, daß der erste Prototyp von CADDY und damit insbesondere der hier beschriebene Editor CADDY-O in so kurzer Zeit erstellt wurde. Weiterhin danke ich meinen Kollegen/innen U. Hohenstein, K. Hülsmann und P. Löhr-Richter für zahlreiche interessante Diskussionen und allen beteiligten Studenten/innen für Ihren Einsatz für CADDY. J. Ebert (EWH Koblenz) danke ich für die Bereitstellung des Graphenlabors.

Literatur

/BNTT 85/ Batini, C./ Nardelli, E./ Talamo, M./ Tamassia, R.: GINCOD: A Graphical Tool for Conceptual Design of Data Base Applications, in Albano/DeAntonellis/DiLeva (eds.): Computer-Aided Database Design: The DATAID Project. Amsterdam: North-Holland 1985, S. 33-51

/EHHLE 89/ Engels, G./ Hohenstein, U./ Hülsmann, K./ Löhr-Richter, P./ Ehrich, H.-D.: CADDY - Computer-Aided Design on Non-Standard Databases, in H. Weber (Hrsg.): Proc. of the First Intern. Conference on System Development Environments & Factories, Berlin, Mai 1989, Pitman Publishing, London 1989

/En 86/ Engels, G.: Graphen als zentrale Datenstukturen 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, Dez. 1986, ACM SIGPLAN Notices, Vol. 22, No. 1, 1987, S. 190-198

/He 89/ Henderson, P. (ed.): Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Conference on Practical Software Development Environments, Boston, Dezember 1989, ACM SIGPLAN Notices, Vol. 24, No. 2, Februar 1989

/HG 88/ Hohenstein, U./ Gogolla, M.: A Calculus for an Extended Entity-Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions, in C. Batini (ed.): Proc. of the 7th Int. Conf. on ER Approach, Rom 1988, S. 1-20

/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", Wagner/Traunmüller/Mayr (Hrsg.), Informatik-Fachbericht Bd. 143, Berlin: Springer 1987, S. 58-88

/HS 89/ Henneman, Chr./ Schacht, J.: Realisierung eines Parsers für eine Datendefinitionssprache eines erweiterten Entity-Relationship-Modells, Studienarbeit, TU Braunschweig 1989

/LS 88/ Lewerentz, C./ Schürr, A.: GRAS - a Management System for Graph-like Documents, in Beeri/Schmidt/Dayal (eds.): Proc. 3rd. Int. Conf. on Data and Knowledge Bases, San Matheo: Morgan-Kaufmann 1988, S. 19-31

/Lu 82/ Lundeberg, M.: The ISAC approach to specification of information systems, in Olle/Sol/Verrijn-Stuart (eds.): Information Systems Design Methodologies, Amsterdam: North-Holland 1982, S. 173-234

/Os 88/ Osterhold, A.: Entwurf und Implementierung eines syntaxgestützten Editors für die Bearbeitung von Datentypspezifikationen, Diplomarbeit, TU Braunschweig 1988

/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 (Versailles), AFCET, Paris 1986, S. 165-176

/Sch 89/ Schönthaler, F.: Rapid Prototyping zur Unterstützung des konzeptuellen Entwurfs von Informationssystemen, Dissertation, TH Karlsruhe 1989

/SG 86/ Scheifler, R.W./ Gettys, J.: The X Window System, ACM Transactions on Graphics, Vol. 63, 1986

/WP 87/ Wasserman, A./ Pircher, P.: A Graphical, Extensible Integrated Environment for Software Development, in ACM SIGPLAN Notices, Vol. 22, No. 1, 1987, S. 131-142