Implementierung einer objektorientierten, kooperativ arbeitenden Softwarekomponente auf Client/Server-Basis im hostorientierten Umfeld der Allianz Lebensversicherungs-AG.

Diplomarbeit

An der Universität Leipzig, Fachbereich Mathematik und Informatik,
Institut für Informatik,
Lehrstuhl für Computersysteme,
sowie bei der
Hauptverwaltung Stuttgart der Allianz Lebensversicherungs-AG

Betreut durch: Herrn Prof. Dr.-Ing. W. G. Spruth,
Herrn Dr. K. Hänßgen
Herrn G. Hahn und
Herrn J. Steckelberg

Leipzig, 9. April 1997

Ulrike Sieber,
geboren am 13.05.1972 in Weimar/Thüringen.

Zum Geleit

Die vorliegende Arbeit befaßt sich mit einer konkreten Fragestellung aus dem Gebiet der Client/Server-Anwendungsentwicklung, wobei das Hauptaugenmerk auf der Verbindung neuer Technologien mit vorhandenen Systemen liegt.

Der aus den Untersuchungen hervorgegangene Prototyp einer Anwendung setzt dabei die Möglichkeiten moderner, graphischer Softwareentwicklungs-Tools, die Client/Server-Funktionalitäten relationaler Datenbanksysteme und das Einbeziehen vorhandener Datenbestände um.

Dieser Prototyp und seine Entwicklungsstufen werden in dieser Arbeit ebenso beschrieben, wie die Funktionalitäten verteilter Datenbanken und die Grundlagen verschiedener Client/Server-Architekturen, sowie eine genaue Untersuchung und Einschätzung der verwendeten Entwicklungswerkzeuge.

Diese Arbeit und die Beispielanwendungen hätten ohne die Hilfe und Unterstützung vieler Menschen nicht so erstellt werden können, wie sie nun vorliegen. Daher möchte ich mich an dieser Stelle bei all jenen bedanken, die zum Gelingen beigetragen haben.

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Spruth, der es ermöglichte, diese Arbeit an seinem Lehrstuhl zu verfassen sowie Herrn Rose, Abteilungsleiter bei der Allianz Lebensversicherungs-AG in Stuttgart, der die Voraussetzungen für die Durchführung des praktischen Teils schuf.

Dank möchte ich auch meinen Betreuern bei der Allianz, Herrn Hahn und Herrn Steckelberg sowie Herrn Dr. Hänßgen, Betreuer am Lehrstuhl Computersysteme, aussprechen.

Für den technischen Support während der Programmierung der Anwendungen möchte ich mich besonders bei Frau Streich, Herrn Dr. Lüdeking, Herrn Pomorin, Herrn Sommerlad und Herrn Ziegler (alle Allianz) bedanken, ebenso bei Herrn Nold (IBM). Erst durch ihre Hilfsbereitschaft konnten viele technische Probleme überwunden werden.

Nicht zuletzt gilt mein Dank auch meinen Eltern und Freunden, die stets für mich da waren und mir die nötige Kraft gaben.

Inhaltsverzeichnis

ZUM GELEIT FEHLER! TEXTMARKE NICHT DEFINIERT.

INHALTSVERZEICHNIS **4**

1. EINLEITUNG **7**

1.1 Einführung in das Thema **7**

1.2 Beschreibung der derzeitigen Situation der Anwendungsentwicklung in der Allianz Lebensversicherungs AG **8**

1.3 Zielsetzung **10**

2. CLIENT/SERVER GRUNDLAGEN **11**

2.1 Mainframe und PC - mögliche Varianten gemeinsamer Nutzung **11**

2.1.1 PC als Terminal 11

2.1.2 Verteilte Verarbeitung 12

2.2 Client/Server-Architekturen **13**

2.2.1 Verteilte Präsentation 13

2.2.2 Verteilungsform geliftete Masken 13

2.2.3 Verteilungsform entfernte Datenbank 16

3. CLIENT/SERVER-DATENBANK-MANAGEMENTSYSTEME **17**

3.1 Grundlagen relationaler Datenbank-Managementsysteme **17**

3.1.1 Entity-Relationship Modell 17

3.1.1.1 Graphische Darstellung eines ER-Modells 18

3.1.1.2 Schlüsselattribute 18

3.1.3 Umsetzung in Tabellen 19

3.2 Benutzerrechte im Datenbanksystem **19**

3.3 Client/Server-Funktionalitäten eines DBMS **21**

3.4 Katalogisierung von Datenbanken **21**

3.4.1 Benutzerprofilverwaltung (UPM) 22

3.4.2 Eintragung eines fernen Knotens 23
3.4.3 Systemdatenbankverzeichnis 24

3.5 Schematische Darstellung eines Client/Server-Datenbankzugriffs 26
3.5.1 Schematische Darstellung des einstufigen C/S-Betriebes 27
3.5.2 Mehrstufiger C/S-Betrieb über ein Gateway 29

4. ERSTELLTE CLIENT/SERVER-ANWENDUNGEN 31

4.1 Konfiguration der benutzen Rechner (Soft- und Hardware) 31
4.1.1 Konfiguration des Client-Rechners 31
4.1.2 Konfiguration des Servers 31
4.1.3 Der Mainframe 32

4.2 Die Beispielprogramme 32
4.2.1 Lokaler Datenbankzugriff 33
4.2.2 Zugriff auf eine entfernte DB2/2-Datenbank 33
4.2.3 Zugriff auf eine Mainframedatenbank 35

5. PROGRAMMIERUMGEBUNG UND WERKZEUGE 39

5.1 Arbeiten mit Projekten - der Workframe 39
5.1.1 Projekte aus dem Ordner „Project Smarts“ 39
5.1.2 Projekte aus dem „Project Template“ 40
5.1.3 Einstellungen im WorkFrame 41
5.1.4 Vererbung von Projektinformationen 43

5.2 Entwicklung einer dynamischen Bibliothek für den Datenbankzugriff 44
5.2.1 Vorgehensweise 44
5.2.2 Anlegen des Projekts 44
5.2.3 Der Data Access Builder 46
5.2.3.1 Generieren der Klassen 47
5.2.3.2 Eigenschaften und Verwendung 50
5.2.4 Erzeugen der *DLL* 51
5.2.5 Arbeiten mit der DLL 51
5.2.6 Binden der Anwendung an die Datenbank 52
5.2.7 Enhanced Data Access Builder 53

5.3 Erstellen einer Applikation mittels visueller Programmierung 55
5.3.1 Erstellen eines Projekts 55

5.3.2 Der Visual Builder 56
5.3.2.1 Einstellungen im Visual Builder 57
5.3.2.2 Laden und Importieren von Klassen und Parts 58
5.3.3 Arbeitsoberfläche Composition Editor /Class Editor/ Part Interface Editor 59
5.3.3.1 Ausgangssituation 62
5.3.3.2 Arbeiten mit Parts 63
5.3.3.3 Verknüpfen der Parts 63
5.3.3.4 Generieren des Quellcodes 67
5.3.4 Compilieren und Binden 69
5.3.4.1 Compileroptionen 70
5.3.4.2 Linkeroptionen 72

5.4 Tools zur Bearbeitung erstellter Programme 73
5.4.1 Debugger 73
5.4.2 Browser 75
5.4.3 Performance Analyzer 78

5.5 Probleme und Problembehebung 80
5.5.1 Fixes 80
5.5.2 Compiler- und Linkerfehler 80

6. ABSCHLIEßENDE BEMERKUNGEN 83

6.1 Einschätzung der erstellten Programme 83

6.2 Visuelle objektorientierte Programmierung 84

6.3 Ausblick 88

7. ANHANG 89

7.1 Glossar 89

7.2 Abbildungsverzeichnis 94

7.3 Tabellenverzeichnis 95

7.4 Literaturverzeichnis 95

7.5 Trademarks/Warenzeichen 97

1 Einleitung

1.1 Einführung in das Thema

Wie in allen Unternehmen, die schon seit Jahren täglich große Mengen Daten verwalten und bearbeiten müssen, findet sich auch in der Allianz Lebensversicherung[1] folgende typische Hardware-Struktur:
Das Kernstück der Datenverarbeitung bildet eine IBM /390, dazu kommen Terminals und sukzessive auch PCs. An ihrem Arbeitsplatz werden die Sachbearbeiter demnächst alle mit PCs ausgestattet sein, in der Anwendungsentwicklung ist dies bereits der Fall.
Fast alle Programme, die derzeit für einen reibungslosen Ablauf der elektronischen Datenverarbeitung im Innendienst sorgen, wurden für den /390-Großrechner entwickelt und laufen dort stabil. Diese Zuverlässigkeit der Programme und auch der immense Umfang der bearbeiteten Informationen macht die Mainframes auch weiterhin unverzichtbar. Zum einen ist die Wartung und Pflege bestehender Programme wesentlich kostengünstiger als die Entwicklung neuer Software, die erst mit langwierigen Tests auf ihre Zuverlässigkeit und Korrektheit untersucht werden muß.
Zum anderen gibt es derzeit keine Alternative für die Bearbeitung und Speicherung von Daten in Großrechnerdatenbanken (z.B. in DB2 für MVS). Kein PC kann momentan mit ähnlichen Datendurchsatzraten und Platten- bzw. Hauptspeichergrößen aufwarten.
Auch wird die Großrechnersoftware immer schwieriger wartbar, denn häufig gibt es kaum noch Mitarbeiter, die bei der Entwicklung von Anfang an dabei waren, und ein Programm im Detail kennen. Oft wurde besonders platzsparend programmiert, was der Übersichtlichkeit alter Programme abträglich ist. Es ist nur eine Frage der Zeit, wann solche alten Programme neu erstellt werden müssen.
Auch findet der PC mehr und mehr Verbreitung - sei es im Büro oder in der Anwendungsentwicklung. Preisverfall bei der Hardware, leistungsfähigere, mächtigere Standard-Programme und die Möglichkeiten der graphischen Aufbereitung von Daten machen den PC attraktiver. Die Bedienung der graphischen Oberflächen ist ganz intuitiv möglich.

[1] Im weiteren oft kurz als Allianz Leben bezeichnet.

Die gestiegene Leistungsfähigkeit der Personalcomputer mit der vorhandenen Soft- und Hardware zu kombinieren, um alle Vorteile modernster Elektronik nutzen zu können, scheint ganz selbstverständlich der nächste Schritt zu sein.

Die Softwarearchitektur, die unter dem Namen „Client/Server“ bekannt ist, und die vorliegende Arbeit setzen hier an. Im Rahmen der Vorstudie/Sollanalyse für die Neugestaltung des Zentralen Inkassos der Allianz Lebensversicherungs-AG sollte herausgefunden werden, ob eine Client/Server-Programmierung unter Nutzung der bereits vorhandenen Großrechneranwendungen möglich ist, und welche Chancen sich daraus für zukünftige Entwicklungen ergeben.

Folgende Konvention soll dabei für die Nomenklatur der vorliegenden Untersuchung gelten: Als Client wird der Arbeitsplatzrechner bezeichnet, der zur korrekten Ausführung und Beendigung eines Programmes Daten eines anderen Rechners benötigt und diese anfordert. Der korrespondierende Rechner, der die angeforderten Daten liefert, sei stets der Server.

1.2 Beschreibung der derzeitigen Situation der Anwendungsentwicklung in der Allianz Lebensversicherungs-AG

Das Bild, das sich in der Datenverarbeitung in der Allianz Lebensversicherungs-AG[2] zur Zeit zeigt, ist geprägt durch die hauptsächlich im Einsatz befindlichen Großrechnerprogramme.

Für die Entwicklung neuer PL/1-Programme bzw. Programmodule stehen viele Standardbibliotheken, sowie eine ausgereifte Testumgebung zur Verfügung. Nicht zuletzt wird jedes Modul vor der Freigabe sorgfältig unter produktionsähnlichen Bedingungen getestet. Dazu gibt es einen umfangreichen Testdatenbestand, der regelmäßig aus der Produktion abgeleitet wird und dessen Konsistenz kontinuierlich überwacht wird. Das sind optimale Bedingungen zur Erstellung funktionstüchtiger, fehlerarmer Software.

[2] Oft kurz mit Allianz Leben bezeichnet.

Dies ist äußerst wichtig, da ein in Produktion überführtes Modul nicht schnell zurückgenommen werden kann, und ein Ausfall der EDV für wenige Minuten schon zu großen unternehmerischen Verlusten führen kann.
Hinzu kommen die für PL/1 geschulten Mitarbeiter, die unter dieser Programmierumgebung hohe Produktivität erreichen und deren Umschulung Zeit und Kosten verursacht. Die Mehrheit der Programme wird deshalb sicher auch weiterhin im Sinne des Investitionsschutzes für den Großrechner erarbeitet werden.
Darüber hinaus liegen fast alle Daten auf Host-Datenbanken. Dazu zählt ein proprietäres Datenbanksystem der Allianz (BALSAM genannt), sowie relationale DB2 für MVS-Datenbanken.

Neben dem Mainframe existieren aber zahlreiche PCs, und das Ziel ist es, in Kürze alle Mitarbeiter damit auszustatten. Diese PCs können mittels *Communications Manager* und der 3270-Emulation auch als Terminal arbeiten und weiter für die Großrechnerwelt genutzt werden. Darüber hinaus bieten sie die Möglichkeit, Software der individuellen Datenverarbeitung, wie Textverarbeitungssysteme, Tabellenkalkulationen, Grafik-programme zu nutzen.
Mit der weiteren Entwicklung dieser Software kommt dem PC besondere Bedeutung zu.
Dem wird in einem Allianz-weiten Projekt Rechnung getragen, welches den Titel „Arbeitsplatz der Zukunft“ („APZ“) trägt. Hier sollen die Vorteile vernetzter PCs mit allen ihren Möglichkeiten genutzt werden. Das betrifft vornehmlich die graphische Darstellung, die interne Vernetzung der PCs sowie die Anbindung an Großrechner und andere Serverrechner, wie Workflow oder NCI[3]. Denkbar wäre auch hier eine Nutzung des Internet.

Auch andere Firmen bieten inzwischen Software an, die sich an diese Voraussetzungen anpassen läßt. Erwähnt sei hier das System SAP / R3, welches folgende Architektur[4] ermöglicht:

- Host weiterhin als Datenbankserver für das relationale DB2,
- Anwendungsserver - zwischen Host und Arbeitsplatzrechner

[3] NCI-non coded information. Z.B. als Grafik aufbewahrte Schriftstücke etc.

[4] Siehe dazu auch [Wen95], S. 283.

- Präsentation auf den Arbeitsplatzrechnern mit graphischer Oberfläche.

Dieses Angebot von SAP richtet sich an Firmen, die ähnlich wie die Allianz eine /390 weiterhin als Kernstück der EDV betreiben werden, aber trotzdem an die Entwicklung auf dem PC-Sektor anknüpfen wollen.
SAP-Produkte kommen in der Allianz Leben bereits zum Einsatz.

1.3 Zielsetzung

In der vorliegenden Arbeit wird untersucht, inwieweit sich vorhandene Datenbestände der Großrechnerdatenbanken mit Hilfe neuer PC-Anwendungsentwicklungs-Tools nutzen lassen. Darüber hinaus wird die visuelle Programmierung bei Implementierung von Client/Server-Programmen bewertet.
Zunächst wird aufgezeigt, welche Formen der Client/Server-Architektur möglich sind (Grundlagen der Client/Server-Architektur - Kapitel 2). Zwei dieser Architekturen werden näher erläutert (Abschnitte 2.2.2 und 2.2.3). Anschließend wird in Kapitel 3 im Rahmen der Darstellung der Client/Server-Funktionalität von Datenbanken eine Übersicht der Prinzipien relationaler Datenbanksysteme gegeben, insbesondere am Beispiel DB2/2. Weitere Grundlagen zu relationalen Datenbanksystemen befinden sich ebenfalls in diesem Kapitel.
Die folgenden Kapitel 4 und 5 gehen über die Grundlagen hinaus und bringen detaillierte Beschreibungen der erzeugten Software. Der Übersicht der Hardware- und Softwarevoraussetzungen, die zur Erstellung der vorliegenden Programme notwendig waren, folgt eine Erläuterung der Beispielprogramme in Kapitel 4.
Das 5. Kapitel gibt eine Übersicht der notwendigen Anpassungen, die an der Arbeitsumgebung vorgenommen werden mußten, sowie eine Beschreibung der verschiedenen Entwicklungswerkzeuge unter Visual Age C++. Es erläutert anhand vieler Beispiele die einzelnen Schritte zur Implementierung eines objektorientierten Client/Server-Programms.
Eine Bewertung der Programme und der Tools in Kapitel 6 soll einen Ausblick auf weitergehende Nutzungsmöglichkeiten dieses Client/Server-Ansatzes geben.

2 Client/Server Grundlagen

2.1 Mainframe und PC - mögliche Varianten gemeinsamer Nutzung

Aus den vorangegangenen Ausführungen ergibt sich, daß in nächster Zeit zentrale Großrechner weiterhin wesentliche Bestandteile der Datenverarbeitung sein werden. Gleichzeitig sollen aber die Vorteile eines PCs stärker als bisher genutzt werden, die Rechenleistung „vor Ort“ nicht nur für Emulationen dienen.

2.1.1 PC als Terminal

Die einfachste Variante des „Nebeneinander“ von PC und Host ist die, daß ein PC wie ein Terminal gehandhabt wird, also lediglich eine 3270-Emulation[5] genutzt wird. Alle Großrechnerprogramme werden wie bisher bedient und der Mitarbeiter hat zusätzlich die Möglichkeit, auf dem PC Berichte, Grafiken oder kleine Datenbanken zu erstellen und dabei PC-Standardsoftware zu nutzen.
Das bietet den Vorteil, daß keine Großrechnerprogramme neu erstellt werden müssen. Es stellt sich aber die Frage, ob die Anschaffung teurer PCs notwendig ist, wenn diese nur die Funktion besserer Terminals übernehmen.
Gerade diese Fragestellung: „Einsatz einer Emulation oder Client/Server-Programmierung“ unter dem Aspekt des finanziellen Aufwandes wird auch in [NC96] diskutiert.

Des weiteren darf nicht unerwähnt bleiben, daß ein Einsatz einer Emulation und damit verbunden die parallele Benutzung von Großrechner- und PC- Software die Einarbeitung des Benutzers in beide Systeme erforderlich macht. Er lernt einerseits mit der Tastatur und Transaktionssymbolen zu arbeiten, aber auch mit Maus und einer graphischen Oberfläche umzugehen. Man kann davon ausgehen, daß die Einarbeitungszeit neuer Mitarbeiter länger ist, je mehr unterschiedliche Systeme sie bedienen müssen.

[5] Allgemeiner: Terminal-Emulation.

2.1.2 Verteilte Verarbeitung

Den nächsten Schritt, den man gehen kann, um die Ressourcen vorhandener *Mainframes* und PCs zu nutzen, ist die Implementierung „echt verteilt“ arbeitender Client/Server-Systeme. Dabei werden verschiedene Stufen der Arbeitsteilung zwischen Client und Server unterschieden. Abbildung 2-1 soll einen Überblick über die verbreiteten Varianten[6] geben. In der oberen Reihe ist der Client dargestellt, in der unteren Reihe der Server.

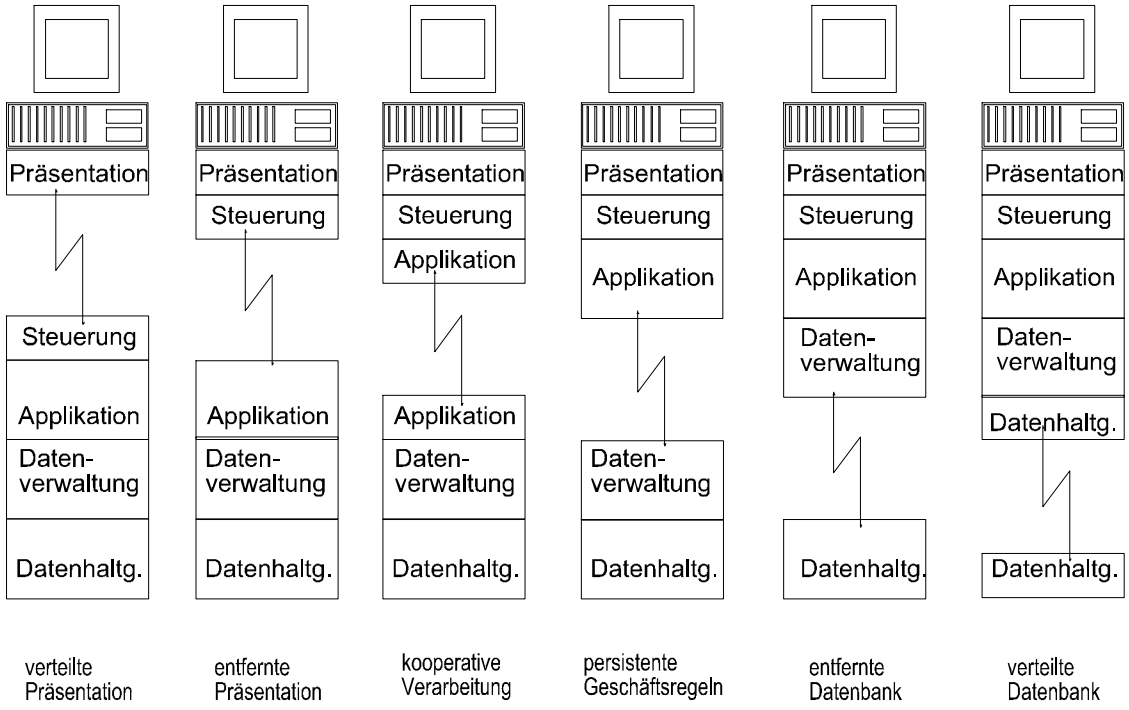

Abb. 2-1: Verteilungsformen einer Client/Server-Anwendung.

In Untersuchungen zu möglichen Verteilungskonzepten von Client/Server-Anwendungen werden für die zukünftige Anwendungsarchitektur in der Allianz Lebensversicherungs-AG nur vier Verteilungsformen unterschieden.[7] Sie lassen sich im wesentlichen in das oben gezeigte Schema einordnen. Daher soll im folgenden mit den in Abbildung 2-1 dargestellten Bezeichnungen für Verteilungen gearbeitet werden.

[6] Nach [Nie95] S.21

[7] Vgl. C/S-AAA Arbeitsmappe vom 22.08.96, Seite 4 „Verteilungskonzepte“.

2.2 Client/Server-Architekturen

2.2.1 Verteilte Präsentation

Gegenstand der folgenden Betrachtungen soll die Client/Server-Verteilungsform „verteilte Präsentation“ sein. Sie wird in der Allianz Lebensversicherungs AG bereits genutzt. Ein Beispiel dafür sind die sogenannten „*gelifteten Masken*“.

2.2.2 Verteilungsform geliftete Masken

Eine Möglichkeit der Client/Server-Implementierung ist die Anwendung eines Programmpakets wie *Phantom* oder *Visual Lift*[8], um den Datenaustausch, der zwischen Anwender und Hostprogramm erfolgt, so darzustellen, als arbeitete man mit einem typischen PC-Programm. (Dies ist auch unter dem Schlagwort "*GUIfizierung*" bekannt, *GUI=Graphical User Interface.*)

Diese Programmpakete bieten die Möglichkeit, den ankommenden 3270-Datenstrom abzufangen und aufzubereiten. Dabei ergeben sich auf dem Server (Großrechner) keinerlei Änderungen. Am PC werden die eintreffenden Daten aufbereitet und über eine graphische Benutzeroberfläche zugänglich gemacht. Nach einer Eingabe durch den Nutzer läuft der gleiche Prozeß in umgekehrter Richtung ab: Die Eingabe wird umgesetzt in ein dem Hostprogramm bekanntes Format und dann abgeschickt. Das heißt, es ist im Dialog mit dem Großrechner nur eine andere Eingabeschnittstelle hinzugekommen. Am Programm selbst wird dabei nichts geändert. Es funktioniert unabhängig davon, wie der Nutzer diesen Dialog sieht. Mit solchen sogenannten „*gelifteten Masken*“ hat man die Möglichkeit, Großrechnerprogramme unverändert belassen zu können, man kann aber andererseits in gewissen Umfang die Vorteile graphischer Benutzeroberflächen nutzbar machen. Für den Anwender ist dann auch kein Medienbruch mehr zu spüren, er arbeitet sich gleich in die graphische Oberfläche ein und muß auch für Host-Dialoge nicht umlernen.

Auf diesem Weg lassen sich jedoch keine graphischen Oberflächen mit voller Funktionalität implementieren. Zudem arbeiten die gelifteten Dialog-Masken nur, wenn

[8] Siehe dazu auch Anhang: „Warenzeichen“.

die Emulation gleichzeitig mitläuft, da eine Emulation die Basis für den Datenaustausch auf diesem Wege ist. Man bezeichnet dieses Vorgehen auch als *thin Client*, da kaum Rechenleistung vom Client übernommen wird, lediglich die Aufbereitung der Masken erfolgt „vor Ort“. Auch besteht die Gefahr, daß auf diese Weise ein Übergang zur kooperativen Verarbeitung verhindert oder aber zumindest verzögert wird.

Von den derzeitigen Host-Dialogmasken ist bisher nur ein kleiner Teil überarbeitet und *GUIfiziert* worden. Die folgenden Grafiken (2-2 und 2-3) stellen beide Versionen der inhaltlich gleichen Eingabemaske einander gegenüber.

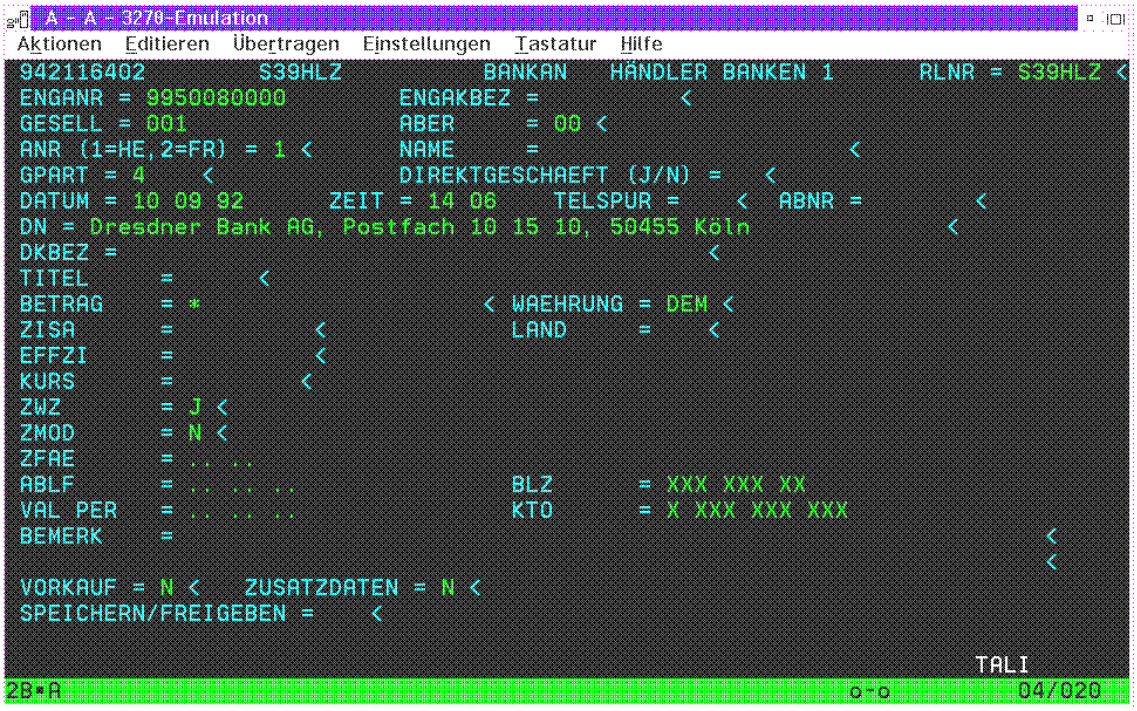

Abb. 2-2: Originale Host-Dialogmaske (aus dem Händlerarbeitsplatz[9]).

[9] Händlerarbeitsplatz bedeutet hier: eine typische Arbeitsumgebung für Mitarbeiter des Wertpapierhandels.

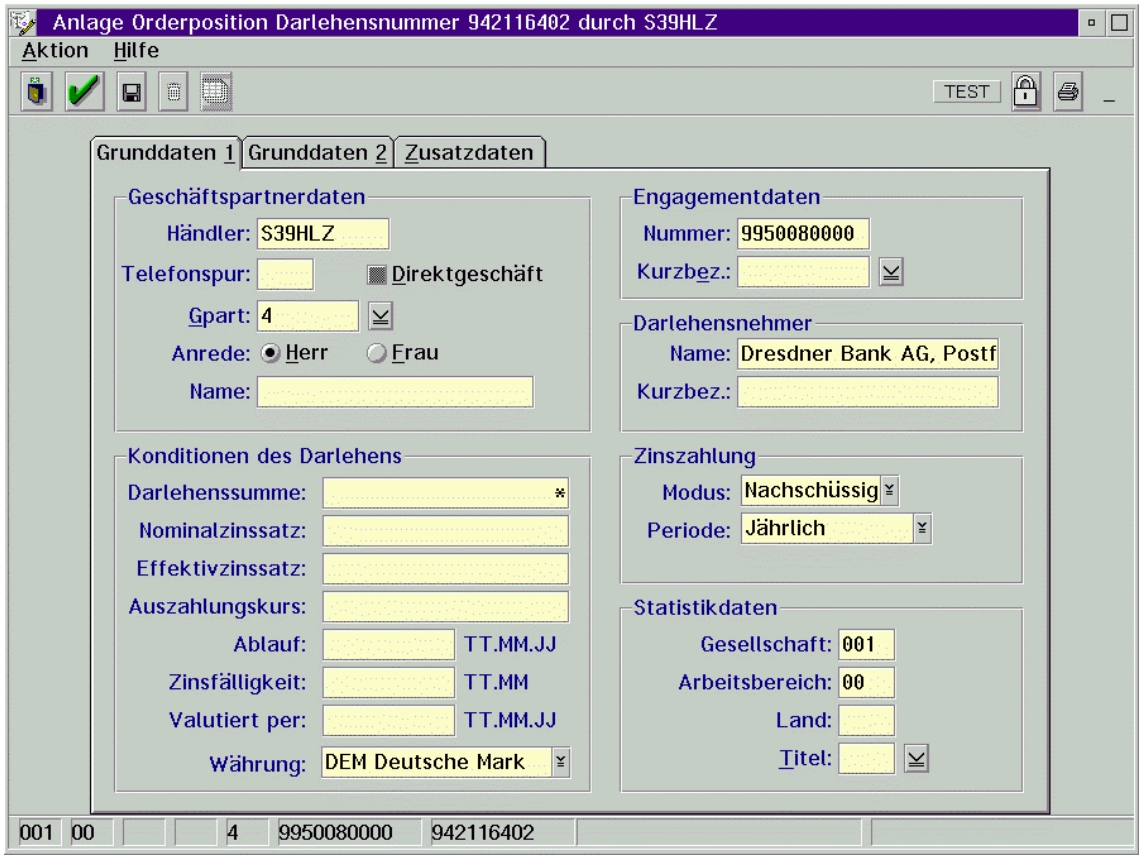

Abb. 2-3: Gleiche Eingabemaske wie Abb. 2-2, geliftet.

Entsprechend der am Beginn des Abschnittes gezeigten Unterteilung (Bild 2-1) würde die Form der beschriebenen Verteilung mittels *gelifteter Masken* der verteilten Präsentation zugeordnet werden. Das unterstreicht [Nie95] S.23:

> „Im einfachsten Fall handelt es sich bei der Netzwerkschnittstelle um dieselbe, die auch für die Ausgabe auf ein Terminal ohne eigenen Prozessor verwendet wird. In diesem Fall wird das Terminal durch einen PC ersetzt, auf dem (in der Regel mit Hilfe eines Werkzeuges) der Terminaldatenstrom gelesen, interpretiert und mit Hilfe der Präsentationsfunktionen in Form einer graphischen Oberfläche dargestellt wird.“

Ähnliche Lösungen finden sich auch im UNIX-Umfeld. Das verbreitete X-Windows arbeitet auf vergleichbare Weise. Jedoch werden hier sämtliche Ein- und Ausgaben über

ein LAN[10] vom Terminal zum UNIX-Rechner (bzw. umgekehrt) gesendet und ausgewertet (z.B. Aufruf eines Kontextmenüs). Bei der verteilten Präsentation in X-Windows ist die Menge der übertragenen Daten zwischen Client und Server daher noch größer, als bei einer Terminal-Emulation. Bei einer Emulation auf einem PC kann einiges (z.B. der Aufruf eines Kontextmenüs) mit lokaler Rechenleistung erbracht werden, und hat somit keinen Verkehr über das LAN zur Folge.

Ausgehend von den oben genannten Bedingungen, ist mit der im Rahmen dieser Arbeit angefertigten Beispielanwendung eine weitere Stufe der Verteilung realisiert worden. Dabei handelt es sich um die Verteilungsform „entfernte Datenbank“.

2.2.3 Verteilungsform entfernte Datenbank

Während im Verteilungsmodell „persistente Geschäftsregeln“ davon ausgegangen wird, daß auf der angesprochenen (fernen) Datenbank sogenannte *stored procedures* vorhanden sind, und keine lokale Datenverwaltung notwendig ist, so ist im Modell „entfernte Datenbank“ die Datenverwaltung (zumindest teilweise) auf dem Client-Rechner erforderlich.
Dieses Modell basiert auf Client/Server-Datenbanksystemen (im vorliegenden Fall DB2/2). Dabei befindet sich der Datenbestand, also die Datenbank, auf dem Server, während die Datenverwaltungsebene des Clients mit der Server-Datenbank über ein sogenanntes API kommuniziert (*API=Application Programming Interface*). Der Zugriff beruht auf der Schnittstelle des *Remote Database Access* (RDA).
Diese Art der Verteilung findet sich häufig in LANs, während die Menge der transportierten Daten einen Einsatz in einem WAN[11] eher unwahrscheinlich werden lassen. Die Client/Server-Funktionalität liegt hierbei gänzlich bei dem verwendeten Datenbanksystem.

[10] LAN=Local Area Network.

[11] WAN=Wide Area Network.

3 Client/Server-Datenbanksysteme

Wie im Abschnitt 2.2.2 dargestellt, kann eine Client/Server-Anwendung mit Hilfe der Client/Server-Funktionalität einer Datenbank aufgebaut werden. In den im Rahmen dieser Arbeit erstellten Anwendungen wurde dazu das Datenbanksystem DB2/2 für OS/2 bzw. DB2 für MVS benutzt. Der nächste Abschnitt beschäftigt sich daher mit den wesentlichen Eigenschaften relationaler Datenbanksyteme.

3.1 Grundlagen relationaler Datenbanksysteme

Das Datenbanksystem DB2/2 ist, ebenso wie das Datenbanksystem DB2 für MVS, ein relationales Datenbanksystem. Es ermöglicht eine Strukturierung der Daten gemäß dem Relationenmodell, sowie Angaben zur Datenintegrität. Es umfaßt darüber hinaus Möglichkeiten zur Manipulation der Daten.

3.1.1 Entity-Relationship-Modell

Beim Entwurf[12] einer relationalen Datenbank kommt das sogenannte *Entity-Relationship-Modell* zum Einsatz. Darin unterscheidet man

- Entitäten
- Relationen
- Attribute
- Integritätsbedingungen

Bevor man eine Datenbank implementiert wird im ersten Schritt der Ausschnitt der realen Welt, der modelliert werden soll, in einem *Entity-Relationship-Modell*[13] dargestellt.

Entitätsmengen werden dabei als Rechtecke und Beziehungen (Relationen) als Rauten dargestellt. Die Kardinalität der Beziehungen wird explizit an den Verbindungslinien angegeben. Das nachfolgende Beispiel (Abbildung 3-1) zeigt einen Ausschnitt der fiktiven Modellierung der Verwaltung eines Yachthafens (Liegeplatzverwaltung).

[12] Zum Entwurf relationaler Datenbanken siehe [Vos94] Abschnitt II.

3.1.1.1 Graphische Darstellung eines ER-Modells

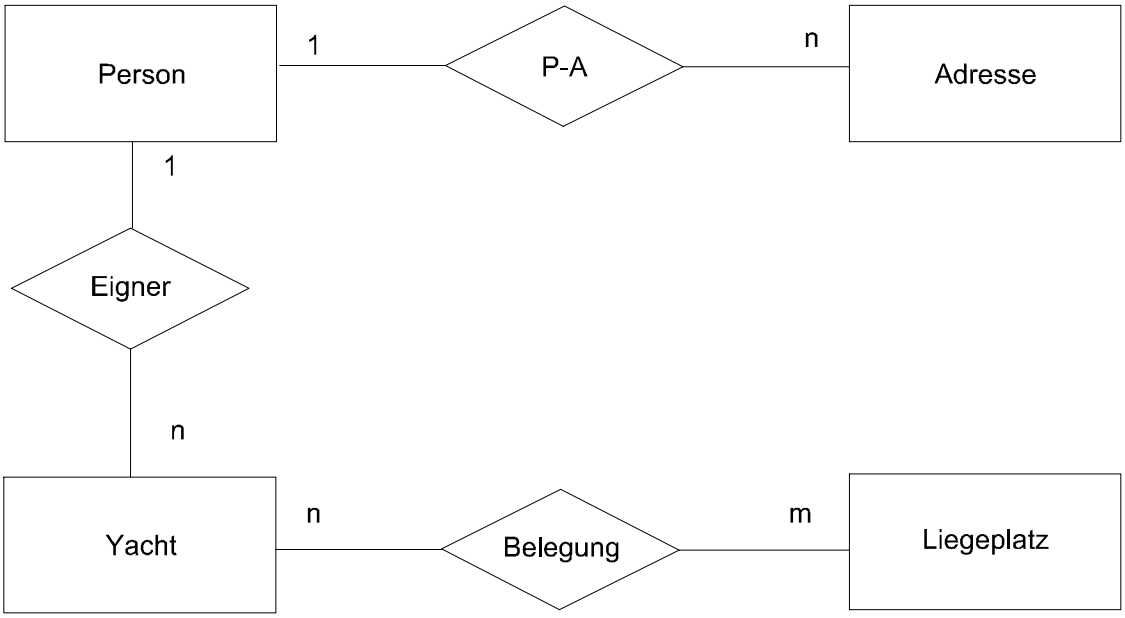

Abb. 3-1: Entity-Relationship-Modell einer Liegeplatzverwaltung (nach [Pü94], S.15).

Dabei ist eine Entität eine Grundeinheit, die in der realen Welt ein konkreter Gegenstand, ein Ereignis oder ein Begriff sein kann. Dabei unterscheidet man *strong entities* (starke Entitäten), das sind Entitäten, die allein existenzfähig sind, und *weak entities* (schwache Entitäten), das sind Entitäten, die nur im Zusammenhang mit anderen Entitäten existenzfähig sind.

Eine Beziehung drückt den Zusammenhang zwischen Entitäten aus. Dabei werden mindestens zwei Entitäten durch eine Beziehung miteinander verknüpft. Ein wichtiges Merkmal einer Beziehung ist ihre Kardinalität (1:1, 1:n, m:n). Attributwerte liefern sowohl über Entitäten als auch über Beziehungen Informationen.

3.1.1.2 Schlüsselattribute

Entitäten können entweder durch einzelne Attribute oder durch Gruppen von Attributen eindeutig identifiziert werden. Diese Attribute heißen *Schlüsselattribute* oder *Schlüssel*. Gibt es kein eindeutig identifizierendes Attribut (oder keine Gruppe), so muß ein künstliches Attribut als Schlüssel eingeführt werden (z.B. eine „Zuordnungsnummer“). Existieren mehrere mögliche Schlüsselattribute, so muß ein sogenannter *Primärschlüssel* ausgewählt werden. Dieser soll stets minimal sein.

[13] Meist kurz: E/R-Modell bezeichnet.

3.1.2 Umsetzung in Tabellen

Nach der Erstellung eines *Entity-Relationship-Modells* kann dieses problemlos in ein Relationenmodell umgesetzt und implementiert werden. Dabei kann etwa von folgender Zuordnung ausgegangen werden:

E/R-Modell	Relationenmodell
Entität	Tabelle
n:m-Beziehung	Tabelle mit Fremdschlüsseln
1:n-Beziehung	Fremdschlüssel
Attribut	Spalte

Tabelle 3-1: Umsetzung vom E/R-Modell in das Relationenmodell.

3.2 Benutzerrechte im Datenbanksystem

Das Datenbanksystem DB2/2 gestattet die detaillierte Vergabe von Benutzerrechten, die in Tabelle 3-2 aufgeführt werden. Dabei umfaßt die jeweils höhere Berechtigung alle untergeordneten Berechtigungen. Berechtigungen können für folgende Datenbankobjekte vergeben werden:

- Datenbanken
- Tabellen
- Sichten
- Indizes
- Packages[14]

Für Tabelle 3-2 gilt, daß die Benutzerrechte die links stehen, die weiter rechts stehenden Rechte beinhalten. So hat z.B. ein SYSADM[15] die gleichen Rechte wie ein DBADM[16], aber er kann darüber hinaus alle Benutzer des Status DBADM verwalten. Ein Benutzer

[14] Package = Zugriffsplan. Enthält die Zugriffe, die ein Optimizer aufgrund der Gegebenheiten der Datenbank für die SQL-Befehle eines Programms auswählt.

[15] SYSADM = Systemadministrator.

[16] DBADM = Datenbankadministrator, Datenbankverwalter

mit Status DBADM hat alle Rechte auf Tabellen, Packages und Sichten. Dagegen hat ein Nutzer, der lediglich Sichten bearbeiten darf, keine Rechte Tabellen zu ändern.

SYSADM			
	DBADM		
		CREATETAB (Datenbank)	
		BINDADD (Datenbank)	
		CONNECT (Datenbank)	
		CONTROL (Index)	
		CONTROL (Package)	
			BIND
			EXECUTE
		CONTROL (Tabelle)	
			ALTER
			DELETE
			INDEX
			INSERT
			REFERENCES
			SELECT
			UPDATE
		CONTROL (Sicht)	
			DELETE
			INSERT
			SELECT
			UPDATE

Tabelle 3-2: Struktur der Berechtigungen unter DB2/2[17]

Wird nun, wie in den vorliegenden Programmen, eine Client/Server Anwendung mit Hilfe der Funktionalitäten eines Datenbanksystems erstellt, so benötigt der Softwareentwickler die Stufe DBADM (Datenbankadministrator), um die notwendigen Operationen durchführen zu können.

[17] Vgl. [Pür94], S.308.

Er benötigt sowohl Zugriff auf Tabellen als auch auf Sichten, er muß Packages binden und anlegen oder auch (wie in der Beispielanwendung) Tabellen anlegen können. DBADM gilt dabei immer nur für eine Datenbank, was nicht ausschließt, daß ein Benutzer auch DBADM mehrerer Datenbanken sein kann. Als DBADM kann der Benutzer DB2/2-Kommandos absetzen, Datenbankwerkzeuge starten und Berechtigungen für seine Datenbank und ihre Objekte vergeben.
In DB2 für MVS existieren noch weitere Abstufungen für Benutzerrechte. Das ist einerseits auf eine andere physische Struktur zurückzuführen, andererseits wird in einem PC-Netzwerk keine so umfangreiche Arbeitsteilung wie in einem Großrechnersystem vorliegen.

3.3 Client/Server-Funktionalitäten eines DBMS

DB2/2 unterstützt die Aufteilung einer Anwendung in eine Anwendungskomponente und eine Datenkomponente. Es fungiert dabei als Daten-Server. Auch der Zugriff auf einen Mainframe-Server wird ermöglicht. Dazu steht das Paket DDCS/2 (*DDCS = Distributed Database Connection Services*) zur Verfügung. DDCS/2 ist eine eigenständige Software. Es basiert auf DRDA (*DRDA = Distributed Relational Database Architecture*[18]). Für den Client ist dabei die Lokalisation des Servers transparent. DB2/2 unterstützt OS/2-, Windows- und DOS-Clients. Der DB2/2 Datenbankserver verarbeitet eingehende Anforderungen synchron.
Der Zugriff auf die entfernte Datenbank erfolgt über die Verteilungsschnittstelle des *Remote Database Access, RDA.* Der Client kommuniziert mit der Datenbank über ein API, welches lokal zur Verfügung steht.

3.4 Katalogisierung von Datenbanken

Damit ein Client auf den Datenbestand eines Datenbankservers (oder auch auf lokale Datenbanken) zugreifen kann, müssen einige Einträge in den sogenannten Systemkatalogen vorgenommen werden. Das kann direkt mit den entsprechenden

[18]DRDA ist ein Regelwerk zur Kommunikation zwischen Datenbanksystemen. Es wurde von IBM herausgegeben.

Befehlen im Befehlszeilenprozessor geschehen oder im Tool „*Database Director*“, welches im Paket DB2/2 enthalten ist.
Wichtige Punkte sind hierbei:

- Einrichten eines Nutzers mit den entsprechenden Werten
- Eintragung des fernen Knotens (Servers)
- Katalogisieren der (fernen) Datenbanken

3.4.1 Benutzerprofilverwaltung (UPM)

Alle Benutzerprofile für DB2/2 werden im *User Profile Management* eingerichtet und gepflegt. Dieses UPM existiert außerhalb von DB2/2 und wird bei der Installation angelegt, sofern es nicht bereits unter OS/2 existierte. Es können dort auch die Benutzerrechte für OS/2 verwaltet werden. Meldet sich ein Nutzer an, so wird seine Kennung und sein Paßwort durch das UPM überprüft und bei Korrektheit werden diese Angaben an den Datenbank Manager weitergegeben.
Es muß stets erst eine Anmeldung erfolgen, bevor mit dem Datenbanksystem gearbeitet werden kann.
Innerhalb einer Client/Server-Umgebung kann die Benutzerkennung auf Client und Server unterschiedlich sein. Ist dies der Fall, so muß erst eine Anmeldung auf dem Client erfolgen, bevor dann eine weitere Anmeldung auf dem Server durchgeführt werden kann. Eine einheitliche Benutzerkennung vereinfacht dies, da die Werte automatisch weitergegeben werden und so nur eine einzige Anmeldung durchgeführt werden muß.

DB2/2 geht stets davon aus, daß es mindestens einen Benutzer der Stufe SYSADM gibt. Die durch den Hersteller eingerichteten Standardwerte hierfür sind: „Benutzer“ und „Kennwort“ bei der ersten Anmeldung. Dieses Standard-Login sollte aus Sicherheitsgründen möglichst schnell entfernt und durch ein neues ersetzt werden.
Bild 3-2 zeigt einen Ausschnitt aus der Benutzerprofilverwaltung.

Abb. 3-2: Ausschnitt aus dem *UPM* mit einem Beispielnutzer.

3.4.2 Eintragung eines fernen Knotens

Damit der Client-Rechner seine Anfragen präzise weiterreichen kann, werden alle fernen Knoten in einem „Knotenverzeichnis“ katalogisiert. Auch hier besteht die Möglichkeit, dies durch die Eingabe von Befehlen (z.B. DBM CATALOG NetBIOS NODE nodename. - für NetBIOS Protokoll) zu realisieren. Andererseits kann der Administrator auch alle notwendigen Einträge im Knotenverzeichnis des *Database Directors* in einer graphischen Oberfläche vornehmen.

Für Änderungen im Knotenverzeichnis benötigt man die Stufe SYSADM. Das Knotenverzeichnis wird angelegt, sobald der erste ferne Server katalogisiert wird.

Notwendige Einträge sind:

- Name
- Protokoll
- Service (für TCP/IP)

Den Eintrag über den Service findet man anschließend in der Datei TCP\ETC\Services wieder. Der Server kann stets nur mit seinem Namen eingetragen werden, die Eingabe der *IP*-Adresse (*IP* = *Internet Protocol*) ist nicht möglich. Die Zuordnung des

Hostnamens zu einer *IP*-Adresse erfolgt auf einem zentralen *Nameserver*. Bei kleinen Netzwerken ist es auch möglich, diese Zuordnung lokal in der Datei TCP\ETC\hosts abzulegen, bei großen Netzwerken wird ein zentraler *Nameserver* wegen seines geringeren Wartungsaufwandes bevorzugt. In Abbildung 3-3 wird ein Ausschnitt aus dem Knotenverzeichnis des *Database Directors* wiedergegeben. Es handelt sich dabei um originale Eintragungen des benutzten Clients.

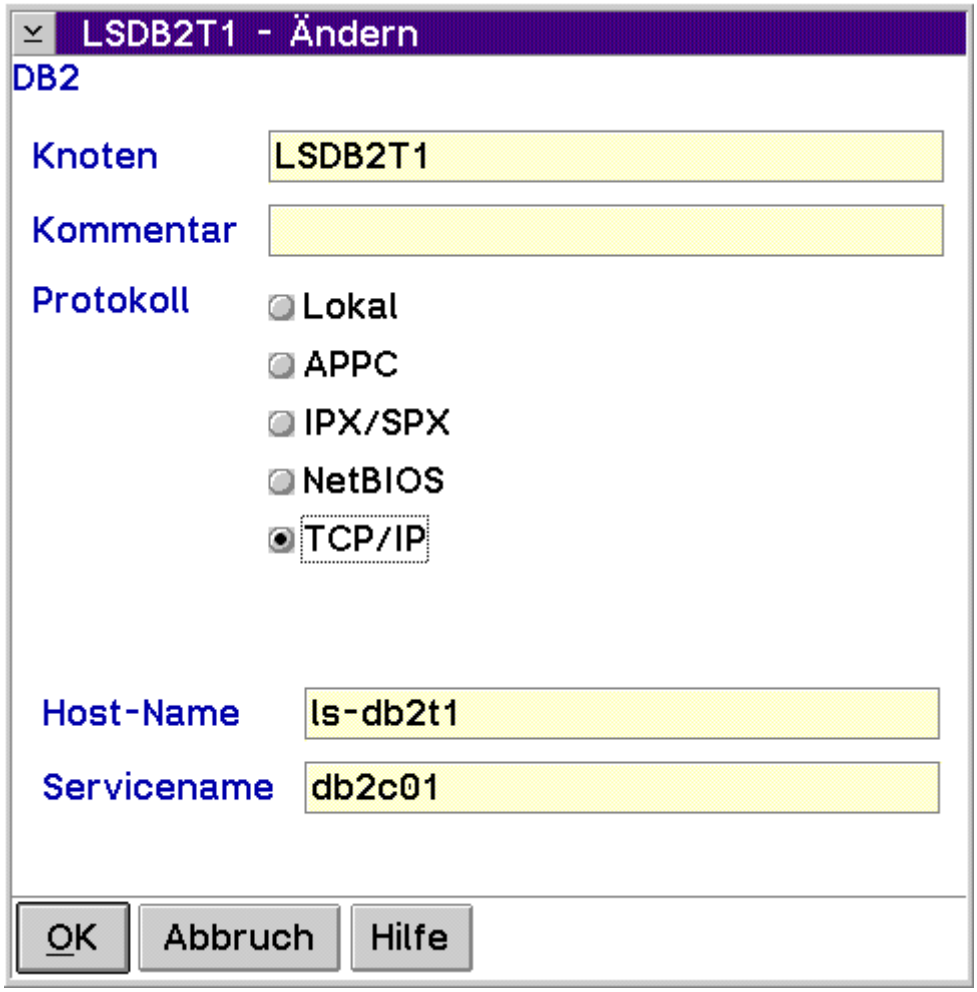

Abb. 3-3: Eintragungen zu einem fernen Knoten unter dem Transportprotokoll TCP/IP.

3.4.3 Systemdatenbankverzeichnis

Weitere Eintragungen müssen im Systemdatenbankverzeichnis vorgenommen werden. Für Änderungen in diesem Verzeichnis benötigt der Benutzer die Stufe SYSADM. Im Systemdatenbankverzeichnis wird jede Datenbank katalogisiert, auf die zugegriffen werden soll. Das Verzeichnis befindet sich auf dem Laufwerk, auf dem auch DB2/2 installiert ist. Es wird angelegt, sobald die erste Datenbank angelegt wird oder der erste Katalogeintrag vorgenommen wird. Das Systemdatenbankverzeichnis enthält folgende Angaben:

Alias	Name zur Referenzierung der Datenbank
Typ	Datenbank auf Server oder lokal
Speicherposition	Laufwerk (lokal) oder Knoten (fern)
Identifikationsüberprüfung	Ort, an dem die Authentifizierung stattfindet
Kommentar	Erläuterungen zur Datenbank

Tabelle 3-3: Katalogeinträge einer Datenbank

Die folgende Detailanzeige zeigt einen Ausschnitt aus dem Systemdatenbankverzeichnis des Clients, der zur Erstellung der Beispielprogramme verwendet wurde.

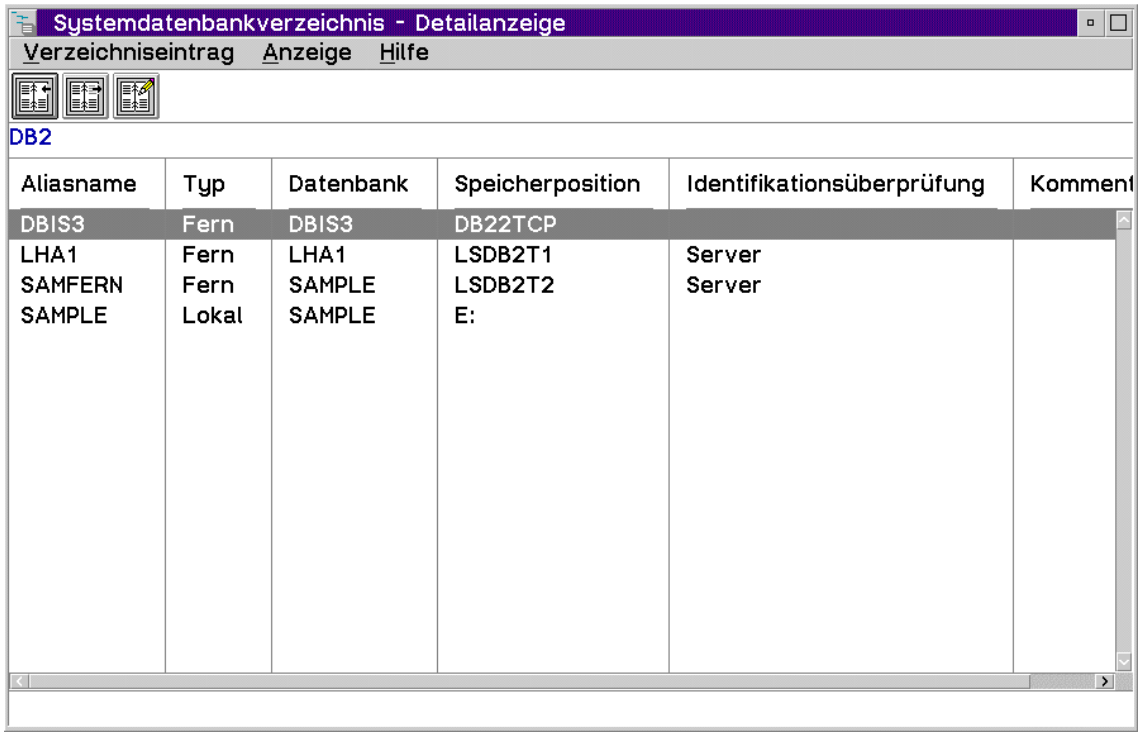

Abb. 3-4: Ausschnitt aus dem Systemdatenbankverzeichnis des Clients.

Die Katalogisierung einer Datenbank kann auch mit dem Befehl „DBM CATALOG DATABASE dbname“ im Befehlszeilenprozessor erfolgen.

Für eine ferne DB müssen zusätzlich folgende Einträge vorgenommen werden:

- Knoten, auf dem sich die Datenbank befindet
- Transportprotokoll
- Ort, an dem die Zugriffsrechte geprüft werden

Diese Eintragungen werden nur dann akzeptiert, wenn der entsprechende Knoten vorher im Knotenverzeichnis ordnungsgemäß katalogisiert wurde. Abbildung 3-5 zeigt diese Eintragungen.

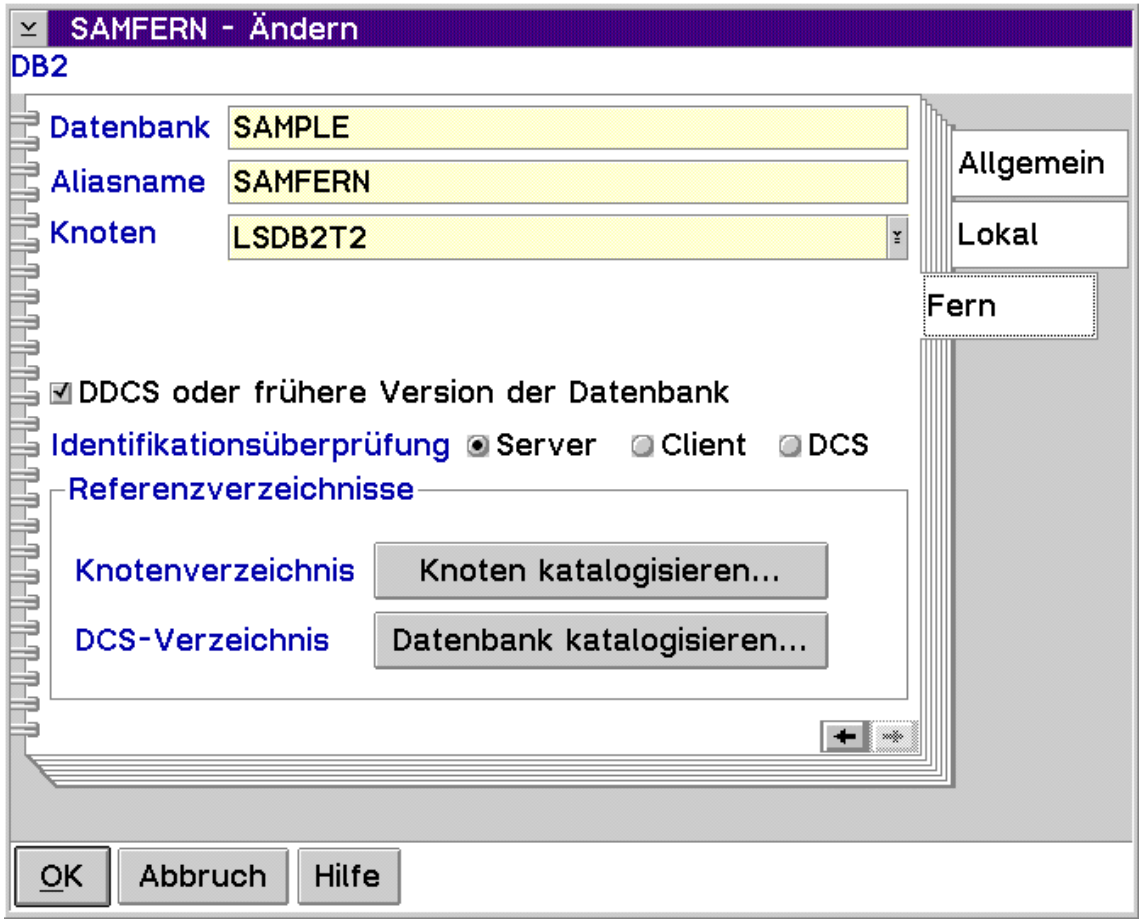

Abb. 3-5: Eintragungen zu einer fernen Datenbank. (im *Database Director*).

3.5 Schematische Darstellung eines Client/Server-Datenbankzugriffs

Nachdem alle Eintragungen erfolgreich durchgeführt wurden und für alle Datenbanken, die bearbeitet werden sollen, die entsprechenden Berechtigungen vorliegen, kann ein Client/Server-Zugriff durchgeführt werden. Dazu wird im Befehlszeilenprozessor erst ein „CONNECT TO <dbname>“ durchgeführt und anschließend die SQL-Abfrage gestartet. Beides kann auch mit dem *Flight-Toolkit*[19] vorgenommen werden. Das *Flight-Toolkit* ermöglicht auch Benutzern ohne *SQL*-Kenntnisse (*SQL = Structured Query Language*), Datenbanken anzusprechen und Anfragen zu stellen. Der Benutzer wird graphisch

[19] Graphisches Abfrage-Werkzeug für DB2/2 Datenbanken, allerdings nicht im DB2/2 enthalten.

geführt. Darüber hinaus gestattet das *Flight-Toolkit* auch, Tabellen anzulegen, Statistiken zu führen und wie gewohnt *SQL-Statements*[20] abzusetzen.

3.5.1 Schematische Darstellung des einstufigen C/S-Betriebes

Im folgenden soll die Abbildung 3-6 deutlich machen, wie das Datenbanksystem anhand der vorgenommenen Katalogisierungen in der Lage ist, eine Anfrage an die entfernte Datenbank zu stellen. Das Bild zeigt, wie die einzelnen Verzeichnisse interagieren.

[20] Datenbankabfrage, die in SQL formuliert wurde.

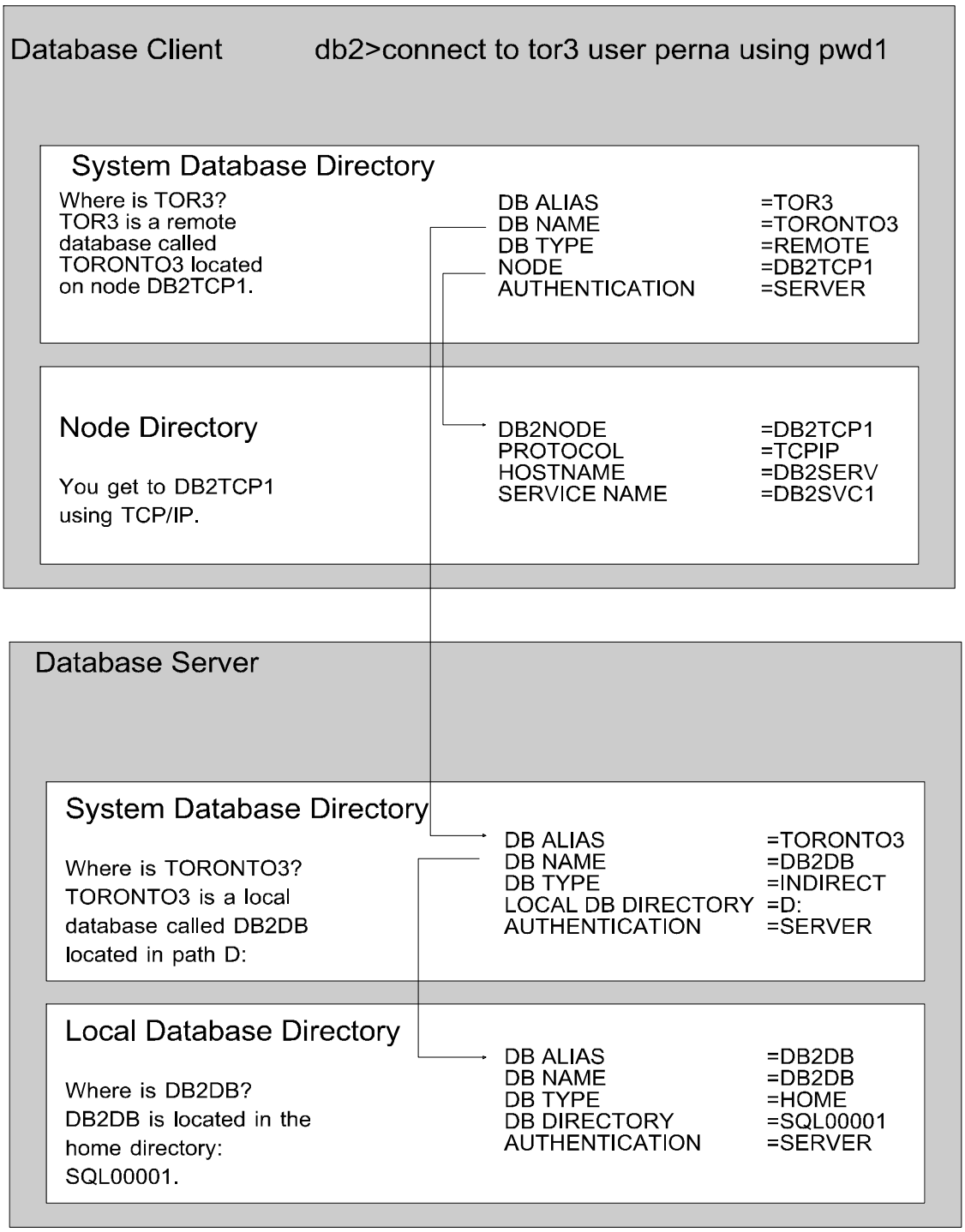

Abb. 3-6: Beispiel für die Interaktion der Datenbankverzeichnisse bei einem Serverzugriff.[21].

Erläuterung:

Die Anfrage wird zunächst an den Client versandt, der prüft, wo sich die Datenbank TOR3 befindet. Er erkennt, daß TOR3 der Alias für TORONTO3 ist, und daß diese Datenbank nicht lokal vorhanden ist. Aufgrund des Knoteneintrags ist er in der Lage, den korrespondierenden Server, auf dem TORONTO3 abgelegt ist, ausfindig zu machen und

[21] Die Abbildung 3-6 wurde dem IBM-Handbuch zu DB2/2 (dem Produkt beigefügt) entnommen.

das richtige Übertragungsprotokoll anzusprechen. Die Anfrage gelangt auf diese Weise an den Server. Dort wird dem Alias TORONTO3 der Datenbankname DB2DB zugeordnet. Diese Datenbank befindet sich auf dem Server auf dessen Laufwerk D:\ im DB2-Verzeichnis SQL00001[22].

Hat der anfragende Client die korrekten Berechtigungen zur Anfrage (die Berechtigungen werden in diesem Beispiel nur auf dem Server überprüft), so werden die entsprechenden Angaben aus der Datenbank gesucht oder Änderungen an der Datenbank vorgenommen, und die Ergebnisse der Transaktion dem Client zurückgesandt.

Diese Art des Server-Zugriffs wurde in einem der Beispielprogramme realisiert.

3.5.2 Mehrstufiger Client/Server-Betrieb über ein Gateway

Ähnlich wie in der einstufigen Client/Server-Architektur, so greifen auch bei der mehrstufigen Architektur die Einträge der System- und Knotenverzeichnisse der Datenbanken ineinander und ermöglichen die Arbeit mit einer fernen Datenbank. Abbildung 3-7 zeigt den Zugriff auf eine *Mainframe*-Datenbank (DB2 für MVS). Der Zugriff wird hier gesteuert über ein Gateway, welches sich auf einem entfernten DB2/2-Server befindet. Die Umsetzung der Protokolle und Anfragen wird dabei von dem Paket DDCS/2 übernommen. Es beinhaltet die DB2-Implementierung der *DRDA-Application-Requesters* (DRDA-AR)-Funktionen. Diese Funktionen unterstützen DB2-Anwendungen bei einem Zugriff auf DRDA-Server, wie z.B.:

- MVS/ESA
- DB2 für VSE und VM
- DB2 für OS/4000

[22] DB2/2 ordnet jeder Datenbank ein Verzeichnis zu, welches fortlaufend numeriert wird, beginnend mit SQL00001.

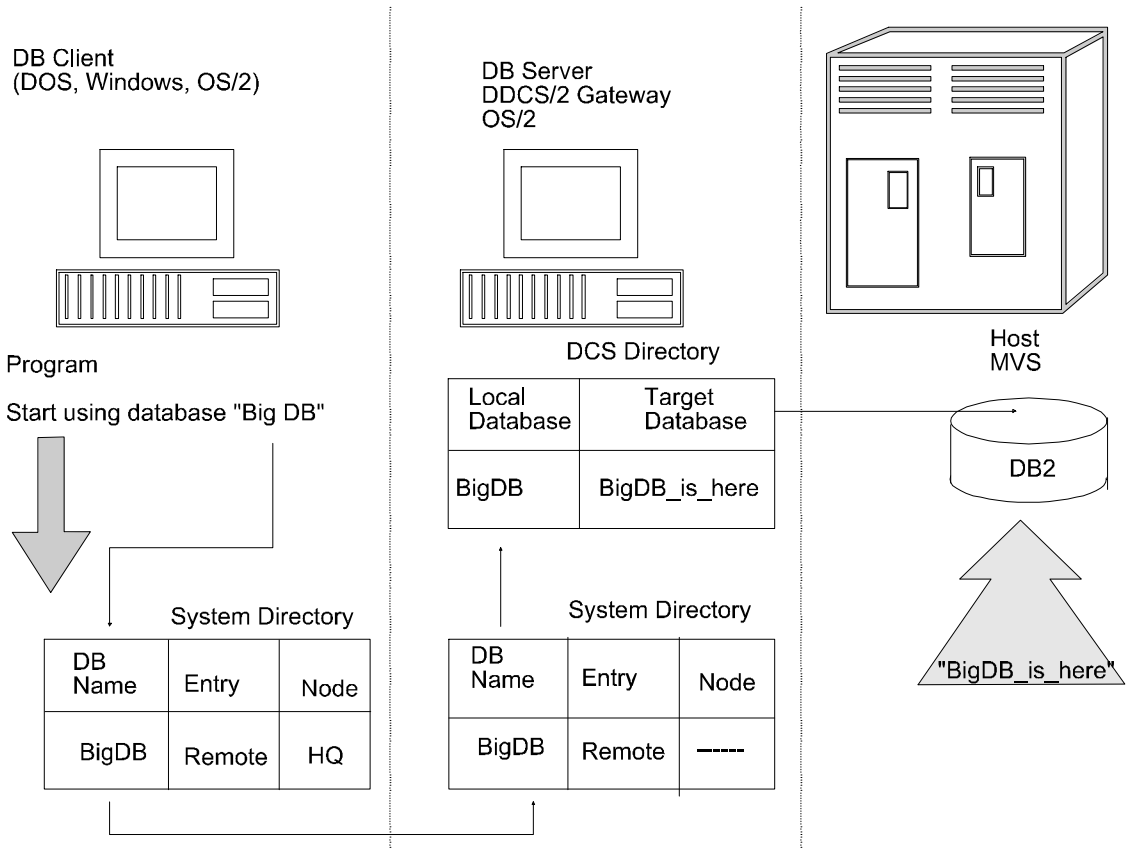

Abb. 3-7: Client/Server-Datenbankanbindung über ein DDCS/2-Gateway[23].

Bis zum Systemverzeichniseintrag des Servers entspricht die Vorgehensweise der des einstufigen Client/Server-Datenbankzugriffs. Der Unterschied besteht darin, daß sich im Systemdatenbankverzeichnis des Servers kein Eintrag über die Lokalität der Datenbank „*BigDB*“ findet. Das zwingt den Server, in das DCS-Verzeichnis (*DCS=Database Connection Services*) zu wechseln und dort nach Einträgen zu suchen. Im DCS-Verzeichnis befinden sich Einträge zu allen relationalen Datenbanken, die über DDCS/2 erreicht werden können. So auch der Eintrag zu „*BigDB*“. DDCS/2 setzt dann die Anfrage um, benutzt ein anderes Protokoll (APPC) und leitet sie an den *Mainframe* weiter.

Diese Art des Zugriffs auf eine Großrechnerdatenbank wurde in einem Beispielprogramm erarbeitet.

[23] Nach [OrfHar93] S.649.

4 Erstellte Client/Server-Anwendungen

Der folgende Abschnitt vermittelt einen Überblick über die im Rahmen der Arbeit erstellten Beispielprogramme. Dazu zählt die Konfiguration der beteiligten Rechner, und die Beschreibung der erstellten Programme sowie die Erläuterung der Vorgehensweise.

4.1 Konfiguration der benutzen Rechner (Soft- und Hardware)

Zur Implementierung der Beispielanwendungen wurde folgende Konfiguration eingesetzt:

4.1.1 Konfiguration des Client-Rechners

Als Client stand ein Personalcomputer mit Intel Pentium Prozessor der Taktfrequenz 100 MHz, einem Hauptspeicher mit 32 MB und einer 1,2 GB Festplatte sowie Netzanbindung zur Verfügung.
Auf dem Client lief das Betriebssystem OS/2 Warp. Als Applikationen waren installiert:

- das Datenbank System DB2/2 2.1 mit dem Client Application Enabler
- das Softwareentwicklungstool Visual Age C++ 3.0 für OS/2[24]
- TCP/IP Version 3.
- Novell Netware Client 4.11
- der Communications Manager 2.1[25]

4.1.2 Konfiguration des Servers

Die Konfiguration des Servers bleibt während der Arbeit völlig transparent. Im speziellen handelt es sich auch hier um einen PC mit Intel Pentium Prozessor, dem Betriebssystem OS/2 Warp, der DB2/2-Serverversion, dem Paket DDCS für OS/2 (als Gateway zum Großrechner), sowie ebenfalls TCP/IP und Communications Manager[26].

---

[24] Siehe Anhang zu Warenzeichen.

[25] Ermöglichte den direkten Zugriff zum Host via 3270-Emulation.

[26] Für LU-Einträge und VTAM-Angaben.

4.1.3 Der Mainframe

Bei dem Großrechner handelt es sich um eine IBM /390-Architektur mit dem Betriebssystem MVS und dem Datenbanksystem DB2. Weitere Anwendungen, die auf dem Großrechner implementiert sind, waren für das Erstellen der Beispielprogramme nicht von Bedeutung. Die folgende Abbildung 4-1 soll die Konfiguration noch einmal zusammenfassend darstellen.

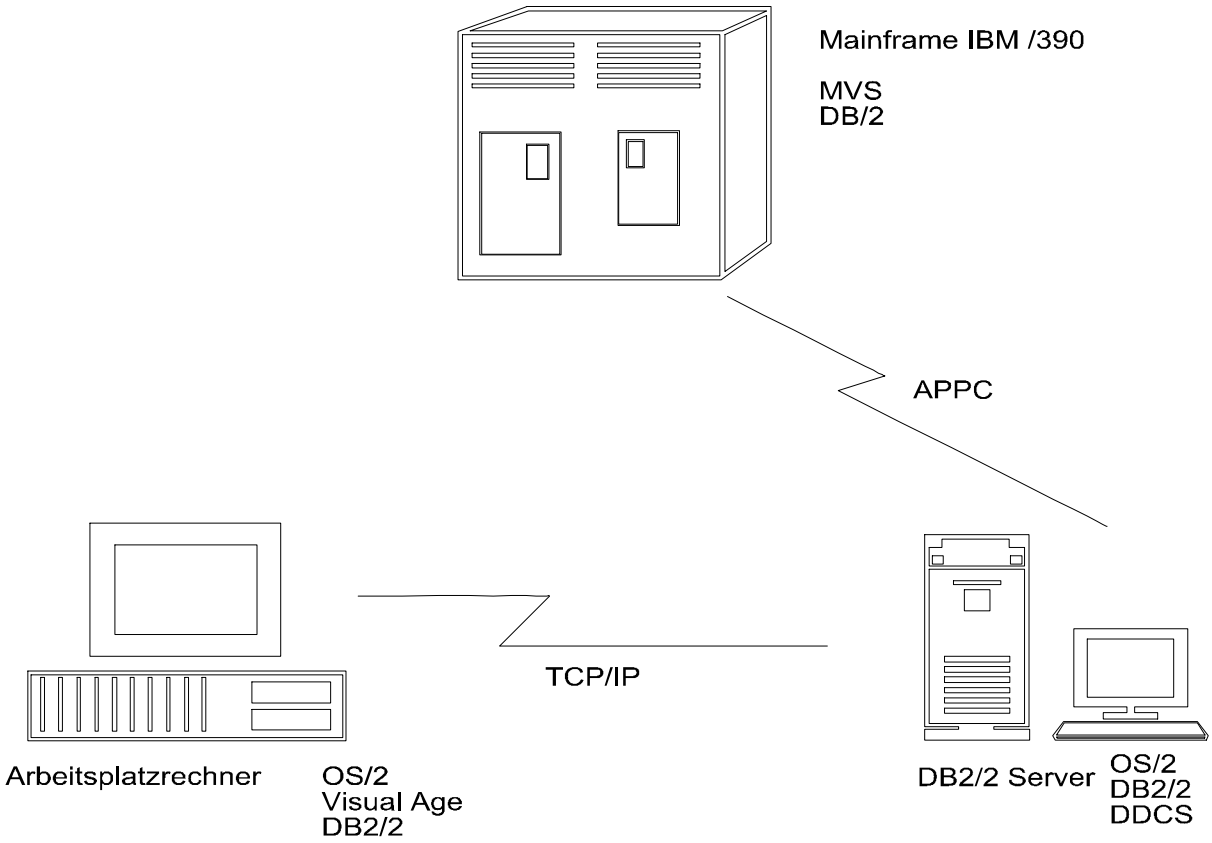

Abb. 4-1: Schematische Darstellung der Konfiguration.

4.2 Die Beispielprogramme

Im Rahmen der vorliegenden Arbeit wurden drei Client/Server-Programme erstellt, die im wesentlichen mehrere „Ausbaustufen“ eines Programmes darstellen.

Der erste Ansatz bestand in einem lokalen Datenbankzugriff, um Fehler bei der Übertragung im Netzwerk auszuschließen. In einer zweiten Stufe wurde auf eine entfernte Datenbank eines DB2/2-Servers zugegriffen, um dann im dritten Schritt eine Datenbank auf dem Großrechner anzusprechen.

4.2.1 Lokaler Datenbankzugriff

In einem ersten Versuch wurde ein Programm in Visual Age C++ komplett visuell erstellt, welches auf eine Datenbank zugreift, die sich lokal auf dem Client befindet. Bei der Datenbank handelt es sich um eine relationale Datenbank in DB2/2, konkret um die Datenbank „DBSAMPLE“, die bei der Installation des Produktes DB2/2 mit angelegt wird. Diese lokale Anwendung schloß etwaige Fehler, die durch die Netzwerkumgebung bedingt wären, aus. Darüber hinaus war die Verwendung einer Datenbank des Herstellers eine Garantie für eine korrekte Datenbank. Fehler, die durch das Anlegen einer inkonsistenten Datenbank hätten entstehen können, wurden somit ausgeschlossen.
Die erstellte Anwendung liest Einträge aus der Tabelle „Employee“. Diese Tabelle ist eine Tabelle der Datenbank DBSAMPLE und enthält z.B. Name, Vorname und Geburtsdatum von Angestellten (Employees). Diese Angaben sind durch einen Schlüssel, der Angestelltennummer (Personalnummer), eindeutig gekennzeichnet.
Während des Programmablaufs wird zunächst eine Verbindung zur Datenbank aufgebaut, dann kann der Anwender eine Personalnummer zur Suche eingeben. Er erhält daraufhin Name, Vorname und Geburtsdatum des entsprechenden Angestellten. Diese Suche kann beliebig oft angestoßen werden.
Ein Test über die Korrektheit der Abfrageergebnisse konnte insofern durchgeführt werden, als daß eine gleichlautende Anfrage entweder im Befehlszeilen-Prozessor oder im *Flight Toolkit* gestartet wurde. Die Ergebnisse wurden miteinander verglichen und stimmten stets überein.

4.2.2 Zugriff auf eine entfernte DB2/2-Datenbank

Die unter 4.2.1 beschriebene Beispieldatenbank DBSAMPLE befand sich auch auf einem DB2/2-Testserver. Das oben beschriebene erste Programm konnte dadurch fast vollständig übernommen werden. Da beide Datenbanken identisch, also auch tabellengleich waren, konnten fast alle Teile der ersten Anwendung auch für den Zugriff auf die entfernte Datenbank benutzt werden.
Dazu wurden zunächst die notwendigen Eintragungen im Datenbanksystem vorgenommen. Im Programm selbst wurde lediglich der Alias der Datenbank ausgetauscht. In der zweiten Anwendung wurden einige weitere Features eingebaut, so

z.B. eine Fehlerbehandlung, eine Menüzeile und Hilfetexte mit Hypertext-Verbindungen. Auch können in diesem Programm alle Feldeinträge automatisch gelöscht werden. Kontrollmechanismen sorgen dafür, daß Abfragen nur nach dem erfolgreichen Aufbau einer Verbindung gestellt werden können. Darüber hinaus befindet sich am unteren Ende des Fensters ein sogenanntes *Information Area* (*Information Area*=Statuszeile), welches den Benutzer über Programmverlauf oder Aktionen informiert. Außerdem gibt es *FlyOver* Hilfen.[27]

Die korrekte Funktion dieses Programmes wurde wie unter 4.2.1 durch parallele Abfragen getestet. Darüber hinaus wurden an der lokalen Datenbank einige Einträge abgeändert, damit überprüft werden konnte, ob das vorliegende Programm auch tatsächlich auf die fernen Daten zugreift. Dies konnte bestätigt werden. Das erstellte Programm lieferte die unveränderten Daten aus der fernen Datenbank. Die veränderten Einträge der lokalen Datenbank wurden nur vom vorangegangenen Programm angezeigt. Abbildung 4-2 zeigt das Fenster der Anwendung 2 nach einer erfolgreichen Abfrage.

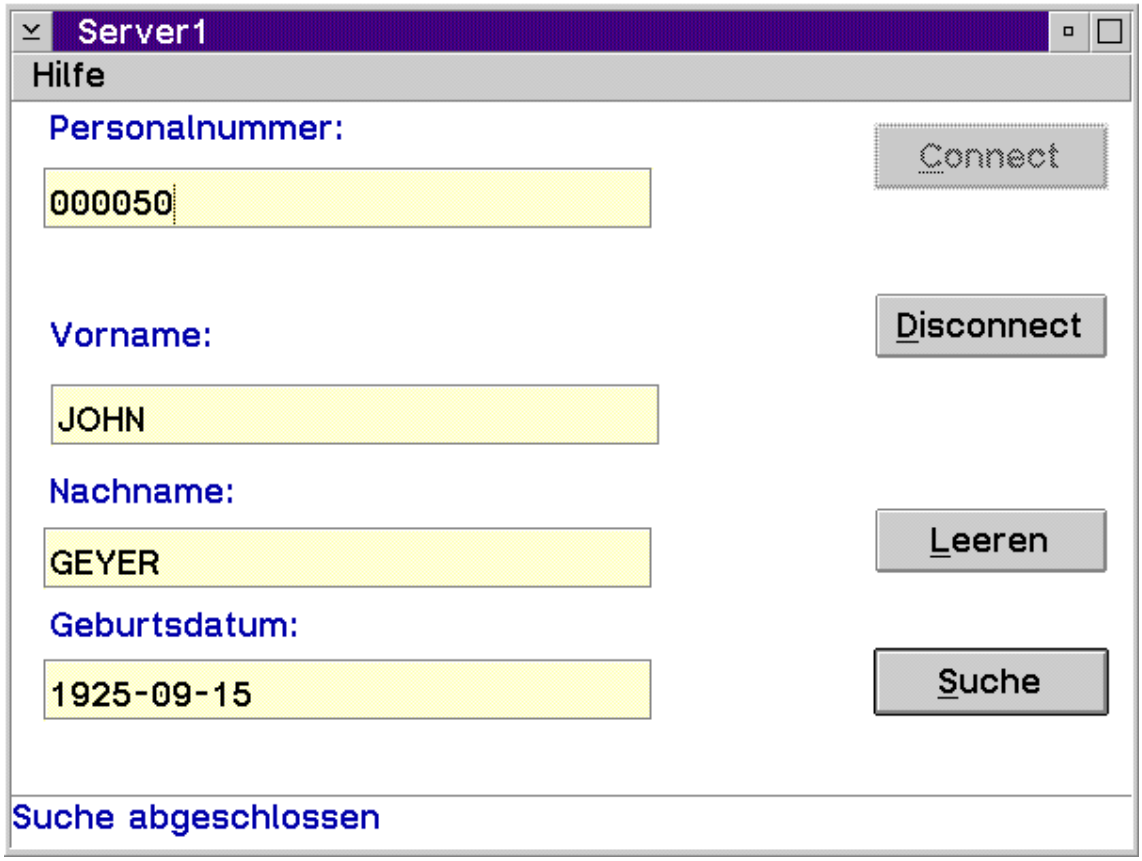

Abb. 4-2: Fenster der zweiten Ausbaustufe der Anwendung; Serverzugriff.

[27] *FlyOver*= kurzer Hinweis, der z.B. über die Funktionen eines Schalters informiert, auf den der Mauszeiger zeigt.

4.2.3 Zugriff auf eine Mainframedatenbank

In der dritten Ausbaustufe des vorliegenden Programmes wurde auf eine DB2-Datenbank unter MVS zugegriffen.

Dazu mußte eine neue DLL für den Zugriff auf die Daten erstellt werden, da es sich um eine völlig andere Datenbank handelte. Die Datenbank repräsentiert einen Ausschnitt aus dem Testdatenbestand und enthält pro Tabelle etwa 1000 Einträge. Die angelegten Tabellen entsprechen den im Einsatz befindlichen Tabellen des zentralen Exkassos (Projekt „ZEx“). Dabei wurde in diesem speziellen Fall nicht auf eine Tabelle, sondern auf Sichten zugegriffen, die jedoch jeweils mit einer Tabelle übereinstimmten.

Das Programm arbeitet - abweichend von den vorangegangenen - bei der Suche nach Einträgen nicht mit einem Schlüsselattribut. Gesucht wird nach der Journalordnungsnummer. Diese Ordnungsnummer ist jeweils für einen Kunden zutreffend und kann bei mehreren Sätzen[28] gleich sein. Das Abfrageergebnis ist somit nicht der Wert einzelner Felder, sondern eine Menge von Sätzen der Sicht. Diese werden in einem sogenannten „Container“ abgelegt, der alle gefundenen Sätze anzeigt. Dabei wird in der Beispielanwendung jedoch nicht jeder Satz vollständig angegeben, sondern nur die relevanten Felder.

Die Struktur eines Containers ermöglicht die variable Darstellung der Ergebnisse. Mit seinen verschiebbaren Balken bietet er auch großen Datenmengen Platz.

Ein weiterer Vorteil dieser Suche: Bei der Eingabe einer nicht vorhandenen Journalordnungsnummer, bekommt der Anwender keinen *SQL-Error* (*SQL-Error*=Fehlermeldung des Datenbanksystems). Dies passiert jedoch bei den beiden anderen Anwendungen, da dort die Abfrage auf ein Schlüsselattribut ausgerichtet ist.

Diese Anwendung zeigt, daß es möglich ist, mit Hilfe der Client/Server-Funktionalitäten des Datenbanksystems DB2/2 auf die Datenbestände einer relationalen Mainframedatenbank zuzugreifen. Bisher erfolgte allerdings nur ein lesender Zugriff, was durch die Arbeit mit Sichten bedingt ist. (Sichten sind normalerweise nur lesbar, aber nicht veränderbar.)

[28] Beispielsweise mehrere Kontenbewegungen für einen Kunden.

Auch bei dieser Anwendung wurde eine Fehlerbehandlung eingefügt, ein Menü, eine Hilfedatei sowie eine Statuszeile und *FlyOver*-Hilfe.

Des weiteren wird in diesem Programm kein Anmelden an der Datenbank explizit gefordert, sondern die Anwendung baut beim Start eine Verbindung zur DB2 Zieldatenbank auf. Die dabei mitgegeben Werte (Benutzerkennung und Paßwort) werden direkt aus dem *UPM* mitgegeben und erfordern damit nur eine einmalige Anmeldung am lokalen Datenbanksystem. Erst wenn eine Verbindung hergestellt ist, wird das Fenster der Anwendung für den Benutzer sichtbar, und er kann sofort mit der Eingabe beginnen.

Die korrekte Arbeitsweise dieser Anwendung wurde getestet. Dazu wurde eine gleichlautende Abfrage direkt in der Terminalemulation unter TSO/ISPF an die DB2 für MVS-Datenbank gestellt. Die Auswahl der Journalordnungsnummern aus den 1000 Einträgen erfolgte willkürlich. Ein Vergleich der Ergebnisse brachte in allen Fällen eine Übereinstimmung. Somit kann von einer ordnungsgemäßen Arbeitsweise des vorliegenden Programms ausgegangen werden.

Zur Unterstützung der Erläuterungen zeigt Abbildung 4-3 das Arbeitsfenster der Anwendung:

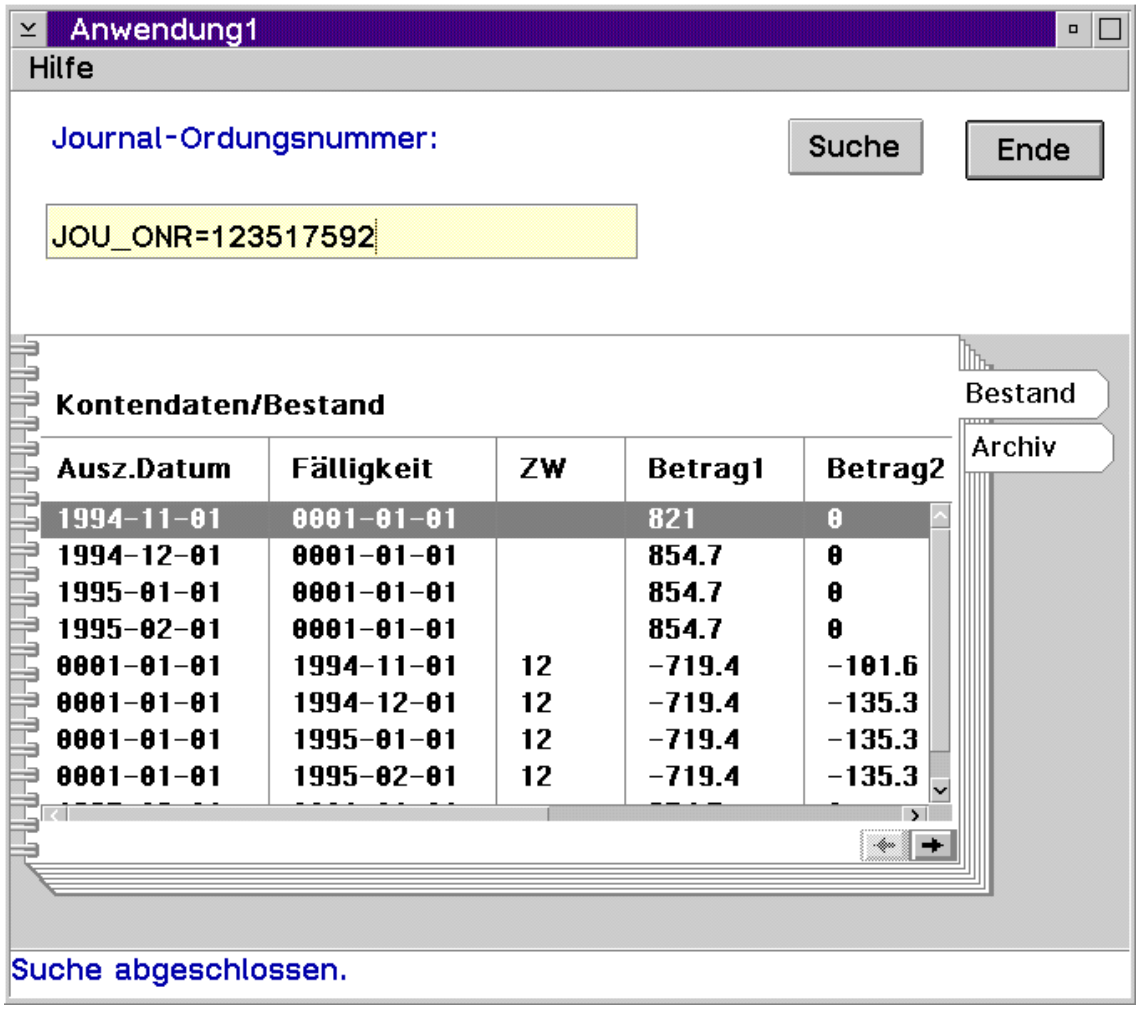

Abb. 4-3: Zugriff auf Mainframedaten in der dritten Anwendung.

5. Programmierumgebung und Werkzeuge

Wenn man Client/Server Programme mit Visual Age C++ und DB2/2 entwickelt, müssen einige Anpassungen vorgenommen werden. Das folgende Kapitel beschreibt die Einstellungen, die dazu notwendig sind. Es umfaßt darüber hinaus eine Erläuterung der Werkzeuge, die dem Entwickler dabei zur Verfügung stehen. Es wird ebenso auf die visuelle Programmierung eingegangen. Probleme, die während der Programmierung auftraten, werden erörtert.

Diesem Kapitel liegen die Erfahrungen aus der Erstellung der in Kapitel 4 beschrieben Anwendungen zugrunde. Die Reihenfolge der Abschnitte spiegelt die einzelnen Schritte zur Implementierung dieser Client/Server-Anwendungen wieder.

Alle Programme, die mit Visual Age C++ erstellt werden, bearbeitet man in sogenannten Projekten. Daher beschäftigt sich der erste Abschnitt speziell mit Projekten und der „Projektzentrale“ - dem *WorkFrame.*

5.1 Arbeiten mit Projekten - der Workframe

Zur Erzeugung von ausführbaren Programmen (.exe-Dateien) oder Programmbibliotheken (.dll-Dateien) öffnet man unter Visual Age C++ ein sogenanntes Projekt. Projekte können aus „*Project Smarts*“ geöffnet werden oder aus dem „*Project Template*“.

5.1.1 Projekte aus dem Ordner „Project Smarts“

Projekte, die unter „Project Smarts“ erstellt werden, sind Projekte die bereits einige spezielle Einstellungen entsprechend des angestrebten Ziels enthalten. Öffnet man den Ordner „*Project Smarts*“ im Ordner *Visual Age*, so erhält man eine Übersicht der verschiedenen, bereits vorgefertigten Projektumgebungen. So kann man beispielsweise ein Projekt erstellen, daß für die Entwicklung einer *Presentation-Manager*-Anwendung[29] genutzt werden soll.

[29] *Presentation Manager*=objektorientierte, graphische Arbeitsoberfläche von OS/2.

Zu jedem vorgefertigten Projekttyp gibt es eine kurze Erläuterung. Wählt man einen Projekttyp aus, so wird automatisch ein Ordner mit dem entsprechenden Namen auf der Arbeitsoberfläche angelegt. Wird dieser geöffnet, so erscheint der *WorkFrame* (die „Projektzentrale“) im sogenannten *Icon-view*[30]. (*Icon-view*=Übersicht über Dateien, aber ohne spezifische Dateiangaben, wie z.B. Dateigröße etc.) Das so erzeugte Projekt enthält nun schon Dateien und wichtige Einstellungen, die z.B. für eine PM-Applikation benötigt werden (*PM=Presentation Manager*). Diese Einstellungen wurden von den vorgegebenen Projekten geerbt. Einige Einstellungen sind darüber hinaus vom Entwickler zusätzlich anzugeben: z.B. das Verzeichnis, in dem sich die Quelldateien befinden, oder der Name des Zielobjekts.

5.1.2 Projekte aus dem „Project Template“

Projekte können ebenso durch ein „Ziehen“ der „*Project Template*“-Schablone aus dem *Visual Age* Ordner oder aus dem Schablonen-Verzeichnis erzeugt werden. Dazu wird die Projekt-Schablone mit der rechten Maustaste gewählt und auf die Arbeitsoberfläche gezogen. Es entsteht dort ein Projekt-Ordner. Dieser enthält alle unter Visual Age verfügbaren Projekt- und Datei-bezogenen Aktionen. Zum Entwickeln einer Anwendung müssen dann weitergehende Einstellungen als bei den vorgefertigten Projekten vorgenommen werden.

[30] Weitere mögliche Varianten der Dateidarstellung im *WorkFrame*: *Details-view* und die Baumstruktur, *tree.*

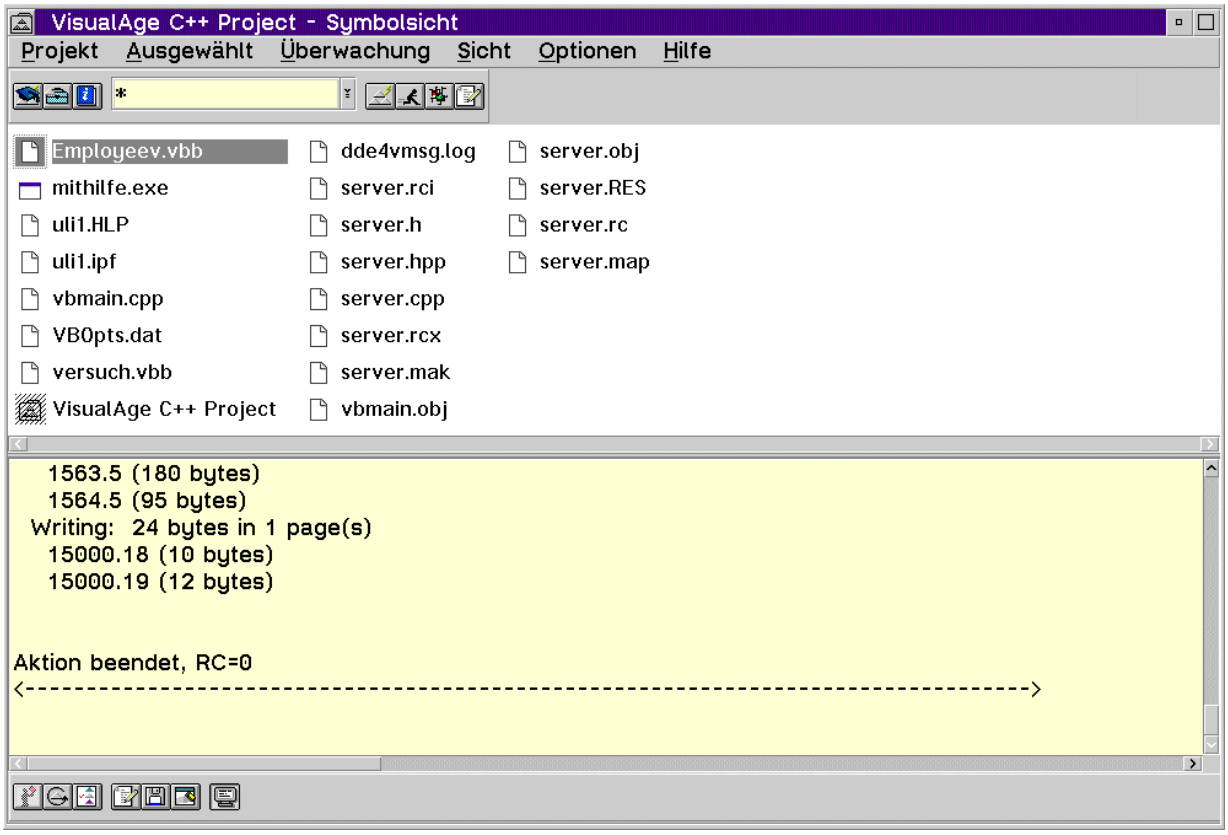

Abb. 5-1: WorkFrame während der Erzeugung des zweiten Anwendungsprogramms.

Abbildung 5-1 zeigt die Entwicklung einer Anwendung im *WorkFrame*. Es handelt sich dabei um Dateien des zweiten Anwendungsprogramms (DB2/2-Server Anbindung) mit dem Projektziel einer ausführbaren Datei. Der *WorkFrame* ist hier im „*Icon-view*“ abgebildet.

5.1.3 Einstellungen im WorkFrame

Im *WorkFrame* können Einstellungen zum Projekt vorgenommen werden, dazu gehören z.B.:

- Angaben zum Elternprojekt (Unterpunkt „Vererbung“)
- Name der Zieldatei des Makefiles (unter „Ziel“)
- Name des Makefiles (unter „Ziel“)
- Arbeitsverzeichnis (bei „Position“)

Der folgende Ausschnitt aus diesen Projekteinstellungen zeigt die Seite, auf der die Angaben zur Vererbung eingestellt werden können. Das vorliegende Projekt erbt seine Einstellungen vom mitgelieferten *IBM-MainProj*. Das ist ein Projekt, in dem alle Werkzeuge eingebunden sind und keine Projekt-Einstellungen vorgegeben sind. Es ist das allgemeinste Projekt.

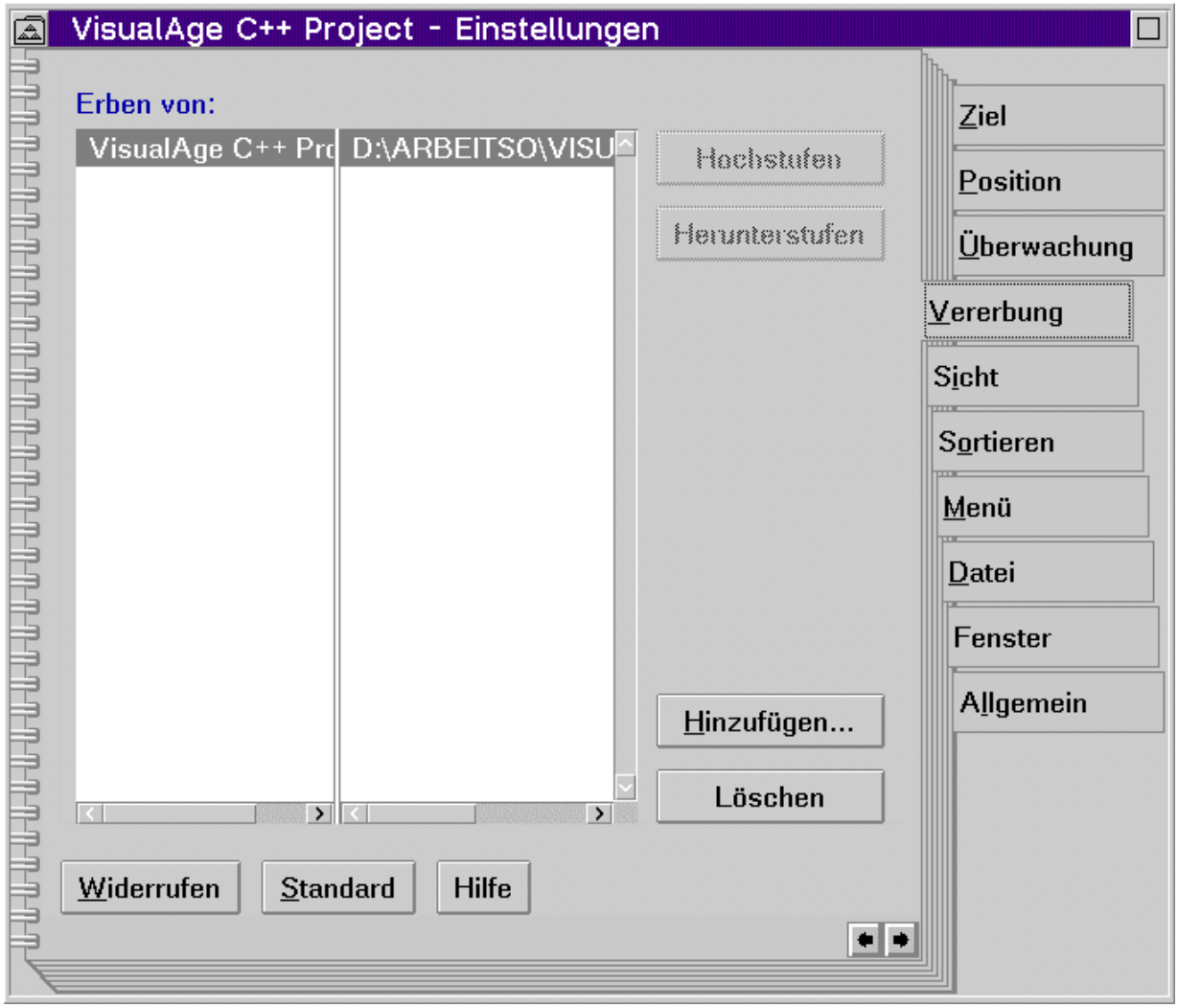

Abb. 5-2: Ausschnitt aus dem „*Project-Setup*“ des *Workframes.*

Im *WorkFrame* können auch Einstellungen zu den Werkzeugen eingetragen werden. Es wird bestimmt, welche Werkzeuge zu verwenden sind. Hier können sowohl Werkzeuge von *Visual Age*, als auch „fremde“ Werkzeuge (z.B. andere Editoren) angegeben werden. Der *WorkFrame* gestattet es zudem, Einstellungen an den einzelnen Werkzeugen vorzunehmen, z.B. Compileroptionen. Diese Werte ändert man im sogenannten „*Tools*-Setup“ (*Tools*=Werkzeuge). Abbildung 5-3 zeigt einen Teil des *Tools*-Setups.

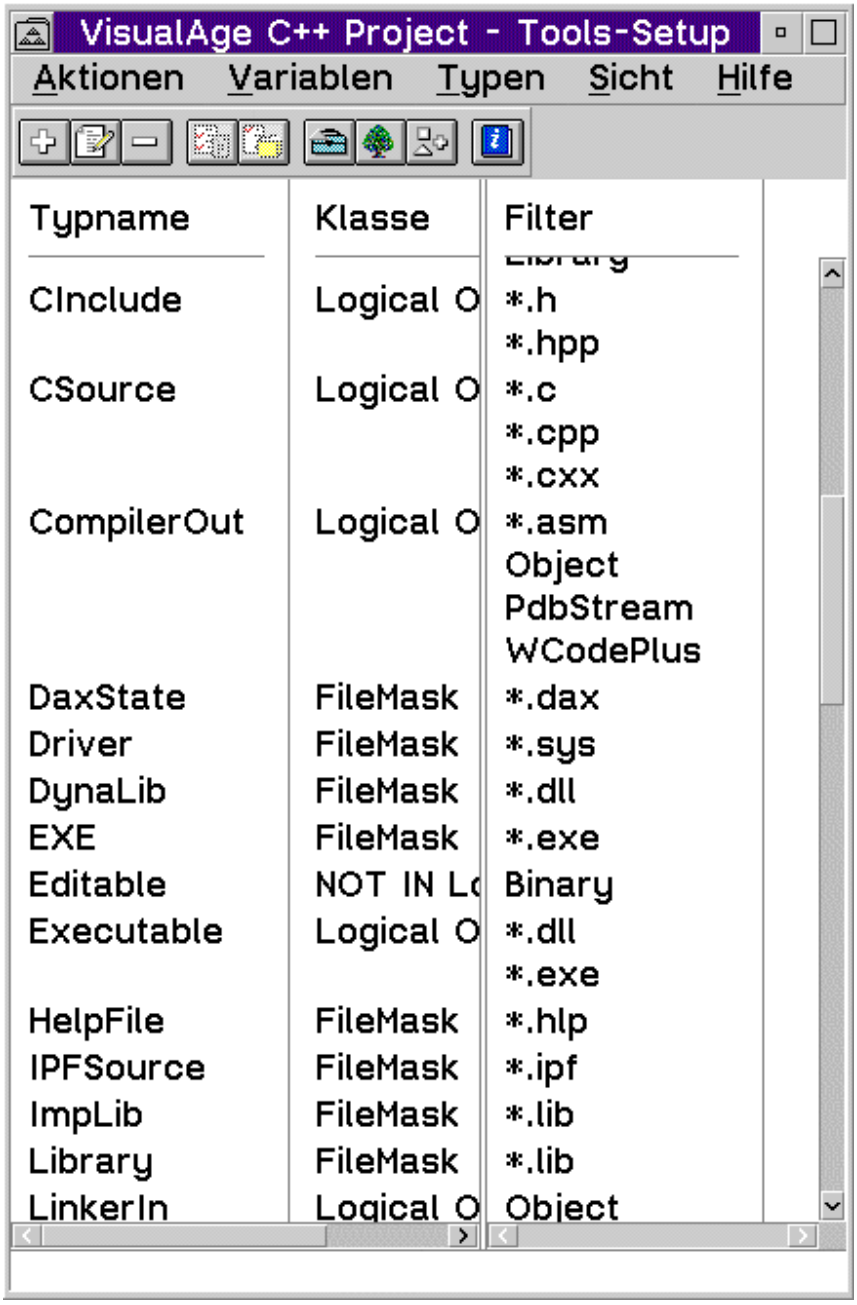

Abb. 5-3: Ausschnitt aus dem Tools-Setup.

5.1.4 Vererbung von Projektinformationen

Unter den beiden vorangegangenen Punkten wurde bereits erwähnt, daß Eigenschaften eines Projekts auf ein anderes Projekt übertragen werden können. Dieser Vorgang wird entsprechend der üblichen Terminologie der objektorientierten Anwendungsentwicklung als

„Vererbung“ bezeichnet. Wesentlich ist dabei, daß die Eigenschaften des „Kindprojekts“ stets vom „Elternprojekt“ abhängig bleiben. Das heißt: werden Eigenschaften im „Elternprojekt“ geändert, so ändern sich auch diese Werte in allen abhängigen Projekten. Darüber hinaus ist es möglich, Einstellungen verschiedener Projekte zu kombinieren, also von mehreren Projekten zu vererben.
Die Angaben zu den Projekten, aus denen geerbt werden soll, erfolgen im *WorkFrame* im Menü „Sicht“, Unterpunkt „Einstellungen“ (siehe dazu Abb. 5-2).

5.2 Entwicklung einer dynamischen Bibliothek für den Datenbankzugriff

5.2.1 Vorgehensweise

Die folgenden Abschnitte zeigen, welche Schritte notwendig sind, um einen Datenbankzugriff auf DB2/2 zu implementieren. Es wird davon ausgegangen, daß der Datenbankzugriff nur einen Teil der Applikation darstellt. Er wird durch eine DLL (*DLL=Dynamic Link Library*, dynamische Bibliothek) realisiert. Dabei wird einem Datenbankobjekt mit Hilfe von Werkzeugen eine sogenannte Klasse zugeordnet. Diese wird für die objektorientierte, visuelle Programmierung verwendet.
Nach den Erläuterungen zum Anlegen eines Projekts, wird das Werkzeug „*Data Access Builder*“ beschrieben, das die Umsetzung von relationalen Datenbanktabellen in objektorientierte Klassen übernimmt. Danach wird die Generierung von Quellcode beschrieben, sowie das Erzeugen der eigentlichen Bibliothek und das Binden der Anwendung an die Datenbank.
Da das Generieren von Klassen mit dem mitgelieferten *Data Access Builder* nur für DB2/2 Datenbanken möglich ist, befaßt sich ein weiterer Abschnitt mit dem *Enhanced Data Access Builder* und dem Generieren von Klassen für DB2-Datenbankobjekte.

5.2.2 Anlegen des Projekts

Zuerst wird, wie unter Punkt 5.1.2 beschrieben, ein Projekt angelegt. Hier kann auf ein Projekt zurückgegriffen werden, daß speziell für die Erstellung eines Datenbankzugriffs

eingerichtet ist. Es befindet sich in „*Project Smarts*“. Bild 5-4 zeigt *Project Smarts* mit dem entsprechenden Projekt.

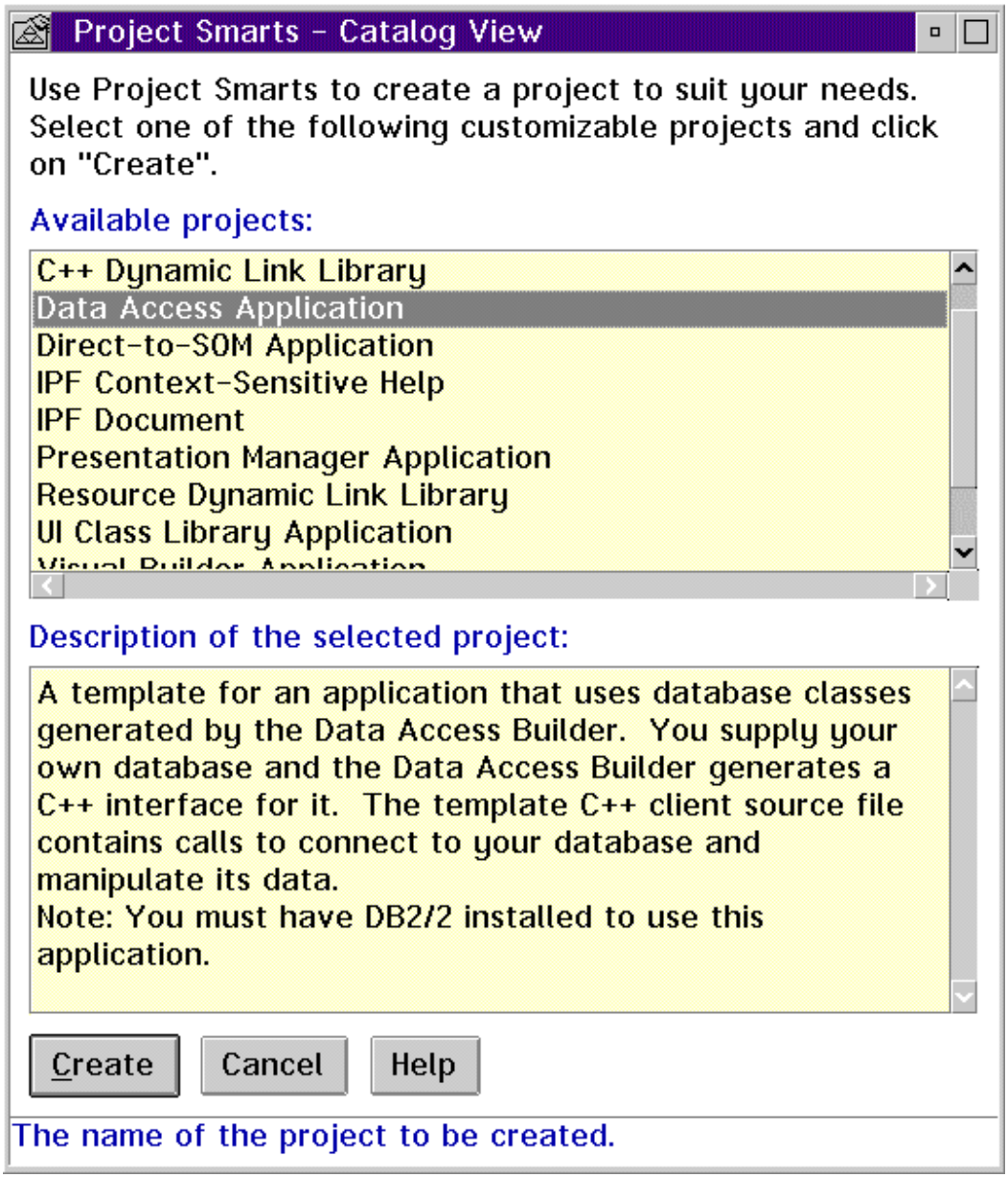

Abb. 5-4: Hauptfenster von „*Project Smarts*“ mit vorgefertigten Projektumgebungen.

Trotz bereits vorhandener Einstellungen muß hier aber von Hand eingegeben werden, daß eine DLL erzeugt werden soll. Wichtig ist auch die Angabe des Arbeitsverzeichnisses. Es sollte mit dem Arbeitsverzeichnis des *Data Access Builders* übereinstimmen. Zur besseren

Übersicht empfiehlt es sich außerdem, für jedes Datenbankobjekt ein eigenes Projekt anzulegen.

Während der Arbeit an diesem Projekt sollte lokal eine Anmeldung im UPM erfolgt sein. Außerdem ist der korrekte Zugriff zur entfernten Datenbank zu gewährleisten, da während des Compilierens eine Verbindung zur Zieldatenbank aufgebaut wird, und dort sogenannte *Packages*[31] abgelegt werden. Das ist ohne eine Anmeldung und entsprechende Zugriffsrechte nicht möglich, der Vorgang würde abgebrochen.

5.2.3 Der Data Access Builder

Der *Data Access Builder* ist ein Werkzeug, das es gestattet, zu einem Objekt (Tabelle, Sicht) einer relationalen Datenbank eine objektorientierte Klasse zu generieren. Bei dieser Umsetzung werden den Spalten einer Tabelle oder Sicht die Attribute eines Objekts zugeordnet. Die typischen Datenbankanfragen, wie *select*, *insert* (*add*), *delete* oder *update*[32], stellen dann die Methoden der neu erzeugten Objekte dar. Diese Objekte (Klassen genannt) können dann in der weiteren Anwendungsentwicklung wie andere, bereits vorhandene Objekte bearbeitet werden.

Es gibt zwei Möglichkeiten den *Data Access Builder* zu starten. Einerseits kann er aus dem Projekt heraus aufgerufen werden (aus dem Menü des *WorkFrame*) oder aber aus dem allgemeinen *Visual Age C++ Tools*-Ordner. Wird der *Data Access Builder* aus einem Projekt heraus gestartet, so gibt es keine Möglichkeit, ein *makefile*[33] mitzugenerieren. Das muß dann durch die Option „*makemake*[34]“ im Projekt geleistet werden, was sich als problematisch herausgestellt hat. Wird dagegen der *Data Access Builder* direkt aus dem *Tools*-Ordner aufgerufen, so wird auch ein *makefile* erzeugt.

---

[31] Vgl. Abschnitt 3.2 dieser Arbeit.

[32] suchen, einfügen, löschen, ändern

[33] Das sogenannte *makefile* ist eine Datei, die Informationen über die zu compilierenden und zu bindenden Dateien sowie deren Abhängigkeiten enthält. Es beinhaltet die Compileroptionen und die Reihenfolge der aufzurufenden Werkzeuge.

[34] Diese Funktion erstellt automatisch ein *makefile*.

5.2.3.1 Generieren der Klassen

Bevor der *Data Access Builder* geöffnet wird, muß im Kontextmenü[35] das entsprechende Arbeitsverzeichnis eingestellt werden. Es sollte das gleiche sein, das im Projekt angegeben wurde (siehe 5.2.2). Beim Start des *Builders* erscheint im Startmenü eine Auswahl, ob neue Klassen zu generieren sind, oder bereits generierte Klassen wieder bearbeitet werden sollen. Abbildung 5-5 zeigt diese Ausgangssituation.

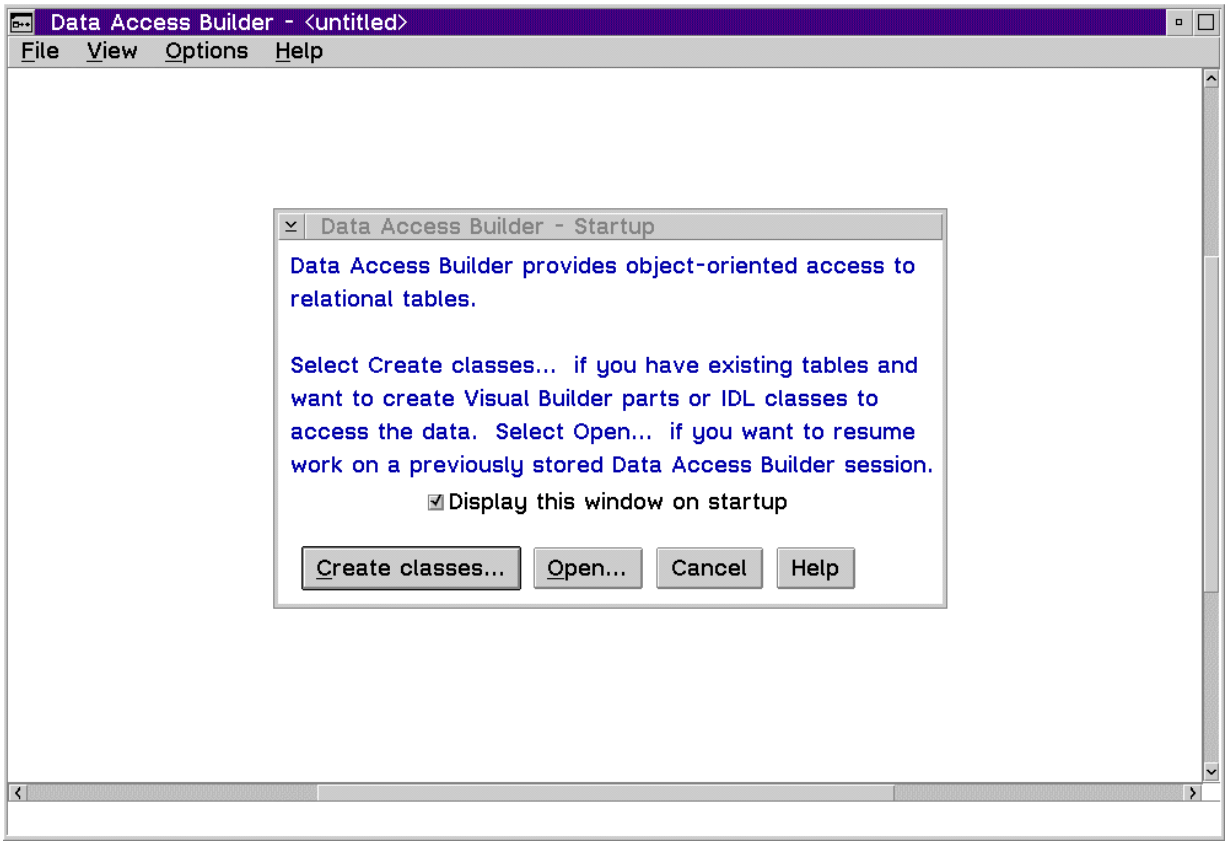

Abb. 5-5: Startmenü des *Data Access Builders.*

Ist „Klassen generieren“ gewählt, so stellt der *Data Access Builder* automatisch eine Übersicht aller Datenbanken auf, die im Systemdatenbankverzeichnis der lokalen DB2/2

[35] Wie unter OS/2 üblich, wählt man auch im *Visual Age* das Kontextmenü eines Objekts mittels rechter Maustaste aus. Darauf wird in den folgenden Abschnitten nicht mehr verwiesen, sondern generell nur noch vom Kontextmenü gesprochen.

Installation eingetragen sind[36]. Die entsprechende Datenbank kann markiert werden und der *Data Access Builder* bereitet eine Liste sämtlicher in dieser Datenbank enthaltenen Tabellen und Sichten auf.

Aus diesen Datenbankobjekten wiederum werden die ausgewählt, zu denen Klassen erstellt werden sollen. Für ein übersichtlicheres Arbeiten bietet es sich an, jeweils immer nur für ein Datenbankobjekt Klassen zu generieren und abzulegen.

Die folgende Grafik zeigt den *Data Access Builder* und eine Übersicht aller Datenbanken, auf die zugegriffen werden kann. Es handelt bei der markierten Datenbank um die Datenbank, die im zweiten erstellten Anwendungsprogramm benutzt wurde. Da noch keine Verbindung aufgebaut ist, sind in Abbildung 5-6 keine Angaben zu den Objekten der Datenbank zu finden.

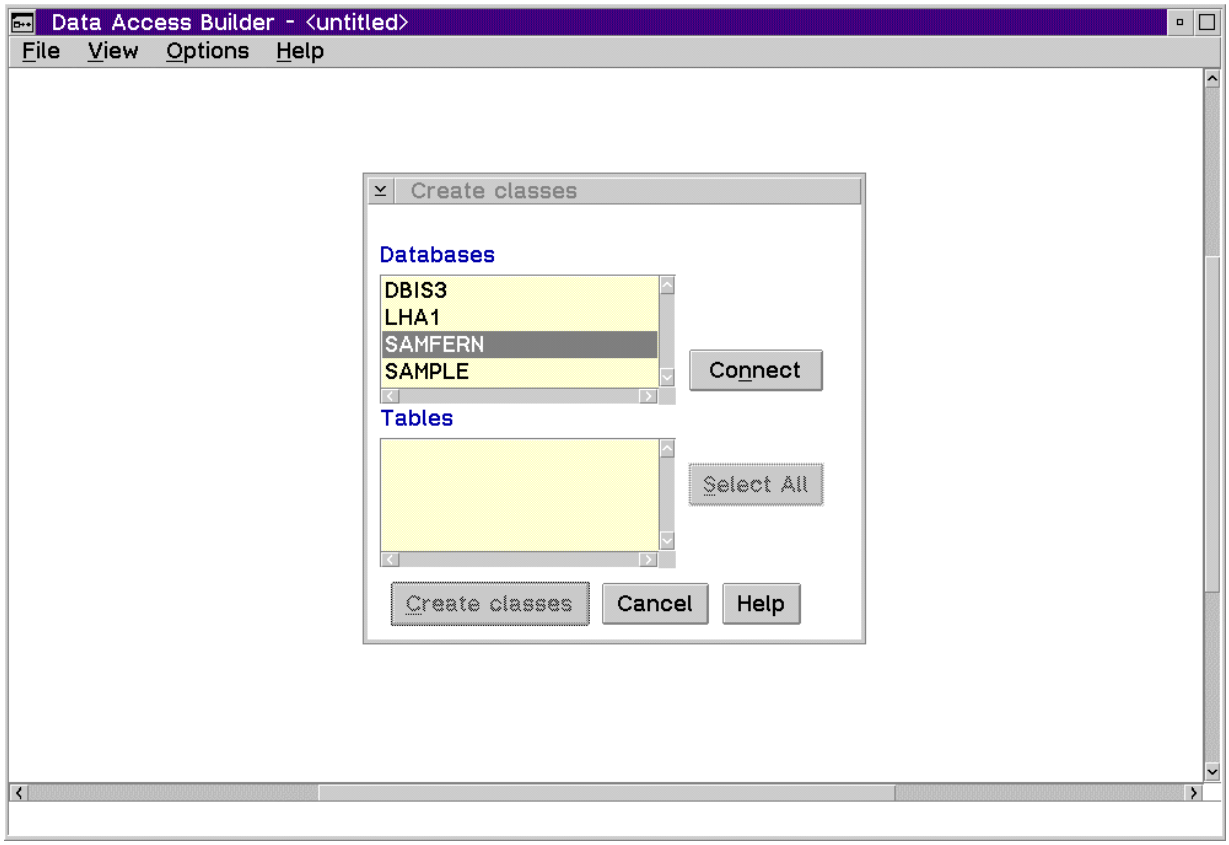

Abb. 5-6: Der *Data Access Builder* - Übersicht über die Datenbanken.

Nach dem Auswählen der Datenbank und einem erfolgreichen Verbindungsaufbau wird die gewünschte Sicht oder Tabelle gewählt und dazu erscheint eine graphische Darstellung des

[36] Vgl. Abschnitt 3.4.3 dieser Arbeit.

Datenbankobjekts sowie das Symbol für eine Klasse (eine Kugel). Über das Kontextmenü des Datenbankobjekts können nun Einstellungen vorgenommen werden, wie z.B. Spalten weglassen oder den *Data Identifier*[37] ändern. Entspricht das veränderte Datenbankobjekt den Wünschen, können jetzt daraus die Klassen generiert werden (Kontextmenü der „Kugel“). Es können sowohl *Visual Builder Parts*[38] (wie hier im Beispiel) als auch *SOM*-Klassen (*SOM = System Object Module*[39]) oder beide generiert werden.

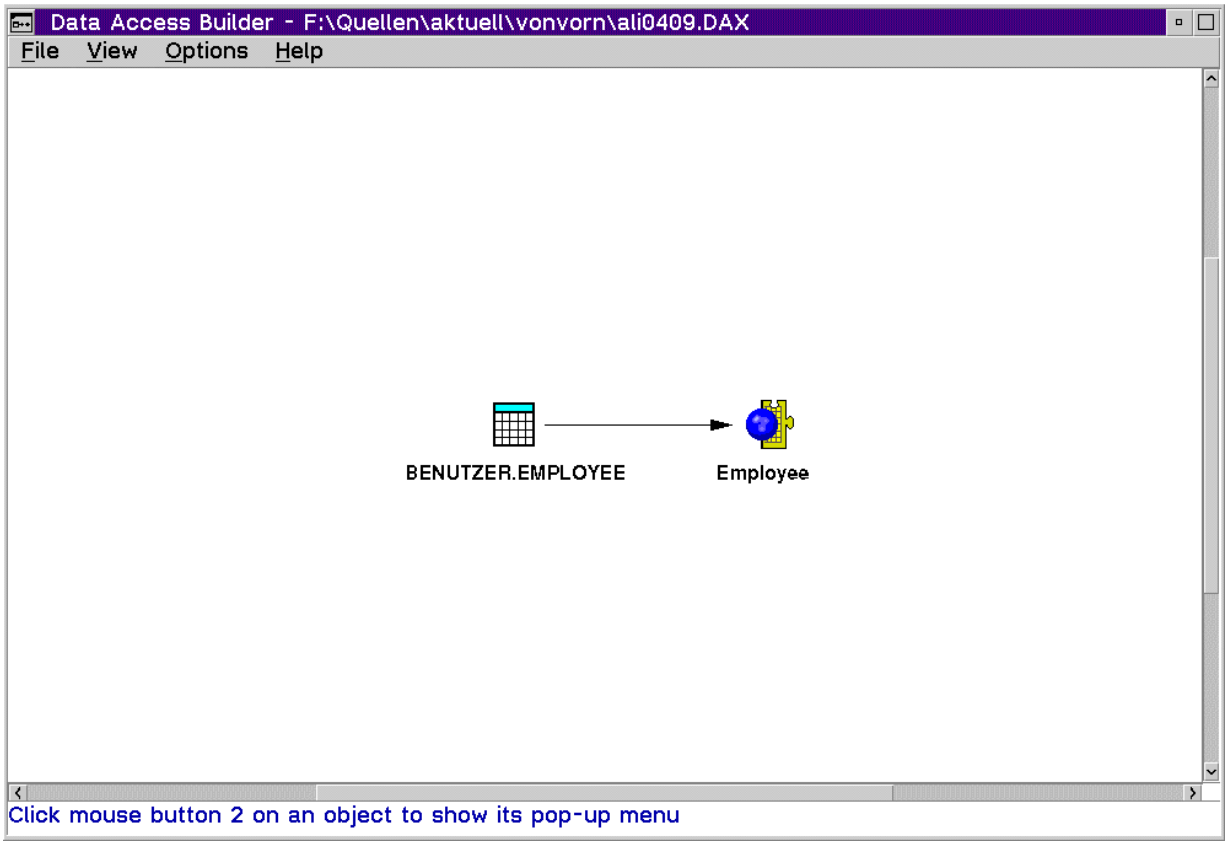

Abb. 5-7: Der *Data Access Builder*. Zuordnung einer Klasse zu einem Datenbankobjekt.

[37] Bezeichnung für das Schlüsselattribut.

[38] Objekte, die im Werkzeug „*Visual Builder*“ verwendet werden sollen.

[39] SOM ist eine spezielle Technologie, sprachunabhängige Objekte zu erzeugen und zu bearbeiten. SOM-Objekte können sowohl von prozeduralen, als auch von objektorientierten Programmiersprachen benutzt werden. SOM ist eine Implementierung der CORBA Vorgaben durch die IBM. (siehe dazu [OrfHar93] S.981 ff)

In Abbildung 5-7 ist zu sehen, daß für die Datenbanktabelle „Employee“ bereits eine Klasse angelegt und Quellcode erzeugt wurde (Puzzleteil hinter der Kugel). Die Farbe des Puzzelteils (gelb) läßt erkennen, daß hier *Visual Builder Parts* generiert wurden.
Es entstanden sechs Dateien mit folgenden Endungen:

- sqc - Die Datei mit dieser Endung enthält das *Embedded*[40] *SQL* für die Datenbankoperationen (*SQL = Structured Query Language*)
- cpp - legt die Klasse an (Quelldatei in C++)
- hpp - *Header*-Datei zu cpp
- def - gibt an, welche Dateien zur Erstellung der *DLL* noch benötigt werden (*includes* aus DB2/2)
- mak - Diese Datei ist das generierte *makefile*
- vbe - enthält Informationen für den *Visual Builder*, sogenanntes *Part Information File*

5.2.3.2 Eigenschaften und Verwendung

Zu jeder Tabelle bzw. jeder Sicht sind nun 2 Klassen entstanden. Das ist einerseits die Basisklasse, zum anderen die Manager-Klasse. Sie ermöglichen unterschiedliche Arten des Zugriffs auf die Tabelle oder die Sicht. Mit der Basisklasse kann jedes Attribut getrennt angesprochen werden, und sie enthält die Methoden *update, insert* (*add*), *delete* und *retrieve*[41]. Mit dieser Klasse können einzelne Zeilen einer Tabelle herausgesucht oder modifiziert werden. Innerhalb einer Zeile kann ein ganz bestimmtes Attribut abgefragt oder geändert werden.
Die Manager-Klasse erlaubt dagegen eine uneindeutige Abfrage[42], d.h. es ist möglich mehrere Sätze auszulesen (z.B.: alle Versicherungen zu einer bestimmten Kundennummer, falls ein Kunde mehrere Versicherungen hat). Diese Klasse bietet die Methode *select* und darüber hinaus einige Grundmethoden der *Visual Builder* Klassen an.

---

[40] Embedded=eingebettet Das bedeutet: SQL-Anweisungen sind in eine andere Programmiersprache eingefügt (z.B. C) und werden von einem sogenannten Precompiler erkannt und vorübersetzt.

[41] *Update*, insert (add) und *delete* sind nur möglich, wenn nicht mit Sichten gearbeitet wird. Sichten sind hingegen meist nur lesbar und erlauben daher nur die Methode *retrieve*.

[42] Z.B.: Abfrage nach einem Nicht-Schlüssel-Attribut, welches auch den Wert Null annehmen kann.

Die bei der Generierung der Klassen entstandenen Dateien werden für die Erzeugung der dynamischen Bibliothek im *WorkFrame* verwendet.

5.2.4 Erzeugen der *DLL*

Die genannten Dateien werden im Projekt im *WorkFrame* zu einer *DLL* in mehreren Zwischenschritten zusammengesetzt. Alle Anweisungen und Einstellungen sind im generierten *makefile* vorhanden, so daß nur der Befehl „*make*“ abgesetzt werden muß.
Während dieses Vorgangs wird sowohl eine .bnd Datei (*Bindfile*) für die Anbindung an die Datenbank erstellt, als auch ein *Package* mit Zugriffspfaden, welches direkt in der Zieldatenbank abgelegt wird. Die Optionen im *Makefile* dazu sind /B für das Erzeugen des *Bindfiles* und /P für das Erstellen des *Package*. Es ist auch möglich, diese Einstellungen so abzuändern, daß zunächst nur eine .bnd - Datei erzeugt wird und die Anwendung später an die entsprechende Datenbank gebunden wird. Das Anlegen eines solchen *Package* ist notwendig, um später mit der Datenbank arbeiten zu können.
Zum Ablegen des *Package* wird eine Verbindung zur Datenbank aufgebaut. Ist eine solche Verbindung nicht möglich, wird abgebrochen. Das kann verschiedene Ursachen haben, so ist z.B. das lokale Datenbanksystem nicht gestartet oder der Programmierer nicht mit den korrekten Daten angemeldet, die auch auf dem fernen Rechner Gültigkeit haben.

5.2.5 Arbeiten mit der DLL

Die auf diese Weise erzeugte *DLL* muß nun in ein Verzeichnis kopiert werden, auf das in der Datei *config.sys* unter dem Kommando LIBPATH verwiesen wird. (Eine Erweiterung dieses Pfades ist genauso möglich.) Eine weitere Änderung ist unter dem Befehl INCLUDE in der Datei *config.sys* vorzunehmen, so daß später der *Visual Builder* auf die entstandenen Dateien zugreifen kann. Natürlich können auch hier umgedreht der INCLUDE-Pfad beibehalten und die aktuellen Versionen der Dateien in ein entsprechendes Verzeichnis kopiert werden.

Die nachfolgenden Ausschnitte aus der Datei „*config.sys*“ sind Angaben, die während der Erarbeitung der Beispielprogramme verwendet wurden.

```
...
LIBPATH=F:\IBMCPP\DLL;X:\vacpp\IBMCPP\DLL;F:\IBMCPP\SAMPLES\TOOLKIT\
DLL;E:\SQLLIB\DLL;E:\SQLLIB\ALT;E:\SQLLIB\FUNCTION;D:\IBMCOM\DLL;.;D:\
MUGLIB\DLL;D:\OS2\DLL;D:\CMLIB\DLL;D:\OS2\MDOS;D:\;D:\OS2\APPS\DLL;D:\I
BMNVDM2\DLL;D:\NW41OS2;D:\NW41OS2\NLS\DEUTSCH;E:\TCPIP\DLL
...
INCLUDE=X:\vacpp\IBMCPP\INCLUDE;X:\vacpp\IBMCPP\INCLUDE\OS2;X:\vacpp\I
BMCPP\INC;X:\vacpp\IBMCPP\INCLUDE\SOM;D:\MUGLIB;E:\SQLLIB\INCLUDE;F:\
quellen\aktuell\vonvorn;\F:\quellen\aktuell;F:\quellen\vbprj\host1;F:\quellen\vbprj\host2
...
SET
LIB=X:\vacpp\IBMCPP\LIB;X:\vacpp\IBMCPP\DLL;F:\IBMCPP\DLL;D:\MUGLIB;E:\S
QLLIB\LIB;
...
SET FINCLUDE=E:\SQLLIB\INCLUDE
SET VBPATH=.;X:\vacpp\IBMCPP\DDE4VB
SET LPATH=X:\vacpp\IBMCPP\MACROS
...
```

Abb. 5-8: Ausschnitte aus der config.sys, Einstellungen der Pfade.

Sind diese Schritte ausgeführt und ein Zugriff auf die erzeugte DLL und die anderen benötigten Dateien gewährleistet, so muß nun die Anwendung an die Datenbank gebunden werden.

5.2.6 Binden der Anwendung an die Datenbank

Die Anbindung an die Datenbank erfolgt im lokalen Datenbanksystem. Im Client-Konfigurations-Verzeichnis können die Datenbanken auf entfernten Knoten eingesehen werden und ein Schalter „Unterstützung“ ermöglicht die Auswahl der *.bnd*-Dateien

(*Bindfiles*), die an die jeweilige Datenbank gebunden werden sollen. Für alle Datenbankobjekte, auf die später im Programm zugegriffen werden soll, ist so zu verfahren. Für diesen Vorgang ist auf der Zieldatenbank die Stufe „*DBADM*“ Voraussetzung. Das Binden der Dateien kann auch über ein DB2/2-Kommando im Befehlszeilenprozessor ausgeführt werden.

5.2.7 Enhanced Data Access Builder

Da sich die Dateistruktur von DB2/2 und DB2 für MVS grundlegend unterscheidet, ist das *Mappen* (Ableiten) von Objekten auf Großrechnern nicht mit dem normalen *Data Access Builder* möglich. Der *Data Access Builder* versucht, eine Übersicht[43] sämtlicher in der Datenbank vorhandenen Tabellen und Sichten zu geben. Allein für die im Beispiel angesprochene Testdatenbank ist die Anzahl dieser Datenbankobjekte so groß (mehrere tausend Tabellen und Sichten), daß es für das Tool nicht mehr handhabbar ist und es zum Systemabsturz kommt. Das Problem besteht darin, daß es keine Möglichkeit gibt, die Datenbanken, auf die man zugreifen möchte. genauer zu spezifizieren. So kann man im *Data Access Builder* keine *Database* angeben, so wie sie in DB2 vergeben werden. Ein Eintrag des Erzeugers (*creators*) der gesuchten Objekte ist nicht möglich.
Mit dem im Herbst 1996 auf den Markt gekommenen „*Enhanced Data Access Builder*“, der eine Einschränkung bei der Suche nach DB-Objekten ermöglicht, kann nun auch problemlos auf Großrechnerdatenbanken unter DB2 zugegriffen werden. Für die vorliegende Beispielanwendung wurde mit einer Beta-Version dieses Tools gearbeitet. Die Abbildung 5-9 zeigt die einstellbaren Filterinformationen des *Enhanced Builders.*

[43] Wie unter 5.2.3.1 in dieser Arbeit beschrieben.

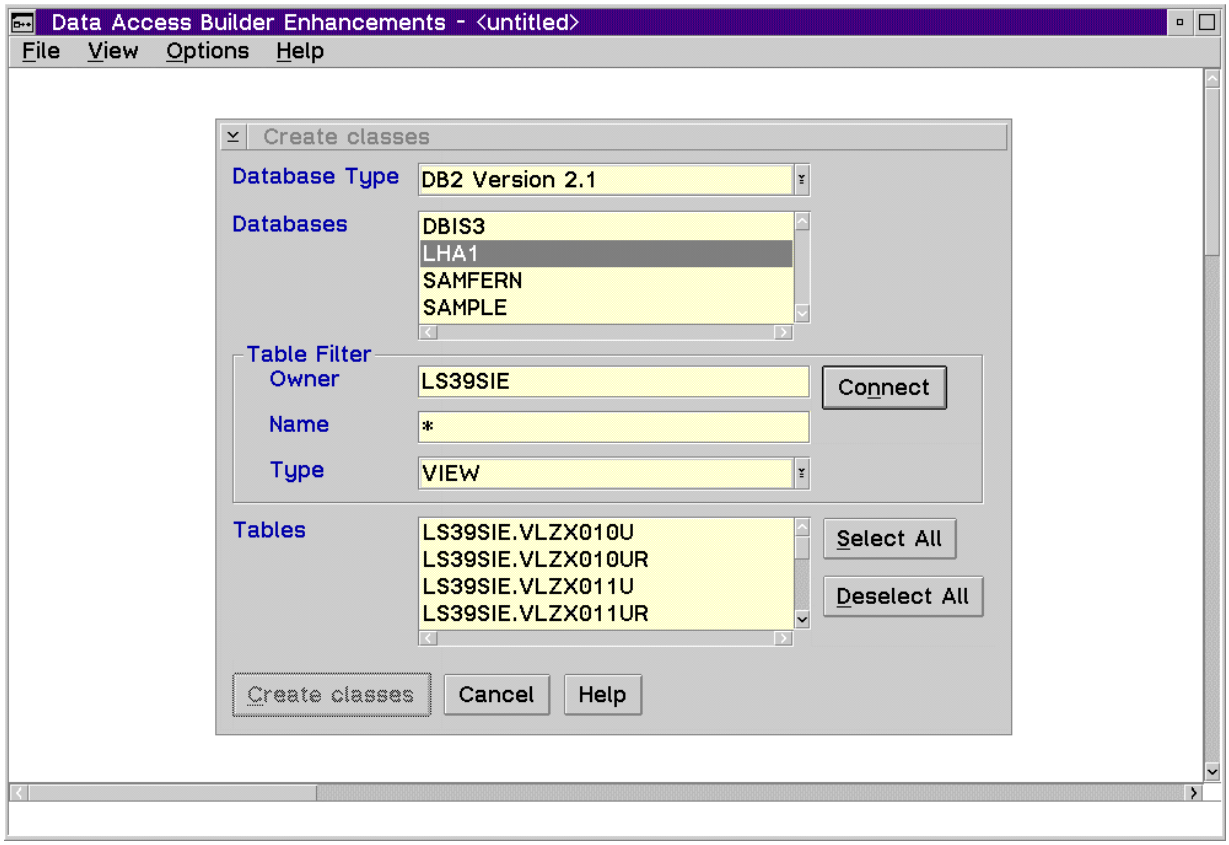

Abb. 5-9: Der *Enhanced Data Access Builder*.

Auch beim *Enhanced Data Access Builder* wird zuerst „generieren“ ausgewählt, dann die Datenbank. Neu ist, daß hier der „Tabellen-Inhaber“ (*owner*) angegeben werden kann und auch die Objekte näher bestimmt werden können. Diese Filterinformationen ermöglichen erst den Zugriff auf die MVS-DB2 Datenbanken.

Das Generieren der Klassen und das Erstellen der DLL funktionieren wie im *Data Access Builder*[44]. Man erhält aber statt eines .sqc-files ein .sqx-file, was ebenso die SQL-Anweisungen enthält, nur in C++-Umgebung und nicht in C.

Zusätzlich zu der bekannten Basisklasse und der Manager-Klasse werden weitere Klassen (sog. *Template*-Klassen) erstellt, die aber in keinem der vorliegenden Programme benutzt wurden.

Abbildung 5-10 zeigt eine Übersicht über die erzeugten Klassen:

[44] Vgl. Abschnitte 5.2.3.1 und 5.2.4 dieser Arbeit.

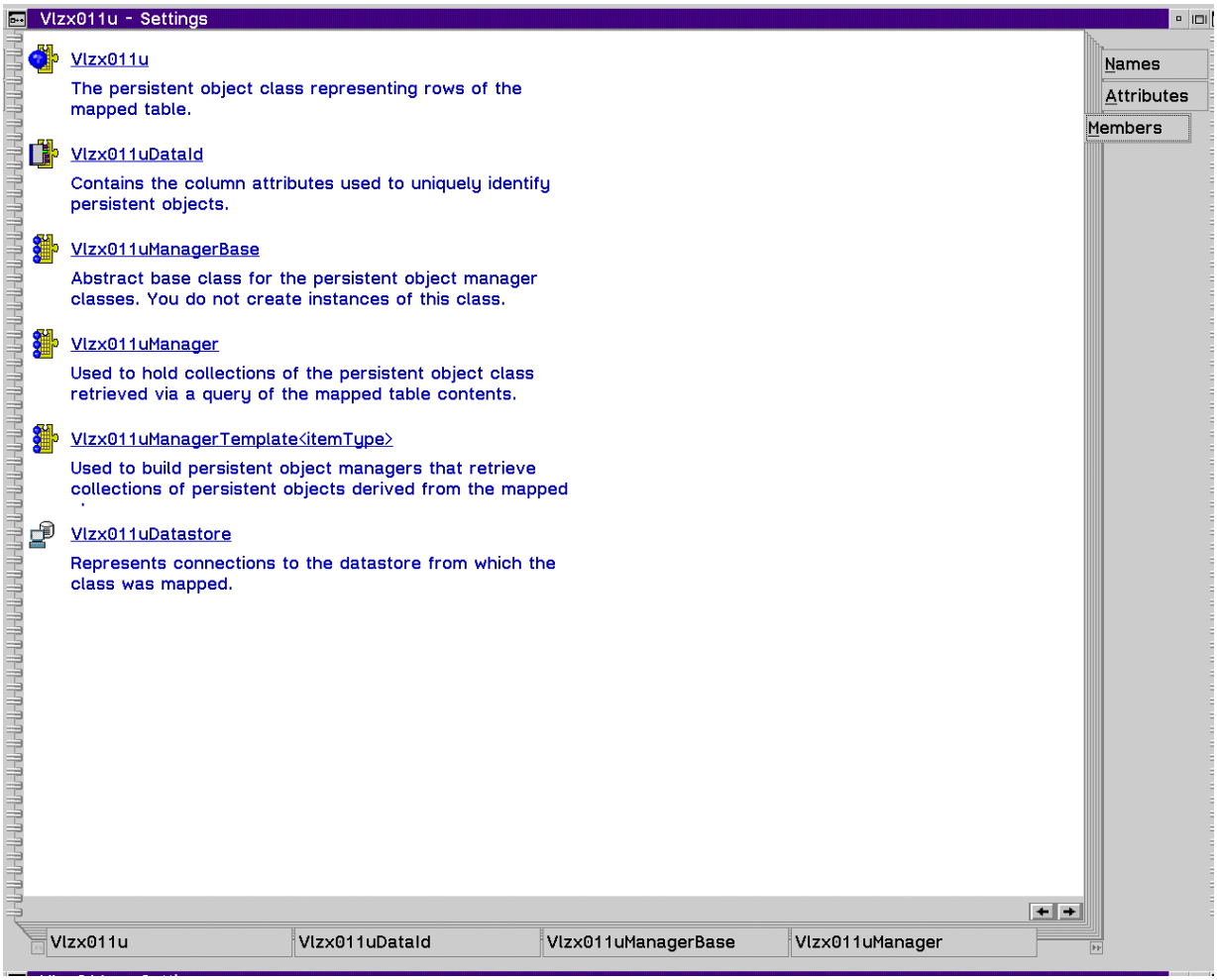

Abb. 5-10: Zusammenfassende Darstellung der mit dem *Enhanced Builder* erzeugten Klassen.

5.3 Erstellen einer Applikation mittels visueller Programmierung

5.3.1 Erstellen eines Projekts

Zunächst wird wieder ein Projekt angelegt[45]. Diesmal soll aber eine ausführbare Datei erzeugt werden. Dazu ist die Einstellung .exe-Datei als Ziel des Projekts notwendig. Auch hier müssen wie für die Erzeugung der DLL Angaben zum Arbeitsverzeichnis und zur Vererbung durchgeführt werden. Ist das Projekt angelegt, kann mit der Arbeit im *Visual Builder* begonnen werden.

[45] Vgl. Abschnitt 5.2.2dieser Arbeit.

5.3.2 Der Visual Builder

Mit dem *Visual Builder*[46] steht dem Anwendungsentwickler ein mächtiges Werkzeug zum visuellen Erstellen eigener objektorientierter Applikationen zur Verfügung. Die Philosophie dieses Werkzeugs ist die Wiederverwendung von Softwarebausteinen. Diese Bausteine werden Parts genannt und können beliebig kombiniert und den eigenen Bedürfnissen angepaßt werden. Dazu stehen dem Entwickler weitere Tools[47] zur Verfügung. Es ist auch möglich, eigene Bausteine zu entwickeln. Dafür sind gute C++-Kenntnisse unabdingbare Voraussetzung. Meist können Anwendungen aber komplett visuell erstellt werden, ohne das Schreiben von Quellcode. Der *Visual Builder* ist mit einer umfangreichen Bibliothek von vorgefertigten Softwarebausteinen ausgerüstet. Diese *Parts* sind Objekte mit Methoden und Attributen.

Das visuelle Programmieren mit dem *Visual Builder* besteht im wesentlichen aus:

- dem Erstellen einer graphischen Oberfläche mit dem *Composition Editor*
- dem Verbinden der verwendeten Bausteine
- dem Generieren von C++-Code mit dem Code-Generator

Abbildung 5-11 zeigt den *Visual Builder*:

[46] Weiterführende Informationen enthält der *Visual Builder User's Guide* oder auch *How Do I...? Visual Builder*.

[47] Composition Editor, Part Interface Editor und Class Editor sowie ein Texteditor

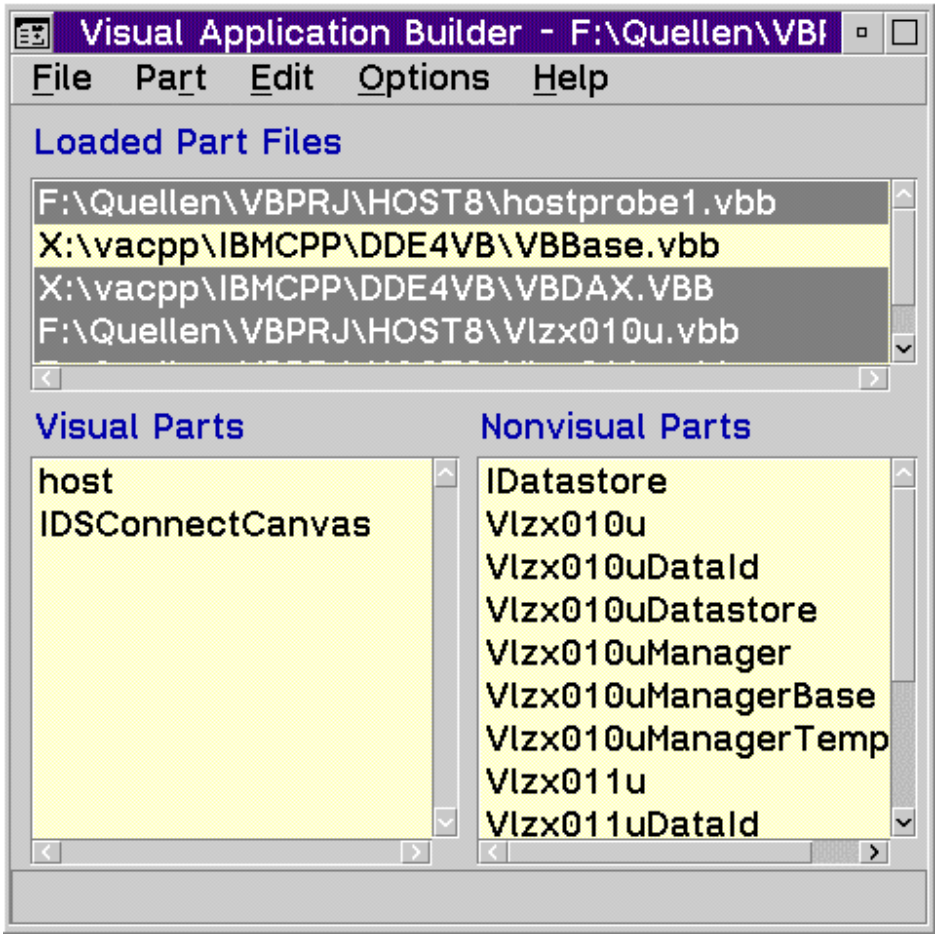

Abb. 5-11: Das *Visual Builder* Fenster mit bereits geladenen Klassen.

5.3.2.1 Einstellungen im Visual Builder

Der Visual Builder ermöglicht eine Vielzahl von Einstellungen. Wichtig für die weitere Arbeit sind die folgenden Eintragungen:

- das Arbeitsverzeichnis (in Übereinstimmung mit dem Projekt)
- Anzeige der kompletten Pfade aller Dateien
- *Makefile* generieren (Menüpunkt „*Options*“)

5.3.2.2 Laden und Importieren von Klassen und Parts

Beim Start des Visual Builders ist stets die Basisklasse VBBase.VBB geladen. Soll eine Anwendung mit Datenbankzugriff erstellt werden, so wählt man zusätzlich die Klasse VBDAX.VBB. Sie befindet sich im gleichen geladenen Verzeichnis[48] wie VBBase.VBB und enthält alle notwendigen Informationen für An- und Abmeldevorgänge an Datenbanken, sowie Möglichkeiten *Commit*-, *Rollback*- oder *Transact*- Statements abzusetzen. Markiert man eine geladene Datei (mit Endung VBB), so werden die darin enthaltenen Teile in der unteren Hälfte des *Visual-Builder*-Fensters angezeigt. Sie werden nach „sichtbaren" (*visual*) und „nicht-sichtbaren" (*nonvisual*) Teilen (*Parts*) geordnet.

Neben dem Laden von Standardklassen hat man außerdem die Möglichkeit, Teile zu importieren. Solche Teile sind z.B. die im Abschnitt 5.2.3 erzeugten *Parts* für die Arbeit mit einer Datenbank. Diese müssen in den *Visual Builder* importiert[49] werden, da noch keine VBB-Dateien dafür angelgt sind. Das Importieren muß vor[50] dem Öffnen des *Composition Editors* geschehen, da sonst nicht mit diesen Teilen gearbeitet werden kann.

Dazu wird ein *Part Information File* (Endung .vbe) importiert, welches alle notwendigen Informationen enthält. Es verweist auf die entsprechenden *DLLs* und *.lib* - Dateien, die zur weiteren Arbeit notwendig sind[51]

Man erhält eine Information über den Import-Vorgang und findet dann im oberen Drittel des Visual Builders die .vbe - Dateien wieder und dazu gehörend im unteren rechten Drittel (überschrieben mit „*nonvisual Parts*") die generierten Klassen. Diese Klassen können nun, um die spätere Arbeit mit dem *Composition Editor* zu vereinfachen, noch in die Palette geladen werden.[52]

Nach Abschluß des Importierens findet man im Arbeitsverzeichnis des Visual Builders zu jedem .vbe-file ein gleichlautendes .vbb-file[53], welches beim nächsten Hochfahren der

---

[48] Beide befinden sich im Verzeichnis \IBMCPP\DDE4VB.

[49] Menüpunkt „Datei" Unterpunkt „Importieren"

[50] Wird dies nicht vor dem Starten des *Composition Editors* getan, kann dieser mit den neuen Parts nicht arbeiten. Man findet auf der Oberfläche dann nur Fragezeichen vor, die nur die Standardeigenschaften aller nicht-sichtbaren Teile haben. Es wird keine Datenbankfunktionalität angezeigt.

[51] Siehe auch Abschnitt 5.2.3.1 dieser Arbeit.

[52] Dazu den *Part* mit der rechten Maustaste wählen. Im aufgeklappten Menü dann „*add to Palette*" angeben.

[53] Bei selbst erstellten Teilen lautet die Endung .vbb, die Standardklassen tragen die Endung .VBB.

Arbeitsumgebung gleich geladen werden kann, so daß nicht jedes Mal neu importiert werden muß.

Bei einer Änderung der Datenbank bzw. der entsprechenden Klassen müssen diese neu importiert, also die Änderungen „nachgezogen“ werden. Nur so ist gewährleistet, daß das Programm die korrekten Daten liefert.

5.3.3 Arbeitsoberfläche Composition Editor /Class Editor/ Part Interface Editor

Oft wird vom *Visual Builder* gesprochen, aber gemeint ist der *Composition Editor*, die Oberfläche auf der die Software entwickelt wird. Der *Composition Editor* ist neben dem *WorkFrame* einer der wesentlichen Arbeitsplätze während der Softwareentwicklung. Hier werden die Objekte auf der Oberfläche angeordnet, ihre Attribute verknüpft und das Zusammenspiel unter den Objekten geregelt. Parallel zum *Composition Editor* steht für jeden Part der dort erstellt oder benutzt wird auch der *Class Editor* und der *Part Interface Editor* zur Verfügung. Zwischen diesen drei Editoren wird leicht per Mausklick umgeschaltet. Ein gedrittelter Kreis zeigt die Zusammengehörigkeit dieser drei Entwicklungseditoren.

Der folgende Bildschirmausschnitt zeigt den *Composition Editor* während der Programmierung der dritten Beispielanwendung (DB2-Mainframeanbindung)

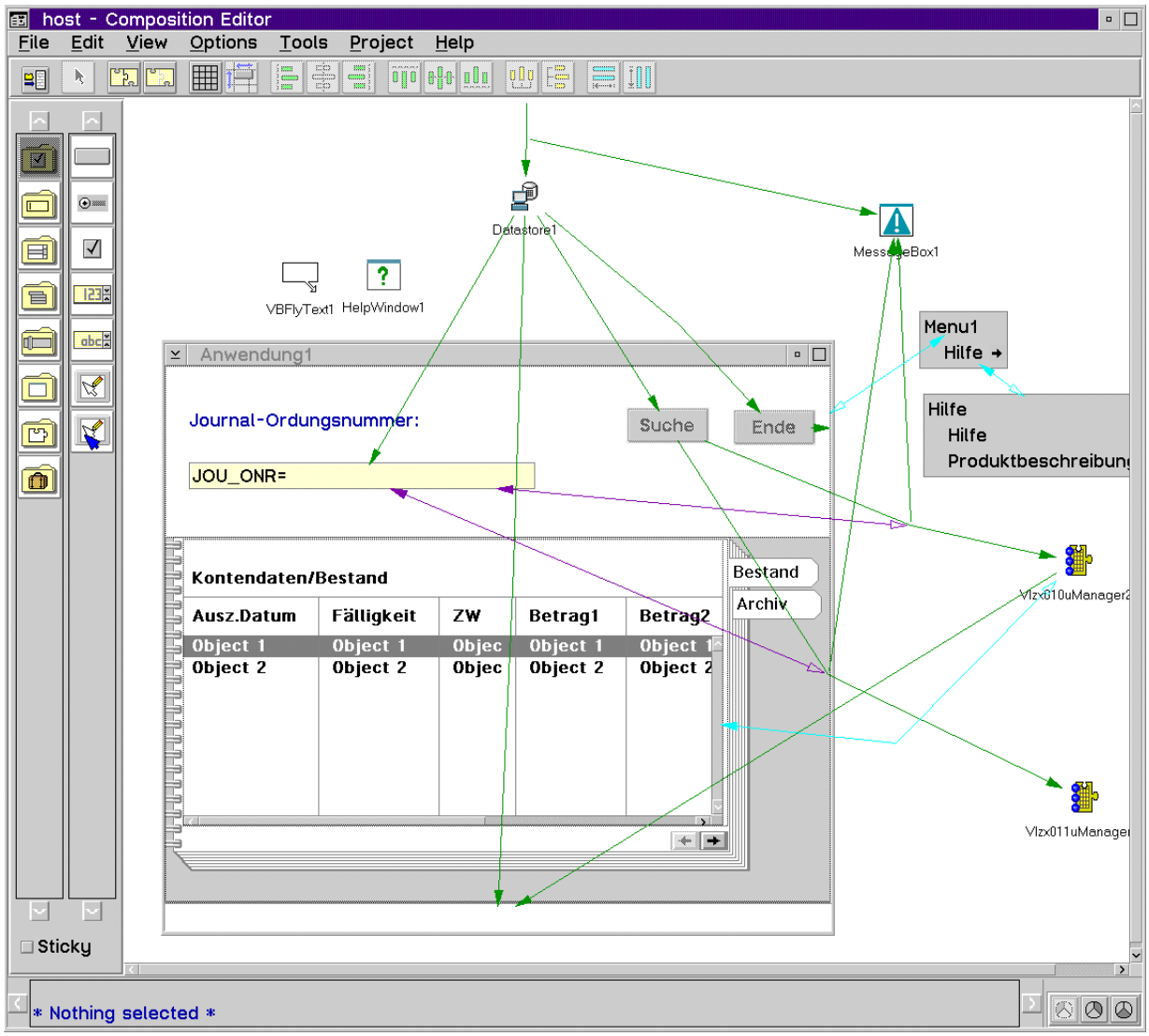

Abb. 5-12: Der *Composition Editor* während der Erstellung der dritten Anwendung.

Beim ersten Start des *Composition Editors* aus dem *Visual Builder* heraus[54], wird der zu erstellenden Anwendung ein Name gegeben, später reicht ein einfaches „*open*“ aus. Zu jeder Anwendung die auf dieser Fläche gestaltet wird, entsteht ein *VBB-file*. Es muß geladen werden, bevor die nächste Sitzung am *Composition Editor* beginnen kann.[55]

Zur Veranschaulichung der anderen beiden Editoren, wird anschließend die gleiche Anwendung im gleichen Status gezeigt. Dazu wird kurz die Funktionalität der beiden anderen Editoren erklärt.

[54] Im Menü „*Part*“, Unterpunkt „*new*“.

[55] Solch eine Datei existiert vor dem ersten Codegenerieren bzw. speichern noch nicht.

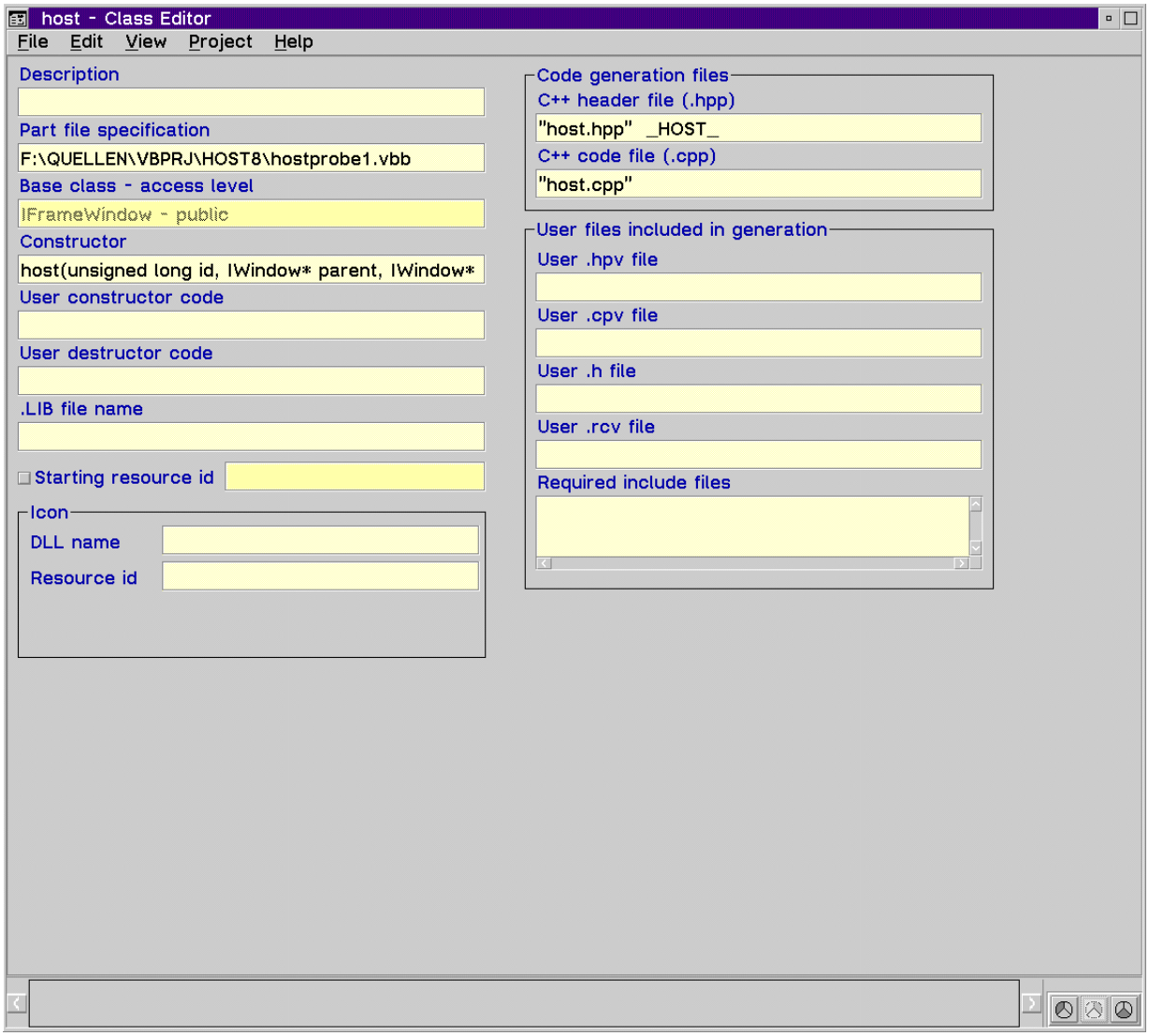

Abb. 5-13: Der *Class Editor* - Ansicht der dritten Anwendung.

Der *Class Editor* gestattet es, eigenen Quellcode zur Anwendung hinzuzufügen (Dateien mit der Endung .cpv und .hpv) sowie sogenannte *Resource*-DLLs anzugeben. Diese beinhalten z.B. Icons für die Anwendung. Darüber hinaus kann für die Klasse eigener *Constructor*- oder *Destructor-Code* eingebunden werden.

Nachstehend wird die gleiche Anwendung aus der Sicht des *Part Interface Editors* dargestellt.

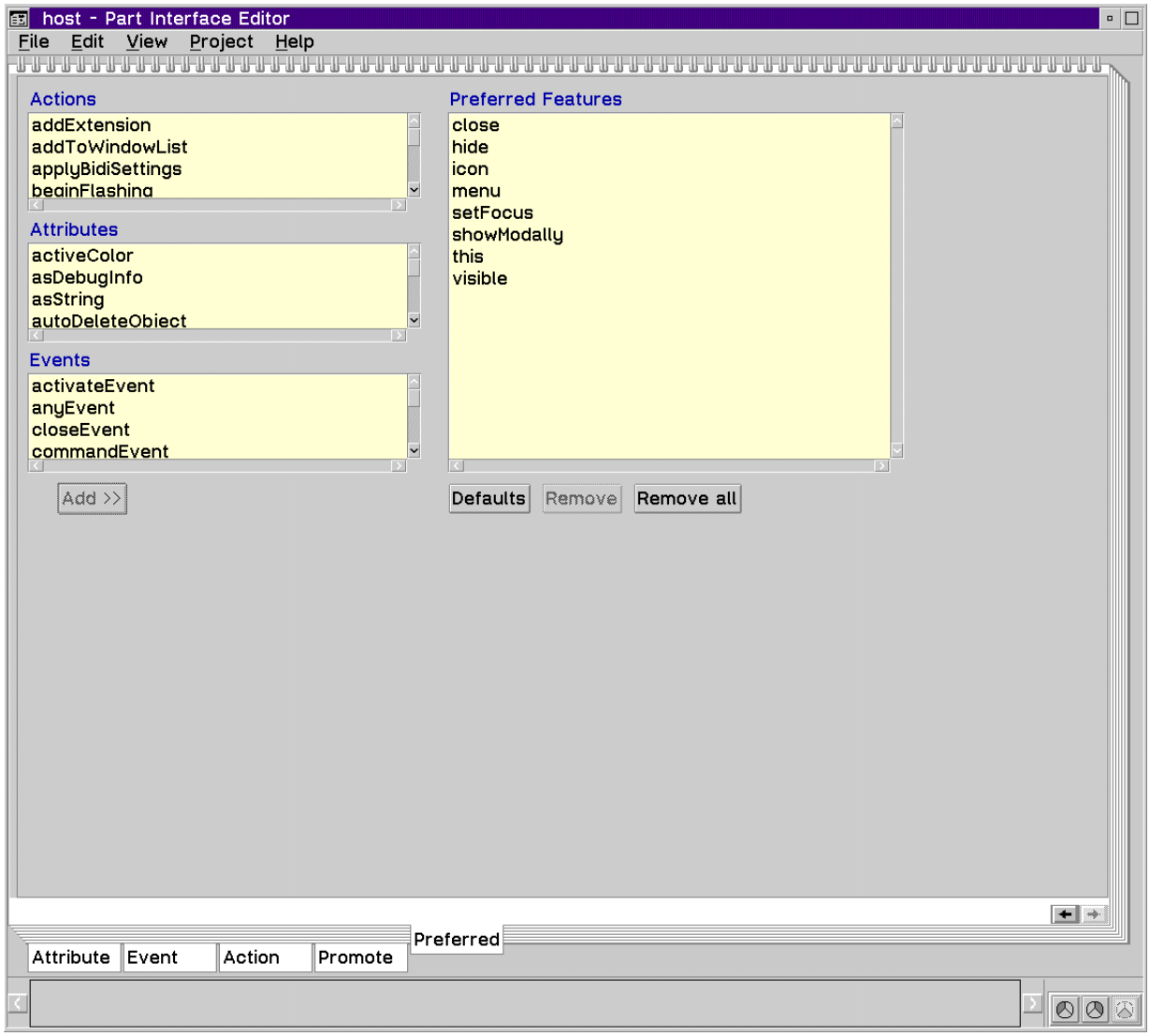

Abb. 5-14: Der *Part Interface Editor* - Ansicht der Host-Anwendung.

Der *Part Interface Editor* gestattet es, die Schnittstelle eines Objekts zu ändern. Im hier abgebildeten Ausschnitt sind die sogenannten „*preferred Features*“ zu sehen, das sind die Attribute/Ereignisse und Aktionen, die von außen aufgerufen oder geändert werden können (z.B. durch andere Ereignisse). Diese Einträge können durch den Entwickler abgewandelt werden, er kann hier eigene Aktionen zur weiteren Verbindung zur Verfügung stellen oder vorhandene abändern.

5.3.3.1 Ausgangssituation

Auf dem Arbeitsblatt („*Canvas*“ genannt) befindet sich ein normales Fenster (*IFrameWindow*), in dem mit der Arbeit begonnen werden kann. Hier werden alle benötigten

Schalter (*Buttons, Sliders, Check-Boxes*) angeordnet, genauso wie Eingabefelder (*EntryFields*) oder *Container*, also alle auch zur Laufzeit sichtbaren Elemente der Anwendung (*visual Parts*). Im Gegensatz dazu werden alle nicht sichtbaren Teile (*nonvisual Parts*) auf der Arbeitsoberfläche neben dem Fenster plaziert. Hierzu gehören z.B. *variable Parts*, die generierten Datenbank-Klassen oder auch die *FlyOverText-Control*. Das sind alles Elemente die im Hintergrund agieren und später im Programmablauf nicht auf der Oberfläche zu sehen sind.

5.3.3.2 Arbeiten mit Parts

Auf dem *Canvas* und im Fenster können nun alle benötigten Teile angeordnet und miteinander verbunden werden. Außerdem können an jedem Objekt weitere Einstellungen vorgenommen werden. Zu jedem Objekt existiert ein Kontextmenü, über das man zu den Einstellungen gelangt. Es ist auch möglich, zu jedem einzelnen *Part* den *Class Editor* oder den *Part Interface Editor* aufzurufen und das Objekt dort zu bearbeiten.
Sind alle Objekte auf der Oberfläche angeordnet, können sie miteinander verknüpft werden. Diese Verknüpfung oder Verbindung der Teile (P*arts*) miteinander ist ein entscheidender Schritt zur Erstellung der Anwendung. Hier zeigt sich, ob die vorangegangenen theoretischen Entwürfe korrekt waren.

5.3.3.3 Verknüpfen der Parts

Verbindungen geben dem Entwickler die Möglichkeit, die Objekte auf sichtbare Weise miteinander zu verknüpfen. Jedes Visual Builder Teil, ob sichtbar oder nicht sichtbar verfügt sowohl über eigene, als auch über Standardattribute, -ereignisse und -aktionen[56]. Alle für eine Verbindung möglichen Optionen eines Objekts lassen sich über das Kontextmenü und den Unterpunkt „*connect*“ ansehen und auswählen. Eine Übersicht über sämtliche *Features* eines Parts[57] läßt sich über Kontextmenü und „*Browse Part Features*“ anzeigen.

---

[56] Im allgemeinen ist immer die Rede von *Attributes*, *Events* und *Actions*.

[57] Auch versteckte *features* sind hier aufgeführt, sie können aber nicht für Verbindungen genutzt werden.

Um zwei Objekte miteinander zu verbinden, muß klar sein, von welchem Objekt aus die Verbindung gezogen werden soll. Mitunter sind Verbindungen nur in eine Richtung gestattet, dann wählt *Visual Age C++* automatisch die korrekte Variante aus. Anderenfalls muß der Entwickler entscheiden, welche Richtung der Verbindung die richtige ist.
Man öffnet das Kontextmenü des ersten beteiligten Objekts und wählt „*connect*“. Daraufhin erscheint eine Liste der gebräuchlichsten Attribute oder *Events* die eine Verbindung anstoßen können. Ist die entsprechende Eigenschaft ausgewählt, wird der Mauszeiger auf das Zielobjekt geführt und dort die gleiche Aktion wiederholt.
Im Anschluß daran erhält man eine Verbindung, deren Farbe und Struktur Aussage über die Art der Verbindung geben. Man unterscheidet:

- Attribut-zu-*Action* Verbindung (grüner Pfeil, einseitig)
- *Event*-zu-*Action* Verbindung (grüner Pfeil, einseitig)
- *Event*-zu-Attribut Verbindung (grüner Pfeil, einseitig)
- Attribut-zu-Attribut Verbindung (hellblauer Pfeil, doppelseitig)
- *Custom logic* Verbindung (dunkelblauer Pfeil, einseitig)
- Attribut-zu-*member function* Verbindung (hellgrüner Pfeil, einseitig)
- Parameter-Verbindung (violetter Pfeil, doppelseitig)

Allen angegebenen Verbindungen ist gemeinsam, daß eine durchgehende Linie eine vollständige Verbindung angibt, während eine unterbrochen dargestellte Linie auf fehlende Parameter hinweist, die entweder über eine Parameter-Verbindung oder über permanent eingestellte Parameter vervollständigt werden müssen.
Bei solchen Verknüpfungen, die nur in eine Richtung arbeiten (einseitige Pfeile), weisen die Pfeilspitzen immer auf das Ziel der Verbindung. Bei bidirektionalen Verbindungen (Doppelpfeile) weist die leere Pfeilspitze immer auf die Quelle und die gefüllte Pfeilspitze immer auf das Ziel. Im folgenden wird kurz auf die Eigenschaften jeder dieser Verbindungen eingegangen.[58]

[58] Eine ausführliche Beschreibung der möglichen Verbindungen gibt der *Visual Builder Users Guide* oder auch *Visual Builder - How Do I ...?* (beide Informationsbücher online verfügbar und im Paket enthalten)

Die Attribut-zu-*Action* Verbindung
verbindet ein Attribut mit einer Aktion. Ändert sich der Wert des Attributs, wird die Aktion angestoßen.

Bei der *Event*-zu-*Action* Verbindung
wird ein Ereignis mit einer Aktion gekoppelt. Tritt das Ereignis ein, so wird die Aktion aufgerufen.

Die *Event*-zu-Attribut Verbindung
verbindet hingegen ein Ereignis mit einem Attribut. Hierbei wird der Wert des abhängigen Attributs nach dem Eintreten des Ereignisses geändert. Im Gegensatz dazu setzt die

Attribut-zu-Attribut Verbindung
zwei Attribute in Abhängigkeit voneinander. Ändert sich der Wert eines Attributes, so wird der Wert des anderen Attributes angepaßt. Diese Verbindung kann in beiden Richtungen funktionieren, und zwar dann, wenn beide Objekte eine „*Set Member Function*“ haben. Hat nur ein Objekt eine solche Funktion, ist es automatisch das Ziel und das andere die Quelle. Eine solchen Verbindung verlangt nie nach Parametern.

Custom logic Verbindung
Hierbei wird entweder durch den geänderten Wert eines Attributs oder das Eintreten eines Ereignisses eine Funktion aufgerufen. Diese Funktion übernimmt meist komplexere Aufgaben, die sonst durch eine Vielzahl von Verbindungen hätten gelöst werden müssen.

Attribut-zu-*Member Function* Verbindung
Eine Änderung des Attributwertes stößt beim Zielobjekt eine *member Function* an.
Damit entspricht sie der *Event*-zu-*Member Function* Verbindung. Sie funktioniert nur, wenn das Attribut der Quelle ein „*Event Attribute*“ ist (andere Attribute haben keinen *Event Identifier*). Kann beim Quellobjekt also keine Wertänderung des Attributes registriert werden, funktioniert diese Verbindung nicht. Vom Prinzip her ist diese Art der Verbindung

gleich einer Verbindung Attribut-zu-Aktion oder Ereignis-zu-Aktion. Der Unterschied liegt in der Zugriffsberechtigung. Auch Aktionen sind in gewissem Sinne *Member Functions*, sie haben jedoch ein sogenanntes *Part Interface* (Schnittstelle) und sind frei zugänglich. *Member Functions* hingegen können nur vom Ersteller des *Parts* benutzt werden.

Die Parameter Verbindung
besteht immer zwischen einem Parameter und einer unvollständigen Verbindung. Sie liefert die erforderlichen Angaben zum Vervollständigen der angesprochenen *connection*[59]. Auch sie ist wie die Attribut-zu-Attribut Verknüpfung bidirektional, wobei hier aber der Parameter stets die Quelle und die Verbindung stets das Ziel ist. Parameter können z.B. Attribute sein. Diese werden dann über ihre *Get Member Function* angesprochen. Parameter können aber auch Rückgabewerte von Aktionen sein (*Action Result*).

Zur Veranschaulichung von Verknüpfungen sei hier in Abbildung 5-15 die Entwicklungsumgebung während der Erstellung der zweiten Anwendung (DB2/2-Serverzugriff) gezeigt:

[59] Verbindung - im Composition Editor stets *Connection* genannt.

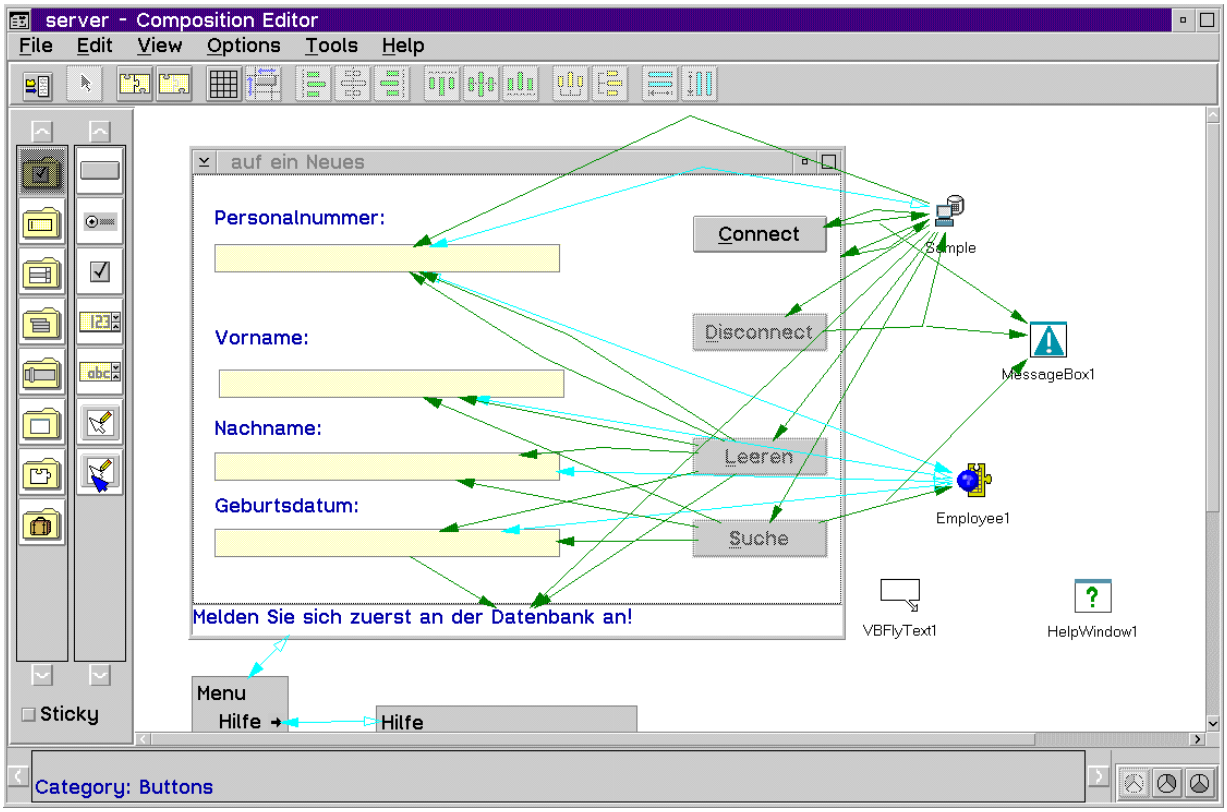

Abb. 5-15: Der *Composition Editor*, Programmierung der zweiten Beispielanwendung.

5.3.3.4 Generieren des Quellcodes

Nachdem alle Verbindungen gezogen wurden, kann der Quellcode generiert werden. Dazu wird im *Composition Editor*, oder in einem der anderen beiden Editoren, im Menü *Parts* der Unterpunkt *Save and Generate* ausgewählt. Im daraufhin aufklappenden Untermenü muß zunächst „*Part Source*“ gewählt werden. Nachdem die Generierung der „*Part Source*“ abgeschlossen ist, wird im zweiten Schritt „*Main for Part*“ generiert[60]. Die gleichen Aufrufe sind auch im *Visual Builder* im Menü „*Parts*“ möglich.

Treten beim Generieren Fehler auf, z.B. wegen falscher Verbindungen, so bekommt der Entwickler eine Meldung und der Vorgang wird abgebrochen.

[60] Es muß stets zuerst die „*Part-Source*“ generiert werden. Besteht die Anwendung aus mehreren Teilen., so ist zuerst für alle Teile „*Part-Source*“ zu generieren, und anschließend „*Main for Part*“ nur im Hauptfenster der Anwendung. Das ist das Fenster, was der Benutzer zuerst sieht.

Ist die Codegenerierung erfolgreich beendet, so existieren im Arbeitsverzeichnis Dateien mit den folgenden Dateiendungen:

- cpp C++-Code-Datei h eine *Resource-header*-Datei für die cpp-Datei (enthält *Resource-IDs* für die *Parts*)
- hpp Eine C++ *header*-Datei für cpp
- vbb *Visual Builder Class (binary) file* (Datei mit Informationen für den *Visual Builder*)
- mak enthält die Daten für das „*nmake*“, (nur vorhanden wenn im *Visual Builder* explizit gewählt)
- vbmain.cpp enthält C++ Hauptprogramm zur cpp-Datei (Entsteht nur, wenn vom *WorkFrame* aus gearbeitet wurde. Sonst trägt die Datei die Endung app.)
- app „Hauptprogramm“ zu cpp, wenn kein *WorkFrame* verwendet wurde
- rc -Datei - *Resource*-Datei, die alle Texte (*Strings*) des Parts enthält (z.B. für Menüs, Schlater...)
- rci - Datei - wie rc, aber für den *Main*-Teil
- h eine *Resource-header*-Datei für die cpp-Datei (enthält *Resource-IDs* für die *Parts*)

Falls für die Anwendung eine Hilfe vorgesehen ist, so befinden sich im Verzeichnis außerdem eine ipf-Datei[61] sowie nach erfolgreicher Übersetzung eine hlp-Datei.[62]

Abbildung 5-16 zeigt den *WorkFrame* mit allen Dateien, die zur Erstellung der Anwendung notwendig sind. Er enthält darüber hinaus die in den Zwischenschritten, während des Compilierens, entstandenen Dateien, sowie einige Dateien, die durch das Benutzen des Browsers oder des Navigators entstanden sind.

[61] Das ist die Datei, die den Quellcode für die Hilfe enthält.

[62] Aufrufbare Hilfe-Datei mit üblichen Funktionalitäten der Online-Hilfen.

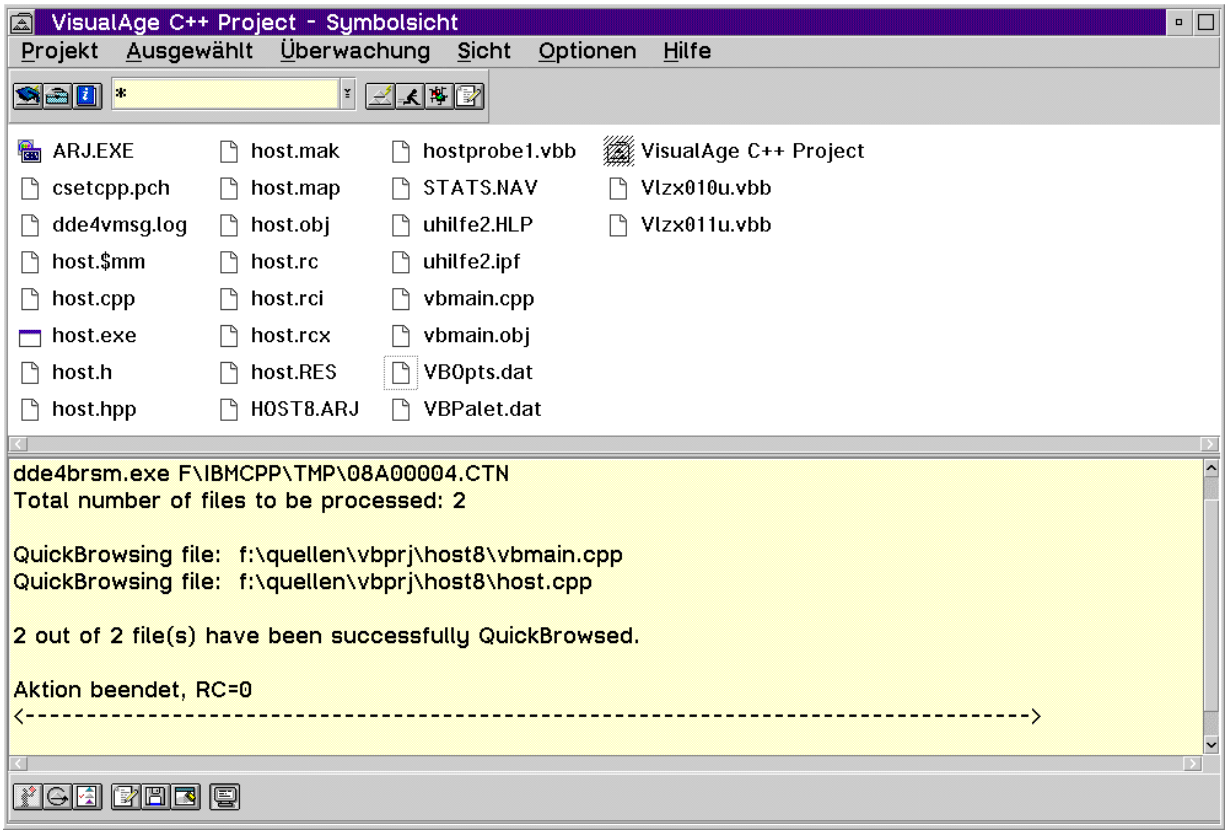

Abb. 5-16: Der *WorkFrame* mit den Dateien der Host-Beispielanwendung.

5.3.4 Compilieren und Binden

Wurde für die Anwendung auch ein *makefile* erstellt, so braucht dieses nur aufgerufen zu werden („*nmake*“), und die entsprechenden Quelldateien werden mit den richtigen Compilereinstellungen übersetzt. Dabei entstehen u.a. eine obj-Datei aus der cpp-Datei, eine vbmain.obj-Datei und eine res-Datei. Alle werden zu einem ausführbaren Programm zusammengefügt. Diese Programmdatei erhält den Namen des .cpp-files, soweit kein anderer Name ausdrücklich angegeben wurde.

Sollte über den *Visual Builder* kein *makefile* mit erzeugt worden sein, so kann mittels „*makemake*“ ein solches erstellt werden. Hierbei muß jedoch auf die richtigen Optionen geachtet werden.

5.3.4.1 Compileroptionen

Wird mit dem *Visual Builder* eine Anwendung entwickelt, so handelt es sich um eine sogenannte *PM*-Anwendung (*PM= Presentation Manager*[63]). Dies sollte unter den Compileroptionen angegeben werden, ebenso wie dynamisches Linken von Bibliotheken. Auch die Einstellungen „*Multithread*“ ist notwendig, sobald mit den *Visual-Builder*-Klassen gearbeitet wird.

Abbildung 5-17 zeigt, wo die Einstellungen der Compileroptionen im *WorkFrame* erfolgen. Hier können alle notwendigen Schalter graphisch gesetzt werden.

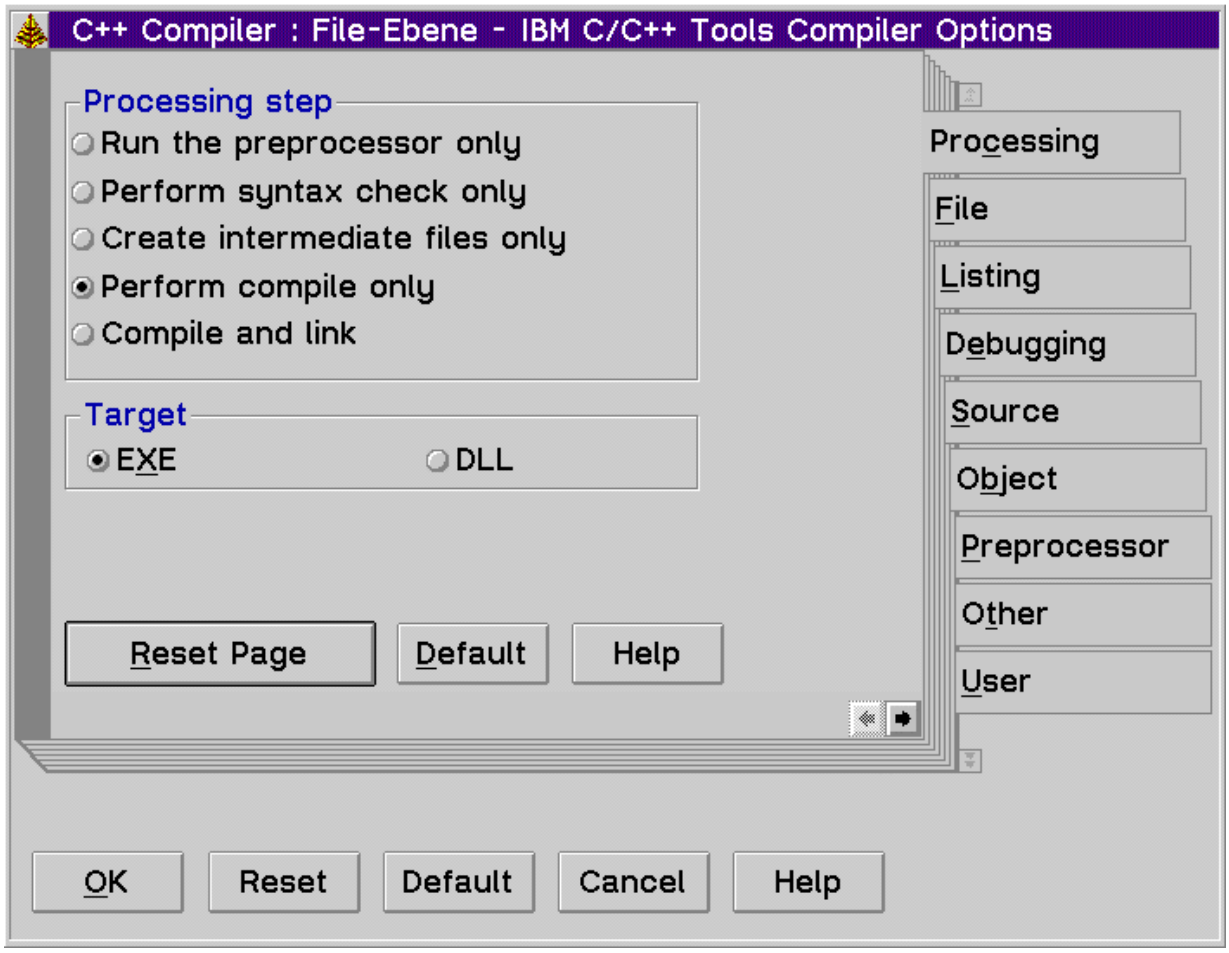

Abb. 5-17: Einstellen der Compileroptionen im *WorkFrame*.

Alle Optionen können auch von Hand, z.B. beim Aufruf des Compilers von der Befehlszeile aus, eingegeben werden. Die Einstellungen sind auch im *Makefile* zu finden. Die folgenden Zeilen stellen nur einen Ausschnitt dar, und sollen einen Teil der Parameter zeigen, und zwar

[63] Objektorientierte, graphische Benutzeroberfläche von OS/2

für die Erstellung der ausführbaren Datei. Es handelt sich dabei um das *Makefile* zur dritten Anwendung.

```
PARTCPPFLAGS=-Fthost -Gm+ -Tdp -Gd+ -I.
APPCPPFLAGS=-Fthost -Gm+ -Tdp -Gd+ -I.

all:   host

host:   host.exe

host.exe:   vbmain.obj host.obj host.res
        icc $(PARTCPPFLAGS) /B"vbmain.obj /pmtype:pm" \
          host.obj \
          /Fehost.exe /Fmhost.map \
          os2386.lib
        rc host.res host.exe
```

Abb. 5-18: Ausschnitt aus dem *Makefile* der Anwendung „host“.

Die einzelnen Parameter tragen dabei folgende Bedeutung:

(siehe Tabelle 5-1, folgende Seite)

Ft(dir)	generiert Dateien zur Template-Auflösung und stellt diese in das Verzeichnis (dir)
Gm+	linkt *Multithread-Library* dazu (sonst: Single Thread)
Tdp	Compiliert alle Quelldateien als C++-Dateien und sorgt für das ordnungsgemäße Auflösen der Template-Funktionen
Gd+	linkt Laufzeitbibliotheken dynamisch
I	sucht Include-Dateien außer im aktuellen Verzeichnis und in den Verzeichnissen des INCLUDE-Pfades (config.sys) auch im angegeben Pfad
/B „/DE/pmtype:pm“	übergibt den Text /DE/pmtype:pm an den Linker
/FeName	Name der zu erstellenden exe-Datei angeben
/FmName	map-Datei für den Linker erstellen und mit Name benennen

Tabelle 5-1 : Übersicht über die Compileroptionen.

5.3.4.2 Linkeroptionen

Für das Linken einer PM-Anwendung zu einer ausführbaren Datei, sollten im *WorkFrame* folgende Einstellungen vorgenommen werden:

- die Modul-Ladeadresse sollte eingeschaltet sein und den Wert 65536 besitzen
- die Anwendung ist vom Typ „PM“
- der Schalter „*Search the extended library*“ ist zu setzen
- unter „*Filenames*“ sind im Feld *Libraries* die .lib-Dateien der Datenbankobjekte einzufügen
- *Templates used* muß angeschaltet sein

5.4 Tools zur Bearbeitung erstellter Programme

Visual Age stellt neben Tools zur Codeerstellung auch eine Reihe von Werkzeugen zur Programmoptimierung und Fehlersuche zur Verfügung. Anschließend sollen Browser, Performance Analyzer und Debugger vorgestellt werden.

5.4.1 Debugger

Der Debugger erlaubt eine Untersuchung des erstellten Programmes hinsichtlich der aufgerufenen Threads, der Variablen, der Registerinhalte, des Inhalts des Call-Stacks oder auch der Abhängigkeiten von Fenstern innerhalb der Anwendung. Während des Programmablaufs können diese Werte verfolgt werden. Um die Beispiele übersichtlich zu halten, wurde die einfachste der Anwendungen gewählt und anschließend einige Sichten des Debuggers gezeigt:

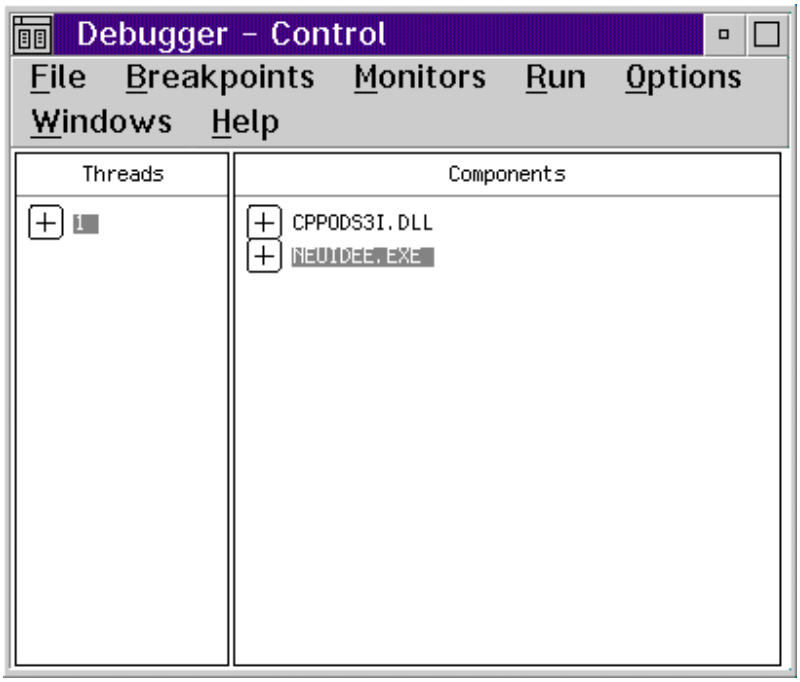

Abb. 5-19: Zentrales Debugger-Fenster.

Von diesem zentralen Debugger-Fenster aus startet man weitere Fenster zur Überwachung von Variablen oder Registerinhalten. Diese Überwachungsfenster werden unter dem Menüpunkt „Monitors“ verwaltet. Die folgenden Auszüge zeigen diese Sichten:

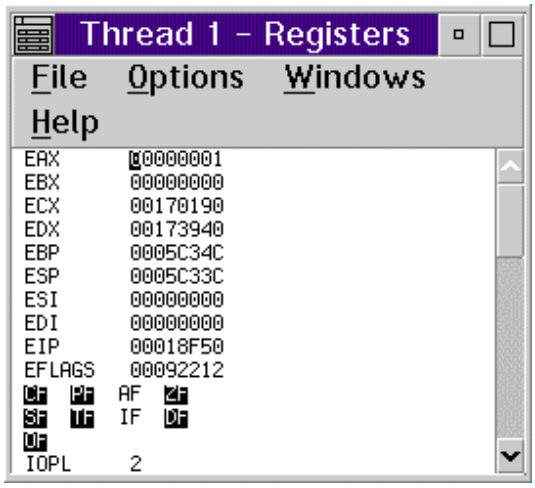

Abb. 5-20: Debugger - Auszug der Registerinhalte.

Abbildung 5-20 zeigt das Fenster, in dem Registerinhalte überwacht werden können. Abbildung 5-21 zeigt die Debugger Sicht auf den Quellcode:

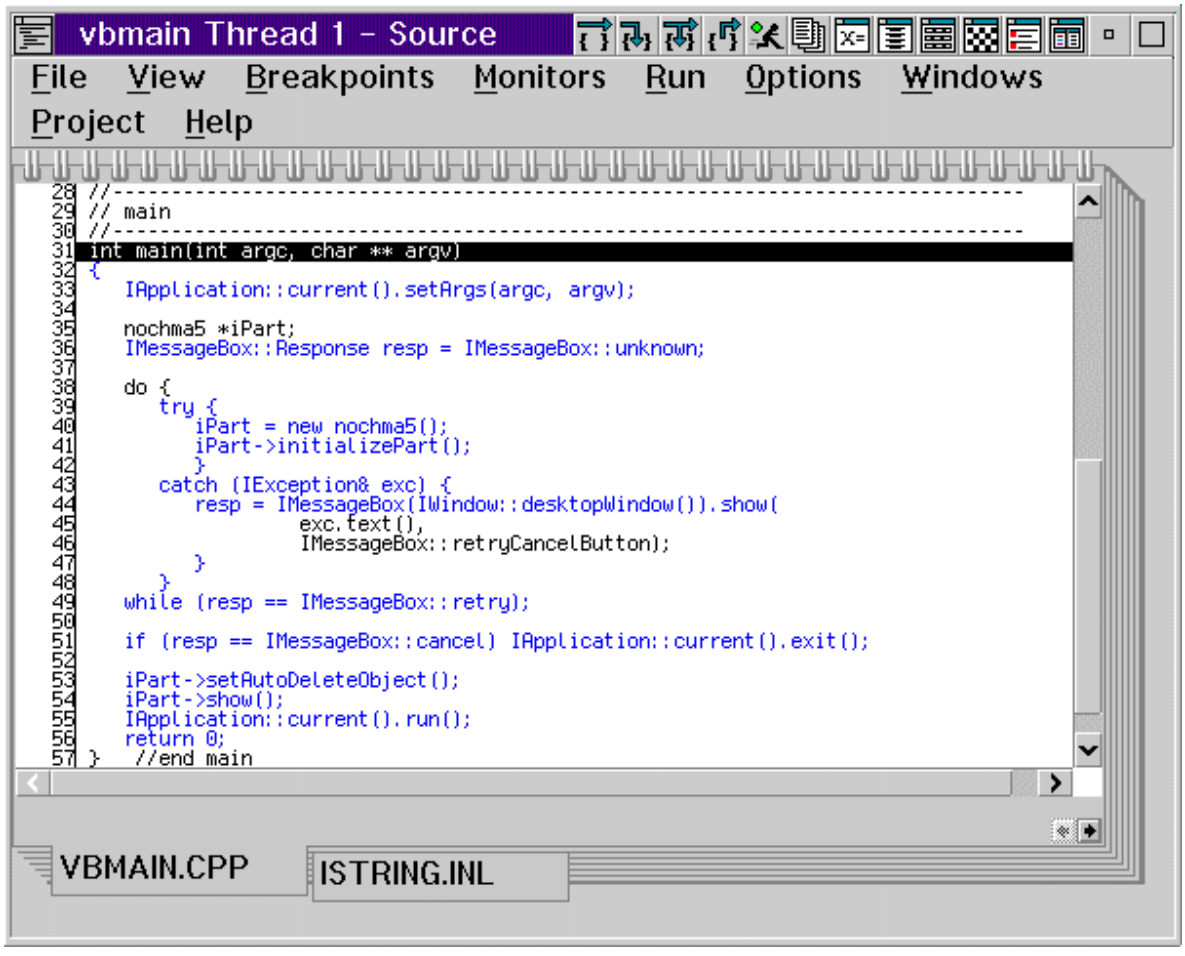

Abb. 5-21: Debbuger. Sicht auf den Quellcode.

Beim Debug-Lauf des Programms, werden die jeweiligen Stellen des Quellcodes automatisch angezeigt, auch wenn es sich um mehrere Dateien handelt. Der Debugger springt dabei zwischen den einzelnen „Laschen“ des *Notebooks*.

Eine weitere interessante Möglichkeit, Abhängigkeiten innerhalb des Programmes darzustellen, ist die Analyse der erzeugten Fenster. Hier kann die Hierarchie der Eltern- und Kindfenster deutlich gemacht werden. In den zu dieser Arbeit entwickelten Beispielen handelt es sich jedoch um Anwendungen, in denen nur ein Fenster existiert, so daß zur Fensteranalyse keine weiteren Ausführungen folgen.

5.4.2 Browser

Eine andere Art von Abhängigkeiten im Programm kann mit dem Browser untersucht werden. Der Browser listet alle im Programm implementierten Funktionen auf. Ausgehend von dieser Übersicht[64] können z.B. der Vererbungs-Graph des kompletten Programms gezeichnet oder auch der Vererbungs-Graph einer einzigen Funktion angesehen werden. Der Browser bietet auch die Möglichkeit, sämtliche *Include*-Dateien eines Programms aufzulisten und deren Abhängigkeiten darzustellen.

[64] Vgl. Abbildung 5-22 in dieser Arbeit.

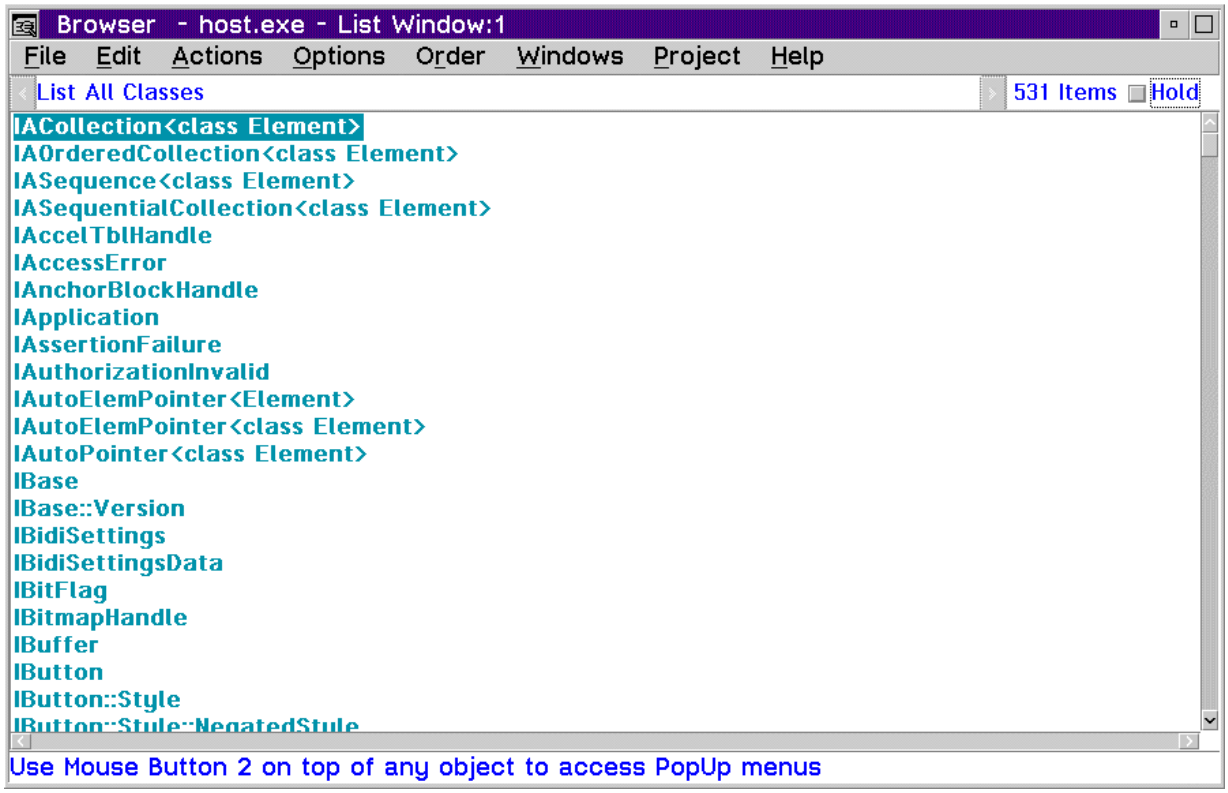

Abb. 5-22: Zentrales Browser-Fenster.

Bild 5-22 zeigt das Hauptfenster des Browsers. Untersucht wurde in diesem Fall das Programm „host.exe“, also die Anwendung mit *Mainframe* Zugriff. Zu dieser Anwendung werden anschließend auch der Vererbungs-Graph und der *Include*-Graph gezeigt.

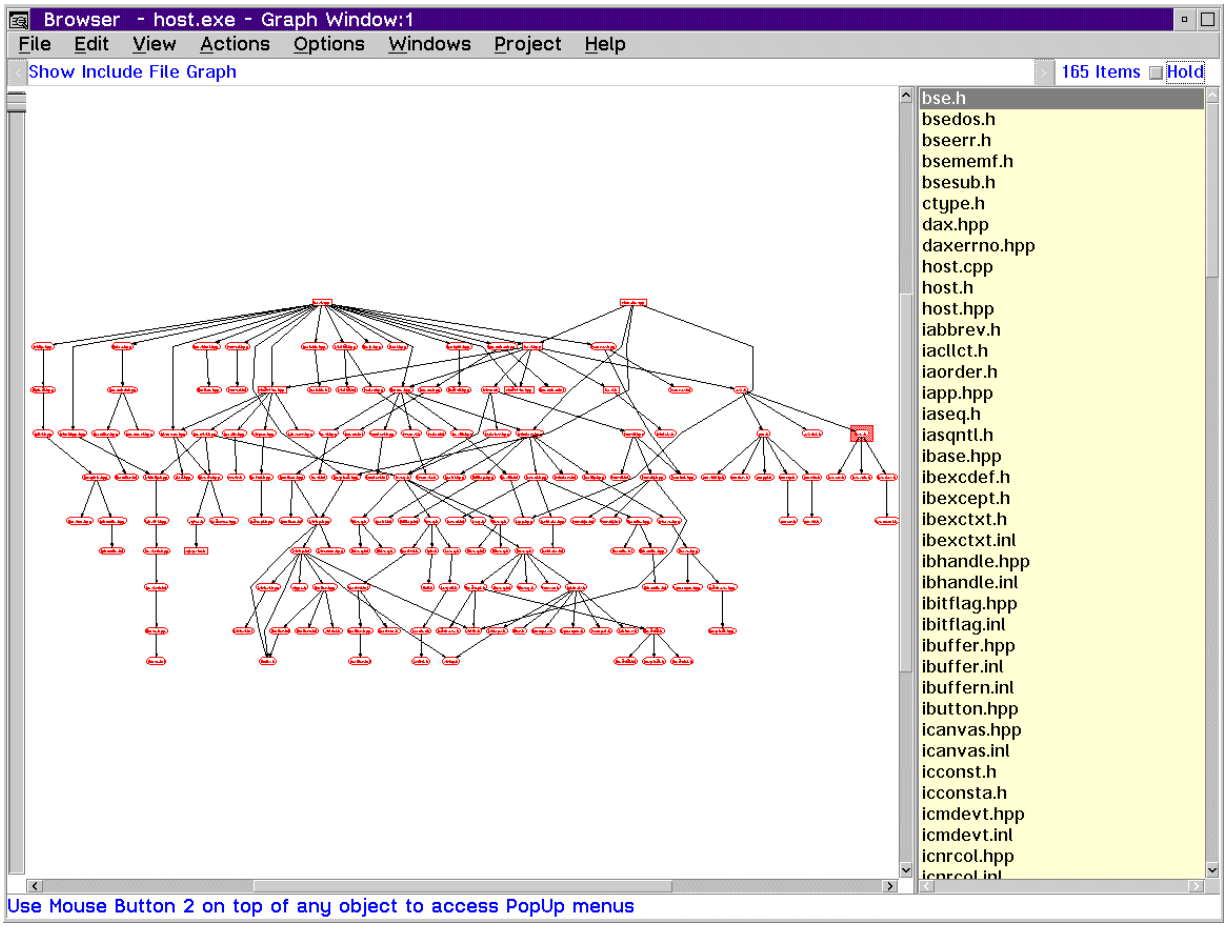

Abb. 5-23: Beispiel für einen *Include*-Graphen.

Das Arbeiten mit dem Browser im *Include*-Graph wird durch viele Extras unterstützt. So kann man beispielsweise eine *Include*-Datei aus der rechten Liste auswählen. Diese Datei wird dann im Graphen umrahmt, so daß man sie leichter auffinden kann (wie in Abbildung 5-23 dargestellt). Balken erlauben das einfache Verschieben und Vergrößern des Ausschnitts. Ist der Ausschnitt stark vergrößert, bietet es sich an, ein Fenster mit einer Übersicht über den Graphen zu öffnen. Das zeigt Abbildung 5-24 am Beispiel des Vererbungs-Graphen.

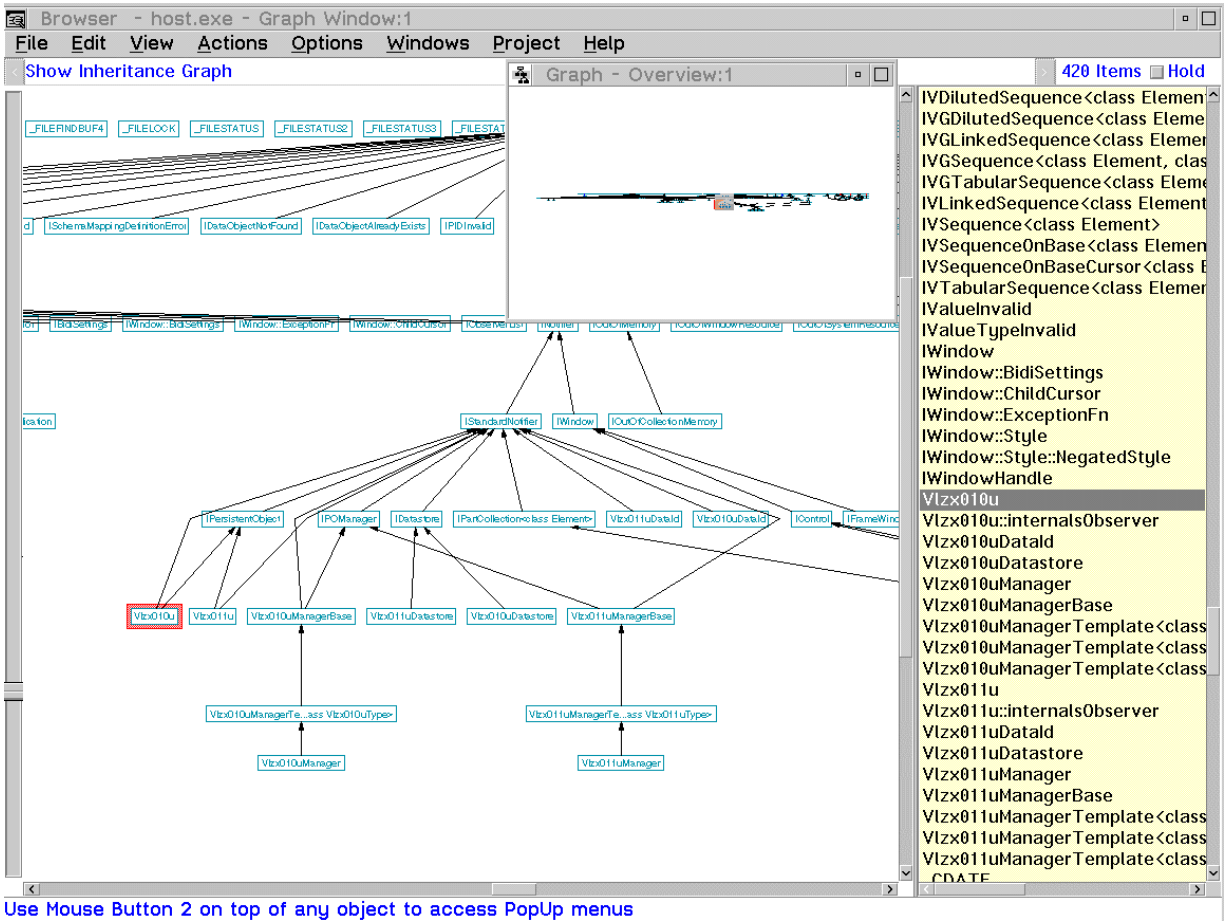

Abb. 5-24: Browser - der Vererbungs-Graph einer Anwendung.

Die gleichen Funktionen wie für den *Include*-Graphen stehen auch für den Vererbungs-Graphen zur Verfügung. In Abbildung 5-24 ist zu sehen, daß in der rechten Spalte ein Objekt ausgewählt wurde. In der Mitte oben (im Übersichtsbild) kann man etwa erkennen, wo sich das gesuchte Objekt befindet. Links in der vergrößerten Ansicht des Graphen ist das Objekt (rot) markiert und somit deutlicher zu sehen. Von hier aus könnten nun alle Objekte gefunden werden, von denen das markierte Objekt Eigenschaften erbt.

5.4.3 Performance Analyzer

Als letztes Tool soll der *Performance Analyzer* vorgestellt werden. Mit dem *Performance Analyzer* läßt sich herausfinden, welche Funktionen des Programms in welcher Reihenfolge aufgerufen werden, und wieviel Rechenzeit für einzelne Schritte notwendig ist.

Abbildung 5-25 zeigt das Hauptfenster des *Analyzers*, von dem aus weitere Untersuchungen vorgenommen werden können.

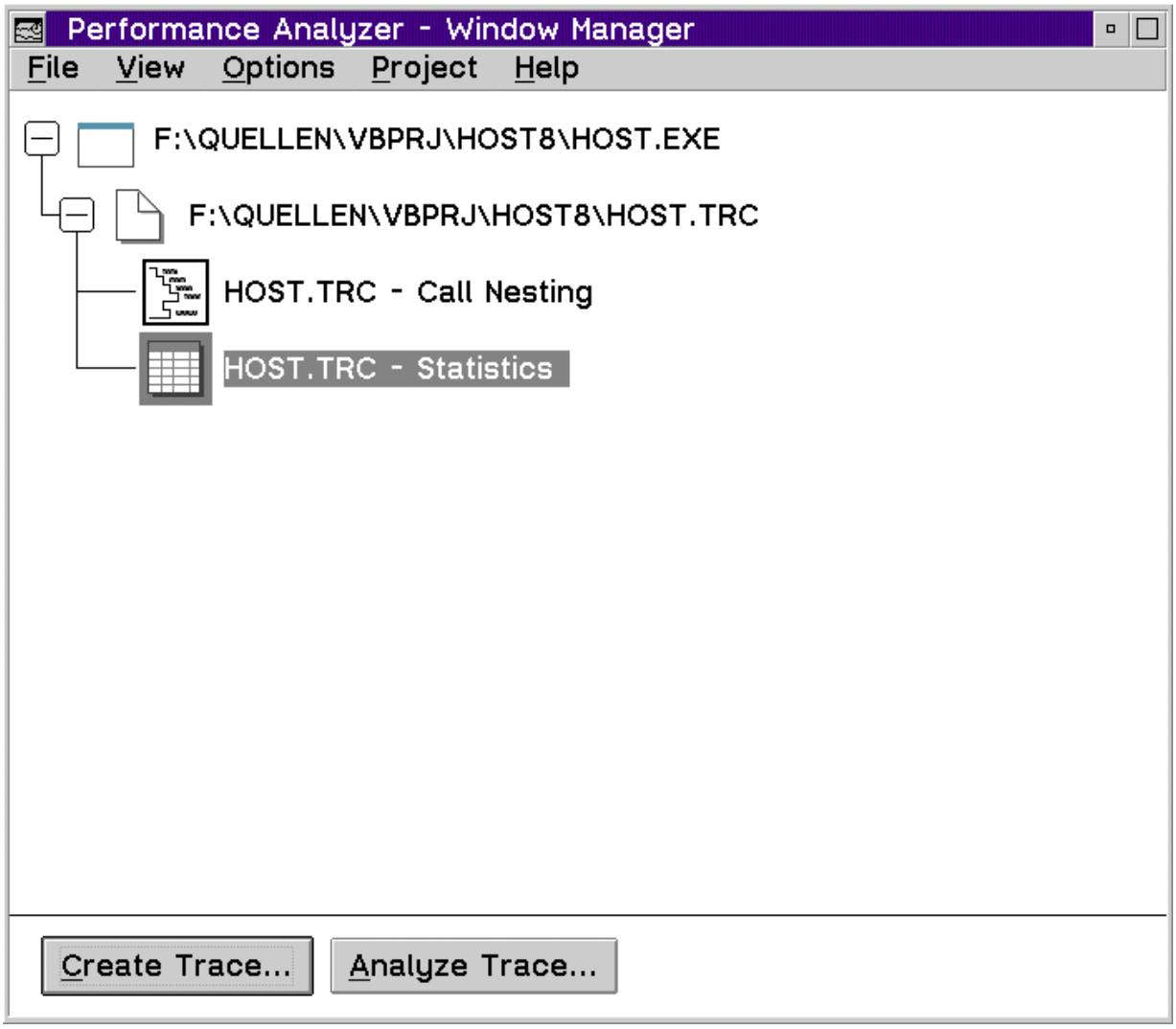

Abb. 5-25: Hauptfenster des *Performance Analyzers*.

Informationen für diese Tools (Debugger, Browser, *Performance Analyzer*) können bereits während des Compilierens mit erzeugt werden. Dann ist es möglich, eine Anwendung zu Analysieren, bevor sie zu einem lauffähigen Programm gelinkt wurde.

5.5 Probleme und Problembehebung

5.5.1 Fixes

Leider muß festgestellt werden, daß das Programmpaket Visual Age C++ 3.0 einige Fehlfunktionen enthält, die nur mit sogenannten *Bugfixes* (fehlerbehebenden Programmen) zu beseitigen sind. Im Laufe der Anwendungsentwicklung wurden mehrere Stufen dieser *Bugfixes* eingespielt.

Eine solche schwerwiegende Fehlfunktion war beispielsweise, daß der *Data Access Builder* keinen Zugriff zu DB2/2 der Version 2.1 und darüber fand, obwohl in allen Beispielprogrammen eine DB2/2 Datenbank dieser Version gefordert wurde. Andere Fehler, wie z.B. des Editors oder auch fehlende *Resource-DLLs* für Icons wurden beseitigt. Die vorliegenden Programme wurden mit folgendem Fixstand erstellt:

- ctc305
- cto305
- ctw302
- ctv304
- ctd301
- ctu302

Diese *Bugfixes* können auf einer CD bestellt[65] werden. Sie sind auch auf vielen FTP-Servern abgelegt. Die einzelnen Buchstaben hinter dem Kürzel „ct“ weisen auf das Tool hin, das erneuert wird.

5.5.2 Compiler- und Linkerfehler

Häufig kommt es beim Compilieren des generierten Codes zu Fehlermeldungen, die sich auf externe Funktionen beziehen. Auch Fehlermeldungen bezüglich Standardbibliotheken treten auf.

Solche Fehler sind oft auf die Benutzung von *Templates* zurückzuführen. Werden Template-Klassen verwendet, so legt *Visual Age* ein Unterverzeichnis unter dem derzeitigen Arbeitsverzeichnis an. Dieses Verzeichnis heißt *Tempinc* und enthält die Dateien (mit

[65] Bei OS/2 Insight AVI-Verlag. Die sogenannten „CSDs“ erscheinen monatlich.

Endungen cpp und h) der Template-Klassen. Diese werden während der Generierung des Quellcodes benötigt.
Während des Compilierens führt die Existenz dieses Verzeichnisses jedoch zu unerklärlichen Fehlern. Deshalb sollte nach Abschluß der Generierung das Verzeichnis entfernt werden.
Eine weitere Ursache für suspekte Fehlermeldungen sind auch die in den Zwischenschritten der Programmübersetzung erzeugten Objekt-Dateien (Endung obj). Leider enthält das durch *Visual Builder* oder *Data Access Builder* erstellte *Makefile* keinen Eintrag zum Löschen dieser Dateien. Deshalb sollten obj-Dateien dann gelöscht werden, wenn an der Anwendung etwas geändert und neuer Quellcode generiert wurde.
Der Compiler ist nicht in der Lage zu erkennen, daß es sich um obj-Dateien handelt, die älter sind als der Quellcode. Er benutzt daher die alten Dateien. So kommt es, daß trotz Änderungen im Programm ständig die gleichen Fehler auftreten. Vor neuerlichem Übersetzen des Programms sind deshalb alte obj-Dateien zu entfernen.
Hinweise zu weiteren Fehlern, die häufig auftauchen und auf die Konfiguration der *Visual Age Tools* zurückzuführen sind, findet man in der Online-Hilfe im Buch „*Frequently Asked Questions*“[66].

[66] Siehe Literaturverzeichnis zu [VAI95]

6. Abschließende Bemerkungen

6.1 Einschätzung der erstellten Programme

Für die vorliegende Arbeit wurden drei Anwendungen angefertigt, die in Kapitel 4 genauer beschrieben sind. Im Rahmen dieser Anwendungsentwicklung wurden graphische Oberflächen erstellt, und Daten aus Datenbanken ausgelesen. Dabei handelte es sich um relationale Datenbanksysteme (DB2/2 und DB2).
Zunächst wurde dabei auf eine lokale Datenbank zugegriffen, in der folgenden Ausbaustufe Daten einer entfernten DB2/2 Datenbank gelesen und im letzten Schritt eine mehrstufige Client/Server-Anwendung entwickelt. Mit dieser letzten Anwendung war es dann möglich, Daten einer Großrechnerdatenbank für den PC verfügbar zu machen.

Die erstellten Anwendungen sollten zeigen, ob es möglich ist, die Vorteile moderner Entwicklungswerkzeuge zu nutzen, dabei aber die gewachsenen Strukturen zu unterstützen und Datenbestände einzubeziehen. Diese Aufgabenstellung wurde gelöst.
Als modernes Entwicklungswerkzeug kam Visual Age C++ Version 3.0 zum Einsatz. Mit diesem Programmpaket war es möglich, mehrere Anwendungen komplett visuell zu erstellen. Alle Programme arbeiten vollständig objektorientiert. Es wurde versucht, die vorhandenen Objekte so zu nutzen, daß keine Codierung mehr nötig war. Auch dies ist gelungen.
Vorhandene Datenbanken (DB2/2 und DB2) wurden genutzt. Besonderes Augenmerk lag dabei auf der Einbeziehung des Mainframes und der Daten, die in DB2 für MVS abgelegt wurden. Es konnte nachgewiesen werden, daß ein solcher Zugriff prinzipiell möglich ist.
Einschränkungen sind dabei jedoch:

- lesender Zugriff
- direkte SQL-Zugriffe

Das heißt: Für den Zugriff auf entfernte Datenbanken wurden die Client/Server-Funktionalitäten von DB2 bzw. DB2/2 genutzt. Eine eigene Schnittstelle wurde nicht implementiert. Somit kann auf die entfernten Datenbanken nur mit direkten SQL-Abfragen

zugegriffen werden. Im Rahmen der Nutzung vorhandener Großrechnerprogramme ist ein solcher Zugriff nicht erwünscht. Vielmehr sollten hier vorhandene Module die Datenbankabfragen vornehmen. Dies ist für die Konsistenz der Datenbank von großer Bedeutung. Weitere Untersuchungen auf diesem Gebiet werden sich daher mit der Anbindung eines Clients an existierende Module beschäftigen.

Eine weitere Einschränkung bedeutet der nur lesende Zugriff auf die Mainframedaten. Da auf der Testdatenbank unter DB2 für MVS nur mit Sichten gearbeitet werden durfte, konnte im Rahmen dieser Arbeit nur nachgewiesen werden, daß ein lesender Zugriff möglich ist. Es ist bisher nicht gezeigt worden, inwieweit auch Änderungen und Einfügungen in die Großrechnerdatenbank erfolgen können. Auch hier könnten weitere Arbeiten ansetzen.

Schließlich wurden die Untersuchungen nur auf DB2 und DB2/2 eingeschränkt, da der *Data Access Builder* nur für diese Datenbanken ausgelegt war. Hier könnten weitere Versuche mit anderen relationalen Datenbanksystemen stattfinden.

Auch die Performance der erzeugten Anwendungen läßt, besonders beim Großrechnerzugriff, zu wünschen übrig. Werden gleiche Abfragen vergleichsweise unter TSO/ISPF und den enstprechenden Datenbankwerkzeugen gestellt (z.B. in einer 3270-Emulation), so ergeben sich in den Antwortzeiten beträchtliche Unterschiede. Ein Grund dafür, daß hier die Client/Server Programme langsamer arbeiten, ist sicher die Weiterleitung der Anfragen über mehrere Stationen. So müssen z.B. die Abfragen über verschiedene Netzwerke unter verschiedenen Protokollen weitergegeben werden. Diese Umsetzung zwischen den Protokollen erfordert einen höheren Aufwand. Auch die graphische Darstellung der Daten vergrößert sicher die Antwortzeit.

Die Auslastung des LANs und des DB2/2-Servers spielen bei den Verzögerungen auch eine wichtige eine Rolle. Hier wären ebenfalls weiterführende Untersuchungen denkbar.

6.2 Visuelle objektorientierte Programmierung

Während der Erstellung der Beispielanwendungen wurde visuell objektorientiert gearbeitet. Diese neue Art des Programmierens bringt mehrere Vorteile. Dazu zählen:

- Mitarbeiter aus Abteilungen/Firmen, für die neue Software erstellt werden soll, können schon von Beginn an sehen, wie das Endprodukt aussehen wird und

können mit an der Gestaltung der Oberfläche arbeiten, da zum Anordnen der Parts kein Fachwissen im Gebiet der DV notwendig ist. Die Software wird ergonomisch gebaut und erlebt eine größere Akzeptanz.

- Alle Module des Programms und ihr Zusammenspiel sind übersichtlich dargestellt. Pflege- und Wartungsarbeiten sowie Erweiterungen der Software werden vereinfacht.
- Ein derart erstelltes Programm kann auch leichter von anderen Anwendungsentwicklern übernommen werden. Der Quellcode muß nicht mehr gelesen werden, um auch Einzelheiten im Programmablauf verstehen zu können.
- Für einfache Anwendungen muß kein umfangreiches Wissen in C++ erworben werden, die Programmstruktur wird auf einer abstrakteren, visuellen Ebene entworfen.
- Syntaxfehler, wie vergessene Semikola etc., treten nicht auf, wenn die Anwendung komplett aus dem *Visual Builder* heraus generiert wurde.
- Die objektorientierte Programmierung unterstützt die Modularisierung großer Anwendungen und die Wiederverwendung von Softwarebausteinen.
- Die Bibliothek vorhandener Objekte kann um eigene Objekte erweitert werden.

Andererseits bringt die Programmierung in solchen Entwicklungsumgebungen auch Nachteile mit sich:

- Fehlermeldungen beim Generieren oder Compilieren der Anwendung verweisen zwar auf die entsprechende Stelle im Quellcode, allerdings muß der Umgang mit diesen Fehlermeldungen erst erlernt werden, da er kaum Rückschlüsse auf Tätigkeiten an der Oberfläche zuläßt.
- Der Programmierer muß auf die bereits vorhandenen Teile zurückgreifen. Erfüllen diese die gestellten Anforderungen nicht, muß er selbst neue Teile kreieren, die zu den vorhandenen passen sollen. Das erfordert wiederum sehr gute Kenntnisse in C++. Das legt den Schluß nahe, daß zusätzlich auch Systemprogrammierer eingesetzt werden müssen, die neue Objekte erstellen und pflegen. Alles in einer Hand zu lassen, würde den Arbeitsumfang beträchtlich erhöhen und von den

fachlichen Problemen ablenken, denen ja durch objektorientierte Programmierung größere Aufmerksamkeit geschenkt werden sollte.

- Änderungen, die in den erzeugten Dateien vorgenommen wurden, werden beim nächsten Generieren überschrieben. Besondere Vorsicht ist also geboten, wenn eigene Prozeduren o.ä. mit in das Programm einbezogen werden sollen.

Zusammenfassend kann gesagt werden, daß die visuelle objektorientierte Programmierung mit einer Reihe von Vorteilen aufwartet. Demgegenüber stehen jedoch noch einige Fehler des Produktes. Wünschenswert wäre eine Fehlermeldung des Compilers, die Rückschlüsse auf die Arbeiten an der Entwicklungsoberfläche des *Composition Editors* zuläßt. Es ist zumindest anfänglich für den Entwickler nicht von großem Nutzen, die entsprechende Stelle des Quellcodes zu sehen, den er nicht selbst geschrieben hat.
Darüber hinaus ist die Einarbeitungszeit in dieses komplexe Werkzeug recht lang, wenn man vorher noch nie mit ähnlichen Tools gearbeitet hat.
Andererseits muß darauf hingewiesen werden, daß Anwendungen recht schnell erstellt werden können, wenn der Programmierer mit dem Produkt vertraut ist. Hier kann eine hohe Produktivität erreicht werden. Als Beispiel sei noch einmal der Vererbungsgraph der dritten Beispielanwendung gezeigt (Abbildung. 6-1). Das Programm wurde in vergleichsweise kurzer Zeit erzeugt. Es wäre nicht möglich gewesen, eine solche Anwendung mit dieser Komplexität in gleicher Zeit ohne die Unterstützung eines Tools wie Visual Age C++ herzustellen.

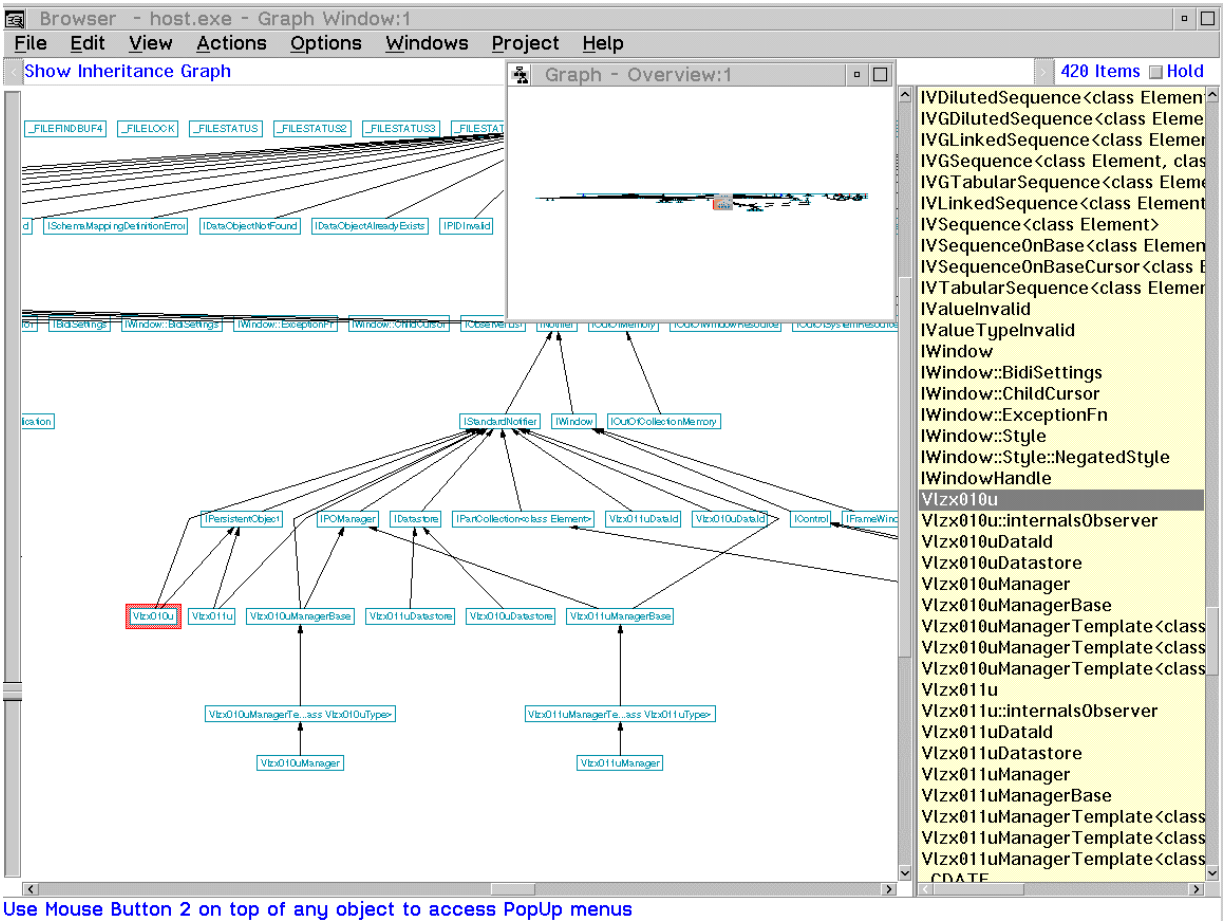

Abb. 6-1: Der Vererbungsgraph der Anwendung „host.exe“.

Gerade die Möglichkeit, komplexe Strukturen innerhalb kürzester Zeit erarbeiten und umsetzen zu können zeichnet die visuelle Programmierung aus.

Trotzdem soll nicht unerwähnt bleiben, daß das Programm-Entwicklungs-Paket *Visual Age C++* die Ressourcen des Rechners[67] voll ausschöpft und man sich während der Entwicklung meist schon die nächste Rechnergeneration oder noch mehr Hauptspeicher wünscht. Das erstaunt, da vom Hersteller als Minimum ein PC mit einem 80386 Prozessor und 24 MB Hauptspeicherkapazität gefordert wird.

[67] Zur Konfiguration siehe Kapitel 4 dieser Arbeit.

6.3 Ausblick

Abschließend kann festgestellt werden, daß mit Hilfe visueller, objektorientierter Programmierung leistungsfähige Software erzeugt werden kann, die die Vorteile alter und neuer Systeme miteinander verknüpft. Trotz anfänglicher Schwierigkeiten wird diese Art der Programmierung in die Anwendungsentwicklung Eingang finden. Sie gestattet es, graphische Oberflächen in kürzester Zeit zu erstellen, Objekte anzulegen, zu pflegen und auch das Zusammenspiel der Objekte im Hintergrund einfach und übersichtlich darzustellen. Sie unterstützt die objektorientierte Modellierung, die wiederum Geschäftsprozesse vielschichtiger abbilden kann. So werden z.B. Aggregation und Generalisierung unterstützt. Prozesse teilen sich nicht mehr in Daten und Funktionen sondern können gesamtheitlich betrachtet werden.

Auch mit der vielfältigen Manipulation von Objekten auf der Arbeitsoberfläche („*Drag and Drop*“) ergeben sich neue Möglichkeiten, die entwickelten Programme mit Standardsoftware zu kombinieren und gemeinsam zu nutzen.

Die objektorientierte, visuelle Programmierung zeigt sich somit als zukunftsträchtige Alternative zur herkömmlichen Anwendungsentwicklung.

7. Anhang

7.1 Glossar

Alias

Eine Bezeichnung, die meist kurz und einfach zu merken ist und die für einen längeren (meist schwer zu merkenden) Namen steht.

API

Application Programming Interface: Anwendungsprogrammschnittstelle. Definierte Schnittstelle, über die Anwendungsprogramme Systemdienste oder Dienstleistungen anderer Programme verwenden können

APPC

Advanced Program-to-Program Communication: Programmierschnittstelle, die eine Programm-Programm-Kommunikation unter Verwendung der SNA LU 6.2 Protokolle gestattet. APPC ist die SNA-Implementierung von CPI-C.

Authentication

Überprüfung, ob die übertragenen Daten (von einer Person/einem Programm) tatsächlich von dieser/m stammen.

Client

Ein Computer oder auch ein Prozeß, der Daten, Services oder Ressourcen anderer Computer oder Prozesse im Netzwerk nutzt.

Client/Server

Anwendungs-Architektur-Modell, nach dem die Client-Systeme die Anwendungsprogramme ausführen. Dazu nutzen sie teils eigene, teils fremde Ressourcen einer oder mehrerer Rechner um ihre Arbeiten durchzuführen. Dabei werden die Anforderungen eines Clients an die Server-Ressourcen über ein Netzwerk geleitet.

Compiler

Computerprogramm, welches Quellcode aus einer Hochsprache in ein ausführbares Programm umsetzt.

CPI-C

Common Programming Interface-Communications. Netzwerkschnittstelle zur Inter-Programm-Kommunikation. Produkt-übergreifender „de facto“-Standard - z.B. von X/Open lizensiert.

Datenbank

Sammlung von Daten, die Fakten über einen spezielle Anwendung der realen Welt darstellen.

Datenbank-Managementsystem

„Schnittstelle“ zwischen den Benutzern und der Datenbank. Ein DBMS verwaltet eine/mehrere Datenbank(en). Es kontrolliert Zugriffe von Benutzern und Anwendungsprogrammen; stellt sicher, daß die Daten gegen Hard-und Softwarefehler resistent sind.

Datenbanksystem

Ein Datenbanksystem besteht aus einem Datenbank-Managementsystem und einer gewissen Anzahl von Datenbanken.

DBMS

Database Management System. Siehe Datenbank-Managementsystem.

DDL

Dynamic Link Library - eine Bibliothek, deren Funktionen erst zur Laufzeit des Programmes hinzugebunden werden (Ggs.: Statische Bibliotheken). Dynamische Bibliotheken können gleichzeitig von mehreren Anwendungen genutzt werden. Sie werden vom Betriebssystem verwaltet.

DRDA

Distributed Relational Database Architecture. Verteilte Architektur relationaler Datenbanken. Eine Architektur der IBM, die es relationalen Datenbanken gestattet, in einem Netzwerk zusammenzuarbeiten und Daten auszutauschen.

GUI

Graphical User Interface: graphische Benutzerschnittstelle. Diese Schnittstelle ermöglicht es dem Benutzer mittels Mausklick Anwendungen zu starten, zu kontrollieren oder Daten zu bearbeiten. Die Arbeitsweise solcher Oberflächen versucht der gewohnten Arbeit am Schreibtisch näher zu kommen und den Benutzer vom Erlernen von Systembefehlen zu befreien. Die eigentliche Datenstruktur kann

verborgen bleiben, der Benutzer arbeitet mit „Ordnern“ oder „Papierkörben“. Beispiel: Windows. Als eine Form der graphischen Benutzerschnittstellen sind auch OOUIs anzusehen.

Host

1) Im TCP/IP-Sinn bezeichnet Host einen Computer, der es einem Anwender erlaubt mit anderen Computern in einem Netzwerk zu kommunizieren.
2) Andere Bezeichnung für Großrechnersysteme, wie z.B. Mainframes.

IP

Internet Protocol: Internet Protokoll. Ein Übertragungsprotokoll der Netzwerkschicht.

LAN

Local Aerea Network: Lokales Netzwerk, örtlich begrenzt auf etwa 1,5 km Ausdehnung. Ein Netzwerk, in dem Computer und Endgeräte (z.B. Drucker) miteinander verbunden sind um Daten auszutauschen und gemeinsame Ressourcen zu nutzen.

LU 6.2

Logical Unit 6.2: Logische Einheit 6.2. Ein Peer-to-Peer Kommunikations-Protokoll, das die Interoperabilität zwischen Programmen unterstützt. Entwickelt durch die IBM, Bestandteil von SNA. Andere Bezeichnung: APPC.

Mainframe

Großrechnersystem, wie z.B. das IBM System/370 oder System/390, meist ausgestattet mit dem Betriebssystem MVS oder VM.

MVS

Multiple Virtual System. IBM-Betriebssystem für Mainframes.

Name Server

Ein Serversystem unter TCP/IP, welches Netzwerkadressen in Namen umsetzt und umgekehrt.

NetBIOS

Network Basic Input Output System. Eine LAN-Schnittstelle zur Inter-Programm-Kommunikation auf der Ebene der Sitzungsschicht (entsprechend dem OSI-Modell).

Objekt

Der Grundbaustein objektorientierter Programmierung sind die Objekte. Sie bestehen aus Daten und Methoden. Methoden sind Aktionen, die auf den Daten des Objekts durchgeführt werden können. Jedes Objekt besitzt eine frei zugängliche Schnittstelle (*Public Interface*) die festlegt, auf welche Art und Weise andere Objekte oder Anwendungen mit dem Objekt interagieren können. Ein Objekt hat auch eine „private“ Komponente: dort werden die Methoden beschrieben.

Objekte gelten als „gekapselt“, d.h. ihre interne Struktur bleibt allgemein verborgen.

Objektorientierte Programmierung

Methode der Programmierung, die auf Objekten aufbaut und darüber hinaus mit Kapselung, Vererbung und Polymorphie arbeitet. So können wiederverwendbare Bausteine erzeugt werden, ohne Sourcecode weiterzugeben.

OOUI

Object Oriented User Interface: Objektorientierte Benutzerschnittestelle. Graphische Oberfläche zum Bearbeiten von Daten oder Ausführen von Anwendungen (ähnliche Funktionalitäten wie GUI). Beispiel: *Presentation Manager*.

PM

Presentation Manager. Objektorientierte graphische Benutzerschnittstelle von OS/2 Warp. (vergleiche OOUI)

Quellcode

„Hochsprachen“-text eines Computerprogramms (z.B. C-Code) Der Quellcode wird häufig auch als *Source Code* bezeichnet.

SNA

Systems Network Architecture. Netzwerkarchitektur der IBM, stark Großrechner-gerprägt.

SOM

Systems Object Model. Eine umfangreiche, Sprach-unabhängige Technologie zum Erzeugen und Manipulieren von Objekten. SOM-Objekte können sowohl in objektorientierten als auch in prozeduralen Programmiersprachen verwendet werden.

SQL

Structured Query Language. Strukturierte Anfragesprache. Genormte Sprache für relationale Datenbanksysteme, die es ermöglicht Daten einer relationalen Datenbank abzufragen oder zu ändern.

TCP/IP

Transmission Control Protocol/ Internet Protocol. Ein weit verbreitetes Netzwerkprotokoll zur Kommunikation zwischen (heterogenen) Rechnern.

Terminal Emulation

Software, die es einem PC oder einer Workstation gestattet, wie ein abhängiges Host-Terminal zu agieren.

Thread

Parallel abgearbeiteter Ablaufpfad innerhalb eines Prozesses (Programms). Vorteil gegenüber parallelen Prozessen: gemeinsamer Adreßraum, einfacherere Verwaltung. Ein Thread verhält sich zu einem Prozess, wie eine Task zu einem Multitasking-Programm.

Verteilte Datenbank

Zu einer verteilten Datenbank gehören mehrere physische Datenbanksysteme, deren Daten dem Anwender wie eine einzige Datenbank erscheinen. Die Datenbanksysteme sorgen dafür, daß Ort und Verteilungsform der Daten für den Anwender nicht sichtbar werden.

VM

Virtual Machine - Betriebssystem der IBM für Mainframes.

7.2 Abbildungsverzeichnis

Abb. 2-1: Verteilungsformen einer Client/Server-Anwendung 12
Abb. 2-2: Originale Host-Dialogmaske (aus dem Händlerarbeitsplatz). 14
Abb. 2-3: Gleiche Eingabemaske wie Abb. 2-2, geliftet 15
Abb. 3-1: Entity-Relationship-Modell einer Liegeplatzverwaltung (nach [Pü94], S.15). 18
Abb. 3-2: Ausschnitt aus dem *UPM* mit einem Beispielnutzer 23
Abb. 3-3: Eintragungen zu einem fernen Knoten unter dem Transportprotokoll TCP/IP 24
Abb. 3-4: Ausschnitt aus dem Systemdatenbankverzeichnis des Clients. 25
Abb. 3-5: Eintragungen zu einer fernen Datenbank. (im *Database Director*). 26
Abb. 3-6: Beispiel für die Interaktion der Datenbankverzeichnisse bei einem Serverzugriff. 28
Abb. 3-7: Client/Server-Datenbankanbindung über ein DDCS/2-Gateway 30
Abb. 4-1: Schematische Darstellung der Konfiguration. 32
Abb. 4-2: Fenster der zweiten Ausbaustufe der Anwendung; Serverzugriff 34
Abb. 4-3: Zugriff auf Mainframedaten in der dritten Anwendung. 37
Abb. 5-1: WorkFrame während der Erzeugung des zweiten Anwendungsprogramms. 41
Abb. 5-2: Ausschnitt aus dem „Projekt-Setup“ des *Workframes.* 42
Abb. 5-3: Ausschnitt aus dem Tools-Setup. 43
Abb. 5-4: Hauptfenster von „Projekt Smarts“ mit vrogefertigten Projektumgebungen 45
Abb. 5-5: Startmenü des *Data Access Builders* 47
Abb. 5-6: Der *Data Access Builder* - Übersicht über die Datenbanken 48
Abb. 5-7: Der *Data Access Builder* Zuordnung einer Klasse zu einem Datenbankobjekt. 49
Abb. 5-8: Ausschnitte aus der config.sys, Einstellungen der Pfade 52
Abb. 5-9: Der *Enhanced Data Access Builder* 54
Abb. 5-10: Zusammenfassende Darstellung der mit dem *Enhanced Builder* erzeugten Klassen 55
Abb. 5-11: Das *Visual Builder* Fenster mit bereits geladenen Klassen. 57
Abb. 5-12: Der *Composition Editor* während der Erstellung der dritten Anwendung. 60
Abb. 5-13: Der *Class Editor* - Ansicht der dritten Anwendung 61
Abb. 5-14: Der *Part Interface Editor* - Ansicht der Host-Anwendung 62
Abb. 5-15: Der *Composition Editor*, Programmierung der zweiten Beispielanwendung. 67
Abb. 5-16: Der *WorkFrame* mit den Dateien der Host-Beispielanwendung. 69
Abb. 5-17: Einstellung der Compileroptionen im *WorkFrame*. 70
Abb. 5-18: Ausschnitt aus dem *Makefile* der Anwendung „host“. 71
Abb. 5-19: Zentrales Debugger-Fenster. 73
Abb. 5-20: Debugger - Auszug der Registerinhalte. 74
Abb. 5-21: Debbuger. Sicht auf den Quellcode. 74
Abb. 5-22: Zentrales Browser-Fenster. 76
Abb. 5-23: Beispiel für einen *Include*-Graphen. 77

Abb. 5-24: Browser - der Vererbungs-Graph einer Anwendung. **78**
Abb. 5-25: Hauptfenster des *Performance Analyzers*. **79**
Abb. 6-1: Der Vererbungsgraph der Anwendung „host.exe“ **87**

7.3 Tabellenverzeichnis

Tabelle 3-1: Umsetzung E/R-Modell nach Relationenmodell **19**
Tabelle 3-2: Struktur der Berechtigungen unter DB2/2 **20**
Tabelle 3-3: Katalogeinträge einer Datenbank **25**
Tabelle5-1 : Übersicht über die Compileroptionen. **72**

7.4 Literaturverzeichnis

[DB295]

DB2-Information. Online-Handbücher des Programmpakets DB2/2 Version 2.1. Eine Auswahl (alphabetisch geordnet):

1) API Reference
2) Application Programming Guide
3) Command Reference
4) Fehlernachrichten
5) Informationen und Konzepte
6) Master Index
7) Problembestimmung
8) SQL Reference
9) Systemverwaltung

[Kra95]

Krantz, Steve. *Real World Client/Server: learn how to successfully migrate to client/server computing from someone who’s actually done it!*. Maximum Press USA, 1995.

[NC96]

„SNA-Connectivity: Die Herausforderung“. *N&C. Networks&Communications*. 9 (1996): 58-68.

[Nie95]

Niemann, Klaus D. *Client/Server-Architektur - Organisation und Methodik der Anwendungsentwicklung*. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1995.

[Pür94]

Pürner, Heinz Axel, und Pürner, Beate. *DB2/2 kompakt - Professioneller Einsatz des Datenbank-Managementsystems unter OS/2*. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1994.

[OrfHar93]

Orfali, Robert, and Harkey, Dan. *Client/Server-Programming with OS/2 2.1 - third Edition*. Van Nostrand Reinhold, New York, 1993.

[Vos94]

Vossen, Gottfried. *Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme*. Bonn; Paris; Reading, Mass. [u.a.]. Addison-Wesley, 1994. 2., korrigierter Nachdruck 1996.

[Wen95]

Wenzel, Paul (Hrsg.). *Geschäftsprozeßoptimierung mit SAP-R/3 - Modellierung, Steuerung und Management betriebswirtschaftlich-integrierter Geschäftsprozesse*. Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1995.

[VAH95] *Visual Age C++ - How Do I ...?* Online Handbücher des Programmpakets Visual Age C++, Version 3.0, alphabetisch geordnet:

1) Browser - How Do I?
2) Data Access Builder - How Do I?
3) Debugger - How Do I?
4) Editor - How Do I?
5) Performance Analyzer - How Do I?
6) Visual Builder - How Do I?
7) Visual Age C++ - How Do I?
8) WorkFrame - How Do I?

[VAI95] *Visual Age C++ Information*. Online Handbücher des Programmpakets Visual Age C++, Version 3.0. Eine Auswahl (alphabetisch geordnet):

1) Building Visual Age C++ Parts for Fun and Profit.
2) C Library Reference
3) C/C++ Language Reference
4) C/C++ Programming Guide
5) C/C++ User's Guide
6) Frequently Asked Questions
7) Guide to Sample Programs
8) Open Class Library Reference
9) Open Classs Library User's Guide
10) Visual Builder Parts Reference
11) Visual Builder User's Guide
12) Welcome to Visual Age C++
13) What's New

Diese Angaben beziehen sich auf: „First Edition, June 1995. This edition applies to Version 3.0 of IBM VisualAge C++ for OS/2 (30H1664, 30H1665, 30H1666) and to all subsequent releases and modifications until otherwise indicated in new editions."

7.5 Trademarks/Warenzeichen

In der vorliegenden Arbeit wird eine Reihe von (Software-) Produkten genannt. Die nachstehende Auflistung nennt die entsprechenden Inhaber der Warenzeichen.

Warenzeichen der International Business Machines Corporation (IBM)

APPC

APPN

Communications Manager

CPI-C

Database Manager

DB2

DB2/2

DDCS/2

MVS/ESA

NetBIOS

OS/2

Presentation Manager

SNA

SOM

System/370, System/390

Visual Age C++

Visual Lift

VM/ESA

3270

Warenzeichen der Entra Data

Phantom

Warenzeichen der Intel Corporation

Pentium

Warenzeichen der Unix Systems Laboratories, Inc.

UNIX

Warenzeichen des Massachusetts Institute of Technology

X-Window

7.6 Ausschnitte aus dem Quellcode

bla

bla

8. Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, daß ich die vorliegende Arbeit selbständig erarbeitet und verfaßt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Ulrike Sieber
Leipzig, 3. April 1997