Dokumentation zu BernHist III

Informatikprojekt von Urs Dietrich und Christian Kaufmann

bei Professor Oscar Nierstrasz

Institut für Informatik und angewandte Mathematik
Universität Bern

Inhaltsverzeichnis

Vorwort

Die vorliegende Arbeit wurde von Christian Kaufmann und Urs Dietrich als Hauptstudiumsprojekt im Fach Informatik durchgeführt. Hauptanliegen war die Portierung der bestehenden Anwendung BERNHIHST, die bis anhin unter VMS lief, auf Windows NT beziehungsweise auf PC. Auftraggeber für diese Arbeit war Prof. Christian Pfister vom Historischen Institut der Universität Bern.

Die vorliegende Dokumentation ist einerseits als Protokoll für die geleistete Arbeit, aber auch als technische Referenz für die Wartung und die Pflege von BERNHIST zu verstehen. Wir verzichten in diesem schriftlichen Teil auf Wiederholungen wie z.B. die Art und Weise, wie BERNHIST entstanden ist und wie es sich bis anhin entwickelt hat. Ebenso lassen wir wichtige technische Teile wie das ER-Schema aus, da dies alles schon in der vorhergehenden Version genauestens beschrieben worden ist.[1]

Nach den einleitenden Kapiteln folgen zu jeder der drei Hauptarbeiten (Datenübernahme, CD-Version, Internetanbindung) je ein separates Kapitel, sowie verschiedene Anhänge.

Der Stand der Dinge entspricht einen Zwischenstand auf dem Weg zu einem kommerziellen Produkt; im Anhang C finden sich zusammengefaßt Vorschläge, Erweiterungsmöglichkeiten und Wünsche, die im Wesentlichen aufgrund der Betatests aufgekommen sind.

[1] Peter Häberli, Klaus Imfeld: HGIS BERNHIST V2.0. Systemdokumentation und Leitfaden zur elektronischen Speicherung und Bearbeitung historisch-statistischer Quellen.

Ziele des Projektes

Im Verlaufe der Arbeit an diesem Projekt wurden die Zielvorgaben nach und nach angepaßt und konkretisiert. Aus der ursprünglichen Variante, die aus einer Telnetapplikation bestand, sollten im wesentlichen zwei Hauptprodukte hervorgehen; eine neue, angepaßte Internetapplikation und eine zweite Anwendung, die lokal auf einem PC verfügbar sein sollte. Mit dem Erscheinen des Berner Atlas, der Ausschreibung eines Lehrerfortbildungskurses für Gymnasiallehrer und nach zahlreichen Tests mit möglichen Anwendern wurde festgelegt, das zweite Produkt in zwei Varianten anzubieten, als Light-Version mit geprüften Datenbeständen und als Gesamtversion mit allen verfügbaren Daten und dem konkreten Hinweis für Forschende, bei Verwendung dieser Datenbestände für wissenschaftliche Publikationen Plausibilitätstests durchzuführen. Die Light-Version soll in erster Linie als Begleitprodukt für den Atlas und als Lehrmittel für Berufsschulen und Gymnasien dienen.

Mit dieser Zielsetzung gingen wir an die Arbeit und merkten rasch einmal, dass die Arbeit damit nicht getan sein würde. Zum einen war die Art und Weise, wie vorhandene Daten in der alten Version auf die VMS upgeloadet wurde, kompliziert. Mit der Umstellung auf einen NT-Server beziehungsweise auf „Wintel-Systeme“ hätte dieser Teil so oder so neu bearbeitet werden müssen. Dadurch, dass wir bestehende, noch nicht eingelesene Datenbestände einzulesen hatten, wurde auch klar, dass das bestehende Fileformat seine Tücken aufweist. Außerdem bestand kein Konzept, wie der Datenfluß als Ganzes sowie die Wartung auszusehen hat.

Unter dem Strich resultiert nun zum einen eine Internetapplikation sowie zwei CD-Versionen und Vorschläge für die nächsten Administratoren des Systems, wie man den Datenfluß kontrollieren könnte und welche Tools für den Anwender im Archiv zur Verfügung zu stellen sind, um eine reibungslose, einfache Möglichkeit zum Erfassen neuer Daten zur Verfügung zu stellen.

Die Tatsache, dass einige „Altlasten“ aufzuarbeiten waren und dass intensive Abklärungen über die Wünsche und Ziele zwischen dem Auftraggeber, den Programmierern und dem breiten Publikum, bestehend aus Schülern, Historikern, Geographen etc., stattfanden, hat dazu geführt, dass wir auf die Ausarbeitung eines kompletten Paketes mit allen nötigen Tools für die weitere Pflege aus Zeitgründen verzichtet haben.

Generelles Vorgehen

Das hier beschriebene generelle Vorgehen gilt für die Arbeiten bis zu dem Punkt, an dem die spezifischen Arbeiten für die Internetversion bzw. die CD-ROM Version beginnen.
Die erste Etappe begann mit der Aufnahme von Gesprächen mit Prof. Pfister als Auftraggeber, dem letzten Programmierteam von BERNHIST und Leuten, die das Produkt seit längerer Zeit kennen und intensiv damit gearbeitet haben. Damit wurden die groben Zielvorgaben festgelegt, spezifische Wünsche für die neue Version herausgearbeitet und mögliche Schwachstellen und Probleme mit der alten Version eruiert. Danach wurde die Funktionalität der alten Version untersucht und grundsätzlich als Leitvorgabe festgelegt, dass sich die neue Version im Funktionsumfang der alten Version anzugleichen habe und dass zusätzlich eine gute Exportfunktion implementiert werden sollte.
Als dieses grobe Raster feststand, wurden die Files mittels FTP auf einen lokalen Rechner übernommen und zuerst einmal anhand des ER-Schemas sortiert; es stellte sich heraus, dass ein Großteil der Files zur Programmierung der alten Oberfläche und der Netzanbindung gedient hatten. Diese wurden aussortiert und als Backup separat gelagert. Danach haben wir das ER-Schema nochmals untersucht. Es erwies sich nach wie vor als richtig und tauglich für die gestellte Aufgabe.
Anschließend wurden die Files Stück für Stück in Access eingelesen, was meist keine Schwierigkeiten machte, außer dort, wo es Sonderzeichen zu behandeln galt; es stellte sich heraus, dass bei einigen dieser Sonderzeichen Fehler auftraten. Dieser Fehler wurde mit einer einfachen Suchen/Ersetzen-Funktion ausgemerzt. Nach Abschluß dieses Schrittes wurden die Datentypen überprüft und in einen geeigneten, von Access bereitgestellten Typ überführt. Mit dem kompletten Vorliegen aller Tabellen konnten die einzelnen Relationen miteinander verknüpft werden. Nach Abschluß dieses Schrittes stand die Grundversion, die einerseits Christian Kaufmann ermöglichte, die Internetanbindung zu programmieren, andererseits Urs Dietrich die Grundlage für die Stand-alone-Lösung lieferte.

Datenübernahme

Bisheriger Datenfluß

Bisher wurden Daten von interessierten Studenten im Archiv oder an der Uni in einem einfachen Textfile aufgenommen. Sie wurden zentral gesammelt und nach einiger Zeit auf die VMS geladen. Das Laden in die VMS-Datenbank geschah in mehreren Schritten mit verschiedenen Programmen. Dabei wurde nach Fehlern geforscht und diese dann per Hand korrigiert. Der genaue Vorgang mit den einzelnen Schritten kann nicht nachvollzogen werden.

Integration alter Daten

Neben der relationalen Datenbank, befanden sich auf der VMS noch eine Menge vorerfasster Textfiles mit Daten. Diese Daten mußten auch noch in die Datenbank integriert werden. Die Verwendung der dazu vorhandenen Programme auf der VMS war nicht möglich. Es gibt verschiedene Gründe dafür: Der Zeitdruck war gross, da die VMS außer Betrieb genommen werden sollte. Eine klare Anleitung für die Programme gab es nicht oder war zumindest nicht auffindbar. Die Datenfiles waren von sehr unterschiedlicher Qualität, was die Einhaltung der Formatierungsrichtlinien betraf. Eine direkte Portierung der VMS-Programme auf Windows NT erschien uns zu kompliziert.

Zur Erstellung der neuen Einleseprogramme wählten wir Delphi (Object Pascal). Auch hier gab es verschiedene Gründe: Der Zugriff auf Access-Dateien ist mit Delphi einfach möglich. Die Komponentenbibliothek von Delphi (VCL) war mir sehr gut bekannt. Zudem waren mit der VCL und aus früheren Projekten sehr gute und mächtige String-Handling-Routinen vorhanden, welche zur Analyse und Verarbeitung der Datenfiles nötig waren.

Grundsätzlich habe ich versucht, die effektive Funktionalität und das GUI (Graphical User Interface) auch im Code in separaten Modulen abzulegen. Das GUI ist "Quick and Dirty", der restliche Code aber bereits sauber programmiert. Die einzelnen Funktionen laufen in der vorliegenden Version stabil, was noch fehlt, ist eine einfache Benutzeroberfläche mit umfassenden Checks, um Fehlmanipulationen abzufangen und zu verhindern.

Das neue Einleseprogramm besteht aus verschiedenen Teilen:

Checking
Diese Funktion liest ein Datenfile, überprüft das Dateiformat und kontrolliert, ob alle verwendeten Themen- und Ortschlüssel in der Datenbank gefunden werden. Bei den Ortschlüsseln werden drei Varianten erkannt: BERNHIST-Code, BfS-Code, sowie die direkte Ortsbezeichnung, zum Beispiel "Burgdorf".

Printing
Ein Unterprogramm um von Datenfiles den Header mit der Themenzuweisung auszudrucken. Das BERNHIST - Datenmodell sieht vor, jedes Thema einem Standardthema zuzuordnen. Zum Beispiel werden Daten zum Thema "Neugeborene" oder "Taufen" beide dem Standardthema "Geburten" zugeordnet. Diese Zuordnung ist für einen Historiker mit Hilfe einer gedruckten Liste einfacher.

Import
Das eigentliche Einleseprogramm, welches die bereinigten Datenfiles in die Datenbank importiert.

Probleme

Die vorerfassten Daten, teilweise Textfiles, teilweise Excel-Tabellen, waren von sehr unterschiedlicher Qualität. Kleinere Unstimmigkeiten wurden von Hand in den Dateien korrigiert. Bei "Fehlern" die häufig vorkamen, versuchten wir, das Einleseprogramm entsprechend anzupassen, was nicht immer einfach ist, da es dabei zu unerwarteten Seiteneffekten kommen kann.
Weiter mußten wir feststellen, dass verschiedene Daten doppelt vorhanden sind. Ein klar strukturierter Ablauf zur Datenerfassung, sowie weitere Tools zur Datenpflege sollten deshalb unbedingt Bestandteil der weiteren Arbeiten an BERNHIST sein.[2]

[2] siehe dazu auch Kapitel "Verbesserungs- und Erweiterungsvorschläge".

Die CD-ROM Version

Einleitung

Die CD-ROM Version wurde unter Access 97 programmiert. Die Gründe für die Wahl dieses Produktes waren vielfältig. Es sollte ein Produkt zur Anwendung kommen, das einen möglichst großen Marktanteil hat, um a) die größtmögliche Sicherheit zu haben, dass es nicht so schnell vom Markt verschwindet und b) um die Wahrscheinlichkeit zu erhöhen, dass interessierte Historiker das DBMS bereits kennen. Schließlich soll das Produkt in Zukunft weiter gepflegt werden, und es ist wünschenswert, dass dies auch Angehörige der Forschungsstelle tun können, wenn Sie bereit sind, sich entsprechend einzuarbeiten. Das betrifft nicht nur BERNHIST III, sondern auch EuroClimHist, das ebenfalls an dieser Forschungsstelle entwickelt wird. Weiter muß die Möglichkeit bestehen, CD-ROM´s mit dem Programm zu vertreiben, ohne dass der Käufer das Datenbankprogramm an sich bereits erworben hat; es muß also möglich sein, eine Runtime-Version zu erzeugen. Damit blieb in der engeren Wahl neben FileMaker, das auf Mac wie auf PC-Ebene einsetzbar ist, eigentlich nur Access, das einerseits eine rasche Entwicklung ermöglicht und andererseits ein Zusatzpaket (ODE-Tools) zur Verfügung stellt, mit denen man bequem Runtime-Versionen herstellen kann. Da für EuroClimHist bereits etliche Routinen und Prototypen in Access bestehen, entschied ich mich für Access.

Vorgehen

Als Ausgangspunkt für die Programmierung wurde die alte Netzversion genommen und zu Beginn versucht, auf einer möglichst einfachen Oberfläche die wichtigsten Elemente anzuordnen. Dazu gehören die Auswahlmöglichkeiten für die Räume, also für Wahl nach Kanton, Landesteil, Amtsbezirk, Kirchgemeinde und Einwohnergemeinde sowie die dazugehörenden Orte. Die zweite Auswahlmöglichkeit betrifft das Thema und die dritte den gewünschten Zeitraum. In der alten Version konnte man eine unbeschränkte Zahl von Orten anwählen, ebenso mehrere Themen und einen Abschnitt auf der Zeitachse. Die jetzige Version erlaubt ebenfalls mehrere Orte, aber nur ein Thema[3].

[3] Weiter unten wird erläutert, weshalb hier nur ein Thema ausgewählt werden kann.

Hier wurde eine erste Testphase eingeschaltet; Christian Pfister begutachtete den Prototypen der Oberfläche, gab Anregungen und korrigierte die grundsätzliche Bedienung, wo er es für nötig hielt. Zu diesem Zeitpunkt wurde auch festgelegt, dass das Formular so gestaltet werden sollte, dass der User sich von oben nach unten, vom Raum zum Thema und schlußendlich zur Zeitachse, durchzuarbeiten hatte.

Nun wurde das Formular gemäß dieser ersten Besprechung angepaßt, verschiedene Möglichkeiten für die Anordnung der Resultate am Bildschirm implementiert und 4 Standardformate für den Export festgelegt[4]. Weiter mußten die Bildschirmmasken für die Anzeige der Resultate programmiert werden. Damit waren die Grundfunktionen der alten Version inklusive der (neuen) einfachen Exportfunktion implementiert. Es folgte als nächster Test eine etwas umfangreichere Ausgabe der Programme an das letzte Programmierteam, das ja bestens mit dem Produkt vertraut war, an interessierte HistorikerInnen und an eine Klasse der Gewerblich-Industriellen Berufsschule Bern, der GIBB[5]. Nach einer längeren Testphase wurde der Prototyp an der Pressekonferenz über den Historischen Atlas des Kantons Bern auch kurz der Presse vorgestellt, um die Arbeitsmöglichkeiten für Schüler mit den beiden Instrumenten Atlas und Datenbank aufzuzeigen. Aus diesem Pool an Betatestern kamen dann Rückmeldungen, die zum Teil sehr unterschiedlich ausgefallen waren; Historiker hatten zum Teil gänzlich andere Vorstellungen und Wünsche im Bezug auf Bedienbarkeit, Exportmöglichkeiten etc. als die testenden Schüler, und die Programmierer der alten Version wiederum achteten mehr auf Fragen der Performance, der Setuproutine oder der Art und Weise, wie die Abfrage im Hintergrund aufgebaut und durchgeführt wird.

Besprochen wurde zu diesem Zeitpunkt die Darstellung der Oberfläche mittels Reiter, ähnlich dem Applet, das auf dem Netz verfügbar ist. Das wäre durchaus eine Alternative gewesen, die ich aber aufgrund der Art und Weise, wie ich die DB abfrage, abgelehnt habe[6]. Zusätzlich kam dazu, dass Prof. Pfister eine Abfrage in der Reihenfolge Raum – Thema – Zeit für angebracht hielt.

---

[4] Als Grundformate stellt Access standardmäßig das Text, das Excel, das RTF und das HTML-Format zur Verfügung. Diese 4 Formate sollten in aller Regel für die Weiterverarbeitung reichen.

[5] An dieser Stelle sei all jenen, die sich als Betatester zur Verfügung gestellt haben, nochmals herzlichst gedankt! Ohne diese Hilfe hätte das Produkt nicht von so vielen unterschiedlichen Usern mit den verschiedensten Ansprüchen, Zugängen zum Thema dieser CD und auf so vielen verschiedenen Rechnerkonfigurationen getestet werden können.

[6] Auch hier wird weiter unten in einem separaten Teil nochmals ausführlicher auf diesen Umstand eingegangen.

Nachdem der Entschluß, grundsätzlich mit einer Oberfläche zu fahren, die nur gerade eine Bildschirmseite für die Abfrage zur Verfügung stellt, gefällt war, ging es an die Detailarbeit. Diese Detailarbeit verlief iterativ so, dass gewisse Features implementiert wurden, die neue Beta den Testern kompiliert wurde, diese ihre Rückmeldungen machten und mich damit wieder an die Arbeit schickten. Dabei wurde immer versucht, einmal diejenigen Punkte zu verbessern oder zu implementieren, die von möglichst vielen der Betatester reklamiert worden waren.

Probleme

Ich unterscheide in diesem Abschnitt drei „Klassen“ von Problemen. Das eine Problem war die Gestaltung des User-interfaces, das zweite die eingesetzten Tools und das dritte die Performance. Natürlich kann diese Grenze nicht scharf gezogen werden, die Problemkreise laufen ineinander über. Trotzdem werde ich für dieses Kapitel diese drei Bereiche getrennt besprechen.

Das User-Interface ist einer der heikelsten Punkte des Projektes; ich als Programmierer konnte mögliche Fehlmanipulationen relativ schlecht finden, da ich gewisse Vorgehensweisen intuitiv anwende, ohne mir zu überlegen, ob das auch der weniger geübte Anwender tut. Hier waren in Bezug auf den zeitlichen Aspekt auch beträchtliche Aufwendungen nötig. Daneben ist viel auch Sache des Geschmacks; was die einen als gelungene Alternative betrachteten, war für andere vom grafischen Standpunkt oder von der Benutzerführung her ungenügend.

Am meisten diskutiert wurde, was die Oberfläche betrifft, die Variante mit Reitern, also die Möglichkeit, für Raum, Thema und Zeit jeweils eine Art Karteikarte zu benutzen. Um auf einem auch relativ schwachen PC eine einigermaßen anständige Performance zu erreichen, schicke ich nach der Auswahl eines Raumes bereits ein SQL-Statement ab, um eine temporäre Zwischentabelle zu bilden. Dieser Vorgang wird ebenfalls nach der Wahl des Themas durchgeführt, um schnell anzeigen zu können, ob sich zu einer Raum/Themakombination auch Daten finden lassen. Der Hauptgrund liegt im Bestreben, später über weniger Tabellen joinen zu müssen, was sich in Tests als sehr langsam erwiesen hat. Damit wird klar, dass der User gezwungen werden muß, zuerst den Raum, dann das Thema und zuletzt die Zeitachse auszuwählen, was durch die Darstellung mit einem Formular, das von oben nach unten ausgefüllt werden muß, unterstrichen wird.

Zur Programmierung wurde einerseits Access 97 verwendet, andererseits die ebenfalls von Microsoft stammenden ODE-Tools, die für den Vertrieb von Datenbankanwendungen bestimmt sind. Um eine Anwendung auf einem fremden Rechner ohne Access installieren zu können, muß die Runtime-Version, die ihrerseits wiederum einige DLL´s mitinstalliert, verwendet werden. Wie sich gezeigt hat, ist das Produkt noch lange nicht voll ausgereift; es werden ohne Vorwarnung DLL´s überschrieben, die teilweise zu ganz eigentümlichen Fehlern führten. Zum Beispiel konnte je nach Konfiguration der RTF-Editor von Windows nicht mehr benutzt werden, die Deinstallation funktionierte über die Systemsteuerung nicht richtig oder das E-Mailtool Eudora verweigerte den Dienst. Sauber nachvollziehbar waren die Fehler nie, sicher liegt es aber unter anderem an der jeweiligen Win95-Version. Weiter werden nicht in allen Fällen die eingestellten Settings für die DB übernommen, was zu Programmfehlern führt.[7]

Ein weiterer Punkt ist die Verwendung von ActiveX-Controls, die sich leicht in die diversen Programme einbinden lassen. Ich benutze zur hierarchischen Ansicht der Themen das von Microsoft mitgelieferte TreeView-Modul, das über seine Eigenschaften und Methoden relativ bequem zu programmieren ist. Hier ergaben sich zwei Probleme: Zum einen ist es nicht möglich, mehrere Elemente des Trees zu selektieren; damit fällt die Suche im Gegensatz zur vorhergehenden Version nach mehreren Themen in einer Abfrage weg. Ich beurteile diesen Aspekt als nicht sehr störend, denn selten werden zu mehr als 5 Themen Daten gesucht. Außerdem wäre es dann beispielsweise möglich, dass man nach zwei Themen sucht, aber nur zum einen der beiden auch Daten zurückgeliefert bekommt; der User wäre zu stark verunsichert und wüsste nicht, ob er etwas falsch gemacht hat. Dazu kommt, dass es gerade für den nicht sehr versierten Benutzer einfacher ist, den Output, den er beispielsweise in Excel in ein Diagramm umwandeln möchte, nur zu einem Thema zu haben.[8]

---

[7] All die genannten Probleme wurden mit der Microsoft-Hotline besprochen; ein ebenso langwieriger wie mühsamer Prozess, da man jedes Mal an einen neuen Techniker verwiesen wird. Eine direkte Nummer zu seinem Betreuer kriegt man nicht, und je nach der fachlichen Kompetenz des Technikers wird das Problem geleugnet oder als schon lange bekannt bezeichnet. Wartezeiten auf Fragen haben auch schon 2 Wochen gedauert, die Lösungen waren zum Teil nicht praktikabel, und beim dritten Anruf wird Geld verlangt. Alles in allem ist Microsoft ein denkbar ungünstiger Partner für eine Zusammenarbeit!

[8] Das oben angeführte Beispiel mag etwas extrem klingen; ich habe aber die Erfahrung gemacht, dass ungeübte Benutzer nach einer Abfrage und dem Export ins Excel vermeintliche Datenbankfehler

Die Performance schien anfänglich ein Problem zu sein, das jedoch bald einmal relativiert wurde. Da die DB nicht wie z.B. ein multimediales Lexikon zum Stöbern in den Datenbeständen einlädt, sondern als reines Recherchewerkzeug eingesetzt wird, ist die Performance sicher auch nicht eines der wichtigsten Kriterien; die Erfahrungen, die ich mit HistorikerInnen gemacht habe, zeigten auf, dass die erreichte Geschwindigkeit auf gängigen Pentiumsystemen durchaus genügt. Überdies wissen die meisten der wissenschaftlichen Benutzer bereits im voraus, welche Datenbestände sie suchen; wenn Sie die Bedienung dann im Griff haben, reduziert sich die Benützung auf das reine Abfragen. SchülerInnen werden nur die abgespeckte Version, die im Wesentlichen auf den Datenbeständen des Atlas beruht, benutzen, so dass hier das geringere Datenvolumen zu einer deutlich besseren Performance führt, so dass auch das „spielen“ mit den Datenbeständen in genügender Geschwindigkeit möglich ist.

ausmachten, weil beim Öffnen des Excelfiles die Spaltenbreiten überall gleich sind und deshalb auf den ersten Blick Resultate abgeschnitten schienen.

Die Internetversion

Einleitung

Neben der CD-Version, soll die Internetanbindung die Möglichkeit schaffen, jederzeit auf die aktuellsten Datenbestände zuzugreifen. Dabei soll sich die Benutzeroberfläche möglichst an die der CD-Version anpassen. Es war klar, dass die Internetanbindung, im Gegensatz zur CD-Version, nicht von einer bestimmten Betriebssystem-Plattform abhängig sein darf.

Daraus entstand folgendes Konzept: Suchen und Bereitstellen der Daten mit Hilfe eines CGI-Programmes. Aufruf des CGI-Programmes über ein einfaches HTML-Formular oder über ein Java-Applet. Das HTML-Formular bietet die Möglichkeit, auch mit älteren und exotischen Web-Browsern schnell auf die Datenbank zuzugreifen. Dafür müssen die Orte und Themen durch Eintippen der jeweiligen ID's gewählt werden. Das Java-Applet bietet eine moderne grafische Benutzeroberfläche, wo Themen und Orte je aus einem übersichtlich strukturierten Baum ausgewählt werden können.

Das Resultat wird in jedem Fall als HTML-Tabelle angezeigt. Die Ausgabe als reines Textfile oder als Excel-Tabelle wurde bewußt nicht integriert. Dies geschah auf Wunsch von Prof. Christian Pfister, der dieses Feature nur in der kommerziellen CD-Version zur Verfügung stellen will.

Vorgehen

Das Vorgehen war vielleicht nicht in jedem Fall der direkteste Weg, da ich mich mit diesem Projekt zum ersten Mal mit CGI-Programmen, Java-Applets die über CGI Datenbankzugriffe machen, sowie dem vorgegebenen Produkt, Microsoft Information Server, beschäftigt habe.

In einem ersten Schritt erstellte ich das serverseitige CGI-Programm und testete dieses mit Anfragen über ein HTML-Formular. Als diese Teile stabil funktionierten, erstellte ich das Java-Applet, welches direkt mit dem CGI-Programm kommuniziert.

Server

Bei der Erstellung des CGI-Programmes waren einige Randbedingungen gegeben: Der Server läuft mit Windows NT 4.0 und Internet Information Server. Die Daten stehen als Access Datenbank zur Verfügung. Aus diesen Gründen und auch weil von den Programmen zum Datenimport bereits einige benötigten Klassen vorhanden waren, fiel die Wahl auch hier auf Delphi (Object Pascal).

Es gibt ein CGI-Programm, welches als Parameter immer ein COMMAND erhält. Je nach COMMAND sind weitere Parameter nötig und wird ein anderes Resultat geliefert. Für folgende Aufgaben sind COMMANDS sind definiert: Liste mit allen Orten - Liste mit allen Themen - Datumsbereich vorhandene Daten zu Thema/Orten - Abfrageresultat aus der Datenbank als HTML-Seite.

Bei der Programmierung habe ich versucht, die Klassenstruktur so zu gestalten, dass allgemein verwendbare Teile, wie Kommunikation über die CGI-Schnittstelle oder das Erstellen von HTML-Dateien von BERNHIST-spezifischem Code getrennt sind. Der Zugriff auf die Access Datenbank erfolgt direkt über die Delphi eigene Database Engine (BDE). Das Programm läuft als EXE-Datei, so dass bei jedem Request ein Connect auf die Datenbank aufgebaut werden muß. Man könnte das verbessern, wenn man das Programm als DLL über die ISAPI-Schnittstelle in den Internet Information Server integrieren würde. Für die gegenwärtigen Zugriffszahlen genügt die einfachere EXE-Lösung vollauf.

Client

Beim Erstellen des HTML-Formulares gab es keine Probleme, es brauchte nur ein paar Eingabefelder, sowie eine Aktion, welche dann die Abfrage startet.

Bei der Gestaltung der grafischen Oberfläche des Applets, habe ich möglichst vieles von der CD-Version übernommen. Das Applet bietet beim Starten die Auswahl der gewünschten Dialogsprache. Dazu wurden sämtliche Texte als Konstanten in einer Klasse zusammengefaßt. Es wird dann jeweils entweder eine französische oder eine deutsche Instanz geladen. Als nächstes werden vom Server die Listen mit Themen und Orten heruntergeladen. Dies geschieht in zwei parallelen Threads, welche neben dem Main-Thread gestartet werden. Das Applet basiert außer der Treeview-Komponente auf

Standard-Java-Controls. Das Abfrageresultat wird jeweils in einem neuen Browser-Fenster angezeigt.

Die Entwicklung erfolgte mit Visual Cafe 2.0 von Symantec. Dieses Tool bietet einerseits das benötigte Treeview-Control und konnte auch die leeren Code-Strukturen für das Eventhandling automatisch generieren. Der erstellte Code basiert weitgehend auf Standardklassen, was es ermöglicht, das Applet klein zu halten. Das JAR-File ist nur 41 Kilobyte gross. Zudem funktionieren die generierten CLASS-Files auch auf Unix-Plattformen, was zum Beispiel bei der Verwendung von Microsoft Visual Java nicht immer der Fall ist.

Probleme

Serverseitig waren die Probleme sehr gering. Die Laufzeitumgebung ist klar definiert, Delphi als Entwicklungsumgebung bietet alle nötigen Tools, um Programmfehler schnell zu finden und zu beheben. Die Zugriffszeiten auf die Datenbank über die BDE sind auch genügend gut, vermutlich sogar schneller als mit der CD-Version, welche mit Microsoft Access selber programmiert wurde.

Clientseitig sieht es etwas anders aus. Java ist in Realität (noch) nicht so plattformunabhängig wie man immer hört. Die aktuelle Version der BERNHIST-Abfrage läuft sicher mit den neusten Version von Netscape (Windows, Unix) und Internet Explorer (Windows). Andere Plattformen haben wir nicht getestet. Das Debugging von Applets in den Web-Browsern ist außerordentlich mühsam, da kein eigentlicher Source-Code-Debugger zur Verfügung steht. Mit der Arbeit wurde deshalb auch klar, warum heute erst sehr wenige interaktive kommerzielle Internetseiten auf Java-Applets basieren. Die meisten arbeiten mit einfacheren HTML-Formularen, weil diese Art deutlich weniger fehleranfällig ist. Es ist zu hoffen, dass in diesem Bereich noch eine bessere Standardisierung stattfindet, vor allem auch in den Details.

Verbesserungs - und Erweiterungsvorschläge

Unserer Ansicht nach sollte ein Tool bereitgestellt werden, das auch auf schwachen Maschinen, ev. in einer 16-bit-Umgebung, die Aufnahme neuer Daten erlaubt. Die neuen Daten können weiterhin in einem Textfile abgelegt werden, sollten aber bereits während der Aufnahme zumindest teilweise kontrolliert werden. Dabei sollten detaillierte Fehlermeldungen dem User erlauben, schnell die Unstimmigkeiten zu finden und sie zu korrigieren.

Damit würde sichergestellt, dass das Einlesen in die Datenbank mit bedeutend weniger Aufwand möglich wäre. Es sollte auch versucht werden, die Daten jeweils sofort nach der Erfassung in die Datenbank zu integrieren. Erfahrungsgemäß sind sowohl die Zeit wie die Finanzen an der Universität knapp; ein Aufschieben der anstehenden Arbeiten führt daher zwangsläufig früher oder später zu größeren Arbeitsschüben, die oftmals zusätzlich mit Terminen kollidieren. Deshalb ist auf alle Fälle darauf zu achten, dass die Pflege kontinuierlich erfolgt.

Zur Datenpflege sollte ein Tool erstellt werden, welches es erlaubt, zum Beispiel doppelte Einträge zu finden oder ganze Datenbereiche einfach einem anderen Thema zuzuordnen. Weitere Inhalte des Pflichtenheftes für dieses Tool müßten noch mit dem BERNHIST - Verantwortlichen, Prof. Christian Pfister besprochen werden.

Dazu kommen weitere Arbeiten, die aus unser Sicht notwendig, um das ganze als kommerzielles Produkt zu vertreiben:

- Kontrolle/Elimination der Daten („fehlerhafte“ Abfragen, Aggregationen etc.)
- Integration des BFS-Codes für den Output
- Endgültiges Pflichtenheft für die kommerzielle Version erstellen und umsetzen (siehe dazu die teils handschriftlichen Notizen der Betatester)
- Ausarbeitung der Oberflächen und Zusatzinfos mit Hilfe von R. Brechbühl
- Ausarbeitung und Fertigstellung am Einlesetool für die Übernahme neuer Datenbestände. Eventuell damit verbundene Änderungen am Fileformat vornehmen.

Erfahrungen

Zwei Erfahrungen sind zentral, die für weitere Projekte dieser Art berücksichtigt werden müssten. Die eine Erfahrung bezieht sich auf die Zusammenarbeit mit dem Auftraggeber, die zweite bezieht sich auf die Pflege von solchen Werkzeugen bei Leuten, die nicht auf professionelle Unterstützung über die Dauer des gesamten Lebenszyklus bauen können.

Wir haben zu Beginn versucht, den Rahmen der zu leistenden Arbeiten abzustecken. Das ist uns mehr oder weniger gut gelungen. Was wir nicht berücksichtigt haben, sind Altlasten des Systems, die erst zutage traten, als es effektiv um die Implementation ging. Konkret heisst das hier, dass der Auftraggeber zwar eine konkrete Vorstellung davon hatte, was das neue Produkt leisten sollte, aber nicht genügend Ueberblick über das gesamte Produkt hatte, um in den Zeitplan Aufgaben wie den längst fälligen Import bestehender Datenbestände einzubeziehen. Dies führte hier dazu, dass ein separates Importtool geschrieben werden musste, das diese Aufgabe erledigen konnte. Weiter war der zeitliche Aufwand für das Testen in echten Umgebungen mit Usern, die zu ganz verschiedenen Zielgruppen gehören, weit zeitraubender als angenommen. Gymnasiallehrer setzten andere Schwerpunkte als ihre Schüler und diese wiederum ganz andere als die mit BernHist arbeitenden Historiker.

Dieser Umstand geht auch stark in die zweite angesprochene Erfahrung hinein; dieses Projekte wurde isoliert betrachtet als eine Portierung von VAX auf NT unter Aufsplitterung der resultierenden Datenbank in eine Stand-alone Lösung und in eine Internetapplikation. Gedanken über weiterführende Arbeiten und das Sicherstellen der Pflege, um mögliche Bugs zu reparieren oder Anpassungen in den nächsten Jahren an Hard- oder Software durchführen zu können, wurden zuwenig gemacht.

Daneben war die Arbeit an einem Projekt mit einem echten Arbeitgeber und einem echten Produkt sicher motivierender als eine Fragestellung mit einem vielleicht akademischeren Hintergrund. Die Zusammenarbeit in einem 2-er Team, das sich von den Kenntnissen her ergänzt, ist sicher ebenso eine Motivation, da die Zusammenarbeit an der Uni immer noch zuwenig gefördert bez. möglich ist und die meisten Arbeiten als Einzelarbeiten durchgeführt werden.

Empfehlenswert für solche Arbeiten scheint mir ein Zeitmanagement, das relativ viel Zeit für Gespräche mit dem Endabnehmer, für das Testen und für „Unvorhergesehenes“ reserviert. Wir schätzen, dass die effektive Arbeit, bestehend aus Portierung,

Programmierung des Einlesemoduls, Programmierung der Web-Anwendung bezw. der Stand-alone Lösung nur gerade die Hälfte der effektiv in Anspruch genommenen Zeit beansprucht hat. Weitere 30% sind etwa für die Tests mit richtigen Usern verbraucht worden, und der Rest verteilt sich auf Abklärungen mit dem Arbeitgeber und mit dem Verfassen der Dokumentation. Diese letzte, abschliessende Arbeit hat, jedenfalls aus meiner Warte, ebenfalls mehr Zeit in Anspruch genommen als zuvor angenommen. Zusammenfassend könnte man sagen, dass mit der eigentlichen Programmierung bezw. der Arbeit am Computer erst die Hälfte der Arbeit getan ist und dass das genaue Abstecken der gewünschten Leistungen zentral ist.

Anhang 1

Spezifikation

Allgemein

Die Auflistung der Arbeiten ist gegliedert in die drei Bereiche Import Daten, CD-Version und Internetanbindung.

Anforderungen an Hard- und Software

- Zielplattform NT, da ein NT-Server am Historischen Institut in Betrieb ist.
- Performance relativ unwichtig, da das Tool mit sehr wenigen Anfragen gleichzeitig konfrontiert wird. Daraus ergibt sich, dass der bestehende NT-Server mit einem PII 400 vollauf genügt.
- Wahl der Software muss auf gängiger, d.h. von der Forschungsstelle eingesetzte Software fallen. Wahl soll auf ein breit eingesetztes Mainstreamtool abstützen, um für weitere Arbeiten ev. auch interessierte Historiker hinzuzuziehen.

Import Daten

Kategorie	Beschreibung der Arbeiten	Status
zwingend	Portierung Daten von VMS auf Windows NT (Datenbank MS Access oder MS SQL Server	erledigt
zwingend	Import Daten, welche im Rohdatenformat (Textdatei) vorliegen.	erledigt
wünschenswert	Überprüfung Daten auf doppelte Einträge	erledigt
wünschenswert	Überprüfung Daten auf Korrektheit und Vollständigkeit, das heisst Überprüfung der Zuweisung der Standardthemen, teilweise überprüfen der erfassten Daten, Plausibilitätsprüfung durch Vergleich von Daten aus verschiedenen Jahren.	offen
wünschenswert	Aggregation von Daten auf Gemeindeebene für Ebenen Kanton / Region / Amt	offen
zwingend	Tool mit einfacher Oberfläche zum Einlesen von weiteren Daten, die mit einem (auch relativ schwachen Laptop) in Archiven aufgenommen werden.	offen

CD-Version (Funktionalitäten)

Kategorie	Beschreibung der Arbeiten	Status
	Implementation mit MS Access	erledigt
zwingend	Einfache grafische Oberfläche zur Eingabe der drei Suchbegriffe "Raum", "Thema", "Zeit"	erledigt
zwingend	Raum: Auswahl ein oder mehrere Orte Auswahl nach einzelnen Kategorien (Region / Amt / Gemeinde) Suchen Ort nach Namen	erledigt
zwingend	Thema:	erledigt

	Auswahl eines oder mehrerer Themen Strukturierte Anzeige der Themata	
zwingend	Zeit: Auswahl des gewünschten Zeitraumes. Das Programm soll aufgrund von Thema und Raum eine Liste mit den Jahren, wo Daten vorhanden sind, zur Verfügung stellen.	erledigt
zwingend	Ausgabe am Bildschirm und Drucker	erledigt
zwingend	Export als Text-, HTML, RTF- oder Exceldatei	erledigt
wünschenswert	Programmversionen in deutsch und französisch	offen
wünschenswert	Programmversion "Light" ohne Druck- und Exportmöglichkeiten. Zudem soll die Datenbank nur die Daten enthalten, welche als Grundlage für den "Historischen Atlas des Kanton Bern" dienten. „Schulversion“	offen
zwingend	Einfache Bedingungsanleitung (Online-Help)	erledigt

Internetanbindung

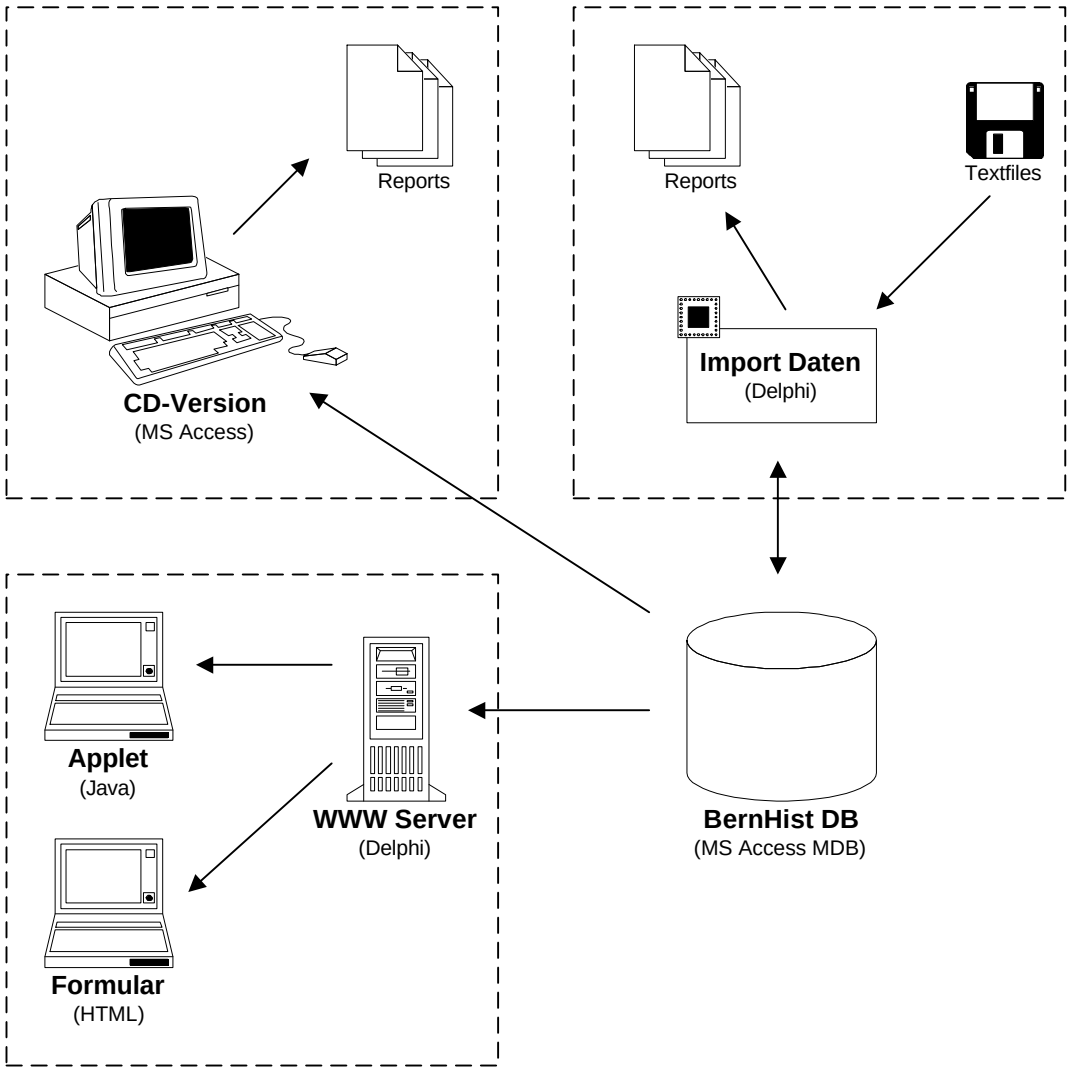
Kategorie	**Beschreibung der Arbeiten**	**Status**
zwingend	Einfache Abfragemöglichkeit über grafische Oberfläche	erledigt
zwingend	Plattformunabhängige Implementation	erledigt
zwingend	Auswahl Raum, Thema und Zeit wie bei CD-Version	erledigt
wünschenswert	Programmversionen in deutsch und französisch	erledigt
zwingend	Datenbasis entsprechend der CD-Version "Light"	offen
wünschenswert	Zugriffmöglichkeit auch mit älteren Webbrowsern (ohne Java VM - Implementation)	erledigt

Anhang 2

Programm Design

Übersicht einzelne Module

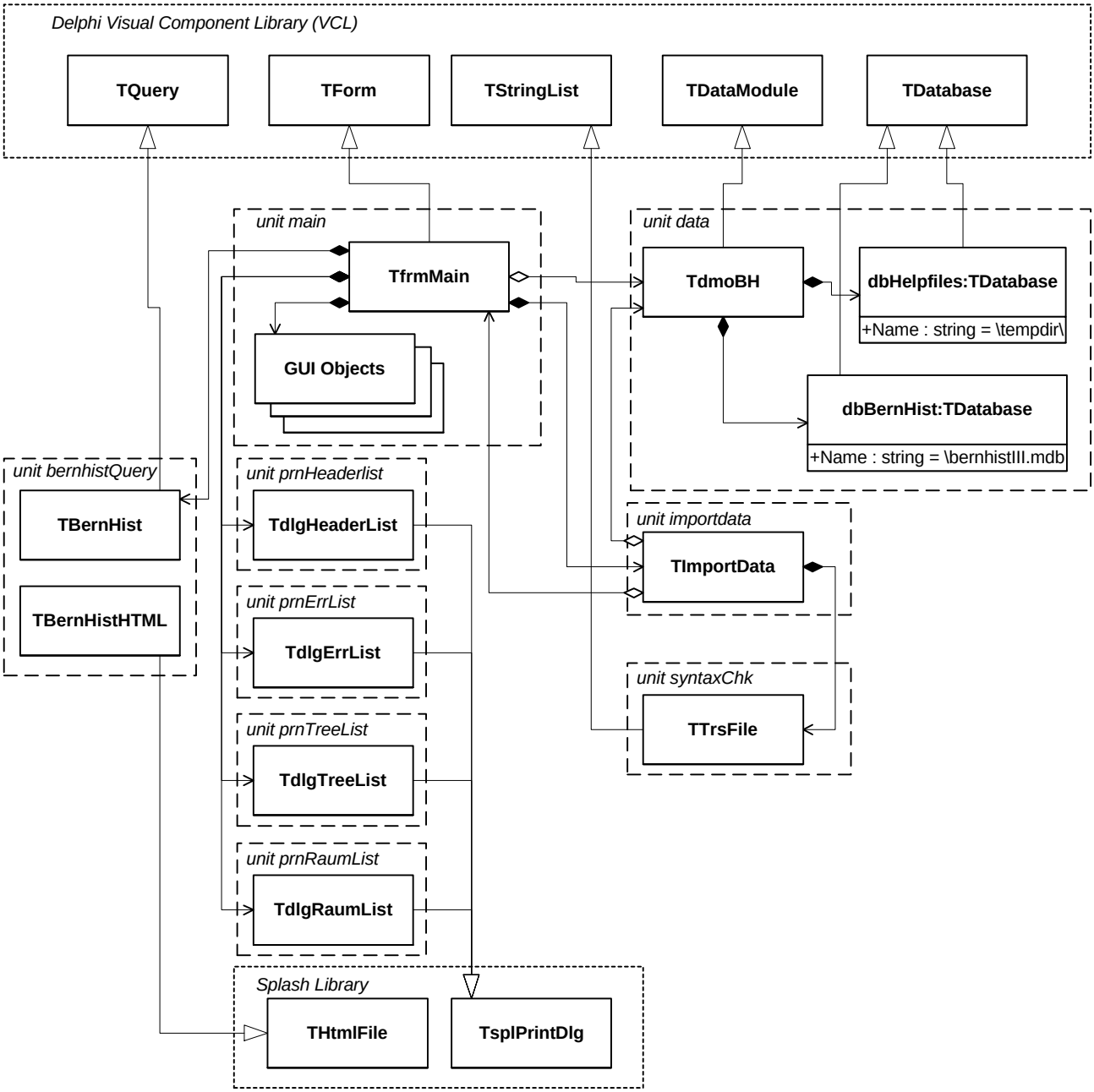
Die folgende Grafik gibt einen schematischen Überblick über alle vorhandenen und verwendeten Programmteile. Die Gruppierung ist dabei nicht unbedingt als physische, sondern als logische Einheiten zu verstehen.
Das Design der einzelnen Einheiten ist in den folgenden Kapiteln im Detail beschrieben. Die Struktur der Datenbank (ER-Schema) ist im Anhang 3 aufgeführt.



Übersichtsschema

Programm Import Daten

Das Importprogramm ist vollständig in Delphi implementiert. Zum Verständnis werden Kenntnisse der Visual Component Library (VCL) vorausgesetzt. Ebenfalls werden verschiedene Funktionen aus der Splash Library verwendet. Diese sind im Source-Code im Detail kommentiert. Auf eine vollständige Beschreibung dieser Klassen und Funktionen wird deshalb verzichtet. Ebenfalls werden die in der Delphi IDE automatisch generierten Methoden für einzelne Ereignisse nicht im Detail beschrieben.



Klassendiagramm des Importtools

Klasse **TBernHist**	inherites from **TQuery**^{VCL}	unit bernhistQuery

Aufgaben

Diese Klasse kapselt alle nötigen Abfragen für die Internetanbindung. Dabei wird über die Param... Attribute und Methoden vor einer Abfrage alle Parameter gesetzt. Anschliessend erfolgt der Aufruf einer Exec... - Methode. In der Resultattabelle kann mit den geerbten Methoden aus TQuery (siehe Delphi Onlinehelp) geblättert werden. Zudem stehen über weitere Properties formatierte Ausgaben des aktuellen Datensatzes zur Verfügung. Alle weiteren Spalten in der Resultatmethode werden über die geerbte Methode FieldByName(aFieldName:string) abgefragt.

Attribute / Properties

+ MdbFileName: string	Dateiname (mit vollständiger Pfadangabe) der verwendeten Bernhist - MDB - Datei.
+ ParamTERM: integer	Key des abzufragenden Standardthemas
+ ParamSTART: integer	Untere Grenze des Abfragezeitraumes (Jahr in Form 1900)
+ ParamEND: integer	Obere Grenze des Abfragezeitraumes
+ TermName:string	Bezeichnung des mit ParamTERM definierten Themas
+ TermType:string	Einheit des mit ParamTERM definierten Themas
+ SysName:string	Bezeichnung des Ortes (Raumes) des aktuellen Datensatzes. Nur gültig nach Anfrage mit ExecForData.
+ SysType:string	Kategorie des aktuellen Ortes (Kanton, Amt, etc.). Nur gültig nach Anfrage mit ExecForData.
+ StartJahr:string	Kleinstes Jahr, für das noch Daten gefunden wurden. Nur gültig nach einer Anfrage mit ExecForYear.
+ EndJahr:string	Grösstes Jahr, für das noch Daten gefunden wurden. Nur gültig nach einer Anfrage mit ExecForYear.
+ Wert:string	Formatierte Version des Datenwertes des aktuellen Datensatzes. Nur gültig nach Anfrage mit ExecForData.

Methoden

+ Create()	Konstruktor. Erstellt alle nötigen TDatabase und TTable-Objekte.
+ Destroy()	Destruktor. Gibt alle Objekte wieder frei
+ ParamSYSRemoveAll()	leert die Liste mit den Key's der abzufragenden Orte (Räume).
+ ParamSYSAdd()	fügt den Key eines Ortes (Raumes) in die Liste ein
+ ExecForYears()	führt aufgrund der gesetzten Parameter für Ort (SYS) und Thema (TERM) eine Abfrage auf der Datenbank durch und liefert als Resultat das minimale und maximale Jahr, wo Daten gefunden werden. Resultat in Properties StartJahr und EndJahr
+ ExecForData()	führt aufgrund der aller gesetzten Parameter eine Abfrage auf der Datenbank durch. Die Resultattabelle wird über die geerbten Methoden First, Next, EOF und FieldByName ausgewertet.
+ ExecForSYS()	Liefert alle Orte einer bestimmten Kategorie (z.B. Kirchgemeinden). Die Kategorie wird der Methode

	ExecForSYS als Parameter übergeben. Die Resultattabelle wird wie nach Aufruf von ExecForData ausgewertet.
+ ExecForTERM()	Keine Parameter nötig. Liefert alle Themen. Die Resultattabelle wird wie nach Aufruf von ExecForData ausgewertet.

Klasse **TBernHistHTML**	inherites from **THtmlFile***Splash*	unit bernhistQuery

Aufgaben

Diese Klasse erweitert die Standardklasse zum Erstellen von HTML-Files.

Methoden

+ Create()	Konstruktor. Fügt in defaultmässig leer erstellte HTML-Datei einen Header für Bernhist - HTML - Dokumente ein.
+ AddFootLines()	Diese Methode fügt am Schluss der HTML-Datei die Fusszeilen für ein Bernhist - HTML- Dokument ein.

Klasse **TdlgErrList**	inherites from **TsplPrintDlg***Splash*	unit prnErrList

Aufgaben

Diese Klasse stellt eine Routine zum Ausdrucken der aktuellen Fehlermeldungen zur Verfügung. Die Fehlermeldungen werden in einem MemoControl des HauptFormulars gespeichert. (siehe auch TfrmMain).

Methoden

+ Execute()	Diese Methode zeigt den Druckdialog an.

Klasse **TdlgHeaderList**	inherites from **TsplPrintDlg***Splash*	unit prnHeaderList

Aufgaben

Diese Klasse dient zum Ausdrucken der Headerinformation von einem oder meherer Rohdatenfiles (Textfiles). Dabei wird bei der Spalteninformation wahlweise das zugewiesene Standardthema mitgedruckt.

Attribute

- files: TsplFileList	Liste mit allen gewählten Textfiles, von welchen die Headerinformationen gedruckt werden sollen. Die Auswahl der Dateien erfolgt in der Klasse des Hauptfensters (TfrmMain) und wird der Methode Execute als zusätzlicher Parameter übergeben.

Methoden

+ Execute()	Diese Methode zeigt den Druckdialog an.

Klasse **TdlgRaumList**	inherites from **TsplPrintDlg***Splash*	unit prnRaumList

Aufgaben

Diese Klasse dient zum Ausdrucken aller vorhandenen Orte (Räume). Die Orte werden dabei alphabetisch sortiert nach Name / Kategorie gedruckt.

Methoden

+ Execute()	Diese Methode zeigt den Druckdialog an.

Klasse **TdlgTreeList**	inherites from **TsplPrintDlg***Splash*	unit prnTreeList

Aufgaben

Diese Klasse dient zum Ausdrucken aller vorhandenen Orte oder Themen in hierarchischer Struktur (baumartig). Dabei werden die Daten aus dem Objekt trvKatalog des Hauptfensters (siehe auch TfrmMain) verwendet. Es kann also jeweils nur der Tree der aktuell geladenen Daten gedruckt werden, entweder Orte oder Themen.

Attribute

- ttt : TTreeType	Typ der zu druckenden Daten (Themen oder Orte)
- tree : TTreeView	Referenz auf TTreeview in Hauptformular (siehe auch TfrmMain)

Methoden

+ Execute()	Diese Methode zeigt den Druckdialog an.

Klasse **TdmoBH**	inherites from **TDataModule***VCL*	unit data

Aufgaben

Dieses Datamodul ist ein Container für alle benötigen Data-Access Objekte wie TDatabase's, TTable's, und TQueries, welche im Importtool benötigt werden

Attribute

+ dbBernHist: TDatabase	Über dieses TDatabase-Objekt geschieht der Zugriff auf die MS Access Datenbank, sowohl lesen als schreiben.
+ dbHelpFiles: TDatabase	Dieses TDatabase-Objekt verweist auf ein temporärers Verzeichnis, wo sämtliche Hilfstabellen (Paradox-Format) angelegt werden.

Methoden

+ FlushAllBuffers()	Mit dieser Methode werden sämtliche Buffers von TTable's-Objekten auf die Festplatte geschrieben.

Klasse **TfrmMain**	inherites from **TForm***VCL*	unit **main**

Aufgaben

Diese Klasse definiert das Hauptfenster des Programms und verwaltet alle Objekte der grafischen Oberfläche. Eine Detailauflistung macht wenig Sinn, da diese Objekte in der Delphi IDE bearbeitet werden können. In der Folge sind also nur die BERNHIST - relevanten Attribute und Methoden beschrieben.

Attribute

- bhDaten: TBernHist	Dieses Objekt ist für den Import nicht nötig. Es ist nur zu Testzwecken für den serverseitigen Teil der Internetanbindung vorhanden

+ memErr:TsplMemo:	Memo - Control in welchem die Fehlermeldungen der letzten Überprüfung gespeichert werden (siehe auch TTrsFile).
+ trvKatalog: TTreeView	Treeview - Control welches entweder alle Orte oder alle Themen in hierarchischer Struktur anzeigt. Der Tree wird mit den Methoden LoadTermNode und LoadSysNode abgefüllt.

Methoden

- GlobalMessageHandler	In dieser Methode wird die Anzeigedauer der letzten Fehlermeldung überprüft und allenfalls die Statuszeile wieder geleert.
- GlobalErrorHandler	Globale Fehlerbehandlungsroutine. Zeigt die Fehlermeldung an und gibt Auswahldialog Ja/Nein ob Programmausführung abgebrochen werden soll
- UpdateStatusBar	Aktualisiert Anzeigen in Statusbar (CAPS, NUM, INS, etc.)
- LoadTermNode()	Lädt alle Standardthemen in das Anzeigefenster. Die Themen werden gemäss der Hierarchie in ein Treeview geladen.
- LoadSysNode()	Lädt alle Ortschaften in das Anzeigefenster. Die Orte werden gemäss der Hierarchie in ein Treeview geladen.
+ DisplayInfo()	Zeigt in der Statuszeile eine Information an
+ DisplayError()	Zeigt in der Statuszeile eine Fehlermeldung an.

Klasse **TImportData**	inherites from **TObject**^{VCL}	unit importdata

Aufgaben

Diese Klasse kapselt alle Methoden, welche zum einlesen einer Rohdatendatei (Textfile) nötig sind. Der Zugriff auf die Textdatei erfolgt mit Hilfe eines TTrsFile - Objektes. Beim Einlesen wird die Integrität der Datenbank sichergestellt. Der Zugriff auf die Datenbank erfolgt über die im DataModul TdmoBH vorhandenen TTable-Objekte.

Attribute

- FName: string	Dateiname der einzulesenden Textdatei.
- FObsKey: integer	nächster freier Primary Key in Tabelle OBS_BASE.
- FRdSourceKey: integer	Key zu aktueller Textdatei in Tabelle ROHDATA_SOURCE.
- FData : TTrsFile	Objekt für Zugriff auf Textdatei.

Methoden

- RemoveOldData()	Diese Methode entfernt alle zu der angegeben Textdatei vorhandenen Daten aus der Datenbank. In der aktuellen Version ist sie noch nicht implementiert, da die Eindeutigkeit der Dateinamen aller Rohdatenfiles nicht sichergestellt ist und sonst plötzlich falsche Daten gelöscht würden.
- MakeRdSourceKey()	Erstellt die zu einem Textfile nötigen Einträge in den Tabellen ROHDATA, SOURCE und ROHDATA_-SOURCE

- MakeSrcTermEntries()	Diese Methode erstellt für alle einzulesenden Spalten neue Einträge in der Tabelle SRC_TERM mit den nötigen Verweisen auf Einträge in der Tabelle STD_TERM.
- ReadDataLine()	Diese Methode liest eine einzelne Zeile aus dem Textfile und erstellt einen oder mehrere Einträge in der Tabelle OBS_BASE.
+ Create()	Konstruktor-Methode.
+ Destroy()	Destruktor-Methode.
+ importTrsFile()	Liest ein Textfile in die Datenbank ein. Der Dateiname wird als Parameter übergeben.

Klasse **TTrsFile**	inherites from **TStringList**^{VCL}	unit syntaxChk

Aufgaben

Attribute

- cThema: string	String mit Themenbezeichnungen für alle Datenspalten, einzele Themas mit Tab getrennt.
- cEinheit: string	String mit Einheiten für alle Datenspalten, einzelne Einheiten mit Tab getrennt.
- cJahre: string	String mit Jahren für alle Datenspalten, einzelne Jahre mit Tab getrennt, Jahre in Form 1999 oder 1998-1999. Falls diese Werte nicht vorhanden sind werden die Standardwerte "datumStart" und "datumEnd" verwendet.
- fName: string	Dateiname der aktuellen Textdatei.
- errors: string	Sammlung aller aufgetretenen Fehler während der Ausführung von CheckSyntax.
- errCount: integer	Anzahl der aufgetretenen Fehler.
- raumColumn: integer	Nummer der Spalte, welche die Angaben für die Ortsidentifikation enthält.
- lastCol: integer	Nummer der letzten Spalte, welche noch gültige Daten enthält.
+ Cols: TBernhistCol[]	Array-Property mit Informationen zu jeder Datenspalte über Thema und Zeitraum.
+ sysCodeType: string	Angabe, in welcher Form die Orte (Räume) gefunden werden. Gültige Werte sind BHIST (Bernhist - Code), BFS (Code Bundesamt für Statistik), KEY (Angabe über Ortsbezeichnung).
+ quellenName: string	Quellenangabe, wird aus dem Header des Textfiles gelesen.
+ datumStart: integer	Defaultwert für untere Datumsgrenze, wird aus dem Header des Textfiles gelesen.
+ datumEnd: integer	Defaultwert für obere Datumsgrenze, wird aus dem Header des Textfiles gelesen.
+ currLine: integer	aktuelle Datenzeile wird in allen "zeilenrelevanten" Methoden verwendet.
+ IsSectionHeader:boolean	true wenn aktuelle Zeile ein Sectionheader ist (beginnt mit eckiger Klammer [ ).

+ IsComment:boolean	true, wenn aktuelle Zeile einen Kommentar enthält (beginnt mit einem Ausrufezeichen).
+ LineSysKey:integer	Liefert die Primary Key eines Records aus der Tabelle STD_SYS. Der Record wird über die Ortsangabe der aktuellen Zeile gesucht.

Methoden

- SetInfo()	Aktualisiert die Infoanzeige im Hauptfenster des Programms
- NewError()	registriert einen Syntaxfehler. Als Parameter wird die Fehlermeldung übergeben.
- CheckNote()	Überprüft einen String ob er eine gültige Anmerkung enthält.
- CheckGE()	Überprüft alle Angaben in der Sektion mit den Grundeinstellungen auf Gültigkeit.
- CheckTS()	Überprüft alle Angaben in der Sektion Themenstandardisierung auf Gültigkeit.
- CheckDatenHeader()	Überprüft Kopfzeilen bei Datenspalten mit Themenbezeichnung und Einheiten.
- CheckDaten()	Überprüft Zeilen mit Datenwerten auf Gültigkeit.
- CheckAnmerkungen()	Überprüft alle Angaben in der Sektion mit den Anmerkungen auf Gültigkeit.
+ Create()	Konstruktor. Erhält als Parameter den Dateinamen. Liest die Textdatei in den Hauptspeicher ein.
+ FindSection()	Erhält als Parameter die Bezeichnung einer Sektion. Setzt currLine auf erste Zeile dieser Sektion
+ ReadHeader()	Liest alle Angaben im Headerbereich ein.
+ CheckSyntax()	Überprüft eine Datei auf Fehler. Rückgabewert ist ein String mit allen Fehlermeldungen.
+ Thema()	Bezeichnung des Themas einer bestimmten Datenspalte
+ Einheit()	Einheit einer bestimmten Datenspalte
+ Jahr()	Jahr in der Form 1999 oder 1998-1999 einer bestimmten Datenspalte

CD-Version

Die Routinen, die in der CD-Version enthalten sind, sind Prozeduren, die nach einem Event aufgerufen werden, normalerweise nach dem Klicken auf eine Schaltfläche oder dem Verlassen eines Formulares. Der Code ist in Visual Basic for Applications geschrieben. Hier werden nur gerade die verwendeten Methoden aufgelistet und wenn nötig kurz beschrieben. Dabei wir ähnlich der Delphi-Dokumentation die Kenntnis der Objekthierarchie von Access vorausgesetzt.

Methoden

ActiveXStr159_GotFocus() Sobald ein Eintrag im Tree angeklickt worden ist, kann der OK-Button für den nächsten Schritt aktiviert werden. Ohne eine Auswahl eines Themas im Tree würde ein Laufzeitfehler auftreten.

ExportButton_Click() Exportfunktionen im HTML, RTF, TXT und XLS-Format. HauptformularVerlassenButton_Click() Verlässt das Hauptformular der Anwendung.

Liste1_GotFocus() Wenn zu Beginn ein Eintrag in der Raumebene gewählt wird, müssen alle anderen Steuerelemente deaktiviert werden, um den Benutzer zu zwingen, Step by Step vorwärts zu gehen.

Liste5_Click() Wenn ein Wert in der Liste5 (Startjahr) selektiert worden ist, muss a) die 2. Liste aktiviert ' werden und b) es sollen in dieser 2. Liste keine Einträge vorselektioniert sein.

Liste6_Click() Der User hat alle relevanten Angaben gemacht und kann nun die Abfrage starten. Aktivierung des Buttons SuchenButton.

NeueSucheButton_Click() Inititalsierungsroutine für das Löschen jeglicher Einträge in Liste 1, 2, 5 und 6 sowie das Zurücksetzen aller Buttons, um die Abfolge der Datenselektion zu gewährleisten.

OK_Button2_Click() Der User kann eine oder mehrere Raumebenen selekieren; aufgrund dieser Auswahl wird dann die Liste mit den Ortsnamen gefüllt.

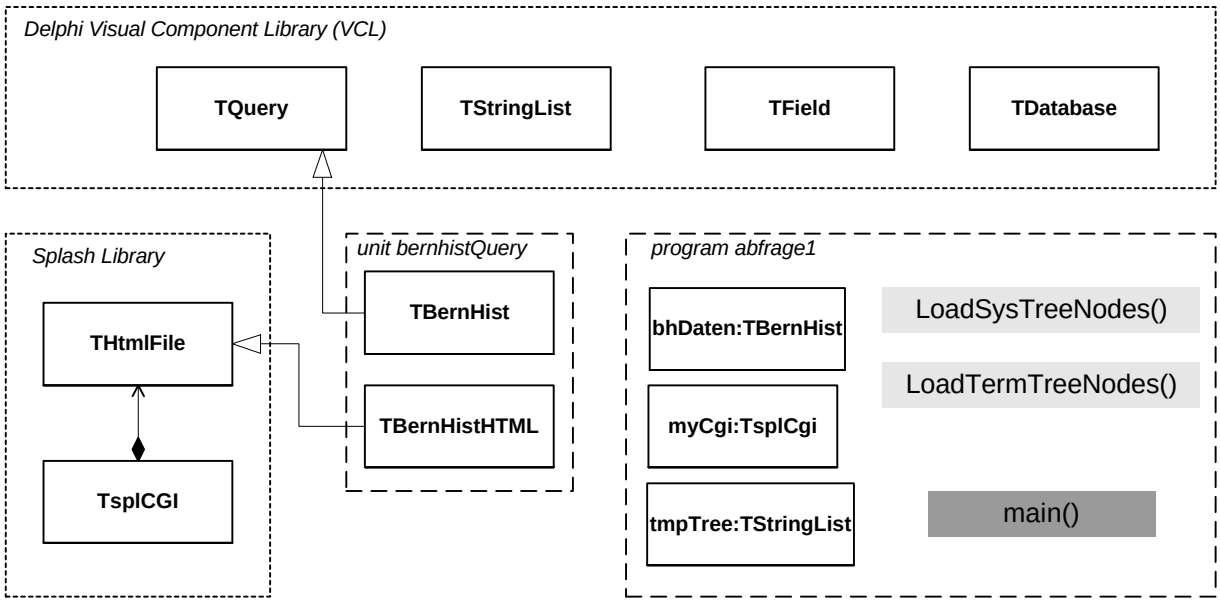
Form_Open(Cancel As Integer) Die Prozedur füllt das TreeView-Control (den Themenbaum) mit Werten. Dazu wird zum einen die Abfrage Hierarchie_1 genutzt, die alle Themen auflistet und die Abhängigkeit dieser Themen untereinander aufzeigt; der Wert im Feld STD_TERM_IN ist der Schlüssel zum übergeordneten Thema von STD_TERM_KEY. Da der Tree zukünftig themenmässig erweiterbar/verkleinerbar sein muss, wird mit dem Modul ListeErzeugen eine Tabelle gefüllt, die so mit STD_TERM_KEY´s gefüllt ist, dass eine sequent. Abarbeitung dieser Liste Stufe für Stufe des Trees aufbaut.

OK_Button4_Click() Beim Verlassen des Steuerelementes TreeView werden diejenigen Datensätze mittels einer dynamischen SQL-Abfrage gesucht, die den Werten in den Listen mit den Räumen und den Themen entsprechen.

SuchenButton_Click()	Wenn der User alle nötigen Werte angegeben hat, kann die Suche starten. Vor der eigentlichen Abfrage werden noch Fehlerüberprüfungen vorgenommen, so z.B., ob das Startjahr auch wirklich kleiner/gleich dem Endjahr ist. Ansonsten wird eine Fehlermeldung ausgegeben.

Internetanbindung (Serverseite)

Der serverseitige Teil der Internetanbindung ist vollständig in Delphi implementiert. Es läuft auf Internet Informations Server unter Windows NT 4.0. Es werden allerdings keine IIS spezifischen Eigenschaften verwendet, die Verwendung von Windows NT ist allerdings zwingend.
Das CGI-Programm wird sowohl für Anfragen über ein HTML-Formular wie auch für Anfragen über das Java-Applet verwendet.
Die verwendeten Klassen sind im Kapitel "Import Daten" bereits beschrieben.



Klassendiagramm

program **Abfrage1**	unit abfrage1

Aufgaben

Dieses Programm führt sämtliche serverseitigen Aufgaben für die Internetanbindung.

Parameter

Das Programm erkennt folgende Parameter, welche sowohl mit der GET als auch mit der POST - Methode übergeben werden können.

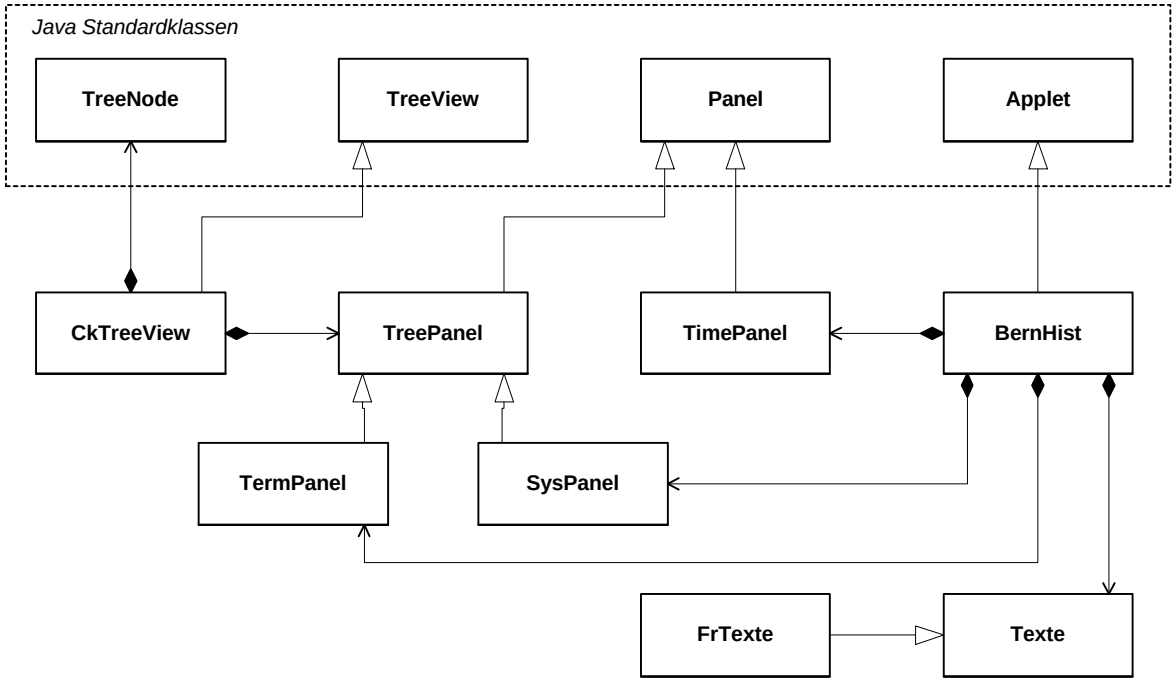
COMMAND	Zahl zwischen 0 und 3. falls nicht angegeben wird 0 angenommen. Der Parameter bestimmt die Art der zurückgegebenen Daten. Dabei gibt es folgende Werte: 0 = HTML-Daten mit Abfrageresultat (Liste mit Daten). 1 = Textdaten mit allen Orten (Räumen) für Tree in Applet 2 = Textdaten mit allen Themen für Tree in Applet 3 = Zeitraum, wo Daten vorhanden als Textdaten.
TERM	Key Id des gewünschten Themas
SYS	Liste mit Key Id's (getrennt mit Komma oder Leerzeichen) der gewünschten Orte (Räume).
START	optional; die Angabe der unteren Jahresgrenze für Daten
END	optional; die Angabe der oberen Jahresgrenze für Daten

Objekte	
myCgi:TsplCgi	Objekt als Interface zur CGI-Schnittstelle.
tmpTree: TStringList	temporäre Liste. wird bei rekursivem Aufbau der Tree's (Räume und Themen) verwendet.
bhDaten: TBernHist	Objekt als Interface zur Bernhist Datenbank. Siehe auch unter Import Daten (Klassenbeschreibung).

Funktionen	
LoadSysTreeNodes()	Funktion, mit welcher die Daten des Raumbaumes rekursiv zusammengestellt werden
LoadTermTreeNodes()	Funktion, mit welcher die Daten des Themenbaumes rekursiv zusammengestellt werden
main() - *Hauptprogramm*	Liest Eingabedaten von CGI-Schnittstelle und stellt je nach COMMAND die Ausgabedaten zusammen und sendet diese an die CGI-Schnittstelle.

Internetanbindung (Clientseite)

Das Java Applet wurde mit Visual Cafe 2.0 von Symantec erstellt. Neben den Java Standardklassen werden die Klassen TreeView und TreeNode aus der Symantec Library verwendet. Auf eine vollständige Beschreibung dieser Klassen wird verzichtet. Ebenfalls werden die Objekte und Klassen für die Windowsoberfläche (Buttons, Labels, etc.) im Klassendiagramm nicht dargestellt.



Klassendiagramm

Klasse **BernHist**	extends **Applet**

Aufgaben

Die Klasse verwaltet die drei verschiedenen Panels mit den Abfrageparametern Raum, Thema und Zeit. Beim Aufstarten wird zudem entweder ein Objekt mit deutschen oder eines mit französischen Texten erstellt.

Attribute

+ txt: Texte	Ein Objekt mit allen Textkonstanten (deutsch oder französisch).
+ mySys: SysPanel	Panel mit Auswahl für Orte (Räume)
+ myTerm: TermPanel	Panel mit Auswahl für Thema
+ myTime: TimePanel	Panel mit Auswahl Zeitraum

Methoden

- loadTexte()	Diese Methode erstellt ein Texte - Objekt und weist allen Controls wie Labels und Buttons die neuen Texte zu.

- setButtonState()	erhält als Parameter eine Referenz auf ein Button - Objekt und eine boolsche Variable. Setzt den Button auf enabled oder disabled.
- selectTab()	aktiviert eines der drei Panels (Raum, Thema, Zeit)
+ init()	geerbte Methode aus Klasse Applet. Hier werden alle grafischen Controls sowie weitere Objekte erstellt.
+ start()	geerbte Methode aus Klasse Applet. Keine Aktionen.

Klasse **CkTreeView** extends **TreeView**

Aufgaben

Erweitert die Klasse TreeView um Funktionalitäten zum suchen.

Attribute

- searchStr: String	Text, welcher gesucht werden soll.
- lastResult: TreeNode	Hier wird der TreeNode der letzten Suche abgespeichert, um bei einem Aufruf "find next" wieder zu verwenden.

Methoden

- searchNode()	interne Hilfsfunktion zur rekursiven Suche durch den Tree.
+ skip()	liefert nächsten TreeNode. Falls ein Ast fertig bearbeitet ist, wird der nächste Node der nächst höheren Ebene geliefert.
+ search()	sucht im Tree einen Eintrag, welcher im Text den Suchstring enthält. Die Suche beginnt entweder beim ersten Node oder fährt beim Resultat - Node der letzten Suche weiter.

Klasse **FrTexte** extends **Texte**

Aufgaben

Klasse mit französischen Texten.

Attribute

diverse Konstanten mit Texten für Labels und Buttons.

Klasse **SysPanel** extends **TreePanel**

Aufgaben

Diese Klasse erweitert ein TreePanel mit der Funktionalität zur Auswahl von Orten (Räumen). Dabei können mehrere Orte gleichzeitig gewählt werden. Die gewählten Orte werden in einer Listbox angezeigt.

Attribute

- selList: Vector	Vector mit Objekten (ID's) der gewählten Räume. selList und lbxSelect werden immer synchron verändert.
+ lbxSelect: List	Listbox mit gewählten Räumen

Methoden

# setTreeInitState()	Initialisierung. Hier expandieren von vier zwei Ebenen des Treeviews.
# loadText()	lädt die sprachspezifischen Texte in Labels und Buttons.
+ selectedSys()	liefert einen String mit allen ID's der gewählten Orte.

Klasse **TermPanel** extends **TreePanel**

Aufgaben

Die Klasse erweitert ein TreePanel um die Funktionalität zur Auswahl eines Themas.

Methoden

+ termAsString()	liefert das ausgewählte Thema als String (Themen-bezeichnung).
+ ehAsString()	liefert die Einheit des gewählten Themas als String.
+ selectedTerm()	liefert die ID des gewählten Themas.
+ loadText()	lädt die sprachspezifischen Text in Labels und Buttons.

Klasse **Texte**

Aufgaben

Die sprachspezifischen Teile der Applikation werden in dieser Klasse gekapselt. Um das Programm zu übersetzen, wird von dieser Klasse abgeleitet und die Konstanten neu definiert.

Attribute

diverse Konstanten mit Texten für Labels und Buttons.

Klasse **TimePanel** extends **Panel**

Aufgaben

Diese Klasse ermöglicht die Auswahl des Zeitraumes und enthält die Funktionen für die Abfragen an den Server. Dabei wird das Resultat in einem neuen Browserfenster angezeigt.

Methoden

+ loadText()	
+ updateInfos()	aktualisiert die Anzeige des gewählten Themas und setzt den Status von verschiedenen Buttons.

Klasse **TreePanel** extends **Panel**

Aufgaben

In dieser Klasse werden sowohl die für den Treeview "Orte" und Treeview "Themen" benötigten Methoden zusammengefasst.

Attribute

- loadThread: Thread	Referenz auf Thread, welcher die Daten für das TreeView über das Internet herunterlädt.
# tmpDataFileName: string	nur für Tests des Applets im Offline-Modus

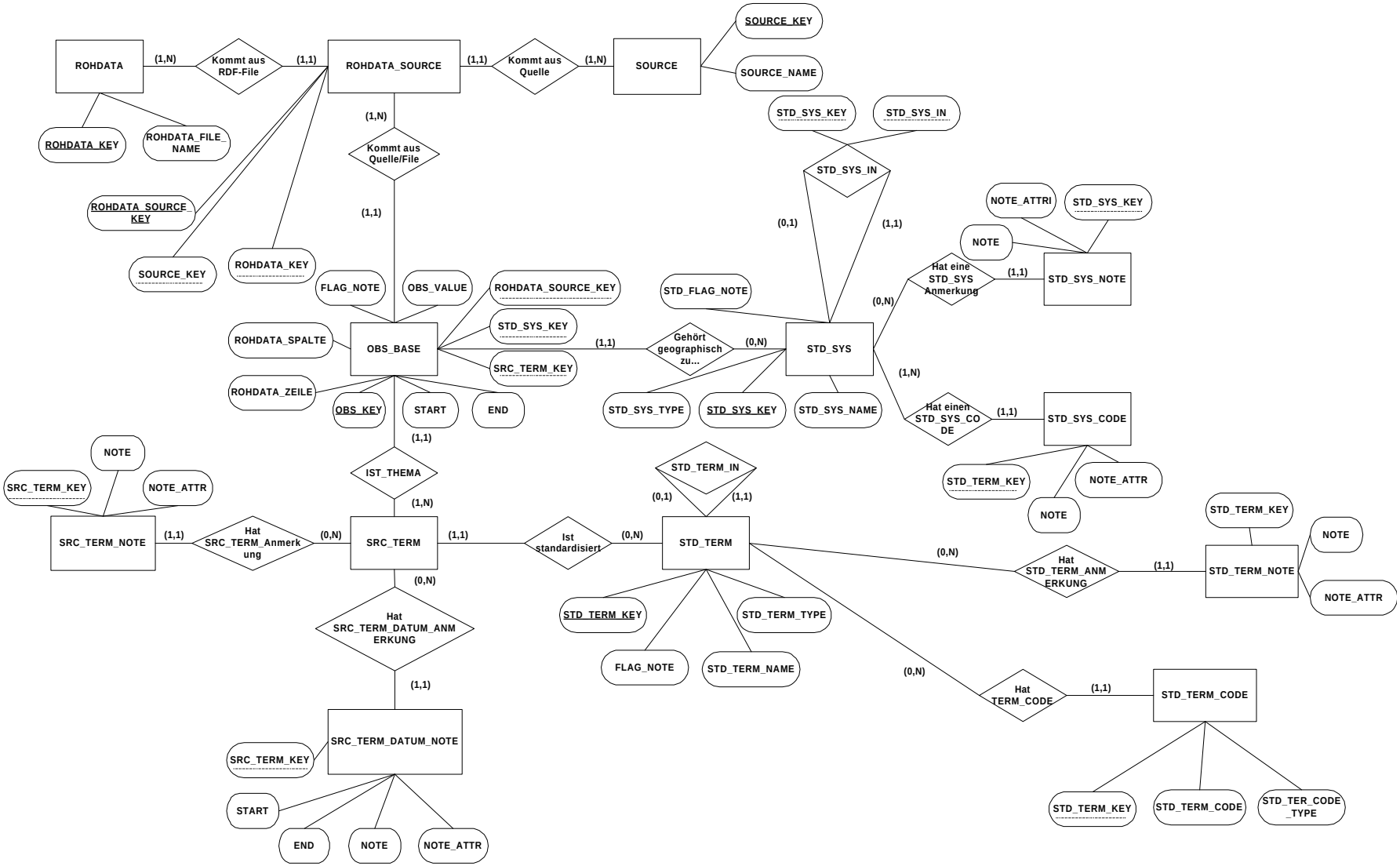
# cmdID: String	gibt an, ob es sich um ein Panel für Räume (1) oder ein Panel zur Themenauswahl (2) handelt.
# tree: CkTreeView	Das Treeview zur Anzeige der Daten
# owner: BernHist	Referenz auf Applet
# allItems: String[]	Array mit allen Einträgen des Trees. Zusätzlich enthält der String die ID der Einträge.

Methoden

# loadTree()	erstellt loadThread und startet diesen.
# loadData()	stellt Verbindung zu Server her und liest empfangene Daten in Array allItems ein
# expandObjects()	rekursiv verwendete Funktion, um aus String im TreeNode-Text die ID abzuspalten und als Datenobjekt des Nodes zu speichern.
# expandTree()	setzt Modus aller TreeNodes auf "expanded" bis zu einem bestimmten Level.
# expandNode()	rekursiv verwendete Funktion, um TreeNodes auf "expanded" zu setzen, wird von expandTree() verwendet.
# *setTreeInitState()*	abstrakt, um in abgeleiteter Klasse Initialisierung abzuschliessen.
+ *loadText()*	abstrakt, um sprachspezifische Texte zu laden.

Anhang 3

ER-Schema



Tabelleneigenschaften

Tabelle: OBS_BASE

Name	Typ	Größe
START	Integer	2
END	Integer	2
STD_SYS_KEY	Integer	2
SRC_TERM_KEY	Integer	2
ROHDATA_SOURCE_KEY	Integer	2
OBS_KEY	Long Integer	4
FLAG_NOTE	Integer	2
ROHDATA_ZEILE	Integer	2
ROHDATA_SPALTE	Integer	2
OBS_VALUE	Single	4

Tabelle: OBS_NOTE

Name	Typ	Größe
OBS_KEY	Long Integer	4
NOTE	Text	255
NOTE_ATTR	Long Integer	4

Tabelle: ROHDATA

Name	Typ	Größe
ROHDATA_KEY	Integer	2
ROHDATA_FILE_NAME	Text	255
STATUS	Integer	2
DATE	Integer	2
TIME	Integer	2
OBS_N	Long Integer	4

Tabelle: ROHDATA_SOURCE

Name	Typ	Größe
ROHDATA_SOURCE_KEY	Integer	2
SOURCE_KEY	Integer	2
ROHDATA_KEY	Integer	2

Tabelle: ROHDATA_SRC_TERM_SOURCE

Name	Typ	Größe
ROHDATA_KEY	Integer	2
ROHDATA_SOURCE_KEY	Integer	2
SRC_TERM_KEY	Integer	2
START	Integer	2

Tabelle: SOURCE

Name	Typ	Größe
SOURCE_KEY	Integer	2
SOURCE_NAME	Text	240

STATUS	Integer	2

Tabelle: SRC_TERM

Name	**Typ**	**Größe**
SRC_TERM_KEY	Integer	2
STD_TERM_KEY	Integer	2
STD_TERM_NAME	Text	255
SRC_TERM_TYPE	Text	255
FLAG_NOTE1	Integer	2

Tabelle: SRC_TERM_DATUM

Name	**Typ**	**Größe**
SRC_TERM_KEY	Integer	2
START	Long Integer	4
END	Long Integer	4

Tabelle: SRC_TERM_DATUM_NOTE

Name	**Typ**	**Größe**
SRC_TERM_KEY	Integer	2
START	Long Integer	4
END	Long Integer	4
NOTE	Text	255
NOTE_ATTR	Integer	2

Tabelle: SRC_TERM_NOTE

Name	**Typ**	**Größe**
SRC_TERM_KEY	Integer	2
NOTE	Text	255
NOTE_ATTR	Integer	2

Tabelle: STD_SYS

Name	**Typ**	**Größe**
STD_SYS_KEY	Integer	2
STD_SYS_NAME	Text	255
STD_SYS_TYPE	Text	255
FLAG_NOTE	Integer	2

Tabelle: STD_SYS_CODE

Name	**Typ**	**Größe**
STD_SYS_KEY	Integer	2
STD_SYS_CODE	Text	255
STD_SYS_CODE_TYPE	Text	255

Tabelle: STD_SYS_IN

Name	**Typ**	**Größe**
STD_SYS_KEY	Integer	2
STD_SYS_IN	Integer	2

Tabelle: STD_SYS_NOTE

Name	Typ	Größe
STD_SYS_KEY	Integer	2
NOTE	Text	255
NOTE_ATTR	Integer	2

Tabelle: STD_TERM

Name	Typ	Größe
STD_TERM_NAME	Text	255
STD_TERM_TYPE	Text	255
STD_TERM_THESA_LEVEL	Integer	2
STD_TERM_THESA_ORD	Integer	2
STD_TERM_FORMEL	Text	255
N_ELEM	Integer	2
FLAG_NOTE	Integer	2
STD_TERM_KEY	Integer	2

Tabelle: STD_TERM_CODE

Name	Typ	Größe
STD_TERM_KEY	Integer	2
STD_TERM_CODE	Text	255
STD_TERM_CODE_TYPE	Text	255

Tabelle: STD_TERM_IN

Name	Typ	Größe
STD_TERM_KEY	Integer	2
STD_TERM_IN	Integer	2

Tabelle: STD_TERM_NOTE

Name	Typ	Größe
STD_TERM_KEY	Integer	2
NOTE	Text	255
NOTE_ATTR	Integer	2

Anhang 4

Tests

Die Tests sind grob in zwei Teile zu unterteilen. Zum einen wurden die User-Interfaces durch Gymnasiallehrer, interessierte Historiker und Schüler von Gewerbeschule und Gymnasien getestet, um die Bedienbarkeit und die Stabilität der Produkte zu testen. Inhaltliche Tests wurden einerseits von uns beiden gemacht, wie z.B. die Suche nach doppelten Datensätzen, andererseits von Prof. Pfister, der die Daten aufgrund aktueller Forschungen sehr gut kennt. Die untenstehende Tabelle gibt Auskunft über die Tests.

Testtyp	**Beschreibung des Tests**	**Reaktion/Massnahme**
User-interfaces	Test sollte Auskunft geben über a) die Bedienbarkeit des Produktes und über die Einarbeitungszeit und b) über Programmierfehler beim Gebrauch des Tools. Tester waren Historiker, Gymnasiallehrer, Schüler, unbekannte User (Internet) und Auftraggeber.	Sammeln der Anregungen/Kritik und Fehlermeldungen. Die Fehler wurden behoben, die Anregungen zusammen mit dem Auftraggeber besprochen und für die laufende Arbeit entweder aufgenommen oder verworfen.
Performance	Die User sollten a) für die Stand-alone Lösung und b) für die Internetapplikation Rückmeldungen geben. Für die Stand-alone Lösung wurde für die verwendete Hardware eine Bestandesaufnahme gemacht, um die Ergebnisse vergleichen zu können. Die Internetapplikation wurde im Klassenverband getestet, um damit eine maximale Auslastung des Systems zu simulieren; aus den Erfahrungen der alten Version ist bekannt, dass bisher nie mehr als drei User gleichzeitig Abfragen auf die DB machten.	Ab einem Gerät mit Pentium 133, 32MB RAM und genügend grosser Festplatte ist die Performance für die Stand-alone Lösung akzeptabel. Problem: Festplattenkapazität. Für die Internetapplikation: Keine negativen Rückmeldungen bei Vollauslastung.
Doppelte Daten	Systematisches Suchen nach doppelten Einträgen durch Chr. Kaufmann und U. Dietrich	-
Richtigkeit der Abfragen	Durch Chr. Pfister, der aus laufender Forschungstätigkeit heraus die Daten sehr gut kennt und andererseits durch Historiker, die das Tool aktiv benutzen und (ebenfalls) stichprobenartig Segmente in den Archiven überprüften.	Teilweise Datenfehler gefunden. Diese Daten werden von Historikern in Archivarbeit überprüft.

Anhang 5

Sourcecode

Der Sourcecode kann auf Wunsch bei den Autoren bezogen werden:

Urs Dietrich	dietrich@hist.unibe.ch
Christian Kaufmann	christian.kaufmann@gmx.net

Der Sourcecode gliedert sich in folgende Bereiche:

- Code Datenimport (Delphi)
- Code zur CD-Version (MS Access)
- Code der Internetanbindung (Java und Delphi)