Jahresschrift für mitteldeutsche Vorgeschichte	78	S. 417 - 442	Halle (Saale)	1996

Zugriff archäologischer Anwendungen auf die Datenbank Arche und Erweiterung ihrer Funktionalität durch Anbindung an ein Geografisches Informationssystem

von Thomas Richter, Halle (Saale)

Im Landesamt für archäologische Denkmalpflege Sachsen-Anhalt (LfA) wird die archäologische Datenbank Arche aufgebaut, die als komplexes Informationssystem für eine Vielzahl von Anwendungen konzipiert ist[1]. Auf diese digitale Datenbasis aufsetzend, werden eine Reihe von Programmen mit Datenbankschnittstelle (im folgenden archäologische DB-Anwendungen genannt) entwickelt, die an die Anforderungen im LfA angepaßt sind. Es handelt sich um Anwendungen zur:

1. Verwaltung und Anfertigung von Stellungnahmen
2. Betreuung der ehrenamtlichen Beauftragten für archäologische Denkmalpflege
3. Verwaltung und Anzeige von Luftbildern

Zur Veranschaulichung der verschiedensten Aspekte und Zusammenhänge ist eine visuelle Darstellung (Kartenarbeit und Anzeige von Bildern wie Luftbildfotos) notwendig. Die Kartenarbeit ist das Einsatzgebiet Geografischer Informationssysteme (GIS). Es werden die Anforderungen an ein GIS und Realisierungsmöglichkeiten für das Zusammenspiel mit den archäologischen DB-Applikationen ausgeführt. Dargestellt wird die enorme Komplexität der zu entwickelnden Programme, die sich daraus ergibt, Informationen aus so unterschiedlichen Informationsquellen wie SQL-Datenbanken und Grafischen Informationssystemen zu verarbeiten und den Datenfluß zwischen diesen zu steuern. Deshalb wird sowohl der Zugriff der Programme auf Datenbanken als auch das Zusammenwirken mit einem GIS vorgestellt. Die von archäologischer Seite gestellten Anforderungen an die Programme und die archäologische Datenbank Arche sind detailliert bei J. Bittner und M. Stock[2] beschrieben.

1. Vorbetrachtungen

1.1. Rechner-Netzwerk

Das Landesamt für archäologische Denkmalpflege Sachsen-Anhalt verfügt über ein heterogenes Computer-Netzwerk auf Ethernet-Basis. Die Vernetzung stützt sich sowohl auf eine moderne strukturierte Twisted-Pair-Verkabelung (die auch den Anforderungen zukünftiger Hochgeschwindigkeits-Netze genügt) als auch auf ein herkömmliches Thin-Ethernet-Netz, das an den zentralen Hub angekoppelt ist. Somit ergeben sich (mit der zur Zeit installierten Netzwerk-Hardware) bestimmte Beschränkungen hinsichtlich der Belastbarkeit des gesamten Netzwerkes (10MBit/s-Netz, Bustopologie, Kollisionen zwischen Datenpaketen), die auch die Auswahl eines geeigneten GIS betreffen.

Als Backend werden zwei UNIX-Server eingesetzt. Ein NeXT-Cube mit NEXTSTEP-Betriebssystem dient zur Netzwerkverwaltung (Userverwaltung, File- und Printserver-Dienste, Einbindung der anderen Rechner in das Netzwerk). Angeschlossen sind etwa 20 mit Betriebssystem NEXTSTEP ausgestattete Computer (eine Reihe älterer NeXT-Workstations, von denen nur die mit Farbgrafik für Kartenarbeit geeignet sind und PCs mit 486DX2- und Pentium-Prozessoren). Für die Arbeit mit Karten und anderem farbigen Bildmaterial (z. B. Luftbilder) sind hohe Bildschirmauflösungen von wenigstens 1024 x 768 Bildpunkten und 8 Bit Farbtiefe und 256 Farben, besser HighColor mit 4096 Farben Voraussetzung. Als Netzwerk-Protokoll wird UNIX-typisch TCP/IP eingesetzt. NEXTSTEP kann im LfA als strategisch wichtigstes Betriebssystem angesehen werden.

Der zweite Server, eine IBM RS/6000 Mod. 530H ist als Datenbank-Server konfiguriert. Auf ihm ist das relationale SQL-Datenbanksystem SYBASE (derzeit in der Version 10.0.1) mit der archäologischen Datenbank Arche installiert. Ausreichend Festplattenkapazität (45 GB) für die Datenbanken und das Karten- sowie weiteres Bildmaterial ist vorhanden.

Weiterhin sind eine Reihe von DOS/Windows-PCs über die Netzwerk-Software Sun PC/NFS in das Netzwerk integriert. Sie nutzen hauptsächlich die Datei- und Druckdienste.

1.2. Einführung in die Datenbankzugriffs-Software für NEXTSTEP

NEXTSTEP ist eines der modernsten Betriebssysteme, die derzeit auf dem Markt verfügbar sind und zeichnet sich durch vollständige Objektorientiertheit bis auf unterste System-Schichten aus. Die funktionelle Oberfläche und das modulare Konzept (das gleichzeitig Vereinfachung durch Wiederverwendbarkeit von Software-Komponenten bedeutet, z. B. das Schriften-Auswahlfenster oder stets gleiche Elemente zum Datei-Öffnen oder -Schließen sowie die Mail-Funktionalität) haben bei den Nutzern im LfA schnell Akzeptanz gefunden.

NEXTSTEP verfügt über geeignete Werkzeuge zur Programmierung von auf der grafischen Oberfläche laufenden Programmen, wie z. B den Interface Builder, mit dem sich einfache Programme schnell realisieren lassen, indem geeignete vorgefertigte Elemente (wie Fenster, Tabellen, Browser, Textfelder, Menüs) aus Paletten mit der Maus gewählt werden und anschließendem Aufbau von Beziehungen zwischen diesen Objekten. Programme werden mit Objective C, einer objektorientierten Erweiterung der Sprache C, entwickelt.

Die Datenbanken werden auf einem SQL-Server der Firma SYBASE angelegt. Ein SQL-Server wird auf einer im Netzwerk laufenden (leistungsfähigen) Maschine installiert und verwaltet zentral die anfallenden Datenbestände nach dem Client/Server-Prinzip. Client-Anwendungen von im Netzwerk installierten Computern senden Anfragen bzw. Datensätze an den Server, der diese verarbeitet und entsprechend reagiert (z. B Ergebnisse einer Datenbankabfrage an die Clients (Frontends) zurückgibt, Sortiervorgänge, Einfüge- und Löschvorgänge auslöst, etc.). Der SQL-Server ist so optimiert, daß er effizient Multitasking- und Multiuser-Operationen ermöglicht.

Folgende Tools zum Zugriff auf SYBASE-Datenbanken sind unter NEXTSTEP verfügbar:

a) isql

SYBASE liefert den isql-Kommandozeilen-Interpreter, der in einem Terminalfenster gestartet werden kann und interaktive Abfragen mit einer erweiterten SQL-Kommandosprache ermöglicht. Mit diesem Tool ist keine programmierbare Interaktion mit anderen auf der grafischen NEXTSTEP-Oberfläche ablaufenden Anwendungen möglich.
SQL (Structured Query Language) ist eine Sprache zum Abfragen, Aktualisieren und Verwalten relationaler Datenbanken. Sie untergliedert sich weiter in DDL (Data Definition Language) zum Anlegen der Datenbankstruktur-Elemente und DML (Data Manipulation Language), die eigentliche Datenabfrage- und Manipulationssprache.

b) dblibrary

Die dblibrary ist eine ebenfalls von SYBASE gelieferte C-Bibliothek und umfaßt eine komplexe Sammlung von in der Sprache C geschriebenen Low Level Funktionen zum Datenbank-Zugriff. Vorteilhaft ist die problemlose Einbindung in eigene Programme und hohe Performance bei Datenbankzugriffen. Nachteilig ist der enorme Einarbeitungsaufwand sowie die notwendige Kodierung ständig benötigter Funktionen, z. B. muß selbst eine einfache Abfrage erst durch eine ganze Serie von Schleifen- und Funktionsaufrufen programmiert werden. Außerdem fehlt die nahtlose Integration in das objektorientierte Objective C Programmiersystem von NEXTSTEP. Es gibt keine vorprogrammierte Anbindung an die verschiedenen grafischen User-Interface-Objekte, wie z. B. Textfelder und Browser. Die Nutzung der dblibrary ist, von Spezialfällen abgesehen (Import von Rasterbildern in Datenbanken), unter NEXTSTEP uneffektiv und nicht empfehlenswert.

c) NeXT Database Kit

Seit Einführung der Betriebssystemversion 3.0 Mitte 1992 liefert NeXT sein Programmier-Frontend zum Zugriff auf Client/Server-Datenbanken, das Database Kit (kurz DBKit) mit der NEXTSTEP Entwickler-Version aus. Es gliedert sich in das Programmier-System von NeXT ein, enthält u. a. eine Interface Builder Palette. Das DBKit ist in Objective C entwickelt und paßt sich in die bestehenden Software-Kits, insbesondere das für die Gestaltung von NEXTSTEP-Programmen essentielle Application Kit, ein. Das DBKit ist aus drei Objektschichten aufgebaut, die allerdings sehr abstrakt gehalten sind.
Die Lieferung durch den Betriebssystem-Hersteller garantiert die Zukunftssicherheit des Kits. Einfachste Anwendungen lassen sich, ohne eine Zeile zu programmieren, mit dem Interface Builder realisieren. Nachteilig ist, daß für so unverzichtbare Routinen wie Suchfunktionen keine vorgefertigten Objekte geliefert werden, sondern erst aus DBKit-Klassen abzuleiten sind. NeXT positioniert das DBKit sowohl als Werkzeug, mit dem einfache Applikationen ohne größere Programmier-Kenntnisse durch die Objekte des User Interface Layers gestaltet werden können, aber auch als Objektbaukasten für professionelle Entwickler, die sich mit genauer Kenntnis der Objektstrukturen und -zusammenhänge ihre benötigten Objektklassen selbst zusammenstellen.

d) Enterprise Object Framework

Das Enterprise Objects Framework (EOF) wurde von NeXT Ende 1994 auf den Markt gebracht. Es ist nicht Bestandteil der NeXT-Entwicklerversion, sondern wird separat vermarktet. Das EOF ist ebenfalls als Bindeglied zur Aufbereitung von Informationen aus unternehmensweiten SQL-Datenbanken gedacht und basiert auf im kommenden OpenStep einfließenden Konzepten. Da noch keine weiteren Fakten zu EOF vorlie-

gen, wird auf das Framework nicht weiter eingegangen. Ob das EOF Akzeptanz für den Datenbank-Zugriff unter NEXTSTEP findet, kann erst die Zukunft zeigen.

e) Software von Third Party Herstellern

Von anderen Herstellern sind das ODB Toolkit, Parabase, das Access Kit und Espresso erwähnenswert. Vorteilhaft ist bei diesen Produkten die Anpassung an die praktischen Erfordernisse der Datenbank-Anbindung gelöst. Sie bieten teils Möglichkeiten der direkten SQL-Programmierung (ODB Toolkit), teils eigene 4GL-Sprachen (Access Kit, Espresso). Geliefert werden weiterhin für die Anwendungsentwicklung noch Interface Builder Paletten mit Objekten zur schnellen Realisierung von DB-Anwendungen.

Diese Systeme kapseln entweder die dblibrary-Funktionen (ODB Toolkit) oder andere, nicht dokumentierte Funktionen (Parabase), durch den Aufbau von komplexen und praktikabel handhabbaren Objekten oder setzen auf dem DBKit auf und erweitern es um zusätzliche Funktionalität. Schlechteres Laufzeitverhalten ist der Preis für die beschleunigte Realisierung von DB-Projekten. Dies gilt ebenfalls für das DBKit. Trotz der beschriebenen Vorteile ist die Einarbeitungszeit recht hoch. Ein Problem ist meist die Zukunftssicherheit solcher Systeme. Mit Aufkommen des weiter unten beschriebenen DBKits wurde z. B. das ODB Toolkit schnell vom Markt verdrängt. Ungeklärt ist die Zukunft der oft kleinen Entwicklerfirmen und die Lauffähigkeit unter kommenden NEXTSTEP-Versionen und OpenStep-Betriebssystemerweiterungen. Es fallen zudem noch pro Arbeitsplatz beträchtliche Anschaffungskosten (Lizenzkosten) an.

Durch die wachsende Bedeutung des DBKit für NEXTSTEP (insbesondere gegenüber den unter Punkt e) beschriebenen proprietären Software-Tools) führt an der Wahl des DBKits zur Entwicklung der benötigten Applikationen kein Weg vorbei. Es ist vergleichbar mit der unter MS Windows führenden ODBC-Schnittstelle.

Generell können auch die unter MS Windows laufenden Computer auf die archäologischen Daten des SYBASE SQL-Servers zugreifen. Dazu ist über die TCP/IP Netzwerk-Software hinausgehend zusätzliche Software seitens des Datenbanksystem-Herstellers SYBASE (NetLibrary für Windows) notwendig, als Bindeglied zwischen der Netzwerk-Software und speziellen Datenbanktreibern. Solche Treibersysteme sind ODBC von Microsoft sowie IDAPI (benötigt zusätzlich den SYBASE SQL Link), das hauptsächlich von Borland entwickelt wurde. ODBC- und/oder IDAPI-fähige Programme (alle neueren auf Client-Server-Prinzipien aufgebauten Datenbanksysteme und Programme) können dann auf die Datenbestände des SQL-Servers zugreifen. Für die NetLibrary sind allerdings zusätzliche Lizenzen zu kaufen, dies richtet sich nach der Anzahl der Arbeitsstationen. Vor allem Microsofts ODBC-System (das inzwischen auch auf OS/2 und einige UNIX-Systeme portiert wurde) findet in der MS Windows Welt zunehmende Verbreitung.

IDAPI (Independent Database Application Programming Interface) ist die Schnittstelle der Firma Borland zum Zugriff auf und zur Weitergabe von Daten. Es spielt keine Rolle, ob die Daten in dateibasierten (dBase, Paradox) oder serverbasierten Datenbanksystemen (Oracle, Sybase, Informix) abgelegt sind. Das Konkurrenz-System zu IDAPI ist ODBC (Open Database Connectivity) von Microsoft mit analoger Zielstellung und hat sich unter MS Windows als De-facto-Standard durchgesetzt. Deshalb integriert IDAPI inzwischen das "übermächtige" ODBC.

Allerdings reichen die personellen und finanziellen Möglichkeiten des LfA nicht aus, um parallel gleichartige DB-Anwendungen der unter dem Aspekt der Programmentwicklung völlig unterschiedlichen Betriebssystemen NEXTSTEP und MS DOS /MS Windows zu entwickeln. Dies ist wohl auch nicht notwendig.

1.3. Auswahl eines Geografischen Informationssystems

Für die digitale Kartenarbeit wurde ein GIS gesucht, das eine ganze Reihe von Voraussetzungen zu erfüllen hat. Dazu zählt:

1. Plattformübergreifende Verfügbarkeit, das GIS sollte Programm-Versionen für NEXTSTEP, aber möglichst auch für MS Windows aufweisen. Daraus ergeben sich gleich weitere Anforderungen. Der Zugriff für alle Rechner-Plattformen soll auf dieselbe Daten- und Kartenbasis erfolgen, um nicht getrennte Datenbestände für NEXTSTEP- und Windows-Versionen des GIS vorhalten zu müssen. Die Bedienung sollte unter beiden grafischen Oberflächen zumindest ähnlich sein (soweit im Rahmen des unterschiedlichen Aufbaus von NEXTSTEP und Windows überhaupt möglich), um den Einarbeitungsaufwand für Programmbenutzer minimal zu halten.
2. Ein wesentliches Kriterium ist die Offenheit zu anderen Systemen. Das betrifft vor allem die Anbindung an Datenbanken und die Möglichkeit des Datenaustauschs mit weiteren Programmen (in unserem Fall archäologische DB-Anwendungen). Es ist aber nicht vermeidbar, daß unter NEXTSTEP und MS Windows unterschiedliche Programmier- und Datenaustausch-Mechanismen zum Einsatz kommen. Schwerpunkt ist auf die Datenaustauschfähigkeiten der NEXTSTEP-Version zu legen. Am geeignetsten sind diese Anforderungen durch einen modernen Client/Server-Aufbau des GIS zu realisieren. Das GIS muß zudem Möglichkeiten der Konvertierung verbreiteter Vektorformate in das eigene Format vorsehen, um vorhandene Datenbestände laden zu können.
3. Fundamentale Eigenschaft von Geografischen Informationssystemen ist die Anzeige von Rasterbildern (Karten) als Hintergrund und unsichtbaren Schichten (Layern), auf denen die verschiedenartigsten Objekte (z. B. Luftbildfundstellen oder von Stellungnahmen betroffene Flächen) als sogenannte Vektorobjekte dargestellt und editiert werden. Zu beeinflussende Eigenschaften sind u. a. Füll- und Umrandungsfarben, Schraffur, Strichstärken und Linientypen sowie der Einsatz von Symbolen. Die Objekte müssen verschoben und in ihrer Ausdehnung verändert werden können. Selbstverständlich ist die nutzerspezifische Abspeicherung der dargestellten Objekte.
4. Da das LfA über ein herkömmliches 10MBit/s Ethernet-Netzwerk verfügt, ist geringe Netzwerk-Belastung beim Laden von beispielsweise Karten durch Einsatz spezieller Verfahren Voraussetzung.

Das einzige Geografische Informationssystem mit diesen Parametern, das gefunden werden konnte (in Zusammenarbeit mit B. I. M. Consulting Magdeburg), ist VISOR der Bremer Firma megatel GmbH.

VISOR arbeitet nach dem Prinzip der verteilten Anwendungen (Client/Server), indem der visor VIEWER (mit VISOR wird im folgenden immer der visor VIEWER bezeichnet) über ein festgelegtes Protokoll mit anderen Applikationen Informationen austauscht[3]. Er übernimmt die Rolle des Servers gegenüber den als Clients fungierenden

anderen Anwendungen. Unter NEXTSTEP werden Distributed Objects (verteilte Objekte, NeXT's Standard zur Realisierung von über Programm- und Rechnergrenzen hinweg netzwerkweit verteilten Objekten) eingesetzt, die Windows-Version verwendet den DDE-Mechanismus (Dynamic Data Exchange). Die archäologischen DB-Anwendungen haben in diesem Rahmen die Rolle der Clients gegenüber dem als Server auftretenden Viewer von Visor inne.

Visor verwendet ein eigenes ASCII-Format, um Vektordaten über Objekte zu laden und zu speichern. Es ist auch möglich, Daten aus Fremdprogrammen zu übernehmen. Ein Weg führt über das weitverbreitete DXF-Format (AutoCAD). Ein Konvertier-Programm setzt DXF-Dateien in das VISOR-eigene VDB-Format um. Dies funktioniert auch in die entgegengesetzte Richtung. Der Import von DXF-Dateien wurde bereits erfolgreich getestet. Ein zweites, im GIS-Umfeld bekanntes Vektorformat (EDBS), kann ebenfalls importiert werden. Dies wurde allerdings noch nicht näher untersucht.

Um mit relativ wenig Hauptspeicher der eingesetzten Computer auszukommen, wird der gesamte Karten-Hintergrund (Rasterbilder im weitverbreiteten TIFF-Format) gekachelt (d. h. in kleine Rechtecke geschnitten) und es werden nur die augenblicklich für die Anzeige benötigten Kacheln im Speicher gehalten. Dies wird durch eine intelligente Software-Steuerung erreicht. Da die digitale Kartengrundlage auf einem Fileserver im Netzwerk gespeichert ist, werden beim Scrollen durch die Karte (es ist im Fenster immer nur ein Kartenausschnitt sichtbar) stets nur wenige Kacheln geringer Dateigröße über das Netzwerk geladen. Die Bedingung minimaler Netzwerkbelastung wird erfüllt.

Der ganze Komplex der Erstellung der digitalen Kartenbasis wurde bereits von R. Weise[4] beschrieben.

2. Spezifische Anforderungen an die DB-Software und das GIS

In den folgenden Ausführungen wird nur noch auf unter NEXTSTEP laufende und zu entwickelnde Software eingegangen.

2.1. Zugriff auf Datenbanken aus den archäologischen Programmen

Die bei J. Bittner und M. Stock[5] und in der Einleitung beschriebenen Programme haben hauptsächlich in der Datenbank Arche gespeicherte Informationen zu verarbeiten. Je nach Applikation sind davon ganz bestimmte Tabellen und deren Beziehungen untereinander betroffen, beispielsweise in der Luftbild-Anwendung neben der Tabelle Luftbilder die Tabellen Gemarkung, Fundstellen, Koordinaten und MTBlaetter. Z. B. wird beim Programmstart ein Browser mit den Arche-Tabellen und Views angezeigt (ein Browser zeigt Daten in einer spaltenorientierten Übersicht an, vom selektierten Begriff abhängige Daten werden in der nächsten Spalte dargestellt). Eine Tabelle oder View ist per Mausklick auszuwählen und danach die gewünschten Spalten (Felder) der Tabelle. Anschließend wird in einem Inspektorfenster eine Suchbedingung, die Suchbegriffe einer oder mehrerer Tabellenspalten in Kombination enthält, zusammengestellt und die Abfrage gestartet. Andererseits können auch aus einer Liste fertige Abfragen, die auf die Problemstellungen der jeweiligen Anwendung zugeschnitten sind, ausgesucht werden, bei denen nur noch die Suchbedingungen einzutragen sind. Ersteres ist dafür gedacht, nicht

nur applikationsspezifische Aspekte zu berücksichtigen, sondern dem Programmnutzer zu erlauben, auch andere in der Arche-Datenbank gespeicherte Informationen für seine Arbeit zu untersuchen. Im zweiten Fall werden hauptsächlich Views (festgelegte Sichten auf die Datenbank, bei denen bestimmte Aspekte der Datenselektion aus mehreren inhaltlich verbundenen Tabellen vorliegen und die bereits innerhalb der Datenbank programmiert sind) verwendet. Als Variante für professionelle Nutzer gibt es dann noch die Eingabe des kompletten Kommandos der Abfrage. Umfangreichere Erläuterungen folgen im Abschnitt zur Programmierung.

Diese Vorgänge sind beliebig oft wiederholbar und Suchbedingungen können erweitert bzw. weiter eingeschränkt werden.

In den einzelnen Applikationen treten immer wiederkehrende Arbeitsvorgänge auf. Das sind:

1. Verbindungsaufbau zur Datenbank Arche,
2. Auswahl der benötigten Tabelle(n) und Tabellenspalten aus einem Browser,
3. Eingabe von Suchbegriffen für bestimmte Tabellenspalten einer Tabelle bzw. über Tabellengrenzen hinweg nach dem Prinzip Query by Example (QBE), d. h. Angabe von Filtern in Kombination,
4. Abschicken der so konstruierten Abfrage an die Datenbank (Select),
5. Entgegennahme der Antwort-Tabelle(n) mit den gefundenen Datensätzen,
6. Anzeige dieser Tabellen mit den gefilterten Datensätzen in Tabellenform und je nach Anwenderwunsch eventuell datensatzweise in einem Formular mit Textfeldern (Beschriftung = Tabellenspalten-Bezeichnung). Bilder und Texte werden in eigenen Fenstern dargestellt,
7. eventuell Vornahme von Änderungen und Einfügungen (Update, Insert),
8. anschließend Absenden dieser Änderungswünsche an den SQL-Server,
9. Rückmeldung an den Programmnutzer bei Scheitern des Updates/Inserts (in der Datenbank gewährleisten Trigger die Konsistenz des Datenbestands und verhindern unzulässige Änderungen an den Tabellen). Ein Trigger löst Aktionen in Form von SQL-Statements an einer Datenbanktabelle (an die der Trigger gebunden ist) ereignisgesteuert (Einfügen, Löschen und Ändern von Datensätzen) aus.
10. Ein Spezialfall ist die Ausführung von Stored Procedures, das sind in SQL geschriebene Programme zur Datenbank-Manipulation, die direkt auf dem Datenbank-Server ablaufen. Diese müssen, zusammen mit einer Anzahl von Parametern, als Befehl an den SQL Server übergeben werden und führen komplexe Vorgänge innerhalb der Datenbank aus. Sie liefern Rückgabewerte, die auszuwerten sind.

Die Zusammenarbeit der archäologischen DB-Programme mit dem GIS VISOR erfordert weitere Arbeitsschritte, die sich, abgesehen vom Datenfluß zwischen VISOR und den Anwendungen, auf die eben genannten Abläufe reduzieren lassen.

2.2. Die Anforderungen an das Geografische Informationssystem VISOR

Aus Anwendersicht sind folgende Punkte zu erfüllen:

1. Es existieren mehrere Schichten von geeichten (geokodierten) Karten (d. h. die Karten sind bezüglich ihrer Koordinaten geeicht), mit
   - unterschiedlichem Maßstab bzw.

– unterschiedlicher Kartenart bei gleichem Maßstab.
Die gewünschte Kartenart kann ausgewählt und die zugehörige Karte geladen und angezeigt werden. Durch Markieren eines Bereichs mit der Maus und Wählen der Zoom-Option wird bei Vorhandensein einer Karte größeren Maßstabs mit mehr Details automatisch dieselbe dargestellt. Dieser Vorgang ist auch umgekehrt, von größerem zum kleineren Maßstab ausführbar. So ist es möglich, schnell von einer Übersichts- zu einer detaillierteren Karte überzugehen und wieder zurück oder von einem Meßtischblatt zu einer geologischen Karte. Die Geokodierung der Karten sorgt dafür, daß immer die richtigen Ausschnitte sichtbar sind (Bereiche gleicher Koordinaten) und die angezeigten Objekte an den richtigen Orten dargestellt werden. Auf den Karten kann der gerade sichtbare Ausschnitt durch Scrollen verändert werden, benötigte Kartenteile lädt VISOR nach. Bei Bedarf kann der Kartenhintergrund auch ausgeblendet werden.

2. Auf den Karten sind beliebige Vektorschichten mit daraufliegenden Strukturen in Form von geometrischen Objekten wie Punkten, Symbolen oder Flächen darzustellen (beispielsweise Verwaltungsgrenzen, von Bauanträgen betroffene Flächen oder Luftbildfundstellen). Es ist eine beliebige Anzahl solcher Vektorlayer gleichzeitig auf den Hintergrund projizierbar.
3. Die einzelnen Objektlayer sind separat und nutzerspezifisch abspeicherbar.
4. Es wird eine Vielzahl von darstellbaren Objekttypen mit variablen Zusatzinformationen (Attributen) unterstützt. VISOR stellt folgende Objekttypen bereit:
   – Punkt, ist als solcher schlecht sichtbar, wird meist mit einem Symbol versehen
   – Linie
   – Kreis und Kreisbogen
   – Rechteck
   – Polygon
   – Polylinie
   – Text
   – Symbol (Icon)

   Attribute sind Rand- bzw. Füllfarbe (mit einstellbarer Transparenz), Strichstärke und Linientyp, Schriftart und -größe sowie Textausrichtung.
5. Die Objekte können bei Anzeige einzeln in all ihren Attributen verändert werden. Das kann interaktiv durch den Nutzer, oder automatisch durch die archäologischen DB-Anwendungen gesteuert erfolgen. Es empfiehlt sich, abhängig von den darzustellenden archäologischen Strukturen, einheitliche Richtlinien für den Einsatz der Objekttypen und Zusatzinformationen festzulegen. Beispielsweise sind Fundstellen in Abhängigkeit von Zeit und Fundart mit verschiedenen farbigen Symbolen anzuzeigen. Die einzelnen Layer mit darauf abgebildeten Objekten sind in Dateiform speicher- und ladbar (VISOR's vdb-Format).

2.3. Das Zusammenwirken zwischen den Datenbank-Anwendungen und VISOR

Als modernes GIS ist VISOR durch andere Anwendungen steuerbar und kann in seiner Funktionalität an ganz spezielle Erfordernisse angepaßt werden (entsprechende Vorbetrachtungen wurden unter 1.3. kurz dargelegt). VISOR kann mit den archäologischen

DB-Anwendungen kommunizieren, um die in diesen zu bearbeitenden Vorgänge optisch aufzubereiten. Zu beachten ist, daß ein Informationsfluß in zwei Richtungen erfolgt: von VISOR zu den archäologischen Anwendungen und umgekehrt.

Spezielle Anforderungen, die über eine Schnittstelle zwischen VISOR und den archäologischen DB-Anwendungen zu realisieren sind, werden in den folgenden Unterpunkten erläutert.

2.3.1. Datenfluß von den archäologischen Anwendungen heraus zu VISOR

Aus den Anwendungen sollen die dort bearbeiteten Vorgänge auf der Kartenbasis sichtbar gemacht werden. Der Programm-Nutzer hat, wie in 2.1. beschrieben, die auf die Karte zu projizierenden Objekte aus der Arche-Datenbank zu selektieren (alle in einem rechteckigen Areal befindliche Luftbildfundstellen durch Angabe der Eckkoordinaten als Suchbedingung). Falls noch nicht aktiviert, kann VISOR direkt aus dem Programm heraus mit der gewünschten Karte gestartet werden. Die im (je nach) DB-Programm dargestellten unterschiedlichsten Sachverhalte wie Fundstellen, ausgegrabene Fundstücke oder Ausmaße der Flächen von Bauobjekten für Stellungnahmen können über einen Programm-Menüpunkt an VISOR übergeben werden. Grundvoraussetzung ist das Vorliegen von Koordinaten-Paaren für an VISOR zu übertragende Objekte, sonst ist keine grafische Visualisierung möglich. Es ist notwendig, im Programm weitere Angaben zu den an VISOR weiterzuleitenden Objekten zu treffen. Um welche Angaben es sich dabei handelt, hängt von den darzustellenden Objekten ab. Bei einem flächigen Bereich, wie einem Verantwortungsbereich von Beauftragten, der durch mehrere Eckpunkte bestimmt wird (Polygon), sind Attribute wie Füll- und Randfarbe in einem Objektinspektor festzulegen. Punktförmige Objekte sind mit Symbolen, die in einer Pickliste des Inspektors zur Verfügung gestellt werden, zu versehen. Um den Aufwand für diese Arbeiten vertretbar zu halten, können ganze Gruppen von Objekten (die tabellarisch angezeigt werden) mit der Maus markiert und auf einmal mit dem gewünschten Symbol belegt werden. Alternativ kann dies auf Wunsch auch automatisch per Programm geschehen, wobei z. B. Typ oder zeitliche Einordnung von Fundstücken mit definierten Symbolen und farblichen Attributen belegt werden. Anschließend werden die Objekte (letztendlich Kombinationen aus Werten verschiedener Felder von Arche-Tabellen) an VISOR gesendet.

Es treten im Arbeitsablauf immer wiederkehrende Abläufe auf, die in allen archäologischen DB-Anwendungen ähnlich sind:

1. Verbindungsaufnahme zu VISOR (eventuell mit vorherigem Starten des VISOR) sowie die Auswahl der Kartenbasis,
2. Zusammenstellung der an VISOR zu sendenden Objekte mit allen zur Anzeige benötigten Eigenschaften, wie geometrischem Darstellungstyp (Kreis, Polygon oder Symbol), Zusatzattributen (Farbe, Linienform, Schriftfont) unter Beachtung des Vorliegens von Koordinaten für die Objekte, n Koordinaten-Paaren bei flächenhaften n-eckigen Objekten, einem Paar bei Punkt-, Kreis- oder Symboldarstellung,
3. Anfordern eines Layers von VISOR, auf dem die Objekte dargestellt werden,
4. Ausschalten der Darstellung von schon vorhandenen Layern, um die Übertragung der eigenen Objekte zu beschleunigen und Einschalten der Objektverfolgung mit dem

Ziel, alle eingefügten Objekte in der Mitte des Karten-Fensters anzuzeigen,
5. Senden der Objekte an VISOR,
6. Rücknahme der Objektverfolgung und Wiedereinblendung der anderen vorhandenen Objektschichten.

2.3.2. Datenfluß von VISOR zu den Client-Anwendungen und zur Arche-Datenbank

Vorbemerkt werden muß, daß eine Anbindung des Geografischen Informationssystems VISOR an die Arche-Datenbank nur über die DB-Applikationen erfolgen kann. Dies ist erforderlich, da in der derzeitigen VISOR-Version (1.4) keine Möglichkeit besteht, ohne Client-Anwendungen z. B. eine Datenbank-Schnittstelle zu realisieren. Das Konzept des Ladens sogenannter Bundles ermöglicht unter NEXTSTEP die einfache Erweiterung von Programmen durch Anbindung zusätzlicher Software-Bausteine. Eine entsprechende Anregung zur Implementierung eines Einsprungpunkts in VISOR zum Anfügen von Programm-Funktionalität ist an die Herstellerfirma megatel GmbH weitergegeben worden. Folgende Arbeitsabläufe sind denkbar:

1. Die DB-Applikationen stellen die interessierenden Daten bereits visuell auf einem Layer von VISOR dar. Der Nutzer möchte nun zusätzliche Informationen zu allen oder einigen dieser Objekte aus der Datenbank abfragen und selektiert diese mit der Maus (was nur erfolgreich ist, wenn die Objekte auf dem gerade aktiven Layer liegen). Dieses Anwählen von Objekten beantwortet VISOR mit einer Nachricht an die Client-Anwendung, die auf die Botschaft reagieren kann. Beispielsweise kann die Applikation das Gerüst einer Suchbedingung erzeugen, mit den Koordinaten der Objekte als einer Bedingung. Der Programm-Benutzer hat dann noch interaktiv die ihn interessierenden Kriterien für die Abfrage einzutragen. Das sollte für fortgeschrittene und mit der Struktur der Arche-Datenbank vertraute Nutzer durch direkte Eingabe der "where-Bedingung" des Kommandostrings möglich sein, aber auch durch Auswahl der Tabellen und Spalten sowie der Wertebereiche im Abfrageinspektor, wie unter 2.1. beschrieben. Nach Ausführen der Abfrage werden die Ergebnisse in der archäologischen DB-Applikation in Tabellenform angezeigt (analog Abschnitt 2.1.).
2. Eine andere Vorgehensweise beginnt in VISOR mit dem Einstellen des gewünschten Gebiets auf der Karte im Fenster des VISOR-Viewers. Allerdings muß eine bereits vorher gestartete Client-Anwendung einen Layer angefordert haben, um die Nachrichten des VISOR an dieses Programm zu leiten. Darauffolgend wird ein rechteckiger Kartenbereich mit der Maus markiert (eine Bereichsabfrage), um z. B. Informationen über in diesem Bereich liegende Luftbildfundstellen zu erhalten. Das GIS schickt eine Meldung an das Client-Programm, daß eine Bereichsabfrage vorliegt und übermittelt die Eck-Koordinaten. Anschließend wird in der DB-Anwendung eine Datenbankabfrage generiert, natürlich unter Einbeziehung der Tabelle Koordinaten mit den Bereichskoordinaten als Bedingung, und der Nutzer kann als weitere Filterbedingungen angeben, wonach gesucht wird (Fundstellen, von Bauanträgen betroffene Flächen oder Luftbildfundstellen). Es folgt wieder das Abschicken der Abfrage. Soll die Bereichsabfrage nicht einen rechteckigen, sondern einen polygonalen Bereich umschließen, ist ein geringfügig modifiziertes Vorgehen notwendig.

Anstatt mit der Maus einen rechteckigen Bezirk zu markieren, ist ein polygonales Objekt mit den gewünschten Eckpunkten zu erzeugen. Durch Festlegung ist die Gesamtzahl der Punkte eines Objekts auf (sicher ausreichende) 100 beschränkt. Alles weitere beschreibt der Arbeitsablauf 1.

3. Programmierung der Schnittstellen zwischen den archäologischen Anwendungen zum SYBASE SQL-Datenbankserver und zu VISOR

3.1. Einführung in die Programmierung unter NEXTSTEP

In Abschnitt 1.2. wurde das Betriebssystem NEXTSTEP bereits kurz vorgestellt. Die Firma NeXT Inc. betont die gegenüber anderen System-Plattformen kurzen Programm-Entwicklungszeiten auf dem eigenen System. NEXTSTEP ist als Plattform zur Programmierung sogenannter Mission Critical Applications besonders geeignet und in manchen Branchen (Finanzwelt, Druckvorstufe) beliebt. Dafür wurden geeignete Voraussetzungen geschaffen:

- Programmierung mit einer objektorientierten Sprache, Objective C, die einerseits weitaus leistungsfähiger als C++ ist (dynamisches Binden und dynamische Typprüfungen zur Programmlaufzeit, Message Handling zwischen Objekten u. a.), andererseits einfacher programmiert werden kann. Objective C ist näher mit Smalltalk als mit C++ verwandt. Die Programm-Performance ist durch die Laufzeit-Umgebung und die Kommunikation mit dem Display-PostScript Window Server nicht so effizient. Die Einarbeitungszeit in Objective C ist geringer als in C++.
- Offenheit, Applikationen können gemischt in Objective C, C++ und Standard C programmiert werden.
- Das Botschafts-Prinzip zum Nachrichtenaustausch zwischen Objekten erleichtert zudem die Programmierung von Routinen, die die Zusammenarbeit zwischen NEXTSTEP-Anwendungen erleichtern. NEXTSTEP-Applikationen sind so meist keine großen monolithischen Programm-Pakete mehr, wie z. B. WordPerfect oder Excel unter MS Windows, sondern die Funktionalität wird auf kleinere, für ihre Aufgabe spezialisierte Programme verteilt. Diese Modularisierung erleichtert die Wiederverwendbarkeit einmal geschriebener Programm-Codes.
- Display PostScript, es spielt keine Rolle, welches Gerät zur Ausgabe gedacht ist, ob Monitor oder Drucker. Der Entwickler muß sich nicht um Eigenheiten der Druckeransteuerung kümmern. PostScript macht NEXTSTEP auch für die Aufbereitung von Daten in der Druckvorstufe für grafische Betriebe interessant.
- In der Entwicklungsumgebung sind eine Anzahl von Software Kits enthalten. Ein Kit ist immer für bestimmte Aspekte der Programmierung verantwortlich. Das Application Kit z. B. enthält alle wesentlichen Basiselemente zur Entwicklung von auf der grafischen Oberfläche lauffähigen Programmen (Fenster-Objekte, Textfelder, Browser, View-Objekte zur Darstellung von Text und Bildern, Browser, komplette Standard-Dialoge zum Datei-Öffnen, Speichern und Drucken einschließlich Druck-Preview, eine Schnittstelle zum Mail-Programm u. v. a. mehr). Die NEXTSTEP-Oberfläche (der Workspace) ist selbst unter Einbeziehung des AppKits programmiert worden. Das Database Kit ist für die Verbindung zu Datenbank-Servern gedacht. Es gibt noch

Music Kits, 3D Kits, ein Indexing Kit sowie die Distributed Objects zur netzwerkweiten Kommunikation von Objekten über Programmgrenzen hinaus.

- Eine leistungsfähige Entwicklungsumgebung aus Project und Interface Builder mit grafischer Schnittstelle zum gdb-Debugger, der zur Fehlersuche und zum schrittweisen Abarbeiten von Programmen (Analyse des Programmablaufs) eingesetzt wird. Im Project Builder wird der Rahmen für einige Projekte geschaffen und gewartet, die vielfältigsten Programmkomponenten integriert. Mittels Interface Builder können, wie weiter oben erwähnt, Programm-Komponenten visuell aus Paletten ausgewählt und in die Anwendung eingebaut sowie Beziehungen zwischen Objekten per Maus generiert werden. Das reduziert den selbst zu schreibenden Quell-Code und ist außerdem eine große Zeitersparnis. Es kann weiterhin auf von Fremdherstellern gelieferte Paletten mit Objekten benötigter Funktionalität zurückgegriffen werden.
- Skalierbarkeit, NEXTSTEP läuft inzwischen auf vier unterschiedlichen Rechnerplattformen. Es handelt sich um die alte NeXT-Hardware auf MC68040-Basis, 486- und Pentium-PCs, die HP9000-Plattform und die Sun SPARC-Architektur. So können Anwendungen unterschiedlichster Bandbreite an Rechner-Hardware und Netzwerk realisiert und bei steigenden Anforderungen auch zukünftig skaliert werden. Das ist im Client/Server-Umfeld von entscheidender Bedeutung.
- Die Verfügbarkeit von NEXTSTEP für bisher vier Architekturen erfordert die Lauffähigkeit der entwickelten Anwendungen auf allen Plattformen, um nicht für jeden Rechnertyp die Programme neu kompilieren zu müssen. NeXT führte als erster Hersteller den Multi-Binary-Support ein. Beim Übersetzen des Programms wird einfach der Maschinencode für alle benötigten Rechner-Plattformen erzeugt und im Programm abgelegt. Beim Starten des Programms prüft das Programm, auf welchem Rechner es läuft und lädt nur den Code für den entsprechenden CPU-Typ.

NEXTSTEP-Programmierung bedeutet objektorientiert programmieren. Entsprechende Konzepte gewannen mit dem Aufkommen grafischer Oberflächen und der Entwicklung von umfangreichen Funktionsbibliotheken an Bedeutung, um dort auftretende Probleme zu vermeiden, wie überproportionale Fehlerzunahme bei Entwicklung großer Projekte, der Zusammenarbeit von Software-Teams und dem Versionsmanagement. In der konservativen Programmierung werden zwar auch Module in Form von Funktionen und Prozeduren eingesetzt, um komplexe Zusammenhänge in kleinere Module zu zerlegen, aber die objektorientierte Programmierung geht einen Schritt weiter. Der erste relevante Aspekt ist die Kapselung der Daten. Die Variablen eines Moduls (die mit ihren Werten den Zustand des Moduls kennzeichnen) und die auf sie zugreifenden Befehlsstrukturen (die das Verhalten solcher Module bestimmen) werden in einer Einheit zusammengefaßt, einem sogenannten Objekt[6]. Von Ausnahmen abgesehen, dürfen auf die Variablen (Attribute bzw. Instanzvariablen genannt) nur die Funktionen des Objekts selbst, die Methoden, zugreifen. Objektorientiert programmierten Applikationen liegt eine Klassenhierarchie zugrunde. Eine Klasse (oder Klassenobjekt) ist so etwas wie eine Schablone eines Objekts, von der bei Bedarf Kopien (Instanzobjekte, im Sprachgebrauch kurz Objekte genannt) erzeugt werden. Alle Klassen werden von einer Basisklasse (Rootklasse), die sämtliche benötigte Grundfunktionalität eines Objekts beinhaltet, abgeleitet, wie Speicherreservierung für die Instanzvariablen und Speicherfreigabe sowie die Fähigkeit auf Botschaften anderer Objekte zu reagieren.

Funktions-Erweiterungen und -Modifizierungen werden durch Vererbung (einem weiteren Prinzip der objektorientierten Programmierung) realisiert. Das Kindobjekt erbt alle Instanzvariablen und Methoden der Elternklasse. Zusätzliche Instanzvariablen und Methoden können definiert werden, um ein Objekt an seinen Aufgabenbereich anzupassen.

Es existiert auch die Möglichkeit, Methoden des Elternobjekts zu verändern, einerseits indem diese selbst in der geänderten Methode aufgerufen wird (Erweiterung der Methodenfunktionalität), oder durch komplette Neuimplementierung der Methode. Diese Fähigkeit unterschiedlicher Objekte auf dieselbe Botschaft (Methodenaufruf) verschieden zu reagieren, wird als Polymorphismus bezeichnet.

Eines der Grundprobleme der objektorientierten Programmierung ist der Aufbau von Klassenbibliotheken, wie der schon besprochenen Kits. Es gehört viel Übung und Erfahrung dazu, Designfehler beim Konzipieren solcher Bibliotheken zu vermeiden. Diese sind nachträglich kaum zu korrigieren. Mit qualitativ hochwertigen Klassenbibliotheken kann der Aufwand bei der Kodierung eigener Projekte gering gehalten werden. Man sollte den Arbeitsaufwand bei der Einarbeitung in diese Bibliotheken trotzdem nicht unterschätzen.

3.2. Allgemeine Überlegungen zu den Programmschnittstellen

1. Angestrebt wird eine vollständige Abdeckung der unter 2. diskutierten Anforderungen.
2. Die beiden Schnittstellen archäologische Anwendung - Datenbanken (Datenbank-Schnittstelle) und archäologische Anwendung - VISOR (VISOR-Schnittstelle) arbeiten unabhängig voneinander (Abb. 1). Auf diese Art und Weise kann eine allmähliche Implementierung der Programmfunktionen erfolgen, z. B. erst Programmierung der kompletten Datenbank-Schnittstelle oder Kern-Teilen derselben und spätere Kodierung der VISOR-Schnittstelle. Diese Vorgehensweise erzwingt eine zusätzliche Vermittlerschicht zwischen beiden Schnittstellen, ermöglicht aber eine starke Modularisierung. Die Schnittstellen können deshalb in anderen Projekten ebenfalls verwendet werden.

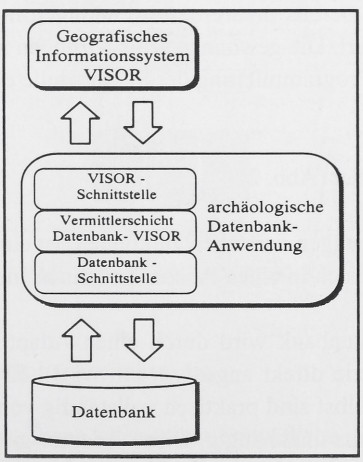

Abb. 1: Datenfluß zwischen den archäologischen Anwendungen sowie dem Datenbank- und dem Geografischen Informationssystem

3. Die Modularisierung wird durch die Erfordernisse der Wiederverwendbarkeit und Zeitersparnis bei der Programmierung und schnellen Verfügbarkeit von Basisfunktionen vorgeschrieben sowie der Vereinfachung der Programmierung durch Einsatz gleicher Objekte und Methoden bei allen archäologischen Anwendungen.
4. Zusammenfassung komplexer Abläufe in nur wenigen einzusetzenden Befehlen. Eine komplette Datenbankabfrage mit allen benötigten Parametern soll beispielsweise mit einem oder einigen wenigen Befehl(en) erfolgen oder analog das Senden einer Reihe von Objekten zur Anzeige im Geografischen Informationssystem.
5. Die beiden Schnittstellen werden entweder durch (Objective) C-Bibliotheken (Libraries) oder Bundles (Verzeichnisse, die vom Interface Builder generierte und in nib-Dateien archivierte Objekte und/oder kompilierten Code sowie andere Resourcen, wie Bilder und Sounds, enthalten) realisiert. Das hat den Vorteil, bei der Entwicklung der archäologischen DB-Applikationen nur Header-Dateien mit Typdeklarationen beim Kompilieren einzubinden und den Code der Schnittstellenelemente nicht ständig mit zu übersetzen. Im Falle von statischen Libraries wird der Schnittstellencode beim Linken angefügt. Diese herkömmliche Lösung hat den Nachteil, daß nicht auf im Interface Builder (IB) aus Objekt-Paletten entnommene und in nib-Dateien archivierte Objekte zurückgegriffen werden kann, alle Zusammenhänge müssen per Quelltext programmiert werden. Es wird erst in der Kodierungsphase entschieden, ob nicht auf Bundles, die auch visuell im IB generierte Objekte enthalten können und zur Laufzeit des Programms geladen werden, zurückgegriffen wird oder eventuell beide Vorgehensweisen kombiniert werden. Die in den Schnittstellen programmierten Objektstrukturen und Befehle sind bei beiden Varianten identisch.
6. Im LfA konzentrieren sich zur Zeit die Anstrengungen auf die Programmierung der Datenbank-Schnittstelle.

3.3. Die Datenbank-Schnittstelle

Im LfA wurden seit 1993 bereits mehrere Anwendungen mit DB-Funktionalität (unter Einsatz des DBKit) realisiert. Die gewonnenen Erfahrungen und Teile des geschriebenen Quellcodes fließen in die Programmierung der Schnittstelle ein.

3.3.1. Der Aufbau des DBKit (Abb. 2)

Um die folgenden Erläuterungen verstehen zu können, ist eine Einführung in das DBKit unumgänglich. Es gliedert sich in einen Access und einen Interface Layer sowie weitere Komponenten.

Die Verbindung zur Datenbank wird durch einen Adapter aufrechterhalten (der im Hintergrund arbeitet und nie direkt angesprochen wird). Er ist das DBMS-spezifische Element. Die Kitklassen selbst sind praktisch vollständig von der Struktur des darunterliegenden Datenbanksystems unabhängig. Diese Adapter gibt es für nahezu alle marktüblichen RDBMS (Relationale Datenbankmanagement-Systeme). NeXT selbst liefert sie für SYBASE und Oracle.

Die Sicht der mit dem DBKit geschriebenen Programme auf die zugrundeliegende Datenbank erfolgt über ein Datenbank-Modell, das mit der DBModeler-Applikation zusammengestellt wird. Im Modell können alle oder eine Auswahl der Tabellen der Datenbank stehen (nicht interessierende Tabellen werden auf diese Art und Weise ausgeblendet). Wesentlich ist noch, daß Beziehungen zwischen Tabellen über sogenannte Relationships direkt im Modell generiert werden. Dies ist bei 1 : 1-Beziehungen eine Alternative zu in der Datenbank abgelegten Views. Generell können auch 1:n- und m:n-Relationen aufgebaut werden.

Die Komponenten des Access Layers warten ein Modell der Datenbank und realisieren die Verbindung zur Datenbank (DBDatabase), puffern die vom DB-Server erhaltenen Daten zwischen (DBRecordStream und -List) und konvertieren Werte in Objective C Datentypen (DBValue). Zum Aufbau von Suchbedingungen verwendet man DBQualifier-Objekte. Zum Abbilden der DB-Elemente Tabelle (Entity im Entity-Relationship-Modell) und Feld, Spalte (Attribut, Property) werden DBEntity- und DBProperty-Objekte verwendet.

Die wichtigsten Objekte der Interface-Schicht sind in einer Interface Builder Palette integriert (DBModule, DBImageView, DBTableView), die beiden letzten erklären sich selbst. Ein DBModule ist ein Management-Objekt, das alle anderen benötigten Objekte des Access Layer für einen Datenbankzugriff automatisch erzeugt und wartet und zudem die Verbindungen zwischen Anzeige-Komponenten, wie z. B. DBTableViews und Datenhaltungsobjekten wie DBRecordLists über DBFetchGroups, koordiniert.

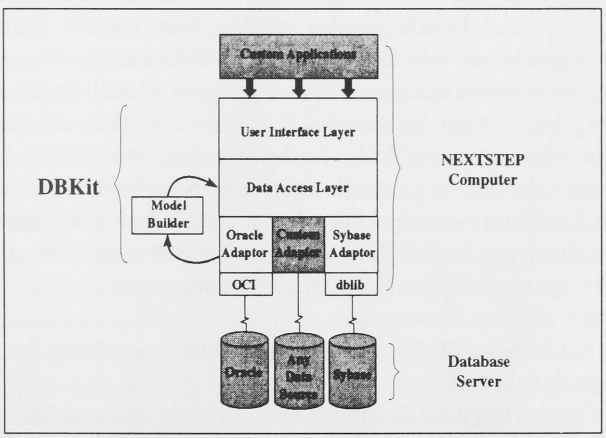

Abb. 2: Aufbau des Database Kit

3.3.2. Komponenten der Datenbankschnittstelle

Die Entscheidung für vorgegebene oder selbst entwickelte Abfragen wird dem Nutzer der Programme in Form zweier Menüpunkte überlassen.

Es werden die unter 2.1. angesprochenen Programmelemente (User Interface Elemente) für eine DB-Abfrage noch einmal aufgelistet:

1. Ein Abfrage-Objekt managt im Hintergrund das Zusammenspiel aller Komponenten,
2. Tabellen- und Spaltenbrowser,
3. Liste mit Abfragen für bestimmte Aufgabenstellungen,
4. Suchinspektor, schränkt den Wertebereich der Datensätze weiter ein,
5. Fenster mit Tabellen für die Ergebnisse der Abfrage,
6. separate Fenster zur Anzeige von Bildern und Textdateien.

Zentrales Objekt ist der AbfrageController. Er regelt das Zusammenspiel der unter Punkt 2-6 beschriebenen Programmteile und wartet eine Liste der vom Nutzer bereits getätigten Datenbank-Abfragen. Jede Abfrage ist ein selbständiges Objekt (DBAbfrage) und enthält verschiedene der oben genannten User Interface Elemente (4-6) sowie DBKit- und weitere Objekte. Abhängig von der Abfrage- und damit den darunterliegenden DB-Strukturen gehören ein DBModule, ein oder mehrere DBFetchGroups und DBRecordLists, Box-Views für den Suchinspektor mit Textfeld-Matrizen und PopUp-Lists für die Filterbedingungen (Punkt 4) sowie Anzeigeelemente für die Abfrageergebnisse (Panels mit DBTableViews, Textfeld-Matrizen und ImageViews, Punkte 5 und 6) dazu. Solch eine Abfrage kann außerdem beliebig oft mit unterschiedlichen Suchbedingungen (bei feststehenden Abfrage-Tabellen und -Feldern) wiederholt werden.

Das zweite Element ist der Tabellen- und Spaltenbrowser (kurz DBBrowser). Aufgabe ist das Auswählen der Tabellen und Spalten für die Abfrage und deren Übergabe an den Such-Inspektor. In der linken Browserspalte werden alle Arche-Tabellen und -Views aufgelistet. Die zweite Spalte zeigt die einzelnen Felder der gerade ausgewählten Tabelle. Der Nutzer selektiert die ihn interessierenden Spalten der gewählten Tabelle und drückt einen Übernahme-Knopf der die Tabellen- und Spaltenparameter übernimmt. Sollen Werte aus mehr als einer Tabelle geladen werden, kann das nur über die erwähnten Datenbank-Views geschehen oder über im NeXT-DBModeler zusammengestellte Relationships, was keine Einschränkung ist, da im Datenbank-Modell auch nachträglich neue Beziehungen aufgebaut werden können. Bei Relationen zwischen zwei Tabellen wird der Name der Relationship mit in die Felder-Spalte eingefügt und in einer dritten Spalte die Felder der zweiten Tabelle, von denen die interessierenden Felder durch den Nutzer ausgewählt werden. Eine vom Basisobjekt abgeleitete Klasse dient als Kontrollobjekt für die untergeordneten Elemente des DBBrowsers. Dieser DBBrowserController sorgt für das Zusammenspiel folgender Objekte (willkürliche Reihenfolge):

- Panel zur Darstellung des Browsers,
- modifizierter NXBrowser (DBBrowser), der auf die Darstellung der Tabellennamen und Spalten spezialisiert ist,
- DBDatabase-Objekt, das ein DB-Modell (mit den Tabellen und Spalten) einliest und dem Controller-Objekt die notwendigen Angaben für die Browser-Spalten in Form von DBEntites und DBProperties zur Verfügung stellt,
- ein Knopf (Button) zur Bestätigung der gewählten Tabelle mit Feldern.

Der DBBrowser blendet das Browser-Panel aus und übergibt dem AbfrageController die durch DBEntities und DBProperties repräsentierten Tabellen und Spalten. Der AbfrageController analysiert dieselben und generiert aus den Angaben ein DBModule mit allen weiteren zugehörigen DBKit-Objekten. Zum Beispiel wird die Anzahl der Fetchgroups und RecordLists von der Struktur eingesetzter Relationen bestimmt. Die neue Abfrage wird in die Abfrageliste eingefügt und der Such-Inspektor gestartet.

Der Menüpunkt 'Spezielle Abfragen' öffnet ein Untermenü, das als Menüpunkte Bezeichnungen verschiedener, auf die jeweilige DB-Anwendung zugeschnittener Abfragen enthält. Die zugehörigen Tabellen und Spalten werden nach Anwahl eines Menüpunkts an den AbfrageController weitergeleitet.

In einer späteren Version der Datenbankschnittstelle wird auch ein Kommandozeilen-Tool entwickelt, das direkte Eingabe von SQL-Statements erlaubt (ebenfalls per Menüpunkt erreichbar). Hauptbestandteile sind für die Befehlseingabe ein Fenster mit Scrollview für die Eingabe des DB-Befehls. Der Befehl wird über einen erweiterten DBBinder (MultiBinder) ausgeführt, dessen Delegate eine modifizierte DBAbfrage ist. Ein Delegate-Objekt reagiert auf bestimmte Nachrichten, die ihm von einem anderen Objekt zugeschickt werden. Es implementiert dazu spezielle Methoden. Das modifizierte Abfrage-Objekt nimmt die Abfrageergebnisse entgegen und stellt sie in Tabellenform dar. Es wird keine nachträglichen Modifikationen von Suchbedingungen oder anderen Befehlsteilen erlauben. Ursache ist die direkte Übergabe des Befehlsstrings an den SQL-Server und das Einsatzziel als Mini-Interpreter mit Ausgabe der Antworttabelle auf der grafischen Oberfläche in einer DBTableView.

Der Such-Inspektor ist ein Fenster, das die ausgewählten Tabellenspalten (der jeweiligen Abfrage) in beschrifteten Textfeldern untereinander darstellt. Bei Feldern mit alphanumerischen Zeichen kann dann ein (Teil-)String des gesuchten Begriffs eingetragen werden. Numerische Felder werden durch zwei nebeneinander stehende Textfelder und eine PopUpList mit mathematischen Operatoren ("=", "< >", usw.) repräsentiert. Zwei Textfelder werden benötigt, um die Eingabe von Zahlenbereichen zu ermöglichen. Die Suchbegriffe verschiedener Felder summieren sich, d. h. es wird eine logische UND-Verknüpfung aufgebaut. Diese Filterfunktionen werden in Form von DBQualifier-Objekten durch die Abfrage-Objekte koordiniert. Im oberen Teil des Such-Inspektors befindet sich eine PopUpList, die alle seit Programmstart aufgebauten Abfragen enthält, um das komfortable Hin- und Herschalten zwischen verschiedenen Abfragen zu erleichtern. Der Such-Inspektor ist per Menüpunkt aufrufbar und arbeitet eng mit dem Abfrage-Controller zusammen. Letzterer baut die darzustellenden Textfelder, PopUpLists und Buttons aus den Tabellenangaben- und Feldtypen zunächst in einer (unsichtbaren) Box zusammen, die dann in den Such-Inspektor eingeblendet und sichtbar gemacht wird. Jede Abfrage erhält eine eigene Box, die auf die Abfrage zugeschnitten ist. Der Vorgang des Hin- und Herschaltens zwischen unterschiedlichen Boxen nennt sich Swappen.

Nach dem Zusammenstellen der Abfrage einschließlich der Suchfunktionen wird über einen Knopf im Such-Inspektor die eigentliche Datenbank-Abfrage gestartet. Der AbfrageController wird benachrichtigt und verständigt die aktive DBAbfrage. Das DBModule desselben führt die Datenbank-Abfrage aus. Die vom SYBASE-Server zurückgelieferten Ergebnisse werden vom Abfrage-Objekt aufbereitet und geeignete Oberflächenelemente erzeugt, die die Daten anzeigen. Die DBAbfrage erzeugt ein Fenster und stellt die Datensätze in einer DBTableView dar. Um nach einer Spalte sortieren zu lassen, genügt es, diese Spalte mit der Maus an die erste Postion zu ziehen.

Per Knopfdruck wird vom Abfrage-Objekt eine Darstellung des gerade in der TableView selektierten Datensatzes in einem einfachen Formular (beschriftete Textfelder) für die Felder der Tabellen vorgenommen. Um 1:n Beziehungen zu visualisieren, werden vom Abfrage-Objekt zwei DBTableViews erzeugt und nebeneinander in einem Fenster

angezeigt. Das Anklicken von einer Zeile der Mastertabelle bewirkt das Laden der zugehörigen Datensätze der zweiten DB-Tabelle in die zweite DBTableView. Die erste Tabelle zeigt z. B. alle Gemarkungen an, die zweite Tabelle zeigt alle in dieser Gemarkung liegenden Fundstellen. Zur Synchronisierung der Daten in den unterschiedlichsten Anzeigeelementen und um auf Maus- und Tastaturereignisse reagieren zu können, richtet das Abfrageobjekt benötigte DBFetchGroups und DBAssociations ein. Im unteren Teil des Fensters sind Knöpfe mit den Standardfunktionen Einfügen, Ändern und Löschen angeordnet: Das ist abhängig von der jeweiligen archäologischen DB-Anwendung, aber auch vom speziellen Sachverhalt und den Zugriffsrechten des Nutzers auf die Daten in der Arche-Datenbank, deren Integrität durch ausgefeilte Mechanismen innerhalb der Datenbank geschützt wird.

Ein spezieller Fall ist das Einblenden von Bildern und Texten. Texte werden als RTF-Dokumente in Feldern bestimmter Arche-Tabellen gespeichert (RTF - Rich Text Format, von NeXT unterstütztes Dateiformat für Texte der Firma Microsoft). Sind Texte anzuzeigen, legt die DBAfrage zusätzliche DBFetchGroups und RecordLists für dieses Feld an und lädt bei Anwahl eines Datensatzes in der TableView den zugehörigen RTF-Text in ein eigenes Fenster (Abb. 3).

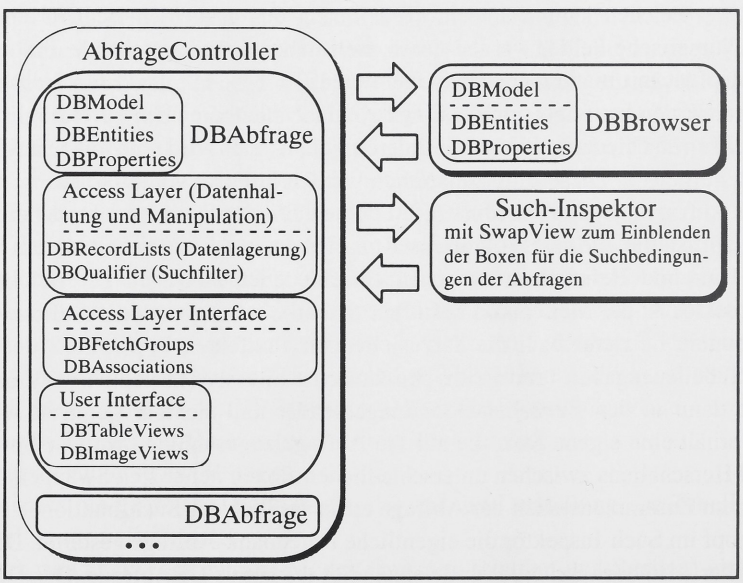

Abb. 3: Schema der Datenbank-Schnittstelle

3.3.2.1. Stored Procedures

Problematisch ist das Ausführen von Stored Procedures mit dem DBKit. Der zur Verfügung gestellte Objekt-Vorrat sieht das Übergeben von SQL-Statements vor, aber nicht die Entgegennahme von komplexen Ergebnissen als Reaktion des Datenbank-Servers. Die von der SQL GmbH Dresden in Zusammenarbeit mit dem LfA entwickelte Arche-

Datenbank enthält zur Überwachung und Steuerung der Aktivitäten in der Datenbank eine Vielzahl solcher Prozeduren. Vorgesehen war von Anfang an Start und Auswertung vieler Prozeduren durch Client-Programme. Im Übernahme-Programm, das Fundberichte bearbeitet und in die Arche-Datenbank einspeist, wird zur Umgehung des o. g. Problems nur eine einzige (hochkomplexe) Stored Procedure eingesetzt, deren Rückgabewerte in eine eigens dafür angelegte Tabelle fester Struktur abgelegt und von der Übernahme-Applikation ausgelesen werden. Diese simple Variante ist für die Vielzahl vorhandener Stored Procedures mit unterschiedlichen Parametern nicht übertragbar.

Gelöst werden kann das Problem durch Einsatz eines MultiBinder-Objekts anstelle eines DBBinders bei der Ausführung von Stored Procedures, das multiple Ergebnisse vom SQL-Server entgegennehmen kann. Eingesetzt wird weiterhin ein BinderDelegate-Objekt, welches jedesmal bei Erhalt einer speziellen Botschaft (diese Methode hat das BinderDelegate-Objekt zu implementieren) einen neuen Container mit dem vom MultiBinder gelieferten Ergebnis füllt und an andere Programmkomponenten zur Weiterverarbeitung übergibt.

3.3.3. Anzeige von Bilddokumenten

Bilder (Luftbilder, Fotos von Fundstücken) werden als externe Bilddateien auf den Festplatten gelagert. In der Datenbank wird nur der Dateiname des Rasterbildes, einschließlich des Pfads, gespeichert. Somit schwillt die Datenbankgröße nicht auf einen mehrstelligen Gigabyte-Betrag an und die Server Performance bei DB-Operationen bleibt vertretbar. Ein anderer Grund für den Verzicht der direkten Ablage von Bilddateien in der Datenbank sind Ladezeiten und Netzwerkbelastung. Der Nutzer kennt beim Absenden einer DB-Abfrage meist nicht die Anzahl der vom SQL-Server übermittelten Antwort-Datensätze. Enthält eines der Felder Bilddaten, können leicht mehrere Megabyte an Daten über das Netzwerk gehen, in ungünstigen Fällen sogar deutlich mehr. Dies ist unbedingt zu vermeiden.

Das Anzeigen von Bildern, aber auch das Speichern in unterschiedlichen Formaten und differierenden Kompressionsalgorithmen wird nicht nur bei den archäologischen DB-Programmen, sondern auch in vielen anderen Anwendungen ständig benötigt. Es ist günstig, die entsprechenden Programmkomponenten eigenständig zu behandeln, unabhängig von der eigentlichen Datenbankschnittstelle, ebenfalls in Form einer C-Bibliothek, die problemlos in die Programme integriert werden kann. Über eine Reihe von Aufrufmethoden kann das Verhalten des Basis-Objekts (BasicPic genannt) gesteuert werden. Bilder können in Abhängigkeit von ihren Abmessungen in einem Fenster mit einer einfachen ImageView oder einer ImageView in einer ScrollView angezeigt werden, was das bequeme Scrollen größerer Bilder erlaubt. Entscheidet man sich für ein Fenster mit ScrollView, wird ein ZoomScrollView-Objekt angelegt, das gleichzeitig über das Einblenden einer Zoom-PopUpList im horizontalen Scroller-Objekt vergrößerte und verkleinerte Darstellungen des Bilds mit automatischer Fenstergrößen-Anpassung unterstützt. Beim Laden mehrerer Bilder nacheinander wird über Aufruf-Parameter gesteuert, ob die Bilder immer im selben oder jeweils eigenen Fenstern sichtbar sind. Bei Bedarf blenden Applikationen in ihr Datei-Menü einen Unterpunkt zum Speichern der Bilder

ein. Wählbar sind TIFF- und EPS-Format, wobei für das TIFF-Format noch die gängigen Kompressions-Algorithmen einstellbar sind. Das Standard-SavePanel von NEXTSTEP wird um die Einblendung einer Box mit Auswahlelementen zur Abfrage dieser Parameter erweitert.

3.4. Die VISOR-Schnittstelle

Die megatel GmbH liefert für die Kommunikation von Client-Programmen mit VISOR die Objective C-Bibliothek libVisor.a und verschiedene Header-Dateien, die in die eigenen Anwendungen eingebaut werden müssen. Hauptbestandteile der Headerfiles ist die Definition des Datentyps GeoObject und unterstützender Typen sowie der Interface-Teil der für die Übertragung von Daten zwischen Applikationen und VISOR essentiell wichtigen Klasse ServerConnection.

3.4.1. Geoobjekte und die Klasse ServerConnection

VISOR kann nur mit Objekten umgehen, die vom Typ GeoObject sind. Das ist keine Einschränkung, da es sich um einen variabel anleg- und manipulierbaren Typ handelt. Alle unter 2.2. angesprochenen geometrischen Objekte mit variabel einstellbaren Attributen sind mit Geoobjekten realisierbar. Der Typ GeoObject ist als C-Struktur (ein aus mehreren anderen Variablen zusammengesetzter Datentyp) angelegt und enthält weitere Unions (Speicherbereich definierter Länge kannVariablen unterschiedlicher Datentypen aufnehmen) und C-Strukturen, um die gesamte Objektvielfalt generieren zu können sowie Variablen zur Identifikation des Geoobjekts und des Vektor-Layers, auf dem es liegt. Nur solche Geoobjekte können zur Anzeige auf Kartenmaterial an VISOR übergeben werden und umgekehrt liefert VISOR auch direkt im Viewer erzeugte oder in ihren Eigenschaften veränderte Geoobjekte an die Client-Anwendung zurück. Selbst Bereichsabfragen werden in Form einer solchen Struktur des Untertyps FLIG_RECT mit den vier Eck-Koordinatenpaaren des Rechtecks zur Markierung des Bereichs übermittelt.

Mit Instanzen der ServerConnection-Klasse nehmen Programme Kontakt zu VISOR auf. Da jede Anwendung ein eigenes ServerConnection-Objekt initiieren muß, können gleichzeitig mehrere Client-Applikationen mit VISOR Kontakt halten und auf getrennten Layern Objekte verwalten.

Die Methoden der ServerConnection-Klasse ermöglichen folgende Operationen:

- Verbindungsaufnahme und -beendigung zu einem VISOR auf dem eigenen oder im Netzwerk befindlichen Computer,
- Setzen eines Delegate-Objekts zur Benachrichtigung von bestimmten Ereignissen durch VISOR. Das Delegate-Objekt hat die Methode "objectReceived: data: by:" zu implementieren, die vom ServerConnection-Object mit den Parametern Botschaftstyp und betroffenes Geoobjekt zum Melden verschiedener Ereignisse ausgeführt wird. Solche Nachrichten betreffen das Selektieren, Anlegen, Ändern und Löschen von Geoobjekten,
- Anforderung und Verwaltung von Layern,

- Mitteilung von Ereignissen an VISOR. Es handelt sich um Methoden zum Anlegen, Verändern, Löschen und Selektieren von Geoobjekten. Als Argument wird ein Geoobjekt übergeben.
- Senden eines Befehls zum Aus- und Einblenden von Layern (freeze und unfreeze), zum Ein- und Ausschalten der Objektverfolgung (traceon und traceoff), zum Zoomen (zoomin und zoomout) sowie dem Verschieben des dargestellten Kartenbereichs (move) an VISOR.
- Das Konvertieren in ein anderes Koordinatensystem von Pixel- zu Geokoordinaten und umgekehrt.

Der Einsatz der dem Programmierer damit in die Hand gegebenen Funktionen ist ausreichend, zwingt ihn andererseits nicht vom VISOR-Viewer bereitgestellte Operationen mit diesen Methoden selbst zu programmieren (wie eine Anbindung an Datenbanken, da in der Regel auf existierende Datenbestände zugegriffen werden soll).

Megatel liefert dem Programmierer noch zwei Demo-Applikationen, die Möglichkeiten der Gestaltung eigener Programme, wie dem Anfordern von Layern, der Reaktion auf VISOR-Ereignisse, dem Laden und Speichern von kompletten Layern mit Objekten am Beispiel demonstrieren. Dieser Code kann für die eigene Programmierung zu großen Teilen übernommen werden.

3.4.2. Komponenten der VISOR-Schnittstelle

Dem Anwender von mit VISOR zusammenarbeitenden Applikationen soll das direkte Hantieren mit Geoobjekten und Layern erspart bleiben. Diese Aufgabe übernehmen im Hintergrund arbeitende Objekte. Diese Objektschicht ist unabhängig von der Herkunft im VISOR-Viewer darzustellender Daten. Sie ist für Aufbau und Abbau der Verbindung zu VISOR, dem Übertragen von Geoobjekten an VISOR, der Entgegennahme von in VISOR erzeugten Geoobjekten und andere im vorherigen Abschnitt besprochene Operationen gedacht.

Ein auf der untersten Ebene liegendes Objekt kapselt die C-Struktur GeoObject in ein Objective C-Object (CGeoObject). Ein Geoobjekt wird so für die objektorientierte Programmierung leichter handhabbar. CGeoObject besitzt noch weitere Eigenschaften und Methoden, die von höherliegenden Software-Schichten angestoßen werden. Es kann aus einem übergebenen Geoobjekt, aber als Spezialfall auch aus einer vdb-Datei initialisiert werden. Wesentlich sind ebenfalls Methoden, die aus dem Geoobjekt bestimmte Angaben wieder herauslösen können, beispielsweise Koordinaten. Ein Objekt vom Typ CGeoObject kann Auskunft über seinen Verbindungs- und Selektions-Status zu VISOR geben, das integrierte Geoobjekt an VISOR senden, die Anzeige des Geoobjekts in VISOR beenden usw.

Die einzelnen zu einer Gruppe anzuzeigender Daten gehörenden CGeoobjekte werden stets auf dem gleichen Layer angezeigt und in einer Liste (GeoList) zusammengefaßt und koordiniert. Das GeoList-Objekt übernimmt auch das Anlegen der in die Liste zu integrierenden CGeoobjekte, die anschließend sofort an VISOR gesendet werden. Dieses GeoList-Objekt sucht auf Anforderung CGeoobjekte mit ganz bestimmten Eigenschaften heraus und übergibt diese Objekte an den Such-Initiator. Es kann Objekte dieses (und

nur dieses) Typs einfügen und löschen. Beim Anlegen einer GeoList fordert diese vom VISOR gleich den benötigten Layer an und gibt ihn beim Löschen der Liste auch wieder frei. Zuvor wird die Verbindung der in der GeoList enthaltenen CGeoobjekte und darauffolgend die CGeoobjekte selbst gelöscht. Initialisierung einer GeoList ist einerseits durch Übergabe eines Datei-Handles zum Laden von Geoobjekten aus einer vdb-Datei und andererseits durch Übermittlung einer Liste mit Geoobjekten vorgesehen.

Übergeordnet ist ein weiteres Objekt, namens VISORController, das automatisch den Kontakt zu VISOR aufbaut (durch Anlegen eines ServerConnection-Objekts, gegebenenfalls wird VISOR erst gestartet), hält und auch beendet. Es managt weiterhin eine Liste mit GeoList-Objekten (legt diese GeoList-Objekte auch an), die ja jeweils eine Gruppe von auf einem Vektorlayer zu projizierenden Objekten enthält. Der VISORController übernimmt die Rolle des Delegate-Objekts und verteilt (je nach vom VISOR gesendeter Nachricht) die Botschaften an die GeoList-Objekte weiter, die ihrerseits betroffene CGeoobjekte informieren (Nachrichten-Kette).

Wird in VISOR eine Bereichsabfrage gestartet, ist eine andere Vorgehensweise nötig. Es wird ohne den Umweg über eine GeoList direkt ein CGeoobjekt angelegt und eine Nachricht an das Delegate-Objekt des VISORControllers gesendet, das darauf reagieren kann. Danach wird dieses temporäre CGeoobjekt sofort wieder gelöscht.

Die VISOR-Schnittstelle enthält User Interface Objekte zum Einstellen der Eigenschaften von in den GeoList-Objekten gespeicherten CGeoobjekten. Zur Anzeige einer GeoList wird eine TableView eingesetzt. Die dort dargestellten CGeoobjekte können gezielt selektiert und in einem Objekt-Inspektor geändert werden. Manipulierbar sind über diesen Inspektor sämtliche Eigenschaften von Geoobjekten (geometrischer Typ, die verschiedenen Attribute wie Farben und Linienstärken, aber auch die Koordinaten). Auf die CGeoobjekte sind Filteroperationen anwendbar (differierende Farbgebung von Symbolen, die Fundstücke unterschiedlichen Typs repräsentieren). Spezielle Methoden sichern das Einstellen aller Geoobjekt-Attribute auch per Programm ohne direkte Interaktion mit dem Programmbenutzer auf Standardwerte.

Dieser streng hierarchische Aufbau der VISOR-Schnittstelle sichert optimalen Aufwand beim Programmieren und die Unabhängigkeit der Schnittstelle von der Quelle der Daten. Eigenschaftsmanipulationen werden an den Geoobjekten vorgenommen, unabhängig von der Datenquelle, die die Geoobjekte initialisiert hat. Transportiert werden stets Geoobjekte, unabhängig davon, ob Daten von der VISOR-Schnittstelle zu Datenbanken oder anderen Quellen transferiert oder der Datenfluß in die umgekehrte Richtung geschieht.

3.5. Vermittlungsschicht zwischen Datenbank- und VISOR-Schnittstelle

Die archäologischen DB-Programme erzeugen jeweils eine Instanz der VISORController-Klasse, die generell zur Zusammenarbeit mit VISOR eingesetzt wird. Weiter notwendig ist ein Management-Objekt, welches sowohl vom VISORController als Delegate-Objekt verwendet wird und die Vermittlerrolle zur Datenbank-Schnittstelle übernimmt, als auch den DB-Anwendungen zur Übermittlung der zu übertragenden Daten zum VISORController dient (DBVISORInterface). Dieses DBVISORInterface-Objekt ist die

Hauptkomponente der Vermittlerschicht und koordiniert die Arbeit weiterer Objekte.

Das DBVISORInterface kann über einen Menüpunkt im archäologischen Programm oder einen Knopf im Ergebnisfenster einer Datenbank-Abfrage aktiviert werden. Beim ersten Fall wird eine neue DBAbfrage angelegt und der Programmbenutzer muß die in VISOR anzuzeigenden Daten zuerst aus der Arche-Datenbank laden. Danach ist der Ablauf für beide Varianten identisch. Das DBVISORInterface untersucht die Tabellen- und Schlüsselfeldbezeichnungen der Abfrage und lädt gegebenenfalls die Koordinaten nach, da ohne Koordinaten die Weitergabe der Sachverhalte an VISOR nicht sinnvoll ist.

An den VISORController können nur Geoobjekte in einer Liste weitergereicht werden. Die Vermittlungsschicht besitzt zu diesem Zweck ein Objekt (GOConverter, Abkürzung für Geoobjekt-Converter), das vom DBVISORInterface die DBRecordList mit den Koordinaten als Argument erhält. Es hat mehrere Aufgaben. Mit Hilfe der Koordinaten werden Geoobjekte angelegt, diese in eine Liste integriert und an das DBVISORInterface zurückgegeben, das seinerseits diese Liste an den VISORController weiterleitet. Die in 3.4. besprochenen Vorgänge legen eine GeoList mit CGeoobjekten an. Das entspricht einem Datenfluß von den DB-Anwendungen zu VISOR.

Da das DBVISORInterface Delegate-Objekt des VISORControllers ist, wird ihm bei einer Bereichsabfrage in VISOR das Geoobjekt mit den Koordinaten des Bereichs übergeben. Das DBVISORInterface leitet das Geoobjekt an den GOConverter weiter, der die Koordinaten aus dem Geoobjekt herauslöst und zurückgibt. Daraufhin wird vom DBVISORInterface eine neue Datenbank-Abfrage (DBAbfrage) initiiert (über den DBBrowser ist festzulegen, wonach in der Arche-Datenbank gesucht werden soll). Um deren Ergebnisdaten im VISOR-Viewer einzublenden, wird die Vorgehensweise des vorherigen Absatzes angewendet. Eine Bereichsabfrage in VISOR ist auf diese Art und Weise in separate Vorgänge zerlegbar, der Koordinaten-Übergabe von VISOR an die Vermittlungsschicht, die eine Datenbank-Abfrage auslöst und zuletzt die Abfrageergebnisse an den VISOR-Viewer sendet (Abb. 4).

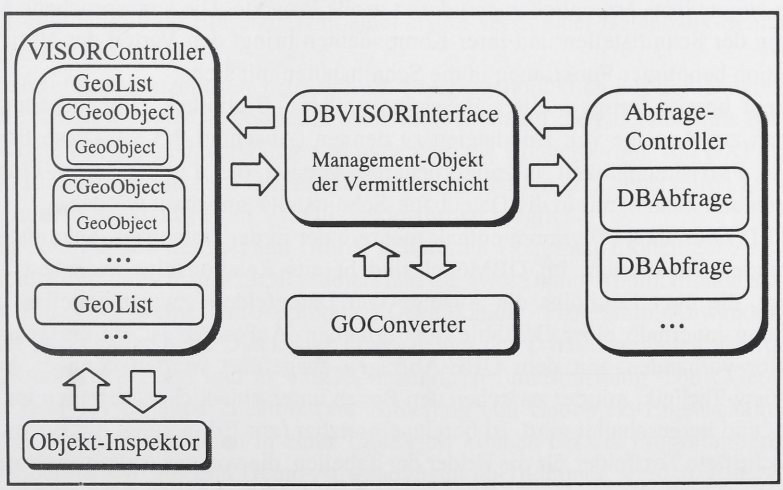

Abb. 4: Vermittlerschicht VISOR- und Datenbank-Schnittstelle

3.6. Realisierungsstand der Schnittstellen

In den vorangegangenen Erläuterungen ist sicher die Komplexität der verschiedenen Komponenten und Zusammenhänge deutlich geworden, angefangen von den Anforderungen an ein Geografisches Informationssystem und die Datenbankzugriffs-Software sowie der Entscheidung für geeignete Systeme wie VISOR und das DBKit mit anschließender Einarbeitung. Die Einarbeitungszeit in diese Software ist hoch. Funktionsumfang und Leistungsfähigkeit sind zu beurteilen und eine Konzeption für die Entwicklung der Datenbank- und VISOR-Schnittstelle für die archäologischen DB-Programme zu erarbeiten und anschließend umzusetzen.

Um auf der grafischen NEXTSTEP-Oberfläche lauffähige Programme mit unterschiedlichstem Anforderungsspektrum entwickeln zu können, bedarf es trotz objektorientierter Programmierung monate- und teilweise auch jahrelanger Erfahrung im Umgang mit NEXTSTEP und der Objective C Programmierung, tieferer Kenntnisse des Betriebssystemaufbaus und der verschiedenen Software-Kits (hauptsächlich Application- und Database Kit) zur Programmierung. Zusätzlich ist die Administration eines SQL-Datenbankservers durchzuführen und Wissen über das relationale Datenmodell und der SQL-Abfragesprache zu erwerben.

Im LfA konzentriert man sich zur Zeit auf die Datenbankfunktionalität. Seit 1992 wurden eine Reihe kleinerer Programme entwickelt, die im Haus eingesetzt werden und auf kleine Datenbanken des SYBASE-Servers zugreifen. Es handelt sich um eine Anwendung zur Verwaltung und Anzeige von Luftbilddaten mit integrierter Anzeige von Luftbildern (Luftbildarchiv), eine Applikation zur Verwaltung personenbezogener Daten der ehrenamtlich Beauftragten für archäologische Denkmalpflege (Beauftragte), ein Programm, das den Bestand des Hauses an Hard- und Software inventarisiert (EDVBestand) sowie eine allgemeine Anwendung zum "Stöbern" in den Tabellen von Datenbanken (DBZugriff). Die vielfältigen dabei gewonnenen Erfahrungen kommen der Entwicklung allgemeiner Schnittstellen zum Datenbankzugriff und zum Geografischen Informationssystem zugute bzw. ermöglichen erst derart große Projekte. Die angesprochene Modularisierung der Schnittstellen und ihrer Komponenten bringt den Vorteil der allmählichen Integration benötigter Funktionen in die Schnittstellen mit sich.

Als ein bereits fertiggestellter Baustein ist die C-Bibliothek "libBildLaden.a" mit Objekten zur Anzeige von Bilddateien zu nennen (Abschnitt 3.3.3.). Diese Bibliothek hat ihre Praxistauglichkeit in einer neueren Version des Luftbildarchiv-Programms bewiesen und kann damit in die Datenbank-Schnittstelle eingebaut werden.

Das EDVBestands-Programm enthält mehrere der in der Datenbank-Schnittstelle einzusetzenden Funktionen. Im DBModel sind bereits Relationships zwischen Tabellen angelegt, die über Equijoins die Anzeige von Datenfeldern zweier Tabellen in 1 : 1-Beziehung innerhalb einer DBTableView erlauben. Außerdem ist ein einfacher Such-Inspektor vorhanden, mit dem QBE-Abfragen ausgeführt werden können. Auch die SwapView-Technik, mit der zwischen den Boxen unterschiedlicher Abfragen im Inspektor hin- und hergeschaltet wird, ist bereits einsetzbar (zur Erinnerung, die Boxen enthalten beschriftete Textfelder für die Felder der Tabellen, die von der Abfrage betroffen sind und in die Filterbedingungen eingetragen werden). In dieser Anwendung sind die Boxen und die in ihnen enthaltenen Textfelder noch statisch im Interface Builder generiert, das

DBAbfrage-Objekt der Datenbank-Schnittstelle wird diese Boxen mit den darin enthaltenen Elementen automatisch aus den Angaben zu Tabellen und Feldern der Abfrage erstellen müssen.

Alle der genannten Programme enthalten Routinen, um Suchoperationen in Datenbanktabellen auszulösen. In der DBZugriffs-Anwendung ist ein Filter-Objekt soweit ausgebaut, daß der Typ des Datenbankfelds berücksichtigt wird und im Falle numerischer Felder Bereichsabfragen unter Einsatz einer Reihe mathematischer Operatoren ausführbar sind. Im EDVBestands-Programm existieren QBE-Abfragemöglichkeiten, die dort aber nur auf Datenbank-Felder mit alphanumerischen Zeichen angewendet werden. In die DBAbfrage-Klasse sind DBQualifier-Objekte mit einer Kombination des Quellcodes aus den Filter-Objekten der beiden angesprochenen Applikationen einzubauen.

Auch ein Beispiel-Objekt für den DBBrowser zur Auswahl der Tabellen und Spalten ist in seinen wesentlichen Eigenschaften bereits fertig und im DBZugriffs-Programm im Einsatz. Bisher kann dieser Browser Felder einer Einzeltabelle (aber auch einer Datenbank-View, die im DBKit wie eine Einzeltabelle behandelt wird) wählen und an ein Weiterverarbeitungs-Objekt übergeben. Die zur Veranschaulichung von Beziehungen zwischen Tabellen essentielle Unterstützung für Relationships fehlt noch.

Das Laden von RTF-Texten aus der Datenbank in ScrollView-Fenster ist ebenfalls im DBZugriffs-Programm enthalten. Man kann diese Applikation schon fast als Prototyp für die Datenbank-Schnittstelle ansehen. Eine der Überlegungen bei der Entwicklung dieser kleinen Anwendung bestand darin, vorbereitende Arbeiten für die DB-Schnittstelle in Angriff zu nehmen.

Die Beispiele geben den Stand der Arbeiten wieder. Viele kleine Strukturelemente, die in die Programmierung der Datenbank-Schnittstelle einfließen, sind bereits vorhanden. Noch fehlt einerseits die Programmierung nicht in den Beispielanwendungen enthaltenen Codes, aber auch die Funktionserweiterung zahlreicher Objekte. Diese sind in ihren Leistungsmerkmalen oft zu starr auf den Einsatzbereich der jeweiligen Applikation ausgerichtet. Mindestens von ebensolcher Bedeutung ist zudem das Zusammenfügen der Softwarebausteine zu einer funktionierenden Einheit. Die geschätzten drei Monate reiner Entwicklungszeit bis zur Einsetzbarkeit einer ersten Version der Datenbank-Schnittstelle sind sicherlich realistisch.

Wie ausgeführt, konzentriert man sich im LfA augenblicklich auf den Datenbankzugriff. Die Arbeit mit VISOR beschränkt sich momentan auf die zu erstellende Kartenbasis[7] und deren Eichung. Zu erwähnen ist außerdem die Konvertierung von DXF- zu vdb-Dateien. Eine Arbeit mit dem GIS ist auch ohne Anbindung an Clients durch Laden und Speichern der Objekt-Layer im VISOR-Vektorformat möglich.

Bisher sind die von der megatel GmbH gelieferten Dokumentationen zur Anbindung von Client-Programmen an VISOR studiert und die zwei Demo-Applikationen analysiert worden. Eine erste kleine Test-Applikation (Applikation - Programm, Anwendung), die aus einer Datenbanktabelle Datensätze (zwei Felder der Tabelle enthalten Koordinaten) liest, Geoobjekte erzeugt und in VISOR anzeigt, ist funktionsfähig. Die CGeoObject-Klasse, Teile des Objekt-Inspektors (zur Änderung von Geoobjekt-Eigenschaften) und die Anzeige von Geoobjekten in einer TableView können bei der Programmierung der VISOR-Schnittstelle aus den Demo-Anwendungen übernommen werden. Eine vorsichtige Abschätzung des Aufwands zur Kodierung von VISOR-Schnittstelle und Vermittlerschicht zur Datenbank-Schnittstelle liegt bei etwa einem Vierteljahr.

Anmerkungen

[1] Bittner/Stock 1994
[2] Bittner/Stock 1994
[3] megatel 1993 – megatel 1994
[4] Weise 1995
[5] Bittner/Stock 1994
[6] NeXT Computer 1993 – NeXT Computer 1993a – NeXT Computer 1993b
[7] Weise 1995

Literaturverzeichnis

Bittner, J./Stock, M. 1994
Probleme beim Aufbau eines digitalen archäologischen Informationssystems - Jahresschrift für mitteldeutsche Vorgeschichte 76, Halle (Saale), S. 271-306

megatel GmbH 1993
Handbuch megatelvisor Version 1.0 für NEXTSTEP - Bremen

megatel GmbH 1994
megatelvisor, Geografisches Informationssystem für MS-Windows, Version 1.5 - Bremen

NeXT Computer, Inc. 1993
NEXTSTEP Object-Oriented Programming and the Objective C Language - NEXTSTEP Developers Library Release 3, New York/Bonn/Madrid

NeXT Computer, Inc. 1993a
NEXTSTEP Development Tools and Techniques - NEXTSTEP Developers Library Release 3, New York/Bonn/Madrid

NeXT Computer, Inc. 1993b
NEXTSTEP General Reference Volume 1 und 2 - NEXTSTEP Developers Library Release 3, New York/Bonn/Madrid

Weise, R. 1995
Erstellen der digitalen Kartengrundlage für ein Geographisches Informationssystem (GIS) - Jahresschrift für mitteldeutsche Vorgeschichte 77, Halle (Saale), S. 339-347

Anschrift

Dipl.-Phys. T. Richter, Landesamt für archäologische Denkmalpflege Sachsen-Anhalt - Landesmuseum für Vorgeschichte, Richard-Wagner-Str. 9-10, D-06114 Halle (Saale)