Geowissenschaften

Institut für Geologie und Paläontologie
der Westfälischen Wilhelms-Universität Münster

Beitrag zur Realisierung portabler Komponenten für Geoinformationssysteme. Ein Konzept zur ereignisgesteuerten und dynamischen Visualisierung und Aufbereitung geowissenschaftlicher Daten.

INAUGURAL-DISSERTATION

zur Erlangung des Doktorgrades
der Naturwissenschaften im Fachbereich Geowissenschaften
der Mathematisch-Naturwissenschaftlichen Fakultät

der Westfälischen Wilhelms-Universität Münster

vorgelegt von

Claus-Peter Rückemann

aus Regensburg

– A. D. 2001 –

Rückemann, Claus-Peter:
Beitrag zur Realisierung portabler Komponenten für Geoinformationssysteme.
Ein Konzept zur ereignisgesteuerten und dynamischen
Visualisierung und Aufbereitung geowissenschaftlicher Daten.

Enthält Literaturverzeichnis und Index.

Überarbeitung `1254` von `dis3.tex`
Erstellt: `1997-09-12 17:54:55`
Letzte Änderung: `2002-01-07 23:51:36`
Formatiert: `2002-01-07 23:57:00`

Hinweis: Diese digitale Version der Dissertation unterscheidet sich nur geringfügig von der eigentlichen gedruckten Version. Die gedruckte Version verwendet u. U. andere Zeichensätze als die digitale Version. Der Text selbst ist unverändert (bis auf diesen Hinweis). Die Anordnung von Überschriften, Abbildungen usw. ist unverändert. Gegebenenfalls werden andere Dateiformate und Komprimierungen verwendet. Der Index und andere Teile wurden sorgfältig an das elektronische Medium angepaßt.

Diese digitale Version ist zum Zeitpunkt der Veröffentlichung über das WWW verfügbar unter

http://wwwmath.uni-muenster.de/cs/u/ruckema

Bitte verwenden Sie zusätzlich diese URL, wenn Sie sich auf diese Arbeit beziehen.

Satz: Claus-Peter Rückemann, Minden, unter Verwendung von TEX und LATEX 2ε unter Linux.
Druck und Bindung: Copy Center CCC GmbH, Coerdestraße, Münster.

Dekan:	Prof. Dr. W. Lange
Erster Gutachter:	Prof. Dr. Th. Kreuser
Zweiter Gutachter:	Prof. Dr. G. Wirtz
Tag der mündlichen Prüfung:	12., 18. und 20. Dezember 2001 (Rigorosum)
Tag der Promotion:	20. Dezember 2001

Meinen lieben Eltern

Vorwort

Diese Dissertation entstand aus der langjährigen Beschäftigung mit verschiedenen Themenstellungen in der Geologie und Geophysik und hat ihren Ursprung in dem engen Bezug zu Problemen der Aufbereitung von Daten in diesen Gebieten.

Möglich wurde die Umsetzung in diese Arbeit aber erst auf Anregung von Herrn Prof. Dr. Thomas Kreuser, dessen Betreuung mich über die Jahre zum Erarbeiten vieler neuer Lösungsaspekte ermutigt und angespornt hat.

Gerade bei der täglichen Forschungsarbeit fallen in vielen Bereichen der Geowissenschaften zunehmend Daten an, die besonders ausgewertet und visualisiert werden müssen.

Das Ziel ist in vielen Fällen die Vermittlung von Einsichten, die auf andere Weise nicht oder nur unvollständig möglich wären.

Viele der erfaßten und bereits vorhandenen Daten oder Proben können nicht losgelöst von ihrem Kontext, im allgemeinen Erhebungslokalität, Fundort oder Entnahmeort, verstanden werden. In vieler Hinsicht haben die meisten neuen Untersuchungen und Nutzungsmöglichkeiten der Daten ihre speziellen Anforderungen an Aufbereitung und Darstellung.

Dies ist insbesondere bei Daten- und Probenmaterial der Fall, das einen engen räumlichen Bezug hat.

Seien es Meßdaten im Rahmen einer geologischen, hydrologischen oder geophysikalischen Kartierung, seien es Proben einer räumlichen Probennahme, kaum eine vorhandene Methode zur Visualisierung und Präsentation löst die teilweise komplexen Aspekte, die in solchen Fällen für eine gemeinsame Vermittlung von reinen Daten und visuellem und multimedialem Kontext wünschenswert wären. In wachsendem Maße sind die Anwendungen und Erkenntnisse, die durch spezielle Visualisierungen und weitergehende interaktive Nutzungsmöglichkeiten geschaffen und erzielt werden, ein wichtiger Anlaß für die Gewinnung neuen Datenmaterials.

Auf der Basis dieser Erfahrungen entstand ein Konzept, das in vielen Anwendungsbereichen die Verwendung der zumeist wenig flexiblen verfügbaren Software durch ereignisgesteuerte und dynamische Komponenten ersetzen kann.

Diese Komponenten lassen sich an die jeweiligen Bedürfnisse individuell anpassen.

In solcher Hinsicht stellt diese Dissertation die Planung, Konzeptionierung und Ausrichtung derartiger Verfahren vor und illustriert eine mögliche Umsetzung an fachbezogenen und einfach gehaltenen Fallstudien, die durch einen Prototyp unterstützt werden, der in den letzten Jahren, eigens für diesen Zweck, zum Nachweis der Sinnhaftigkeit des Konzepts entwickelt worden ist.

Eine große Bedeutung hat dabei die Umsetzung des Konzepts und daher liegt ein besonderes Gewicht auf Aspekten zur Planung und Entwicklung.

Die Prinzipien zur Nutzung der Ereignissteuerung und verschiedener Daten mit intuitiv verständlicher Struktur werden so vorgestellt, daß zum Verständnis und für die Anwendung keine detailierten Kenntnisse der Implementierung des Prototyps, der Anwendung von Skriptsprachen oder weitreichende Programmierkenntnisse erforderlich sind.

Da der zugrundeliegende allgemeine Lösungsweg unabhängig von dem vorliegenden Prototyp ist und dieser inzwischen zu umfangreich ist, um alle Funktionen oder gar Teile der Entwicklung in dieser Dissertation darzustellen, ohne den Leser unnötig von der Kernidee abzulenken, wird dieser zur zusätzlichen Vertiefung mit weiterem Material, speziell angefertigten Daten und weiteren entwickelten Programmkomponenten, im Internet zur Verfügung gestellt.

Nicht zuletzt hoffe ich, durch diese Dissertation einen kleinen Teil der Freude an der Arbeit mit der thematisch vielseitig herausfordernden Aufgabenstellung und daher auch bedingt langjährigen Beschäftigung mit dieser bis dahin offenen Problematik vermitteln zu können und die daraus gewonnenen positiven Erfahrungen einer interessierten Öffentlichkeit nahezubringen.

Kurzdarstellung

— **Kurzdarstellung** —

Beitrag zur Realisierung portabler Komponenten für Geoinformationssysteme.
Ein Konzept zur ereignisgesteuerten und dynamischen Visualisierung und Aufbereitung geowissenschaftlicher Daten.

Claus-Peter Rückemann

2001

Stichworte: Geoinformationssysteme; GIS; Geowissenschaften; ereignisgesteuerte Visualisierungen; dynamische Visualisierungen; Konzept; Ereignissteuerung; dynamische Kartographie; Objektgraphik; Objektdaten; ereignisorientierte Daten; Ereignisdaten; ereignisaktive Objekte; Skriptsprachen; Quellentext-Daten; offene Daten; Internet; portable Komponenten; Entwicklung; modular; Modularisierung; Prototyp; Tcl/Tk; Perl; GNU; Linux

Verfügbarkeit: `http://wwwmath.uni-muenster.de/cs/u/ruckema`
`http://www.unics.uni-hannover.de/cpr/gis`
`http://www.RecommendedLinks.com/gis`

Die vorliegende Dissertation beschäftigt sich mit den Einsatzmöglichkeiten von Skriptsprachen und auf Quellentext basierenden, persistenten Objektdaten für geowissenschaftliche und damit verbundene wissenschaftliche Anwendungen.

Die Untersuchungen konzentrieren sich auf das Ziel, eine Form von Daten zu schaffen, die durch ein neues Konzept und die damit verbundene Erweiterung des Funktionsumfangs für die Erschließung vielfältiger Einsatzbereiche geeignet ist. Die Grundlage für diese konzeptionellen Erweiterungen bildet die Erhöhung des Informationsumfangs verschiedener geowissenschaftlicher und insbesondere räumlicher Daten. Die besondere Ausrichtung liegt auf portabler ereignisgesteuerter und dynamischer Visualisierung und ihrer Eignung für den Einsatz auf lokalen Rechnersystemen und in Netzen wie dem Internet.

Diese Arbeit stellt ein neues und durchgehendes Konzept vor, auf dessen Grundlage sich Verfahren entwickeln lassen, die eine Lösung für zahlreiche Probleme dynamischer Kartographie und multimedialer Präsentation im Bereich der Geowissenschaften sind und als Hilfsmittel für die Auswertung und weitergehende Vermittlung von Meßdaten und Ergebnissen herangezogen werden können.

Eng verbunden mit dem entwickelten Konzept und den eingeführten Quellentext-Daten sind die neuen Begriffe Objektgraphik, ereignisorientierte Daten (Ereignisdaten) und ereignisaktive Objekte.

Es werden die Erfahrungen aus Planung, Entwicklung und Test eines Prototyps vermittelt und an verschiedenen fachbezogenen Fallstudien diskutiert, die in den letzten Jahren zum Nachweis des hier vorgestellten Konzepts entwickelt worden sind.

Aufgrund des umgesetzten Konzepts bietet der Prototyp eine sehr hohe Flexibilität, insbesondere beim Einsatz von Ereignissen für die Nutzung neuer Möglichkeiten zur dynamischen Visualisierung.

Dieser Prototyp und die dafür erzeugten Daten werden im Internet zur Verfügung gestellt.

Alle Entwicklungen und Tests wurden auf frei verfügbaren und offenen Betriebssystemen unter Verwendung freier, offener Entwicklungswerkzeuge durchgeführt und stehen prinzipiell auf jeder modernen Hardware-Plattform zur Verfügung.

Es wird ein Ausblick auf zahlreiche zukünftige Entwicklungen und konkrete Möglichkeiten für Verfahren und Anwendungen gegeben, die auf diesem Konzept beruhen.

Abstract

— Abstract —

Contribution to the Implementation of Portable Components for Geographic Information Systems.
A Concept for Dynamical and Event-driven Visualization and Preparation of Geoscientific Data.

Claus-Peter Rückemann

2001

Keywords: Geographic Information Systems; GIS; geosciences; event-driven visualizations; dynamic visualizations; concept; event control; dynamic cartography; object graphics; object data; event-oriented data; event data; event-active objects; scripting languages; source code data; open data; Internet; portable components; development; modular; unitization; prototype; Tcl/Tk; Perl; GNU; Linux

Availability: `http://wwwmath.uni-muenster.de/cs/u/ruckema`
`http://www.unics.uni-hannover.de/cpr/gis`
`http://www.RecommendedLinks.com/gis`

This thesis concentrates on the employment of scripting languages and source code based persistent object data for geoscientific and related scientific applications.

The object of the analyses is to create a data design that allows to open up multifarious uses by a new concept going along with the extension of the functional range. These conceptional extensions are based on the increase of informational extent of various geoscientific and especially of spatial data. Special bias concentrates onto portable event-driven and dynamic visualization and its applicability for purposes on local host systems and on networks like the Internet.

This work introduces a new and complete concept based on which techniques can be developed for solving numerous problems with dynamic cartography and multimedial presentation regarding geosciences and as an aid for interpretation and advanced mediation of measured data and results.

Closely tied with the concept established here the following terms have been introduced with this work and have been thoroughly used with source-code-based data: object graphics, event-oriented data (event data) and event-active objects.

Experiences are shown resulting from planning, development and testing of a prototype being discussed using a number of problem centered case studies that have been developed over a couple of years to proof the concept presented here.

Based on the concept put into practice the prototype offers a very high flexibility especially regarding to the application of events for using new facilities for dynamic visualization.

This prototype and the data generated along with its development are provided on the Internet.

All of the developments and tests were performed using freely available and open operating systems utilizing free and open development tools and are on principle available for any modern hardware platform.

An outlook shows numerous future developments and concrete potentialities for using techniques and applications based on this concept.

Zusammenfassung

Fragestellung

Die vorliegende Dissertation stellt ein Konzept zur Schaffung von ereignisgesteuerten und dynamischen GIS-Komponenten vor, die Datenmaterial auf Basis von Quellentext verwenden.

Es wird ein durchgängiges Verfahren zur Entwicklung derartiger Komponenten an einem funktionstüchtigen Prototyp vorgestellt. Die zentrale Frage hinsichtlich dynamischer Visualisierung lautet dabei:

Wie können räumliche Daten möglichst flexibel, effizient und portabel mit Daten für Ereignisse verbunden und gleichzeitig die Gegensätzlichkeiten zwischen Datenmaterial und Programm verringert werden?

Konzept und Verfahren

Das Konzept zur Entwicklung anwendungsorientierter Lösungen mittels dynamischer Visualisierung ist in mehrere, miteinander verzahnte Schritte aufgeteilt, die folgendermaßen zusammengefaßt werden können.

1. Zunächst erfolgt die Ermittlung des Problembereichs bezüglich Ereignissteuerung und dynamischer Visualisierung.

2. Die Ermittlung der funktionalen Anforderungen wird im Kontext mit den Erfahrungen bezüglich existierender Anwendungen und Werkzeuge erarbeitet.

3. Die Systemplanung zur Auswahl geeigneter Verfahren berücksichtigt die notwendige Modularisierung und einen adäquaten Schichtaufbau.

4. Alle geeigneten Anteile zu entwickelnder Komponenten werden bezüglich einer Umsetzung mit Mitteln von Skriptsprachen konzipiert. Dafür können bezüglich Aufbau und Modularisierung verschiedene Skriptsprachen eingesetzt werden. Alle grundlegenden Anteile einer Komponente sollen autark sein.

5. An allen Stellen, an denen es angemessen erscheint, werden vorzugsweise Daten auf der Basis einer hochsprachlichen Programmiersprache, beispielsweise als Quellentext (z. B. Objektgraphik), verwendet.

6. Auswahl der Werkzeuge und Verfahren zur Realisierung der Teilanforderungen erfolgen hinsichtlich der Eignung für eine flexible Kombination von Werkzeugen und Verfahren.

7. Die Entwicklung einer Realisierung bzw. eines Prototyps vervollständigt den theoretischen Teil der Planung und zeigt Machbarkeit, Vorteile und zukünftige Erweiterungsmöglichkeiten auf.

Die ersten beiden Schritte definieren das Problem und grenzen es ein, der dritte bis sechste Schritt zielen auf die notwendigen technischen Funktionalitäten und der siebte Schritt gibt eine angepaßte Lösung, auf der zukünftige Entwicklungen aufbauen können.

Zu beachten ist, daß das Konzept auf die Verwendung und Entwicklung geeigneter Verfahren ausgerichtet ist und die Realisierung einer Lösung einschließt.

Diese Lösung kann nur als sinnvolle Lösung angesehen werden, wenn sie offen für zukünftige Weiterentwicklungen ist. Erreicht wird diese Möglichkeit unter anderem durch die ausgewählte Architektur und einen hohen Grad an Modularität, nicht zuletzt aber auch durch den Einsatz frei verfügbarer und offener Entwicklungswerkzeuge.

Dementsprechend ist die Diskussion um Werkzeuge und Verfahren zwingend erforderlich.

Da dieses Verfahren konsequent auf eine Lösung ausgerichtet ist, schließt die vorliegende Dissertation die Planung, den Entwurf und die prototype Implementierung einer vollständigen Komponente mit der geforderten Ereignissteuerung ein.

Der vorliegende Prototyp wurde für verschiedene Einsatzbereiche getestet und wird von der Planung und Entwicklung bis zu möglichen geowissenschaftlichen Anwendungen in Geologie, Hydrologie und Umweltwissenschaften diskutiert.

Ergebnisse

Die wesentlichen und neuen Beiträge, die in dieser Dissertation vorgestellt werden, sind in folgenden Punkten zusammengefaßt:

- Einführung eines Konzepts basierend auf Objektgraphik und die Definition eines neuen Datentyps *Objektgraphik* zur Nutzung erweiterter ereignisorientierter und dynamischer Funktionen.
- Entwicklung eines Verfahrens zur Erstellung dynamischer Visualisierungen unter Einsatz von Objektgraphik.
- Formulierung spezifischer Bedürfnisse bezüglich der Nutzung von Ereignissteuerung und dynamischer Visualisierung für verschiedene Aufgabenbereiche in den Geowissenschaften.
- Gegenüberstellung verschiedener Arten von Daten auf Basis von Objektgraphik bzw. Quellentext.
- Anleitung zum systematischen Einsatz von Anwendungen und Daten auf Basis von Quellentext am Beispiel eines Prototyps.
- Ein frei verfügbarer Prototyp für eine Pilotanwendung zur Nutzung von Objektgraphik.
- Einsatzmöglichkeiten von Objektgraphik im Bereich Klient-Server.
- Nutzung von Ereignisdaten für dynamische Visualisierungen mittels ereignisaktiver Objekte.
- Vorverarbeitung und Aufbereitung von konventionellen Daten zur Nutzung für Ereignisdaten.

Das Konzept, das in dieser Dissertation für die Entwicklung von ereignisgesteuerten und dynamischen Komponenten zur Visualisierung geowissenschaftlicher Daten eingeführt wird, weicht grundlegend von den verbreiteten Verfahren aus dem Umfeld der Entwicklung bisher zur Verfügung stehender räumlicher Informationssysteme ab.

Sein Potential ist die grundlegende Modularisierung, basierend auf Teillösungen mit bester Eignung und nicht eine monolithische Gesamtarchitektur. In dieses Konzept fügt sich das Datenmaterial nahtlos ein.

Damit entspricht das Konzept den speziellen Anforderungen an moderne Informationssysteme, die auf Komponenten aufbauen und eine hohe Modularität und Flexibilität benötigen.

Insbesondere der grundlegende Aufbau erweiterter Daten auf der Basis von Quellentext einer geeigneten Datensprache schafft zahllose Einsatzmöglichkeiten.

Sowohl Anforderungen an die Software, als auch die Spezifikation der Software sind im Kontext formuliert.

In den Entwicklungen zu der vorliegenden Dissertation wurde großer Wert auf die offene Umgebung gelegt sowohl bei den Entwicklungswerkzeugen als auch der Handhabung der eigentlichen Daten, insbesondere hinsichtlich zukünftiger Erweiterungen. Wie die Flexibilität des entwickelten modularen Prototyps gezeigt hat, kann für alle Bereiche der Entwicklung frei verfügbare und offene Software eingesetzt werden.

Die einfache und flexible Erweiterbarkeit derartiger Entwicklungen ist gerade im wissenschaftlichen Bereich von besonderer Bedeutung. Aufgrund des transparenten Systemaufbaus reichen die Eingriffsmöglichkeiten von Kernel-Modulen bis zur interaktiven Oberflächengestaltung einzelner Komponenten.

Ein weiterer überzeugender Grund zur Verwendung einer skriptfähigen Sprache mit einem offenen Interpreterkonzept, wie bei Tcl/Tk, ist der hohe Grad an Kontrolle über dynamische Visualisierungen, der sehr viele neue Möglichkeiten eröffnet.

Die Erfahrungen aus Planungen, Entwicklungen und Tests anhand von fachbezogenen Fallstudien zu dem neu entwickelten Prototyp belegen die besondere Eignung und hohe Flexibilität bezüglich des Einsatzes von Ereignissen für die Nutzung zur dynamischen Visualisierung.

Der praktische Nutzen des vorgestellten Verfahrens wird an durchgängigen Beispielen aus verschiedenen Bereichen der Geowissenschaften demonstriert.

Es werden zukünftige konzeptbezogene Entwicklungen und Erweiterungen diskutiert, die über den Rahmen dieser Dissertation hinausgehen, insbesondere über die Funktionen des entwickelten Prototyps.

Letztendlich stehen mit Fertigstellung dieser Arbeit alle Teile des Prototyps und eine ganze Reihe von angefertigten Daten und Demos, in dem Zustand, in dem sie für die Darstellungen in dieser Dissertation erforderlich waren, im Internet frei zur Verfügung, ergänzt durch eine Vielzahl zusätzlicher Erweiterungen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung und Problemstellung

Situation

Im Mittelpunkt dieser Dissertation steht die Auseinandersetzung mit dem Bedarf geowissenschaftlicher Visualisierung und Auswertung bezüglich ereignisgesteuerter und dynamischer Verfahren.

Neben der Entwicklung eines Konzeptes anhand der Diskussion der zu berücksichtigenden Aspekte, bildet die Entwicklung eines exemplarischen Prototyps den Schwerpunkt dieser Arbeit.

Die Entwicklung dieses zunächst in seinem Bearbeitungsaufwand als sehr hoch einzuschätzenden Prototyps wird dabei als Problemlösungsprozeß zu der offenen Fragestellung nach einer portablen Implementierung an wichtigen Punkten diskutiert.

Auf der Grundlage dieses Konzeptes und den Erfahrungen aus der Entwicklung des Prototyps lassen sich geeignete Verfahren für die ereignisgesteuerte und dynamische Visualisierung mittels Komponenten für geographische Informationssysteme verstehen und entwickeln.

Die zentrale Frage lautet:

Wie können räumliche Daten möglichst flexibel, effizient und portabel mit Daten für Ereignisse verbunden und gleichzeitig die Gegensätzlichkeiten zwischen Datenmaterial und Programm verringert werden?

Geographische Informationssysteme (GIS) haben in Geographie, Geologie und Geophysik seit Anfang der achtziger Jahre des zwanzigsten Jahrhunderts erheblich an Bedeutung gewonnen [Zeh1993].

In diesem Rahmen hat die Vielfalt, Komplexität und Verbreitung von Informationssystemen in den Geowissenschaften zugenommen. Geographische Informationssysteme haben bis heute ihre Anwendungen in fast allen Bereichen des Lebens gefunden, da die Bedeutung räumlicher Informationen nur selten mehr außer acht gelassen werden kann.

Mit fortschreitender Vernetzung steigt die Bedeutung von Informationen rasant. Räumliche Informationen werden immer besser verwaltet, analysiert und präsentiert. Zu diesem Zweck werden vielfältige Komponenten benötigt, die flexible Lösungen bieten.

Neben den informatischen Aspekten geht es aber insbesondere auch um die Anwendungen. Gerade in den Geowissenschaften werden spezielle Entwicklungen und individuelle Anpassungen zunehmend zu einer zwingenden Grundlage für viele Aspekte der Forschung.

Inzwischen haben Informationssysteme weitgefächerte Einsatzfelder in verschiedensten Fachbereichen. Die Geoinformatik verbindet die Geowissenschaften mit der Informatik.

Zum einen haben die Entwicklungen in den letzten Jahren gezeigt, wie übergreifend Problemstellungen aus der Geoinformatik sind und daß in vielen Bereichen Geoinformatiker und geoinformatisches Wissen benötigt werden. Zum anderen ist eine ausgeprägte Praxisorientierung in den Geowissenschaften notwendig, um den steigenden Bedarf an Lösungen in der Zukunft zu decken.

Ein Konzept oder eine Komponente für ein GIS kann heute nicht unabhängig von einer Vielzahl anderer Komponenten entwickelt werden. Ein fundamentaler Gesichtspunkt eines GIS ist die *Integration modularer Teile in heterogene Systeme*.

Moderne Systeme sind aus *Anwendungskomponenten* aufgebaut, welche die Basis für eine *Entwicklungsplattform* darstellen. Aus Sicht der Entwicklung sind die einzelnen Anwendungskomponenten aus Entwicklungskomponenten aufgebaut.

Bei der bestehenden Komplexität moderner GIS-Anwendungen muß das Konzept für die Integration von GIS-Applikationen so *portabel* wie möglich sein, um den höchsten Grad an Verwendbarkeit möglichst *vieler* Funktionen *unterschiedlichster Systeme* zu gewährleisten.

Portabilität kann allgemein folgendermaßen definiert werden:

Eine portable Implementierung kann mit geringerem Aufwand auf ein neues System gebracht werden als eine Neuimplementierung erfordert.

Innerhalb einer komplexen Applikation weisen Komponenten in der Regel unterschiedliche Grade an Portabilität auf (Systemzugriffe, Graphik etc.).

Portabilität derart verschiedener Komponenten, wie sie ein GIS erfordert, ist nicht implizit durch die Verwendung portabler Entwicklungswerkzeuge gegeben, sondern eröffnet sich erst durch Konzeption eines *Abstraktionsniveaus* und die Schaffung geeigneter und beliebig offener *und* geschlossener Entwicklungsebenen.

Zwar haben auch geschlossene Systeme ihre Vorteile, doch die grundlegenden Vorteile offener Systeme sind *Stabilität* und *Erweiterbarkeit* bzw. *Skalierbarkeit* und *Konfigurierbarkeit*.

Bei den heute existierenden Informationssystemen wird häufig nur mit dem Ziel von Erweiterungen entwickelt. Nicht ohne Grund sind viele Funktionen oft nur mit einer *aufgesetzten* Makroprogrammierung erreichbar. Ein grundlegendes Überarbeiten der Basis solcher Systeme ist meist zu aufwendig. Gerade für spezielle Bedürfnisse einer Reihe von modernen Anwendungsgebieten führt dies oft zum Einsatz von sehr *beschränkt anpassungsfähigen* und *wenig skalierbaren* Lösungen.

Die Grundlagen für zahlreiche Auswertungen und Visualisierungen sind der Inhalt und die Strukturierung des Datenmaterials und seine Anbindung an nutzbare Funktionen.

Anlaß zu dieser Arbeit war das Fehlen von Komponenten mit einer engen Anbindung von Ereignissen an Geodaten bzw. Datenobjekte für eine effiziente Nutzung zur dynamischen Visualisierung.

Unter einem Ereignis (engl.: „event“) versteht man ein Geschehen, das in einem gegebenen Kontext eine Bedeutung hat und sich zeitlich und räumlich lokalisieren läßt.

Eine Komponente (engl.: „component“) ist ein ausführbares Softwaremodul mit eigener Identität und wohldefinierten Schnittstellen. Es kann sich dabei beispielsweise um Quellentext (engl.: „source code“), Binärcode, dynamisch ladbare Bibliotheken oder ausführbare Programme in Maschinensprache handeln. Man kann auch von Software sprechen, die unabhängig verwendet werden kann, eine Schnittstelle, ein Typmodell, eine Implementierung besitzt und einen ausführbaren Teil enthält.

Objekte sind konkret vorhandene und agierende Einheiten mit eigener Identität und definierten Grenzen, die Zustand und Verhalten kapseln. Der Zustand wird repräsentiert durch die Attribute und Beziehungen, das Verhalten durch Operationen bzw. Methoden. Jedes Objekt ist ein Exemplar (hier synonym: Instanz) einer Klasse. Das definierte Verhalten gilt für alle Objekte einer Klasse gleichermaßen, ebenso die Struktur ihrer Attribute und ihr Zustand. Die Werte der Attribute sind jedoch individuell für jedes Objekt. Jedes Objekt hat eine eigene, von den Werten seiner Attribute unabhängige, nicht veränderbare Identität.

Fast jede Programmiersprache hat zumindest zu einem Teil objektorientierte Elemente. Beispielsweise ist bereits die Verwendung des Plus-Operators + für ganze Zahlen (engl.: „integer“) *und* Fließkommazahlen (engl.: „floats“) ein „Überladen“ eines Operators (engl.: „operator-overload“), das in fast allen Programmiersprachen seit langem verwendet wird.

Eine einheitliche und exakte Festlegung, was eine objektorientierte Sprache ausmacht, existiert nicht. Man kann strenggenommen derzeit bei keiner existierenden Programmiersprache von vollständig idealer Objektorientierung sprechen, auch wenn einige Sprachen dem schon recht nahe kommen.

Objektorientierte Programmiersprachen stellen jedoch eine Reihe von grundlegenden Konzepten bereit. Die wichtigsten sind:

- Objekte sind abstrakte Einheiten.
- Objekte sind Exemplare einer Klasse.
- Klassen können Eigenschaften vererben.
- Verweise auf Objekte sind dynamisch. Die dynamische Bindung ermöglicht Polymorphismus.

Hinsichtlich der Thematik dieser Dissertation liegt es insofern nahe, zu untersuchen, ob eine Programmiersprache oder ein ähnliches „Datenformat“, objektorientiert oder nicht objektorientiert, für bestimmte Aufgaben bei der Handhabung und Darstellung geowissenschaftlicher Daten herangezogen werden kann.

Neben diesen Eigenschaften müssen aber auch mehr oder weniger konventionell aufgebaute Daten unterstützt werden.

Bis zu 85 Prozent der Kosten beim Einsatz von GIS-Technologie entfallen auf die Daten bzw. die Konvertierung von Daten. Meist liegen diese in proprietären und oft wenig dokumentierten kryptischen Formaten vor [Göb1999]. Weltweit handelt es sich bei den diesbezüglichen Ausgaben jährlich um mehrstellige Milliardenbeträge.

Aus diesem Grund werden zahlreiche Versuche unternommen, übergreifende Lösungen, wie z. B. das Open Geospatial Datastore Interface (OGDI), zu entwickeln [CLGM1998].

Andere Paradigmen gehen von einer Integration kognitiver und geometrischer Ansätze für das Verständnis

der Relationen zwischen Rasterdaten und Vektordaten zur Beschreibung des Raumes aus und bringen zahlreiche Aspekte der GIS damit derzeit in einen überdisziplinären Kontext [Edw1996].

Es bleiben aber Probleme, denn konventionelle GIS sind in der Regel

- wenig flexibel,
- monolithisch,
- proprietär

bezüglich Quellentext, Datenformaten und Einsatzmöglichkeiten.

Dies macht sich bei der Entwicklung und Erweiterung meist noch deutlicher bemerkbar als bei der einfachen Anwendung.

Ziel und Grundidee

Das Ziel dieser Arbeit ist ein Konzept, das die Entwicklung von Verfahren erlaubt, die eine flexible Nutzung von Ereignissen in Verbindung mit geowissenschaftlichen Daten ermöglichen, wie dies beispielsweise für eine weitergehende dynamische Visualisierung benötigt wird.

Eine solche Umsetzung wird anhand einer möglichst effizient entwickelten und effizient einsetzbaren Prototypisierung eines portabel erweiterbaren Modulkonzepts demonstriert. Derartige Module können mit einer einfach zu bedienenden Schnittstelle für den Endanwender und einer professionellen Entwicklungsumgebung in einem breitem Spektrum an Anwendungsgebieten eingesetzt werden. Dies vermittelt einen beispielhaften Ausblick auf mögliche Schnittstellen zu einer Vielzahl von GIS-Modulen und Komponenten aber auch zu unterstützender Software. In vielen Spezialfällen kann eine derartige Lösung mit geringstem möglichen Aufwand eingesetzt werden.

Der hier entwickelte Prototyp ist zudem durch seine in diesem Zusammenhang außergewöhnliche Datensprache spezialisiert auf Ereignissteuerung und möglichst flexiblen Umgang mit diesbezüglichen Daten und Anforderungen. Er sollte daher als beispielhafte Anwendung des Verfahrens betrachtet werden und nicht als eigenständige Applikation.

Aufgrund der Notwendigkeit einer Prototypisierung haben einige Bereiche den Charakter einer Machbarkeitsstudie. Struktur der Entwicklungen, Werkzeuge und zeitliche Planung sind daher ebenso von Bedeutung.

Die zu entwickelnden GIS-Komponenten müssen neben den Möglichkeiten zu einer effizienten und raschen Entwicklung folgende Eigenschaften aufweisen:

- Ermöglichung der Visualisierung und Verarbeitung geologischer, geophysikalischer und anderer flächenbezogener und punktbezogener Daten.
- Ereignisgestützte dynamische Visualisierung als essentieller Bestandteil.
- Trennung von Entwickler- und Anwenderumgebung auf einfache Weise.
- Verwendbarkeit professioneller und flexibler Bearbeitungs- (engl.: „processing“), Datenbank- und Visualisierungskomponenten.
- Einfache Handhabung durch den Endanwender unter Verwendung von elementaren Werkzeugen.

Außer den beschriebenen Zielen sind einige Nebenbedingungen von nicht zu vernachlässigender Wichtigkeit, da sie vor allem über die Entwicklungseffizienz entscheiden.

Zu der Entwicklung dürfen für alle essentiellen Aufgaben ausschließlich nicht proprietäre, *offene* und *frei* verfügbare Werkzeuge und Komponenten verwendet werden, um die *langfristige Unabhängigkeit* des Projektes zu gewährleisten [Aut1999c].

Besondere Herausforderungen sind die Planung der Komponenten und der Einsatz der Entwicklungswerkzeuge, die gewährleisten müssen, daß auch eine *einzelne* Person in einigen Jahren verschiedenste Aspekte einer derartigen Applikation realisieren kann.

In der Komplexität der Komponenten ist ein derartiges System in seiner Gesamtheit weit umfangreicher, als selbst die Entwicklung anspruchsvoller Animationssysteme [Röh1994], da jede derartige Komponente ein optionaler Teil der Entwicklung sein kann. Um diese Komplexität auf einen angemessenen Umfang einzuschränken, sind einige Basisfunktionen definiert.

Die Basisfunktionen müssen umfassen:

- Graphische Benutzerschnittstelle.
- Schaffung modularer und portabler Ebenen der Teilkomponenten.
- Möglichkeiten zur Verwendung mehrerer höherer Programmiersprachen.
- Integration einer Systemumgebung.
- Datenbank für Rasterdaten, Vektordaten und Ortsdaten (engl.: „site data“).
- Schichtungsfunktionen (engl.: „layering“).

- Bildbearbeitungsfunktionen (engl.: „image processing functions“).
- modulare Funktionen zur Datenmanipulation.
- Import- und Exportfunktionen.
- Möglichkeiten zur Parallelisierung.
- Möglichkeiten zur Internationalisierung.
- Generierung von Quellentext.
- Integration einzelner Module in bestehende Systeme.

Diese Arbeit entwickelt ein *Konzept* zu den angesprochenen Forderungen und demonstriert dies anhand der *Entwicklung eines geeigneten Prototyps* für eine Komponente. Diese Komponente weist Funktionen auf, die über die angesprochenen Basisfunktionen hinausgehen.

Konventionen

Einige notations- und gestaltungstechnische Konventionen der gedruckten Version dieser Dissertation sollen helfen, das Lesen und das Verständnis zu erleichtern.

Aufgrund des teilweise interdisziplinären Charakters der Thematik muß darauf hingewiesen werden, daß einige Begriffe je nach Zusammenhang traditionell verschiedene Aspekte aufweisen. Aufgrund ähnlicher Basisbedeutung sind sie aber nicht durch verschiedene Bezeichnungen unterschieden. Solche Begriffe sind beispielsweise Objekte (z. B. bezüglich Objektorientierung, Canvasobjekte, Graphikobjekte), Ereignisse (z. B. bezüglich X Window System, Tcl) oder Quellentext (z. B. bezüglich Anwendungen mit offenem Quellentext, Quellentext-basierte Daten). Die jeweilige Bedeutung geht aus dem Zusammenhang hervor.

- Elemente aus Programmiersprachen, Aufrufe von Programmen, Teile von Programm-Quellentext, Angaben von Netzadressen und ähnliches sind durch eine nichtproportionale Schrift ausgezeichnet. Beispiel: `#!/usr/bin/perl`.
- Beispiele von Quellentext, `HTML` und ähnlichem sind funktionsfähig hinsichtlich der Aspekte, die sie verdeutlichen sollen und halten sich an den Stand der Technik. Für gewisse Umgebungen sind jedoch gegebenenfalls spezielle Anpassungen notwendig. Für zukünftige Entwicklungen sind Aktualisierungen mit ebenso großer Wahrscheinlichkeit erforderlich. Für strenge `HTML` bzw. XML Konformität können derzeit z. B. die frei verfügbaren Werkzeuge Validator[1] und `tidy`[2] verwendet werden. Für Syntaxprüfung von Tcl/Tk Programmen kann z. B. `procheck` aus der Entwicklungsumgebung TclPro[3] verwendet werden.
- Die in Abbildungen verwendeten Graustufen sind aufgrund der vielfältigen dargestellten Abhängigkeiten nur für die jeweilige Abbildung von Bedeutung. Gleiche Graustufen in verschiedenen Abbildungen deuten nur auf vergleichbare Zusammenhänge hin, wenn dies möglich war. Wenn der Zusammenhang nicht unmittelbar deutlich ist, wird im Einzelfall explizit darauf hingewiesen.
- Internet Adressen werden, außer in der Literaturliste, in Fußnoten angegeben, wenn sie als zusätzliche Information dienen. Dies fördert den Lesefluß und hilft Mißverständnisse mit Trennungen in den Adressen zu vermeiden. Andere Fußnoten werden nicht verwendet. Zu den Adressen sind, soweit vorhanden, Datum der Erstveröffentlichung (V:), der letzten Änderung (Ä:) und der Verfügbarkeit bzw. des letzten Zugriffs (Z:) angegeben. Ist ein Dokument vor Abschluß dieser Arbeit nicht mehr unter der angegebenen Adresse verfügbar (n. v.) oder existieren keine Hinweise auf Änderungen (k. A.), so ist dies angegeben.
- Literaturreferenzen erfolgen im Text in eckigen Klammern unter Angabe eines eindeutigen Kürzels, das aus den Anfangsbuchstaben der Namen der Autoren und der Jahreszahl des Erscheinungsdatums gebildet wird. Die zugehörigen Quellenangaben finden sich nach diesen Kürzeln geordnet im Literaturverzeichnis. Wenn möglich und notwendig wurden die zur Verwendungszeit gültigen Netzadressen mit Zugriffsdatum angeführt.
- Die Bedeutung von Akronymen, Abkürzungen und dergleichen wird im Glossar aufgeführt.

[1] `http://validator.w3.org` ..... [V: k. A.] [Ä: k. A.] [Z: 02.03.2001]
[2] `http://www.w3.org/People/Raggett/tidy` ..... [V: k. A.] [Ä: k. A.] [Z: 02.03.2001]
[3] `http://sourceforge.net/projects/tclpro` ..... [V: 2000] [Ä: k. A.] [Z: 25.01.2001]

- Markennamen, Handelsnamen, Warenzeichen und dergleichen sind im Anschluß an das Glossar angegeben.
- Viele Begriffe sind mit einem Eintrag im Index vertreten. Wichtigere Einträge sind **fett** und Definitionen *kursiv* hervorgehoben.
- Für englische Begriffe wurde in allen Fällen, für die dies angemessen erschien, der deutsche Begriff verwendet. Die englischen Begriffe sind zur Information aufgeführt und als solche kenntlich gemacht. Beispiel: Ereignis-Schleife (engl.: „event loop“).

1. Informationssysteme

1.1. Grundlegende Definitionen

Definitionen übergeordneter Begriffe sind im allgemeinen ein Problem und insbesondere bei komplexen Systemen, zu denen auch GIS oder LIS zählen. Für diese Dissertation sind folgende Definitionen nützlich:

1. **GIS:** Eine bestimmte Art von Informationssystem, das sich mit geographischen bzw. räumlichen Daten befaßt.

2. **System:** Ein System ist eine Gruppe verknüpfter Entitäten und Aktionen, die zu einem gemeinsamen Zweck interagieren.

3. **Informationssystem:** Ein Informationssystem ist eine Gruppe von Prozessen, die, angewandt auf weniger verknüpfte Daten, Informationen liefern, die für Entscheidungsprozesse (engl.: „decision-making“) nützlich sind. Ein vollständiges Informationssystem umfaßt Funktionen zur Erfüllung dieser Aufgaben, die z. B. Beobachtung, Messung, Beschreibung, Erklärung, Vorhersage und Entscheidungshilfen betreffen.

4. **Geographische Daten:** Geographische Daten beinhalten Daten, die einen räumlichen Bezug haben. Darunter fallen vier integrierte Komponenten: Lokationen, Attribute, räumliche Beziehungen, Zeit. Ein Informationssystem nutzt in der Regel sowohl räumliche als auch nichträumliche Daten. Häufig sind räumliche Daten in GIS geographisch referenzierbar. In vielen Fällen sind damit Operationen für Aufgaben im Bereich der räumlichen Analyse verbunden.

GIS werden verwendet, um beschreibende Daten und räumliche Daten (engl.: „spatial data“) zu verwalten, zu visualisieren und zu analysieren. Bei den räumlichen Daten handelt es sich um Objekte mit Dimension oder physikalischer Ausdehnung. Haben diese Daten eine Relation zur Oberfläche der Erde, spricht man von geographischen räumlichen Daten (engl.: „geographical spatial data“). Geographische räumliche Daten haben Lokation, positionsunabhängige Attribute und räumliche Beziehungen (Topologie).

Karten werden verwendet, um räumliche Daten oder Ergebnisse darzustellen oder zu analysieren und um verschiedene dieser Eigenschaften zu kombinieren.

Reports werden benutzt, um Tabellen, Listen etc. auf Abfragen hin auszugeben. In den meisten Fällen sind auch diese Daten räumlich verknüpft. Viele dieser Daten bilden die Grundlage für eine Darstellung durch Karten, aber dies ist nicht der einzige Verwendungszweck.

Eine sehr verbreitete Aufgabe von GIS ist die Entscheidungsunterstützung im Bereich Bodennutzung, Transport, Probennahme und anderer räumliche verteilter Entitäten. Die Verbindung zwischen diesen Elementen des Systems ist der räumliche Bezug bzw. die Geographie.

1. **LIS:** Ein Land Information System (LIS) bezeichnet ein räumliches Informationssystem, das seinen Schwerpunkt im Bereich von Katasterdaten, wie z. B. Parzellen, Stromleitungen, Straßen etc., hat.

2. **Unterschied GIS/LIS:** Die Übergänge sind fließend. Im allgemeinen aber hat ein GIS Schwerpunkte in Richtung Datenanalyse und Modellierung und ein LIS in Richtung Datenbankoperationen und Informationsdienste.

Die folgende Auswahl aus der Fülle an alternativen Namen zeigt die Vielfalt an Einsatzgebieten und Orientierungen:

- Multipurpose Geographical Data System
- Multipurpose Input Land Use System
- Computerised Geographical Information System
- System for handling natural resources inventory data
- IBIS - Image-Based Information System

- Land Resource Information System
- Spatial Data Management and Comprehensive Analysis System
- Planning Information System
- Resource Information System
- Natural Resource Management Information System
- Spatial Data Handling System
- Geographically Referenced Information System
- Geo-Information System
- Spatial Information System
- Environment Information System
- AGIS - Automated Geographical Information System
- Multipurpose Cadastre
- Land Information System
- AM/FM - Automated Mapping and Facilities Management
- KBGIS - Knowledge Based Geographical Information System

Einige grundlegende Begriffe zum Umfeld von GIS finden sich z. B. in den zahlreichen Glossaren zum Thema GIS im Internet [Bla1995].

Anwendungen von GIS finden sich heute außer in der Forschung und Umweltplanung in vielen Bereichen, wie Behörden, Militär und z. B. in der Touristik. Nicht allein aus diesem Grund werden flexible und anpassungsfähige, insbesondere aber anpassbare Lösungen immer wichtiger.

1.2. GIS

1.2.1. Aufgaben

Die Vorteile, die durch konventionelle Informationssysteme entstehen, treffen auch auf GIS zu. Spezielle GIS werden im wesentlichen aber für folgende Aufgaben benötigt.

1. Reduzierung der Datenredundanz.
2. Integration von Datenmaterial verschiedener Quellen.
3. Erhaltung der Datenkonsistenz.
4. Vereinfachung der Aktualisierung von Daten.
5. Flexibilisierung bei Datenspeicherung und Datenzugriff.

1.2.2. Ursprung

GIS sind das Produkt einer langen Entwicklung konvergierender technologischer Entwicklungen und traditioneller Disziplinen. GIS ist aufgrund des Potentials für eine Fülle an Disziplinen, die mit räumlichen Daten arbeiten, lange Zeit als „weiterführende Technik“ (engl.: „enabling technology“) bezeichnet worden.

Diese Disziplinen haben durch ihr Bemühen dazu beigetragen, Techniken zu entwickeln, die in ihrer Integration derartige Systeme mit Funktionalität füllen. Zu den Schwerpunkten gehören Integration, Modelling, Analyse.

GIS nimmt gelegentlich für sich in Anspruch, die Wissenschaft räumlicher Information zu sein.

In Abbildung 1.1 (Seite 9) sind die Disziplinen zusammengestellt, die maßgeblich zur Entwicklung von GIS-Technologien beigetragen haben.

Die Geographie hat eine lange Erfahrung im Umgang mit räumlichen Daten und liefert Methoden und Theorien zur Analyse räumlicher Daten und den damit verbundenen Forschungsfeldern.

Kartographie beschäftigt sich zu einem wesentlichen Teil mit der Darstellung räumlicher Informationen. Die Kartographie ist eine der wichtigsten Datenquellen für GIS-Daten und Referenz für weitreichendes Kartendesign.

Fernerkundung stellt eine wichtige Quelle geographischer Daten da, die als digitale Luft- oder Raumaufnahmen gemacht werden. Die Fernerkundung liefert Techniken, um Datenerfassung und Datenbearbeitung (engl.: „processing“) kontinuierlich rund um die Welt zu einem niedrigen Preis zu ermöglichen. In der Kombination mit ergänzenden Daten in einem GIS kann in Echtzeit räumliche Information bereitgestellt werden.

Vermessung (engl.: „surveying“) und Photogrammetrie liefern hochqualitative Daten über die Position von Katasterobjekten (Parzellierung, Gebäude etc.) sowie über Topographie.

Die Informatik trägt aus verschiedenen Arbeitsgebieten zu GIS bei:

Computerunterstütztes Design (CAD) liefert Software, Techniken für Dateneingabe, Darstellung und Visualisierung sowie Repräsentation.

Fortschritte in der Computergraphik liefern Hardware und Software zur Handhabung und Darstellung graphischer Objekte.

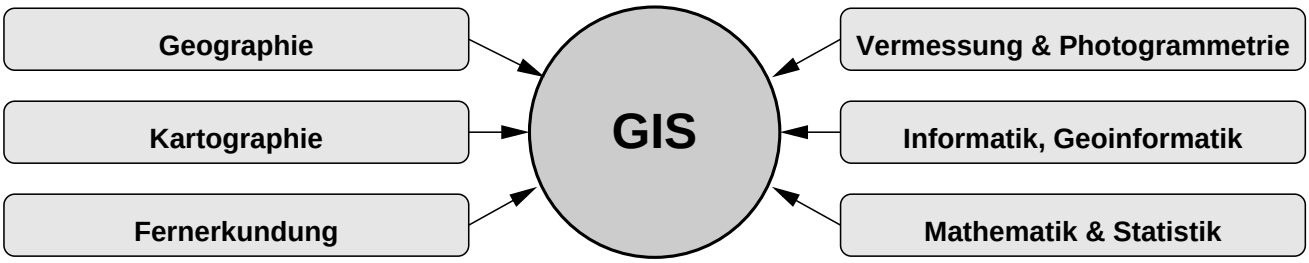

Abb. 1.1: Beitrag zur Entwicklung von GIS-Technologien

Database Management Systems (DBMS) stellen Methoden zur Repräsentation von Daten in digitaler Form bereit und fördern Systemdesign und den Umgang mit großen Datenmengen, insbesondere Zugriff und Handhabung.

Methoden künstlicher Intelligenz (KI, AI) nutzen Computer, um Entscheidungen aufgrund von Dateneigenschaften zu treffen und damit menschliche Intelligenz und Entscheidungsfindung zu emulieren.

In der Mathematik und Statistik beschäftigen sich verschiedene Gebiete insbesondere im Bereich Geometrie und Graphentheorie mit Themen, die in engem Zusammenhang zu dem Design von GIS und der Analyse räumlicher Daten stehen. Insbesondere die Statistik hilft dem Verständnis von Fehler und Unsicherheit in GIS-Daten.

1.2.3. GIS-Komponenten

Die im allgemeinen notwendige Hardware für den Betrieb eines GIS umfaßt Computer bzw. Zentrale Recheneinheiten (engl.: „Central Processing Unit“, CPU), die mit Massenspeichern (Festplatten, CDROM, DVD, Bandlaufwerke etc.) und Peripherie (Digitizer, Scanner, Drucker, Plotter) sowie einer Display-Einheit (Visual Display Unit, VDU) verbunden ist. Hinzu kommen zahlreiche optionale Einrichtungen, wie Kommunikationsgeräte, Netzschnittstellen und Satellitenempfänger.

Die Software eines GIS basiert in der Regel auf vier grundlegenden Modulen:

- Dateneingabe und Verifikation.
- Datenspeicherung und Datenmanagement.
- Datentransformation und Datenmanipulation.
- Datenausgabe und Präsentation.

Grundlegende *Anbindungen von Funktionen*, wie mittels Ereignissen oder der Nutzung von flexiblem Skripting *fehlen jedoch in den meisten Fällen* und sind, wenn überhaupt, dann nur rudimentär oder systemspezifisch ausgeprägt. *Gerade diese Anbindungen sind aber für eine dynamische Visualisierung von besonderer Bedeutung*. Hier wird der Begriff Anbindung verwendet, um den dynamischen Charakter zu betonen und um sich von Begriffen wie Verknüpfungen oder Zuweisungen abzuheben, die häufig in statischem Zusammenhang verwendet werden.

Abbildung 1.2 gibt einen Überblick über die wichtigsten Module eines GIS, die wie Subsysteme zusammenarbeiten.

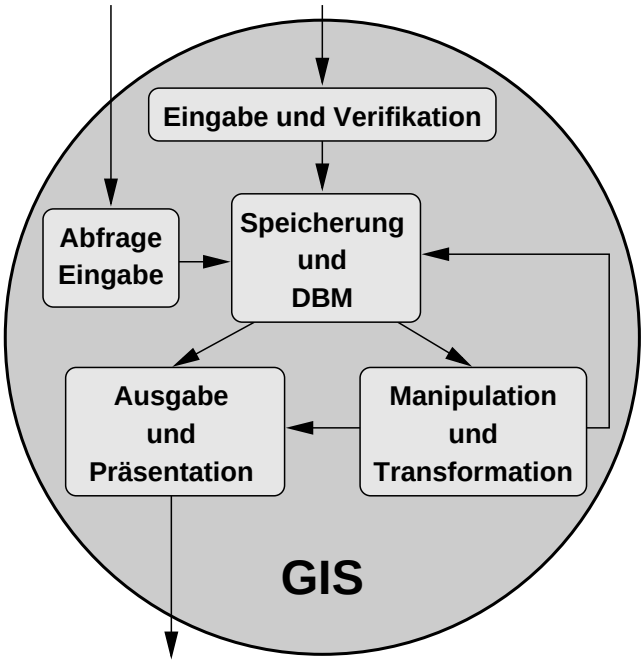

Abb. 1.2: Softwaremodule eines GIS

1.2.4. Organisation

Hardware und Software eines GIS bestimmen die Art und Weise in der räumliche Informationen von und mit einem System bearbeitet werden können, sie garantieren aber keine effektive oder gar effiziente Nutzung. Um dieses zu erreichen, muß das System in einen geeigneten organisatorischen Kontext integriert werden (Abbildung 1.3).

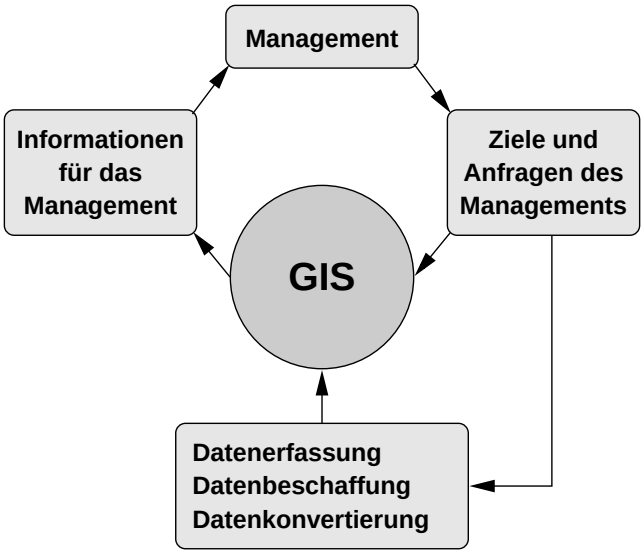

Abb. 1.3: Organisatorischer Kontext eines GIS

1.3. Datenformate

1.3.1. Herkunft

GIS-Daten stammen in der Regel entweder aus computergestützten Kartierungen bzw. Aufzeichnungen (CAD-CAM) oder der Fernerkundung (engl.: „remote sensing“, RS).

CAD-CAM Daten sind Vektordaten, (engl.: „vector data“, engl.: „arc-node data“) während es sich bei Satellitendaten und Luftbilddaten aus der Fernerkundung, bedingt durch die Natur dieses Datenmaterials, um zellenartig strukturierte Rasterdaten (engl.: „raster data“, engl.: „cell data“) handelt.

Vektordaten sind Daten, die Informationen als geometrische Punkte, Linien, Flächen und dergleichen speichern und gegebenenfalls topologisch verknüpfen.

Rasterdaten sind Felder von Bildpunkten. Datenquellen für Rasterdaten sind typischerweise Satellitenbilder, Luftbilder oder digitalisierte (bzw. „gescannte“) Karten. Für die meisten echten Analysen geowissenschaftlicher Daten muß eine Georeferenzierung vorgenommen werden. Das bedeutet, zu jedem Bildpunkt sind die Koordinaten der zugehörigen Lokation auf der Erde bekannt.

Die Analyse dieser beiden Datentypen basiert auf grundlegend unterschiedlichen Algorithmen.

Während Vektordaten meist für Karten und Zeichnungen eingesetzt werden, finden digitale Rasterdaten insbesondere bei der Analyse des Datenmaterials Verwendung.

Ein vollständiges GIS ist letztendlich eine Kombination aus einer Datenbank mit spezifischen Eigenschaften und speziellen Methoden. Aufgrund der oft sehr großen Datenmengen waren solche Systeme bis vor wenigen Jahren auf professionelle Systeme wie Unix Workstations und Großrechner (engl.: „mainframe“, Hauptgerät) beschränkt.

In der Regel ist der Kostenfaktor für das digitale Kartenmaterial der bedeutendste finanzielle Teil beim Einsatz eines Geoinformationssystems. Die Kosten für die Hardware und Software fallen dabei meist gar nicht mehr ins Gewicht.

1.3.2. Politische und kulturelle Unterschiede

Seit den neunziger Jahren des zwanzigsten Jahrhunderts werden verstärkt Spezialformate entwickelt, die dringend benötige spezielle Eigenschaften haben, aber auch eine weitere Diversifikation der geographischen Datenformate zur Folge haben. Ein Beispiel ist das General Data Format (GDF), das vom CEN TC278 für europäische digitale Straßenkarten [OII1998] zur erhöhten Standardisierung (EUROGI[1]) [EUR1996] entwickelt wird.

Der wesentliche Anlaß für die Entwicklung dieses Formates und der einhergehenden Datenbanken sind die grundlegenden politischen, kulturellen und ökonomischen Unterschiede zwischen Europa und den Vereinigten Staaten von Amerika [SGX1994].

Aufgrund der individuellen Gegebenheiten im Großraum Europa ist die Koordination bei der Entwicklung von Kartendatenbanken von zentraler Wichtigkeit. Aufgrund der langjährigen Entwicklungen und vorangetrieben von der Europäischen Union sind in diesen Bereichen Standards für Kartendatenbanken in Europa weitaus fortgeschrittener, als beispielsweise in den Vereinigten Staaten und anderen Ländern.

Alle diese Bestrebungen richten sich auf die Bereitstellung wissenschaftlicher Informationen, dokumentarischer Daten und Geodaten sowie die kartographische Bearbeitung räumlicher Daten. Sie dienen zu Zwecken der Forschungssteuerung (engl.: „controlling“), -planung und -koordinierung, als auch zur Verbesserung interner und externer Informationsabläufe und bilden damit eine Basis wissenschaftlicher Arbeit in vielen Bereichen.

[1] `http://www.eurogi.org` ..... [V: k. A.] [Ä: k. A.] [Z: 30.01.2001]

1.3.3. Datenstrukturen

Räumliche Lokationen und damit auch geographische Positionen eines räumlichen Objektes können durch zwei- bis vierdimensionale Koordinaten in einem geographischen Referenzsystem dargestellt werden.

Attribute sind beschreibende Informationen über bestimmte räumliche Objekte. Sie enthalten häufig keine unmittelbare Information über die räumliche Lokation, können aber mit den räumlichen Objekten verknüpft werden. Aus diesem Grund werden Attribute oft auch als nichträumliche Information (engl.: „nonspatial information“, engl.: „aspatial information“) bezeichnet.

Die folgenden Abschnitte stellen konventionelle 2D-Daten (Punktdaten, Liniendaten, Polygondaten und verbundene Attribute) anhand eines einfachen Beispiels dar. Derartige Daten werden häufig nicht in direkt lesbarer Form von verschiedenen GIS verwendet, der Aufbau und Inhalt sind aber vergleichbar.

1.3.4. Punktdaten

Im Format für Punktdaten erhält jeder Punkt eine eigene Zeile. Der erste Eintrag der Zeile ist eine Identifikationsnummer (ID-Nummer), der zweite die x-Koordinate, der dritte die zugehörige y-Koordinate.

Ein Datenformat für Punktdaten hat den folgenden Aufbau (Abbildung 1.4).

```
1 200567.345 456543.234
2 257658.458 462789.435
3 261340.456 470600.370
```

Abb. 1.4: Datenbeispiel: Konventionelle Punktdaten

1.3.5. Liniendaten

Für jeden Bogenzug in einem Datensatz wird eine ID-Nummer in einer eigenen Zeile vergeben und anschließend die x-y Koordinatenpaare in eigenen Zeilen. Der Bogenzug wird durch das Auftreten des Kennzeichners `END` beendet. Das Ende eines Datensatzes kann durch ein zusätzliches `END` angegeben sein.

Ein Datenformat für Liniendaten hat den folgenden Aufbau (Abbildung 1.5).

```
1
456234.612 856234.611
456981.347 867923.456
457840.608 866454.333
489333.444 890544.677
END
2
645700.957 700623.787
653000.456 710756.854
632977.234 715000.746
END
```

Abb. 1.5: Datenbeispiel: Konventionelle Liniendaten

1.3.6. Polygondaten

Polygondaten sind vergleichbar mit den Liniendaten aufgebaut, mit einem kleinen Unterschied. Jeder Polygonzug in Polygondaten hat eine ID-Nummer und ein x-y-Koordinatenpaar für den Ausgangspunkt („seed value“) in einer Zeile. Alle weiteren Koordinatenpaare in den folgenden Zeilen definieren den Bogenzug bis zum letzten Punkt.

Wenn mehrere Polygone den Bogenzug definieren, ist der letzte Punkt eines Polygons der erste Punkt des nächsten Polygons.

Polygonzüge, die Inseln beschreiben, werden von einigen Informationssystemen doppelt definiert: Zum einen durch ihre eigene Definition und zum anderen über eine ID-Nummer `-99999`.

Auf diese Weise können bestimmte Strukturen von diesen Systemen gesondert behandelt werden.

Für einfache Fälle kann die Attributinformation über die ID-Nummer ausreichend sein. Für komplexere Fälle muß eine getrennte Datei verwendet werden, die Tabellen mit weiteren Attributen zu den ID-Nummern bereithält (engl.: „lookup tables“).

Der Bezug zu den Koordinaten in den zugehörigen Datensätzen wird durch die ID-Nummer hergestellt.

Ein Beispiel einer solchen Datei mit ID-Nummer und einigen weiteren Attributen kann folgendermaßen aussehen (Abbildung 1.6):

```
1:DEUTSCHLAND:14900:89000000
2:USA:15000:248000000
3:UK:12000:55000000
```

Abb. 1.6: Datenbeispiel: Konventionelle Attributdaten

1.4. OpenGIS

1.4.1. Datenbarriere

Die wichtigsten Aspekte, an denen gearbeitet werden muß, um die räumliche Datenbarriere (engl.: „geospatial data barrier problem“) zu überwinden, sind Datentranslation und Datenstandardisierung.

OGDI enthält daher u. a. Komponenten, um verschiedene Datenformate in eine einheitliche transiente Datenstruktur zu überführen, um Koordinatensysteme, kartographische Projektionen und plattformabhängige Datenrepräsentationen anzupassen, Geometriedaten und Attributdaten zu extrahieren.

Das transiente Datenmodell kann mit zwei grundlegenden Datentypen umgehen, Vektordaten und Rasterdaten. Die Vektordaten werden in 4 Untertypen von Merkmalen (line, area, point, text) und 3 Untertypen von Grundstrukturen (engl.: „primitives“) eingeteilt.

Bei der Datentranslation spielen folgende Aspekte eine wichtige Rolle:

- Verschiedene räumliche Datenformate haben unterschiedliche Vollständigkeitsgrade. Konvertierungen führen daher in den meisten Fällen zu Verlust an signifikanter Information.
- Datentransformationen werden in der Regel lokal („offline“) durchgeführt. Daher müssen lokal auch große Speichermedien belegt werden.
- Jeder Applikationshersteller muß Konverter für die geläufigsten Datenformate entwickeln und auf aktuellem Stand halten, was zu einem erheblichen Aufwand führt.
- Aufgrund von Einschränkungen bestimmter GIS müssen spezielle zugeschnittene Versionen der räumlichen Daten für verschiedene Softwareprodukte erzeugt werden.
- Die Verwendung von Konvertierungsprogrammen fördert die Verwendung proprietärer Datenformate für Datensätze.
- Schwierigkeiten und Kosten der Datenintegration führen zur Verbreitung von unvollständigen de-facto Industriestandards, je nach den Produkten der Anbieter mit den größten Marktanteilen.

Das OpenGIS Consortium[2] ist ein Forum für die GIS-Gemeinschaft, um zur Interoperabilität von Geodaten und Verfahren beizutragen [BM1998].

1.4.2. OGDI Schnittstelle

Die OGDI Schnittstelle ist durch eine Tcl/Tk Programmierschnittstelle (Tcl/Tk API) und eine ANSI C Schnittstelle unter Unix/Linux erreichbar. Durch die Funktionalitäten werden u. a. möglich:

- Abbildungen auf eine einheitliche transiente Datenstruktur
- Anpassungen von Koordinatensystemen und Projektionen
- Transformationen plattformabhängiger bzw. betriebssystemunabhängiger Repräsentationen
- Zurückholen (engl.: „retrieval“) geometrischer Daten und Attributdaten

Der Zugriff erfolgt direkt lokal oder über das Netz („remote“) für externe Datensätze. Dazu wurde ein Protokoll (GLTP, `gltp`) entwickelt, das auf einem entfernten Computer eine C Schnittstelle (API) simuliert. Das Protokoll wird durch einen Daemon (GLTPD) bereitgestellt. Das `gltp` ist zustandsorientiert (engl.: „statefull“) im Gegensatz z. B. zu `http` und ermöglicht daher über *einen* Serverprozeß mit einer Art Gedächtnis die Koordination aufeinander bezogener Anfragen mehrerer Klienten.

Derzeit ist die Implementierung von OGDI nach Rücksprache mit den Entwicklern noch in einer entscheidenden Entwicklung. Die Integration eines vollständigen OGDI Umfanges in größere GIS ist daher derzeit noch nicht erfolgt bzw. solange zurückgestellt, bis OGDI in einem fortgeschrittenerem Umfang entwickelt ist. Eine Verwendung war daher zu der Zeit der Entwicklungen für dieses Projekt noch nicht praktikabel, eine zukünftige Verwendung ist aufgrund der Schnittstellen möglich. Die Aktivitäten auf diesem Gebiet sind aber hinsichtlich der nächsten Jahre sehr vielversprechend.

1.4.3. Offene räumliche Informationssysteme

Für zahlreiche konventionelle Funktionen kann ein System, wie das Geographical Resources Analysis Support System (GRASS) eingesetzt werden [Gar1993], das insgesamt seit 1982 entwickelt wird. Bei GRASS handelt es sich nicht um ein GIS, sondern um eine flexible Sammlung von Werkzeugen [Net2000], die die

[2] `http://www.opengis.org` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

Bearbeitung und Interpretation von Raster- und Vektordaten ermöglichen [NB1998] [Alb1993] [Wes1991]. Diese Flexibilität[3] erlaubt einen Einsatz, der weit über die Bearbeitung von Satellitendaten hinausgeht und von hydrologischer Modellierung und Analyse, über Landschafts-Management, Habitat-Modellierung, Städteplanung, Meteorologie, Medizin, Politik bis hin zum Management archäologischer Fundstätten und zur Simulation von Waldbränden reicht. GRASS ist das derzeit vollständigste verfügbare und offene System zur Unterstützung von konventionellen GIS-Applikationen [Sch2001]. Seit 1999 steht GRASS im Sinne der freien Softwareentwicklung unter der GPL. Die weitreichende Modularität von GRASS erlaubt fast unbegrenzte Entwicklungsmöglichkeiten, doch gerade Erweiterungen zum Einsatz von Verfahren für interaktive und dynamische Anwendungen sind, wie dies auch bei anderen GIS vergleichbaren Umfangs der Fall ist, im zugrundeliegenden Konzept nicht vorhanden.

Insbesondere sind die meisten Programme separate Einzelprogramme und in höheren Programmiersprachen geschrieben und müssen für verschiedene Plattformen kompiliert werden.

Da keine gemeinsame Oberfläche existierte, wurde vor einiger Zeit begonnen eine Oberfläche in Tcl/Tk zu implementieren.

Dies ermöglicht in Zukunft eine komfortable graphische Steuerung auf der Bedienebene.

Die Einzelprogramme basieren jedoch auch bei diesem System nach wie vor auf konventionellen Datenformaten, so daß Ereignisse und Dynamik nicht Bestandteil der eigentlichen Daten werden können.

Hier würde ein unvertretbar hoher Entwicklungsaufwand erforderlich sein, Datenformate um die Speicherung von Ereignissen zu erweitern und um alle benötigten Programme zur Handhabung von Ereignissen umzuschreiben.

In gleicher Weise würde eine Aufbereitung der Daten für eine interaktive Nutzung im Netz (z.B. in WWW-Klienten) auch nach der Umgestaltung der einzelnen Programme eine Konvertierung bedeuten.

Eine Nutzung mit bestehenden Systemen für diesen Zweck wäre bezüglich Ereignissteuerung und dynamischer Visualisierung, auch nach erheblichem Aufwand, nur sehr eingeschränkt möglich.

[3] `http://www.geog.uni-hannover.de/grass` ..... [V: 1996] [Ä: k. A.] [Z: 30.01.2001]

2. Ein Konzept zur dynamischen Visualisierung

2.1. Ein neues Konzept

Das Ziel dieser Arbeit ist die Entwicklung eines Konzeptes, das die flexible Nutzung von Ereignissen (s. Definition auf Seite 2) in Verbindung mit geowissenschaftlichen Daten ermöglicht.

2.1.1. Zielgerichtete Fragestellungen

Für zahlreiche naturwissenschaftliche Aufgabengebiete, insbesondere in den Geowissenschaften, ist eine flexible Visualisierung notwendig. Dies ist oft mit bereits vorhandenen Methoden möglich. In den meisten Fällen ist jedoch der Grad der Konfigurierbarkeit, Skalierbarkeit, dynamischen Funktionalitäten und Interaktivität nicht ausreichend.

Um für diese Fälle Kriterien für die Notwendigkeit einer speziellen Eigenentwicklung zu ermitteln, spielen einige Fragen eine besondere Rolle.

1. Ist die Problemstellung mit bekannten Mitteln vollständig lösbar?
2. Sind notwendige Werkzeuge verfügbar und bezahlbar?
3. Ist die Umsetzung mit vorhandenen Mitteln technisch zu aufwendig oder unverhältnismäßig?
4. Welche Arten von Daten sind zu berücksichtigen?
5. Werden zu einem großen Teil dynamische Visualisierungen benötigt?
6. Ist für einige Aufgaben eine standardisierte Skriptsprache wünschenswert und vorhanden?
7. Genügt die Modularität vorhandener Möglichkeiten?
8. Sind eigenständige lokale Applikationen ebenso wichtig, wie Applets (Mini-Programme) oder Optionen für Klient-Server Betrieb?
9. Ist der Grad an Erweiterbarkeit bei vorhandenen Lösungen ausreichend?
10. Sind bei Verwendung vorhandener Mittel bereits Erweiterungen für mögliche zukünftige Aufgaben verfügbar?
11. Sind die vorhandenen Lösungen plattformunabhängig und portabel?

Kaum eine vorhandene Software wird eine zufriedenstellende Antwort auf alle diese Fragen geben, daher wird eine wesentliche Voraussetzung für eine Lösung sein, eine intelligente Kombination verschiedener geeigneter Mittel und Verfahren für das jeweilige Teilproblem zu entwickeln.

2.1.2. Forderungen und Umsetzung

Es sollte möglich sein, mit einem flexiblen Ansatz eine geeignete Lösung zu finden. Dieser Ansatz beinhaltet die folgenden Punkte:

1. Wenn eine Lösung allein mit existierenden Mitteln nicht möglich ist, sind gegebenenfalls Neuentwicklungen notwendig.
2. Werkzeuge für fast alle Entwicklungs- und Anwendungsaufgaben stehen frei zur Verfügung.
3. Bei einer Eigenentwicklung kann präzise auf die notwendigen Bedürfnisse eingegangen werden.
4. Im optimalen Fall sollen Rasterdaten, Vektordaten, beschreibende Daten usw. in verschiedener Weise verwendet werden können. Die unterschiedlichen Daten sind die Basis aller Informationssysteme. Eine starke Integration verschiedener Daten und Funktionen ist ein Kernpunkt bei der Entwicklung eines neuen Systems. Ein solches Modell wird im folgenden mit dem Begriff Objektgraphik bezeichnet.

5. Dynamische Visualisierungen ermöglichen durch ihre weitreichende Interaktivität erst die Darstellung bestimmter Sachverhalte, z. B. wenn zur Laufzeit eine dynamische Berechnung erforderlich ist.

6. Frei verfügbare Skriptsprachen sind inzwischen weit verbreitet und für verschiedene Einsatzgebiete geeignet.

7. Mit der Kombination von höheren Programmiersprachen und Skriptsprachen ist eine sehr flexible Modularisierung durchführbar. Bei geeigneter Vorgehensweise können individuelle Komponenten entstehen, die über Schnittstellen (z. B. IPC) miteinander kommunizieren und Daten gemeinsam nutzen können.

8. In einer Kombination aus höherer Programmiersprache und Skriptsprachen kann auf einfache Weise Rücksicht auf Konfigurierbarkeit, Nutzung von Applets (Mini-Programme), Klient-Server Betrieb und weitere Aspekte genommen werden. Die Schaffung von Komponenten und Modulen kann auf dieser Basis sehr effizient sein.

9. Skriptsprachen erlauben eine effiziente Erweiterung graphischer Oberflächen. Höhere Programmiersprachen können für ressourcenintensive Aufgaben eingesetzt werden.

10. Für höhere Programmiersprachen existieren viele frei verfügbare Bibliotheken und Erweiterungen. Für Skriptsprachen wie Tcl/Tk und Perl ganze Archive mit tausenden frei verfügbarer Erweiterungen.

11. Viele der frei verfügbaren Implementierungen von Programmiersprachen und Werkzeugen sind auf zahlreiche Plattformen portiert und können quasi plattformunabhängig eingesetzt werden.

2.1.3. Vorgehensweise

Verschiedene Schritte sind zur Umsetzung des Konzepts notwendig.

1. Der Problembereich bezüglich Ereignissteuerung und dynamischer Visualisierung muß ermittelt werden. Das bedeutet, daß u. a. die Gründe für eine geringe Eignung konventioneller Verfahren zu untersuchen sind.
   Dies geschieht im Abgleich mit existierenden Lösungsansätzen. Es muß u. a. festgestellt werden, welche Möglichkeiten in Teilbereichen des Einsatzes akzeptabel wären, ob diese kombinierbar und entwicklungstechnisch ausreichend erweiterbar sind.

2. Die funktionalen Anforderungen müssen ermittelt werden. Die Erfahrungen bezüglich existierender Ansätze, Anwendungen und Werkzeuge müssen in den Kontext zur Spezifikation der Anforderungen eingehen.

3. Eine Systemplanung muß durchgeführt werden. In diesem Rahmen wird die Auswahl geeigneter Verfahren getroffen und die notwendige Modularisierung und der Schichtaufbau festgelegt. Die Ergebnisse der vorangestellten Punkte gehen mit ein.

4. Die Teile der zu entwickelnden Komponenten werden auf ihre Umsetzung mittels Skriptsprachen untersucht und entsprechend konzipiert. Modularisierung und Schichtaufbau gehen in die Konzipierung ein.

5. Die Teile der Komponenten, bei denen der Einsatz einer Skriptsprache angemessen erscheint, werden auf ihre Zusammenarbeit mit möglichen Daten untersucht, die auf der Basis einer hochsprachlichen Programmiersprache aufgebaut werden können. Diese können z. B. als Quellentext mit besonderen Eigenschaften (im folgenden als „Objektgraphik“ bezeichnet), aber auch in daraus abgeleiteten Formen verwendet werden.

6. Es werden Werkzeuge und Verfahren zur Realisierung der Teilanforderungen ermittelt. Dies erfolgt insbesondere hinsichtlich der Eignung für eine flexible Kombination von benötigten Werkzeugen und Verfahren.

7. Es wird ein Prototyp implementiert, der den theoretischen Teil durch praktische Anwendung ergänzt. Dieser Prototyp zeigt Machbarkeit, das Erreichen möglicher Vorteile durch den Einsatz auf Quellentext basierender Daten und die Eignung für zukünftige Erweiterungen auf. Durch die zwingende Umsetzung eines Prototyps kann u. a. festgestellt werden, ob die Kapazitäten, die in der Systemplanung zur Softwareentwicklung ermittelt wurden, einer realistischen Einschätzung entsprechen.

Jeder dieser Punkte kann, je nach Umfang der Planungen, einen nicht generell festlegbaren Aufwand beinhalten. Bei kleineren Entwicklungen kann der Aufwand einige Tage bedeuten, bei großen Projekten einschließlich eines Prototyps mehrere Jahre. Dieser Aufwand relativiert sich, wenn der Prototyp konkret als Ausgangspunkt für die Lösung bestimmter Probleme erarbeitet wird. Durch starke Modularisierung und autarke Teile von Komponenten können so recht schnell zielgerichtete Teillösungen für die vielfältigen Anwendungen räumlicher Informationssysteme erarbeitet werden.

2.2. Ein neuer Datentyp: Objektgraphik

Karten sind eine zweidimensionale Darstellung mehrdimensionaler Objekte. Eine zweidimensionale Repräsentation einer sich ständig verändernden Welt hat u. a. den Nachteil, daß viele Informationen nicht in einer einzigen Karte dargestellt werden können.

Um dies näherungsweise zu kompensieren, können mehrere Karten bzw. Datenebenen übereinandergelegt werden.

Doch auch durch eine solche Schichtung (engl.: „layering“) sind viele Informationen noch nicht erreichbar, denn das eigentliche Ziel ist nicht die getreue Nachbildung der Realität, sondern eine Abstrahierung, mit der Aufgabe, Informationen zu erschließen und zu konzentrieren.

Es kann daher sinnvoll sein, einzelne Punkte bzw. Objekte in einem dreidimensionalen Kartenstapel mit speziellen Informationen zu verknüpfen.

Mit Hilfe eines entsprechenden Programms sollte es möglich sein, eine Karte darzustellen. Es sollte auch möglich sein, Objekte aus mehreren Karten zu überlagern und einzelne Elemente in einer entsprechenden Vektorgraphik zu erkennen, zu verschieben oder zu verändern. Unter Umständen ist es auch möglich, bestimmte Punkte auf den betreffenden Karten mit bestimmten Informationen auszustatten.

Es ist jedoch z. B. ohne weiteres möglich, Objekte für eine Karte zu definieren, Objektgruppen zu bilden, Eigenschaften von Gruppen zu verändern und Objekten bestimmte Ereignisse zuzuordnen. Sobald derartige Funktionen benötigt werden, ist ein reines Vektorformat nicht mehr geeignet, da es diese Informationen nicht transportieren kann.

Abgesehen von den Vorteilen, die auch ein einfaches Rasterformat gegenüber Vektorgraphiken haben kann, ist doch der nächste Schritt zu einem Objektformat naheliegend, in dem auch derartige Informationen transportiert werden können.

Wenn ein derartiges Format ausschließlich aus Quellentext einer Programmiersprache besteht, kann es auch Programmelemente beinhalten, beispielsweise Prozeduren oder Bedienelemente zu einem bestimmten Datensatz.

Die so erreichbare Fülle an Funktionalität kann selbst mit erheblichem Aufwand durch die Entwicklung von Ergänzungen zu vektororientierten Programmen nicht erreicht werden.

Diese Grundlage beinhaltet, daß Kartenmaterial in einem derartigen Format so gestaltet werden kann, daß es seine eigene Oberfläche und spezifische Eigenschaften in sich trägt und damit kein weiteres Graphikprogramm zur Betrachtung oder Manipulation notwendig ist.

Der Unterschied zwischen einem Programm und dem Format in der gleichen Sprache kann beispielsweise durch spezifische Definitionen festgelegt sein. Das Resultat ist eine Kombination aus objektorientierten Programmfunktionen, Vektorgraphik und Hypertext.

Bisher besteht jedoch ein Mangel an geeignetem Datenmaterial, das die Voraussetzungen für die Überführung in ein solches Objektformat erfüllt.

Eine notwendige Konvertierung zur Ausnutzung eines solchen Formats ist daher sehr zeitaufwendig und mit vielen manuellen Schritten verbunden. Die Vorgehensweise ist je nach Einzelfall verschieden, da es sich in den meisten Fällen um Daten handelt, die nicht für eine derartige Verwendung konzipiert waren. Gleich bleibt, daß die notwendigen Schritte immer diejenigen sein müssen, die zu einem gültigen Objektformat führen.

In dieser Arbeit werden diesbezüglich folgende Begriffe verwendet:

1. **Objektgraphik:** Das Modell eines Datenmaterials, das auf Quellentext einer Programmiersprache basiert und mit dem auf native Weise Rasterdaten, Vektordaten, beschreibende Daten und weitere Objektdaten sowie Funktionen verwendet werden können, wird als Objektgraphik bezeichnet, wenn auf einzelne Elemente und ihre Eigenschaften intuitiv dynamisch und benutzerdefiniert zugegriffen werden kann.

2. **ereignisorientierte Daten:** Bei der Anbindung von Ereignissen an Elemente einer Objektgraphik mittels nativer Verfahren und benutzerdefinierter Funktionen entstehen ereignisorientierte

Daten, kurz Ereignisdaten, die sogenannte *ereignisaktive Objekte* enthalten.

2.2.1. Grundlegende Voraussetzungen

Gängige Formate zum Austausch von Daten zwischen verschiedenen Geoinformationssystemen verwenden einige mathematische Grundelemente zur Abbildung verschiedener geographischer Strukturen.

Es wird im allgemeinen zwischen POINT, LINE und POLYGON Daten unterschieden. Diese Typen können zweckmäßigerweise z. B. für folgende Strukturen verwendet werden:

- POINT: Meßpunkte, Bohrstellen für Öl und Wasser, Wetterstationen.
- LINE: Zentrale Linienzüge für Straßenzüge, Autobahnen, Bahnschienen, Flüsse.
- POLYGON: Geschlossene Flächen und Bereiche, wie Reservoirs, Inseln, Umrisse von Regionen, Local Government Areas (LGA).

Die beiden grundlegenden Ansätze für die Repräsentation räumlicher Daten sind:

- Rasteransatz: Zellen.
- Vektoransatz: Punkte, Linien, Polygonzüge.

2.2.2. Weitergehende Forderungen

Folgende Voraussetzungen werden an ein zu entwickelndes Format gestellt.

Äußerer Aufbau:

- Das Format muß möglichst engen Bezug zur Syntax einer bereits auf mehrere Systeme portierten Programmiersprache haben.
- Die Syntax des Formats muß die Möglichkeit bieten, interpretiert werden zu können, ohne kompiliert werden zu müssen.
- Das Format sollte über verschiedene mathematische bzw. geometrische Grundelemente verfügen (Punkte, Linien, Polygonzüge usw.).
- Das Format sollte ausschließlich durch den ASCII Zeichensatz (7-bit) abgebildet werden können.

Inhaltlich, bezogen auf Elemente:

- Es sollen alle grundlegenden Informationen herkömmlicher portabler Datensätze durch das Format abgedeckt werden können.
- Es müssen den einzelnen Grundelementen Attribute beigefügt werden können. Dies schließt die Angabe von Füllfarben für geeignete Elemente ein.
- Jedes Grundelement sollte einfach identifizierbar sein.
- Die Funktion einer ID-Nummer sollte von einem wählbaren Attribut zur Verfügung gestellt werden.
- Das Format sollte eine Sortierung von Elementen mit einfachen Mitteln erlauben.
- Bei einem zusammengesetzten Format sollte eine einfache Filterfunktion die Trennung von LINE-, POINT- und POLYGON-Daten ermöglichen.
- Das Zusammenfügen von Datensätzen sollte ohne großen Aufwand auf jeder Systemplattform und jedem Betriebssystem ohne spezielle Werkzeuge durchzuführen sein.

Durch die Attribute und die interpretierenden Komponenten sollen die Grundelemente Objektfunktionen erhalten. Die GIS-Komponente, die zur Visualisierung entwickelt wird, muß verschiedene Manipulationen einzelner Objekte erlauben und Funktionen, beispielsweise die Schichtung (engl.: „layering“) von Objekten, zur Verfügung stellen.

2.2.3. Anbindung multimedialer Komponenten

Durch geeignete Ergänzungen des Datenmaterials und grundlegend erweiterte Software kann eine Bereitstellung von Ereignissen und Funktionen erreicht werden, die das Spektrum an Einsatzmöglichkeiten erheblich vergrößert. Damit erhöht sich auch der Grad an Komplexität des Gesamtsystems.

Über die Zuweisung zu den Attributen können einzelne Elemente der spezifizierten Objektgraphik mit Ereignissen und diese mit einer Vielzahl von Funktionen verknüpft werden.

Diese Möglichkeit zur modularen Gestaltung der einzusetzenden Komponenten erlaubt daher beispielsweise den Aufruf externer Programme, wie Editoren,

Bildverarbeitungssysteme, Datenbanken und einer Fülle weiterer multimedialer Komponenten. Eine Ereignisanbindung von Audio- und Video-Daten an beliebige Objekte ist damit beispielsweise implizit vorhanden.

2.3. Die neue Struktur: Softwareebenen und thematische Ebenen

Die Eigenschaften eines Systems, das dieses hier Objektgraphik genannte Verfahren nutzt, sollten die Eigenschaften des Datenmaterials in verschiedener Hinsicht unterstützen.

Die Softwareebenen werden hier vor den thematischen Ebenen behandelt, da dies für das Verständnis der gebildeten Schichten thematischer Funktionalitäten hilfreich sein kann.

2.3.1. Entwicklerebenen und Benutzerebenen

Für ein Projekt der Komplexität einer GIS-Applikation, welche die beschriebenen Eigenschaften nutzt, ist eine Abstufung folgender Art wünschenswert:

1. Entwickler- und Benutzerebenen.
2. Programmentwicklung, Kernsystem.
3. Programmentwicklung, Komponenten.
4. Programmentwicklung, individuelle Funktionen und Anpassungen, graphische Benutzeroberfläche (GUI).
5. Visualisierung, Bearbeitung, Manipulation etc.

Damit soll ermöglicht werden, daß Entwickler und Benutzer auf einem unterschiedlichem Niveau mit dem System produktiv sein können, sprich neue Kernfunktionen entwickeln oder lediglich eine Umsetzung ihrer Daten erreichen können.

Das Kernsystem kann als ein Teil dienen, um verschiedene Komponenten zu verbinden.

Diese Komponenten müssen von dem Kernsystem loszulösen und in definierbarer Weise autark sein.

Die Entwicklung individueller Funktionen, die Erstellung einer eigenen graphischen Benutzeroberfläche und ähnliche Aufgaben sollen separat möglich sein, ohne Komponenten oder gar das Kernsystem verändern zu müssen.

Visualisierung, Bearbeitung und Manipulation soll auch mit anderen Mitteln erfolgen können, als denen der Komponenten oder des Kernsystems, beispielsweise mit verschiedenen externen Applikationen.

2.3.2. Thematische Ebenen

Konventionelles Datenmaterial besteht im wesentlichen aus Rasterdaten und Vektordaten sowie verschiedenen weiteren Daten, z. B. zur Beschreibung von Teilen des Datenmaterials.

Kommen zu solchem Datenmaterial noch Daten zur Interaktivität bzw. zur Steuerung hinzu, so sind zusätzliche Funktionen notwendig. Desto mehr sollten sie aber in gleicher Weise natürlich behandelt werden können, wie es auch für die übrigen Daten der Fall ist.

Die grundlegenden benötigten thematischen Funktionalitäten sind in Abbildung 2.1 (Seite 20) zusammengefaßt.

Die Funktionalitäten sind zur besseren Übersicht in virtuelle thematische Schichten gruppiert.

Jede dieser Schichten enthält in der Horizontalen die betreffende Thematik, eine benötigte Anwendung, grundlegende Mittel zur Handhabung der Daten bzw. Funktionen dieser Schicht und wichtige Vorgänge zur Kommunikation zwischen den Schichten.

In der Vertikalen ist ein Block mit essentiell benötigten Datentypen (Schicht 2 – Schicht 5) hervorgehoben (gestrichelt). Diese Daten umfassen beispielsweise geologisch oder hydrologisch hervorzuhebende Flächen, Meßdaten zu Lokationen, Beschreibungen und Anmerkungen oder Luftbilder. Unter Beschreibung ist z. B. auch ein Photo des Ortes einer Probennahme zu verstehen. Dieser Block enthält neben den konventionellen Daten auch die Daten zur Interaktivität, die für die Auswertung und weitere fachbezogene Darstellung notwendig sind.

Um diese Daten (in dieser Darstellung handelt es sich um vier Datentypen) ist eine Schicht aus Metadaten (Schicht 1) wünschenswert, mit deren Hilfe sich Objekte bestehend aus Daten der Datentypen zusammenfassen und damit z. B. über Funktionen, z. B. per Programmierung, manipulieren lassen.

Um eine interaktive Darstellung der Daten zu ermöglichen, muß eine Schicht zur Visualisierung (Schicht 6) vorhanden sein. Wie diese genau aussieht und mit welchen Mitteln, z. B. graphische Bibliotheken oder Canvas („virtuelle Leinwand“), sie umgesetzt wird, muß hier noch nicht festgelegt werden.

Da die Visualisierung interaktiv über Ereignisanbindungen gesteuert werden soll, ist eine Ereignissteue-

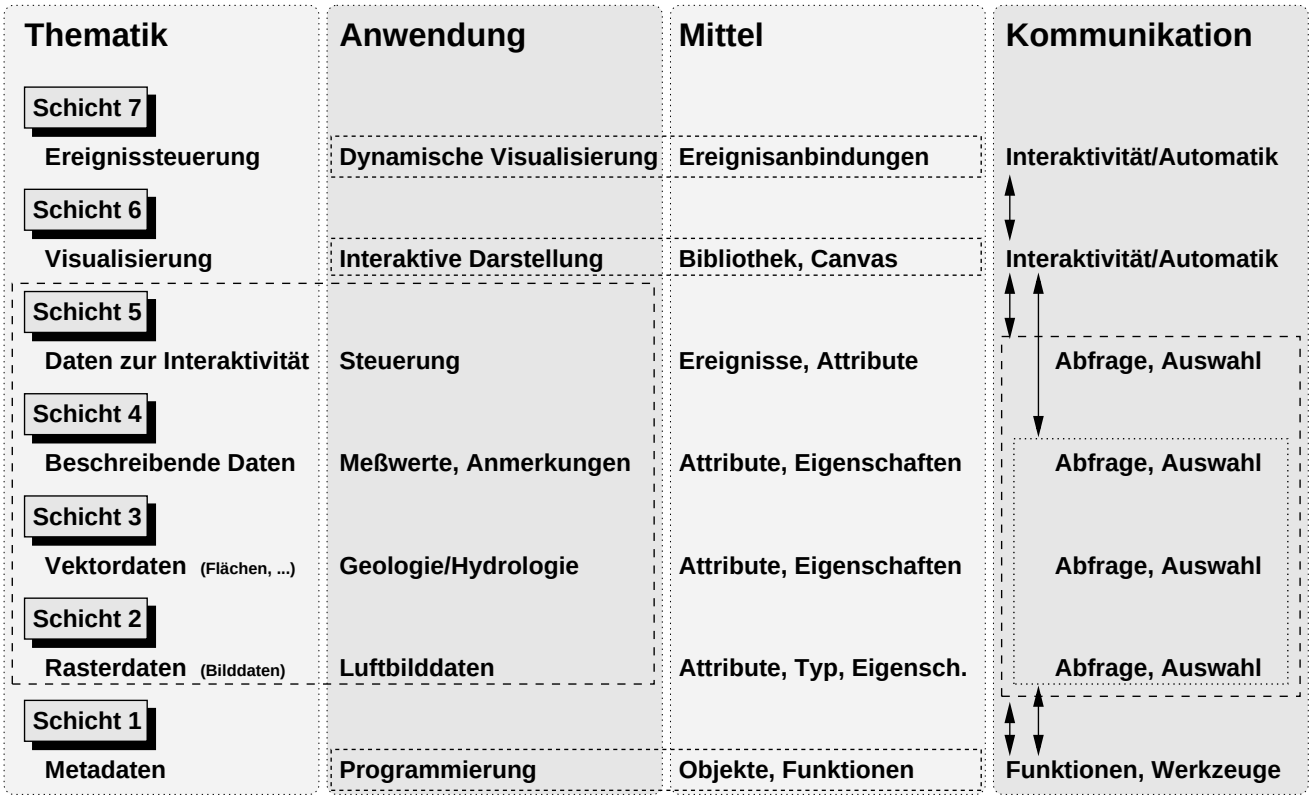

Abb. 2.1: Die neue Struktur: Benötigte thematische Funktionalitäten

rung notwendig (Schicht 7).

Zusammengefaßt wird die dynamische Visualisierung über Ereignisanbindungen, die interaktive Darstellung über Bibliotheken und Canvas und die Programmierung mittels Objekte und Funktionen realisiert.

Soweit zu Thematik, Anwendungen und Mittel. Die Kommunikation zur Nutzung der Funktionalitäten kann beliebig komplex sein. Hervorgehoben sind:

- Ereignissteuerung und Visualisierung sollten interaktiv und automatisch, d. h. durch direkte Benutzereinwirkung und durch indirekte Prozesse, z. B. Einflußnahme von außen, bidirektional verbunden sein. Das bedeutet, Ereignisse können Einfluß auf die Visualisierung nehmen, ebenso wie die Visualisierung z. B. eigene Ereignisse bereitstellen kann.

- Visualisierung und eigentliche Daten sollen bidirektional über die Daten zur Interaktivität und direkt miteinander verbunden werden können.

- Mittels der übergreifenden Metadaten soll ermöglicht werden, daß Werkzeuge oder benutzerdefinierte Funktionen Zugriff auf alle Daten haben.

Über diese Aufgaben hinaus sind weitere Funktionen realistisch. Der weitere Fortgang der Planung wird entscheiden, welche mit den vorhandenen Mittel in geeigneter Weise umsetzbar sind. Beispielsweise ist denkbar, daß über die Metadaten nicht nur der Zugriff auf die eigentlichen Daten, sondern auch auf Visualisierung, Ereignissteuerung und damit auf Teile der Umgebung erreicht werden kann.

Solche Möglichkeiten müssen aber gleichfalls durch Mechanismen eingeschränkt werden können, um in einer endgültigen Version die notwendige Sicherheit zu gewährleisten und Abhängigkeiten verschiedener Daten und Komponenten zu verringern.

3. Systemplanung für einen neuen Prototyp

3.1. Planung eines Prototyps

3.1.1. Grundlegendes

Entwurf („Design“), Erwerb notwendiger Komponenten und Implementierung eines GIS sind in aller Regel bedeutende Investitionen bezüglich personeller und finanzieller Mittel.

Besonders wichtig ist die Eingrenzung des Problems. In fast allen Fällen besteht dieses im Fehlen von Eigenschaften und Funktionen bestehender Software und Methoden.

3.1.2. Planungsablauf

Alle wichtigen Punkte vom Anlaß der Entwicklungsüberlegungen bis zum Stadium nach der Entwicklung eines Prototyps werden im folgenden beschrieben.

Um die Aspekte für die hier behandelten Untersuchungen und Entwicklungen kurz zu fassen, sind wichtige Zusammenfassungen an dieser Stelle *kursiv* gesetzt.

1. **Systemplanung**

   a) Problemerkennung
      - Detailierte Übersicht über den technologischen Markt
      - Faktoren Angebot-Anschub (engl.: „supply-push“)
      - Faktoren Nachfrage-Bedarf (engl.: „demand-pull“)
        – Routineaufgaben
        – Institutionalisierte Aufgaben
        – Nachfragetendenz (engl.: „affective demand“)
        – Informationen über GIS
          ∗ Status existierender GIS
          ∗ Richtung der Entwicklungen der GIS-Industrie
          ∗ potentielle GIS-Applikationen im eigenen Unternehmen

   b) Entwicklung zum Management Support
      Nicht erforderlich.

2. **Studie zum Funktionsbedarf** (engl.: „Functional Requirements Study“, FRS) Diese Studie wird während der Entwicklung erweitert und aktualisiert und bildet die Voraussetzung für einen Projektplan. Es wird die Vorgehensweise bei der Durchführung geplant, die Zielgruppe festgestellt und es werden Befragungen insbesondere hinsichtlich des Projektplans durchgeführt.
   Festlegung der Funktionen hinsichtlich der Entwicklung eines Prototyps.

3. **Systemevaluierung**

   a) Projektplan/Strategiekonzept
      ➢ Integration in ein größeres (eventuell bereits bestehendes) Projekt:
        Nein.
      ➢ Zentrale oder verteilte Entwicklung:
        Zentrale Entwicklung.
      ➢ Anzahl der Entwickler:
        1 Entwickler, Teilzeit.
      ➢ Anzahl Anwender der Komponente:
        Einzelne Personen oder kleine Gruppe.
      ➢ Automatisierte/manuelle Schritte:
        Beides.
      ➢ Geschwindigkeit/Zeitraum der Entwicklung:
        ca. 3 Jahre.
      ➢ Prioritäten Entwicklung, Ergebnis:
        Entwicklung: schnelle Entwicklung, geringe Kosten.
        Ergebnis: Prototyp zur Demonstration flexibler Anbindungen von Ereignissen und Quellentext-basierter Daten.
      ➢ Leitung der Entwicklung intern/extern:
        Intern.
      ➢ Finanzierung:
        Privat.

b) Abfrage und Anforderung von Vorschlägen (engl.: „Request for Proposals“, RFP)
Kann nicht durchgeführt werden.

c) Hardware, Entscheidungen
Unix/Linux Workstation und Peripherie, privat verfügbar.

d) Software, Entscheidungen
Offene und freie Entwicklungsumgebungen und Werkzeuge.
Die durchgeführten Entwicklungen können jedoch nicht als offener Quellentext (engl.: „open source“) nach der Definition offener Quellen (OSD) im engeren Sinne bezeichnet werden [Aut1999c]. Offene Quellen bedeutet, daß die Quellen oder Teile davon öffentlich verfügbar sind, es bedeutet aber nicht automatisch, daß es sich um frei verfügbare Software handeln muß.

4. **Bewertung**
Die Bewertung (engl.: „benchmarking“) ist das zentrale Element zur Minimierung der Gefahren der Systemauswahl. Beim Bewertungs-Test wird in der Regel die Ausrüstung vom Hersteller oder Vertreiber bereitgestellt, die Daten und Vorgaben vom Kunden, der auch die anfallenden Kosten übernimmt.
Kann nicht durchgeführt werden.

5. **Pilot Projekt**
Das Pilot Projekt wird nach der Fertigstellung eines weitentwickelten Prototyps durchgeführt und soll Erfahrung zur bevorstehenden Eignung und Vermarktung liefern.
Kann nicht durchgeführt werden.

6. **Kosten/Nutzen-Analyse** für Weiterentwicklung.
Es werden die Kosten definiert, der Nutzen ermittelt und eine Gegenüberstellung von Kosten und Nutzen angestrebt hinsichtlich:
– möglicher Alternativen
– quantitativen Supports
– zukünftigen Planungen
Kann nicht durchgeführt werden.

3.1.3. Unterschiede zu kommerziellen Entwicklungen

Jeder der fünf Hauptpunkte von der Problemerkennung bis zum Pilot Projekt benötigt selbst in größeren Entwicklungsabteilungen in der Regel mindestens mehrere Monate bis zu einem zeitweisen Abschluß.

Für Weiterentwicklungen nach dieser Phase ist der Entwicklungszyklus (engl.: „project life cycle“) nach der Kosten/Nutzen-Rechnung bei kommerziellen Applikationen in aller Regel der ausschlaggebende Faktor, sofern das Projekt nicht aus anderen Gründen weiterentwickelt wird.

Hier kann z. B. eine nichtkommerzielle Entwicklung Vorteile für Kunden und Anwender haben.

3.2. Planung der Komponenten

3.2.1. Anforderungen an die Software

Durch die Vielschichtigkeit der Basiskomponenten müssen die verwendeten Entwicklungswerkzeuge Methoden zur effizienten Realisierung vollständig unterschiedlicher Funktionalitäten bieten.

Wichtige Funktionalitäten sind in diesem Zusammenhang:

Entwicklungstechnisch:

- Verwendung eines Compiler, kein offener Code.
- Portables GUI und portable Entwicklungskomponenten.
- Offene, erweiterbare GUI- und Programmier-Schnittstelle.
- Modularisierung der einzelnen Elemente.
- Autarke Eigenschaften (engl.: „standalone“) der wesentlichen Funktionen.
- Trennung von Entwickler- und Anwenderebene (eventuell Verwalterebene).
- Portable, integrierbare und flexible Dokumentation [Lam1985] [Kop1994] [Kop1995] [Kop1997], Ansätze für Literate Programming (z. B. TEX/LATEX etc.) [Knu1984a] [Knu1984b].

Datentechnisch:

- Schichtung (engl.: „layering“) nur begrenzt notwendig.
- Import und Skalierung von ArcView Daten und/oder anderem Datenmaterial.
- Import und Skalierung punktueller Daten.
- Export von Daten (ArcView, ASCII).

- Export von Protokollen etc. (LaTeX, DVI, PostScript, HTML, ASCII/ISO Latin 1).

Weiterführende Aspekte:

- Graphiksystem, flexible externe GIS-Funktionen (GRASS).
- Verwendung mit portablen Datenbanken/ Formular Generator u. ä. (GROK, INGRES etc.).
- Integration wissenschaftlicher Problemlösungen (Fortran, C, C++).
- Möglichkeit zu Klient-Server Einsatz.
- Möglichkeit zum Einsatz von portablen Skriptsprachen (Perl[1], Tcl[2], `sed`, `awk`).
- Integration von Object Request Broker Technologien (ORB)/Common Object Request Broker Architecture (CORBA) [MP1999].

Es existiert auf absehbare Zeit keine einheitliche Lösung aller dieser Probleme. So kann es z. B. wünschenswert sein, einen Großteil aller Arbeit mit dem System auf Basis von Kleinrechnern durchzuführen, einzelne Aufgaben jedoch sollten auch auf Großrechneranlagen erledigt werden können, wenn dies erforderlich ist. Es ist jedoch nur möglich, einzelne Module geeignet zu gestalten, wenn aus der Erfahrung geeignete Konzepte gewählt oder geschaffen werden. Da auch dieser Ansatz nicht auch nur annähernd die denkbaren Einsatzgebiete erfassen kann, muß die Möglichkeit vorhanden sein, das System auch nach Fertigstellung der meisten Komponenten z. B. um ein Modul vollständig neuer Art zu ergänzen, ohne andere Komponenten erheblich modifizieren zu müssen.

3.2.2. Beispiele für Entwicklungswerkzeuge

Die Entwicklungswerkzeuge, Systeme, Bibliotheken und die weitere Umgebung sollen selbstverständlich ebenso diesbezügliche Anforderungen erfüllen:

- Alle essentiellen Teile sollen mit freien Quellen („GNU is Not Unix“, GNU) als offene Software (engl.: „Open Source“, auch „OpenSource“) vorliegen.
- Alle Werkzeuge sollen kostenfrei verfügbar sein.
- Die eingesetzten Methoden sollen standardisiert sein oder einem offenen de-facto Standard entsprechen.
- Die Verwendung soll hinsichtlich der genannten Punkte eine weitergehende Eignung für den Einsatz aufweisen.
- Die Werkzeuge sollen auf möglichst viele Plattformen portabel/portiert sein.
- Bei Verwendung muß eine Vermarktung unter der GPL *und* kommerziell erlaubt sein. Dies betrifft daher z. B. auch Bibliotheken und die Art der Erstellung ausführbarer Programme [Dri2001a].

Die Systeme, die für Entwicklung und Tests zum Einsatz kommen, sind:

- GNU/Linux[3]
- FreeBSD[4]
- (Solaris)
- (AIX)
- ((NT/Win*))

Basisversionen verschiedener Distributionen der verwendeten Systeme können als CDROM Abbild über die angegebenen Adressen kostenfrei aus dem Internet bezogen werden.

Die Arbeitsoberflächen (engl.: „desktops“) bzw. Fenster-Manager (engl.: „window manager“) unter denen derzeit getestet wird, sind:

- KDE[5]
- GNOME[6]
- Enlightenment Window Manager[7]
- Window Maker[8]
- bis FVWM

[1] `http://www.cpan.org` ..... [V: 1995] [Ä: k. A.] [Z: 25.01.2001]
[2] `ftp://ftp.scriptics.com/pub/tcl/` ..... [V: 1997] [Ä: k. A.] [Z: 25.01.2001]
[3] `http://www.linuxiso.org` ..... [V: 1999] [Ä: k. A.] [Z: 25.02.2001]
[4] `ftp://ftp.freebsd.org/pub/FreeBSD/releases/i386/ISO-IMAGES` ..... [V: 1999] [Ä: k. A.] [Z: 25.02.2001]
[5] `http://www.kde.org` ..... [V: 1997] [Ä: k. A.] [Z: 25.02.2001]
[6] `http://www.gnome.org` ..... [V: 1998] [Ä: k. A.] [Z: 28.02.2001]
[7] `http://www.enlightenment.org` ..... [V: 1998] [Ä: k. A.] [Z: 26.02.2001]
[8] `http://www.windowmaker.org` ..... [V: 1998] [Ä: k. A.] [Z: 26.02.2001]

3.2.3. Beispiele für integrierbare Software

Im folgenden sind beispielhaft Software, Programme und Konzepte aufgeführt, die sich für eine Nutzung mit dem entwickelten Prototyp anbieten. Dies ist nur eine kleine Übersicht, die eine Vorstellung von den beabsichtigten Möglichkeiten vermitteln soll.

System, Programmierung und Entwicklung:

GNU/Linux[9] (s. auch FSF[10], `linux.de`[11]), Skripting (Tcl/Tk, Perl[12], bash, `sed`, `awk` etc.), Compiler[13] (gcc, g77 etc.), Tcl/Tk Entwicklungswerkzeuge (TclPro `prodebug`, `procheck`, usw.), Debugger (`gdb`), Tcl Compiler, Tcl Bytecode Compiler, Java/JDK, Externe Bibliotheken (`Tk`, `wx*`, vtk[14], ACE etc.) `make` [SM2000], `configure`, `xtags` etc., Versionskontrolle via CVS und RCS, Literate Programming und Dokumentation (`cweb`[15], LaTeX[16], `de.comp.text.tex`[17], `comp.text.tex`[18] etc.), [Ess2000], Editoren (emacs, xemacs[19], `vim`[20] etc.).

Anwendungen und Plattformen:

- Apache WebServer, Tcl WebServer (tclhttpd[21]).
- PHP[22].
- Graphik etc. (xfig, GIMP[23], netpbm etc.).
- Datenbank (Informix, Adabas, grok, INGRES, db++ etc.).
- X Window System oder Microsoft Windows.
- GRASS.

GRASS ist die einzige vollausgereifte Sammlung von GIS-Werkzeugen, die auch im Quellentext frei verfügbar ist und aufgrund ihrer stetigen Weiterentwicklung eine kontinuierliche Eigendynamik hat.

Linux eignet sich hervorragend zum Einsatz auf der Entwicklerseite, in gleichem Maße inzwischen auch auf der Anwenderseite. Linux ist bezogen auf Entwicklungswerkzeuge, Stabilität, Skalierbarkeit, Umfang und Funktionalität, Preis und zumindest in subjektiver Hinsicht auch bezüglich des Bedienkomforts allen derzeitigen Microsoft Windows Oberflächen und Systemen (3.1/95/97/98/CE/NT/2000/XP) überlegen.

Aufgrund der konzeptionell angestrebten Portabilität der Entwicklungen ist eine Verbreitung des Entwicklungssystems von untergeordneter Bedeutung.

Als Entwicklungssystem wurde eine aktuelle Version mit stabilem Kernel und umfangreichen Entwicklungswerkzeugen gewählt [S.u1997] [S.u2001].

Gerade bei der Arbeit mit großen Datenmengen, insbesondere bei Bildmaterial, erhöhen die Eigenschaften dieses Systems die Leistungsfähigkeit des Gesamtsystems erheblich, z. B. durch Zwischenspeicherfunktionen (engl.: „cache“) und dynamische Pufferfunktionen (engl.: „buffer“), verzögertes Schreiben und weitere Eigenschaften (engl.: „delayed write“ und engl.: „read ahead“). In Kombination mit fast beliebiger Vergrößerung des virtuellen RAM, ist in Einzelfällen auch eingeschränkte Hardware einsetzbar.

Die inzwischen bequeme Installation, Deinstallation und Benutzerverwaltung, Datenimport und -export, ausgezeichnete Druckertreiber und Druckfilter und flexible portable Sicherungsmöglichkeiten unterstützen die Arbeit.

Die angebotenen GNU/Linux Distributionen („GNU is Not Unix“, GNU [Dri2001b], Namensgebung zur Distanzierung vom ursprünglichen Unix von AT&T und Western Electric) bieten zudem einen reichhaltigen Umfang an Programmkomponenten für die verschiedensten Aufgabenbereiche.

Die Wahl der graphischen Benutzerschnittstelle (Graphical User Interface, GUI) fiel auf das X Window System, das auf Unix Rechnern sehr verbreitet

[9] `http://www.linux.org` ... [V: 1993] [Ä: k. A.] [Z: 25.01.2001]
[10] `http://www.fsf.org` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[11] `http://linux.de` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[12] `http://www.perl.org` ... [V: 1996] [Ä: k. A.] [Z: 25.01.2001]
[13] `http://www.fsf.org` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[14] `http://www.cs.rpi.edu/~martink/` ... [V: 1996] [Ä: k. A.] [Z: 25.01.2001]
[15] `http://www.dante.de` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[16] `http://www.dante.de` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[17] `news:de.comp.text.tex` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[18] `news:comp.text.tex` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[19] `http://www.xemacs.org` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[20] `http://www.vim.org` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[21] `http://www.scriptics.com/software/tclhttpd` ... [V: 1997] [Ä: k. A.] [Z: 25.01.2001]
[22] `http://www.php.org` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[23] `http://www.gimp.org` ... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

ist, weil es seit langer Zeit deutlich leistungsfähiger ist, als z. B. typische PC-Benutzerschnittstellen, gegenüber anderen bekannten Oberflächen eine eingebaute Netzwerkfähigkeit besitzt (Internet RFC 1013) und zudem für ein weites Feld besserer Hardware erprobt ist. Die bedeutendste frei verfügbare Implementierung für verschiedenste Plattformen ist das XFree86 des XFree86 Projekts[24]. Das X Window System regelt die netzwerktransparente Bild- und Eingabeübermittlung zwischen Klient und Server und ermöglicht netzwerkweiten Zugriff. Das X Window System ist streng genommen kein simples GUI, sondern ein Protokoll, das beschreibt, wie sich ein solches System verhalten soll. Es ist damit ein Werkzeug, das den Einsatz vielseitiger graphischer Fenster-Manager und damit verschiedener vollständiger graphischer Benutzeroberflächen (GUI) und den Einsatz vollständiger moderner Arbeitsoberflächen (engl.: „desktops“) ermöglicht.

Insbesondere ist die Oberfläche des X Window Systems ereignisorientiert bzw. differentiell. Es müssen nicht kontinuierlich alle möglichen Geschehnisse abgefragt und damit auch übertragen werden, sondern es wird zunächst vereinbart, welche Ereignisse von Interesse sind.

Die aktuellsten sich etablierenden Entwicklungen graphischer Oberflächen sind die objektorientierten und CORBA- bzw. DCOP-basierten Arbeitsoberflächen K-Desktop Environment (KDE) und GNU Network Object Model Environment (GNOME) [RM2000]. Selbst auf diesem Niveau graphischer Oberflächen spiegelt sich die Unix Toolbox Philosophie wieder, größere Aufgaben durch die Verbindung grundlegenderer Komponenten flexibel zu lösen.

Allerdings erwies sich CORBA für die Entwicklung von größeren „Desktop“ Anwendungen als zu langsam. Aus diesem Grund wurde beispielsweise für die Entwicklung von KDE ab Version 2 DCOP entwickelt und seitdem verwendet [Kir2000].

Die erste Version des X Window Systems wurde 1984 am Massachusetts Institute of Technology (MIT) ursprünglich zu Schulungszwecken entworfen.

Das X Window System, kurz X11, ist eine klassische Klienten-Server (engl.: „client-server“) Applikation und als solche nicht Kernel-orientiert. Durch das X Window System wird nicht nur eine graphische Oberfläche bereitgestellt, sondern ein Klienten-Server Protokoll festgelegt, das die Kommunikation zwischen den X-Klienten und dem X-Server regelt.

X bietet zahllose Möglichkeiten für die Umsetzung verschiedenster Software und zeichnet sich durch eine hervorragende Stabilität aus. Auf GNU/Linux Systemen wird standardmäßig eine freie Version XFree86 verwendet, die kontinuierlich weiterentwickelt wird.

Die Schichtung der graphischen Oberfläche auf einem X Window System läßt sich vereinfacht folgendermaßen (Abbildung 3.1) schematisieren:

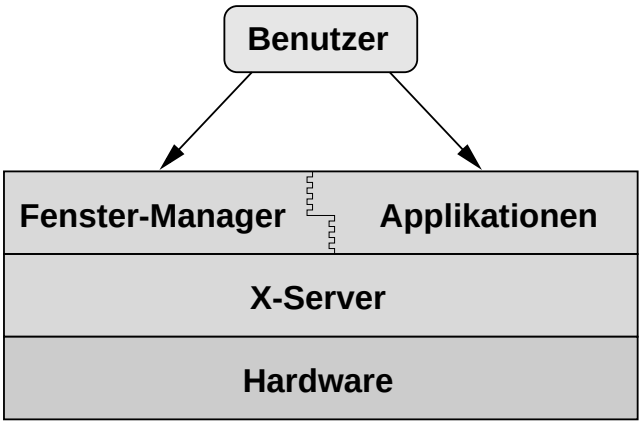

Abb. 3.1: Schichtung der graphischen Oberfläche X Window System

Neuentwickelte Arbeitsoberflächen („Desktop-Umgebungen“), z. B. GNOME und KDE, spielen bei Ausnutzung der Funktionalitäten von X-Server und Fenster-Manager eine immer größere Rolle und komplementieren und erweitern die Funktionalitäten in vielfältiger Weise [RM2000].

Microsoft Windows ist von Microsoft weit verbreitet worden und könnte daher zumindest derzeit noch zum Einsatz auf der Anwenderseite kommen.

Ein Eingriff in die Instabilitäten und Unzulänglichkeiten von Microsoft Windows ist aufgrund der nicht offenen Vermarktungspolitik von Microsoft aber leider nicht möglich. Dies ist mit ein Grund für die über Jahre bestehenden Instabilitäten und den geringen innovativen Fortschritt seit den ersten Versionen bis in die Gegenwart.

Ein Einsatz kam aufgrund der geringen Eignung für die geplante flexible Entwicklung und hinsichtlich der breiten Verfügbarkeit besser geeigneter Systeme nicht in Betracht.

Für die Teile der Komponenten der GIS-Applikation, die eine portable graphische Oberfläche angemessen erscheinen lassen, können folgende Beispiele genannt werden (Tabelle 3.1). Diese Werkzeuge werden in der Regel auch als Werkzeuge zum „Bau graphischer Oberflächen“ (engl.: „GUI builder“) bezeichnet.

[24] `http://www.xfree86.org` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

GUI Builder	*Copyright/Hersteller*
XForms	© 1995 T. C. Zhao and Mark Overmars
Qt	© 1995–1997 Trolltech AS GPL seit 06.09.2000
X-Designer	© Imperial Software Technology
InterViews/	© 1987–1991 Stanford University
Fresco	© 1991 Silicon Graphics, Inc.
CORBA	© Object Mangement Group [Ama1997]
Tcl	© The Regents of the University of California, Sun Microsystems, Inc., Dr. John Ousterhout [Ous1994] [Wel1997]
wxWindows	© 1993–1997 AIAI, University of Edinburgh [Sma1996]
Empress	© Empress
TeleUSE	© Thomson Software Products

Tab. 3.1: Übersicht der berücksichtigten GUI Builder

Es ist die Summe an Eigenschaften, die wxWindows gegenüber anderen Bibliotheken für die Entwicklung stabiler Oberflächen auszeichnet.

- Unterstützung von X Window System und MS Windows.
- Objektorientierter Ansatz (C++) mit API.
- Unterstützung von dynamischen Bibliotheken.
- Kapselung der X-Motif-Aufrufe.
- Offener Quellentext.
- Unterstützung von GNU C++.
- Graphische Primitive (engl.: „graphics primitives“).
- Unterstützung von Interprozeß-Kommunikation (engl.: „interprocess communication“).
- Freie Verfügbarkeit.
- Integration von Hypertext.
- Unterstützung des Internets (engl.: „Internet support“).

wxWindows hat eine minimale Unterstützung von Farbpaletten (engl.: „colour maps“), jedoch zum Zeitpunkt der Planung keinen OLE-2 Support und Restriktionen bei Eltern-Kind Relationen (engl.: „parent-child relations“). Diese Einschränkungen sind nicht von übergroßer Bedeutung, zeigen aber, daß auch hier nicht alle plattformspezifischen Eigenheiten abgedeckt werden. Vielmehr ist wxWindows gerade für die konventionelle Entwicklung sehr großer Projekte geeignet. Für dynamische und kleinere modulare Komponenten sollten flexiblere Lösungen existieren. Diese Lösungen sind Inhalt der folgenden Diskussion.

Gerade hier bietet Tcl/Tk einen geeigneteren Ansatz. Aufgrund seiner besonderen Eignung sei hier vorweggenommen, daß die besonderen Eigenschaften von Tcl/Tk zu einer Verwendung geführt haben und daher in den folgenden Kapiteln ausführlicher diskutiert werden.

Für einen weitergehenden Einsatz erscheint eine Kombination mit Java und Markup-Sprachen [Sch1999], wie der am Europäischen Kernforschungszentrum CERN entwickelten Hypertext Markup Language (HTML) oder TEX/LATEX sehr naheliegend.

HTML basiert auf der Structured General Markup Language (SGML, ISO 8879, aus dem Jahr 1986). Mit dem verwendeten HTML lassen sich zahllose Erweiterungen, wie JavaScript, Cascading Style Sheets CSS, Extented Server Side Includes XSSI etc. nutzen. Neuere Weiterentwicklungen die vom W3C[25] vorangetrieben und koordiniert werden, wie z. B. XML [LS1999] [Eck2000], erweitern die Funktionalitäten für die Zukunft erheblich und stellen die langfristige Verwendbarkeit sicher.

Produkte, die aufgrund geeigneter Bedingungen in die engere Wahl kommen, sind in Tabelle 3.2 aufgeführt.

Produkt	*Lizenz*
Tcl/Tk	Public Domain
GRASS	Public Domain, seit 1999 GPL
wxWindows	Freeware
GNU C++	Freeware, GPL
GNU/Linux System	Freeware, GPL

Tab. 3.2: Produkte in der engeren Wahl

Für die Visualisierung seien weiterhin folgende Beispiele genannt (Tabelle 3.3).

[25] http://www.w3.org ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

Produkt	*Copyright/Hersteller/Referenz*
VTK	[SML1998] [SM1998] [Kle1997]
PHIGS/PEX	ISO/ANSI 9592
OpenGL	© 1997 Silicon Graphics Inc. [WND1997]
Mesa	© Brian Paul

Tab. 3.3: Produkte im Bereich Visualisierung

Einige der genannten Bibliotheken basieren auf Motif. Motif [Kob1991] ist ein weit verbreiteter Industriestandard zur Entwicklung von Benutzerschnittstellen. Seit einiger Zeit ist Motif im Quellentext frei verfügbar.

Neuere sehr mächtige Entwicklungen (z. B. für die Entwicklung von KDE und GNOME) haben jedoch bereits sehr weite Anwendungsbereiche gefunden. Dies wird Motif auf absehbare Zeit jedoch nicht endgültig verdrängen.

Das Visualization Toolkit (VTK[26]) ist ein C++ und Tcl objektorientiertes 3D-Visualisierungssystem. VTK[27] unterstützt vor allem OpenGL, GL, XGL, Starbase, Mesa und steht auch unter Linux[28] zur Verfügung.

Die Open Graphics Library (OpenGL, GL) bietet über das Graphics Library Utility Toolkit (GLUT) und die Graphics Library Utility Library (GLU) eine portable, plattform- und betriebssystemunabhängige Software-Schnittstelle (engl.: „software interface“) für Graphik-Hardware. Die Mesa 3D-Graphikbibliothek ist das entsprechende frei verfügbare Pendant zur OpenGL. Der Schwerpunkt dieser Bibliotheken ist die Erstellung von errechneten („gerenderten“) dreidimensionalen Graphiken und bewegten Animationen. Möglichkeiten zur Gestaltung von Benutzeroberflächen oder auch Druckausgabe sind in diesen Bibliotheken nicht enthalten [PR1996].

Unter Unix stehen verschiedene Kommandointerpreter (sogenannte *shells*) zur Verfügung, die unter einer graphischen Oberfläche eine ideale Möglichkeit zur transparenten Modularisierung bieten. Damit ist gewährleistet, daß zwischen der graphischen Oberfläche und den integrierten Programmkomponenten eine offene Schnittstelle (engl.: „interface“) zur Verfügung steht, die separat modifiziert werden kann und flexible Möglichkeiten u. a. zur Prozeßsteuerung, Parallelisierung und Subshell-Vererbung bietet.

3.2.4. Komponentenebenen für die geplanten Entwicklungen

Damit ist für die geplanten Entwicklungen eine effiziente Trennung der Komponentenebenen nach Funktionalität möglich (Tabelle 3.4).

Komponente	*Beispiel zur Realisierung*
1. GUI	Tcl/Tk (`wish`)
2. Schnittstelle (GUI/extern)	Shell (bash [NR1998], ksh [Ros1993] usw.)
3. externe Komp.	C++, Fortran, Tcl, Perl usw. sowie beliebige Bibliotheken

Tab. 3.4: Komponentenebenen nach Funktionalität

Jede dieser Ebenen ist für eine zu entwickelnde GIS-Applikation aus folgenden Gründen nützlich:

1. a) Die graphische Oberfläche ist offen. Es kann flexibel bestimmt werden, welche Teile der Oberfläche später vom Anwender modifiziert werden dürfen.
   b) Die Entwicklungsumgebung der graphischen Oberfläche ist objektorientiert. Die Bibliotheken können beliebig um eigene C/C++ Funktionen erweitert werden.

2. a) Die Shell bietet eine Programmierumgebung, mit der alle Aufgaben einer Schnittstelle (engl.: „interface“), z. B. zu anderen Anwendungen, abzudecken sind. Beliebige Teile der Schnittstelle können so gestaltet werden, daß sie vom Anwender beliebig modifizierbar sind.
   b) Auf Mehrprozessor-Computern kann z. B. auf dieser Ebene eine weitestgehende Parallelisierung einzelner Funktionen vorgenommen werden. Dies stellt für den Anwender eine einfach zu nutzende Möglichkeit dar, Aktionen gleichzeitig, d. h. pseudoparallel, auszuführen.

3. a) Jede bestehende Komponente kann nach den technischen Gegebenheiten und Funktionen der Komponente in die Oberfläche integriert werden. Diese Programme müssen nicht im Quellentext vorliegen, um integriert werden zu können.

[26] `http://www.kitware.com/vtk.html` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[27] `http://www.cs.rpi.edu/~martink/` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[28] `http://kitware.com/vtkhtml/vtkdata/VTK_Linux_WWW/vtk_body.htm` ..... [V: k. A.] [Ä: k. A.] [Z: 01.07.2001]

b) Integrierte Komponenten können ihre Selbstständigkeit bewahren und lassen sich sowohl im Stapelverarbeitungsmodus (engl.: „batch modus“) als auch interaktiv betreiben.

c) Teile der graphischen Oberfläche können als eigenständige Komponente oder Programm entwickelt werden.

d) Spezielle Programmteile können individuell optimiert werden. Dies ist z. B. insbesondere bei numerisch anspruchsvolleren Aufgaben sehr wünschenswert.

e) Komponenten können leichter auf Groß- und Superrechner übertragen werden. Entsprechende Fortran-Programme werden immer einen erheblichen Geschwindigkeitsvorteil gegenüber C++ Implementierungen haben [Kno1997]. Vorteil einzelner Programmiersprachen und Verfahren können so optimal genutzt werden.

3.2.5. Parallelisierung über die Schnittstelle

Parallelisierung bezeichnet, im Kontext dieser Arbeit, eine günstige Verteilung der Ansprüche eines Prozesses auf die Systemressourcen. Parallelrechner gewinnen bei der derzeitigen Hardwareentwicklung zunehmend auch bei Klein- und Kleinstrechnern an Bedeutung. Ein Prozeß kann bezüglich seiner Beanspruchung der Systemressourcen auf drei Arten charakterisiert werden (Tabelle 3.5).

Schwerpunkt	*Beispiel*
CPU intensiv	numerische Aufgaben u. ä.
I/O intensiv	Einlesen oder Wegschreiben größerer Datenmengen
interaktiv	Benutzerführung

Tab. 3.5: Spezifikation eines Prozesses nach Beanspruchung von Systemressourcen

Auf einem Multitasking-Einprozessorsystem im Gegensatz zu einem Multiprocessing-System, lassen sich nur Prozesse effizient parallelisieren, die aus diesen drei Punkten jeweils verschiedene Schwerpunkte haben. Ein Beispiel ist das Einlesen vieler kleiner Datensätze, die über einen aufwendigeren Algorithmus bearbeitet werden müssen. Dabei können lediglich I/O und Bearbeitung mit einem Prozessor parallelisiert werden. Eine Parallelisierung der Bearbeitung der einzelnen Datensätze wäre jedoch mit nur einem Prozessor aufgrund der Umschaltprozesse eher leistungssenkend. Dieses Phänomen wird gewöhnlich als „Dreschen“ (engl.: „trashing“, soviel wie Dreschen, Flattern, Überlasten) bezeichnet und tritt insbesondere auf Einprozessorsystemen auf.

3.3. Aufbau

3.3.1. Architekturmuster

Ein Entwurfsmuster beschreibt einen Lösungsweg [GHJV1994]. Ein Architekturmuster ist ein Entwurfsmuster (engl.: „design pattern“) für die Lösung von Problemstellungen beim Grobentwurf.

Bei dem geplanten Konzept handelt es sich um den Aufbau einer Mehrschichtarchitektur. Diese Architektur soll modular aufgebaut sein.

Da eine Implementierung nicht zwingendermaßen objektorientiert sein sollte, sind die im weiteren verwendeten Begriffe in Analogie, aber nicht streng im Sinne objektorientierter Softwareentwicklung zu verstehen.

Je nach Komponente müssen Mechanismen existieren, die eine Handhabung von Eigenschaften, Zuständen, Schnittstellen, Implementierungen usw. ermöglichen. Einige grundlegende Lösungswege sind:

Dekorierer: Die Eigenschaften der Teile der Komponenten und der Daten sollen unabhängig von Klassen dynamisch veränderbar und erweiterbar sein.

Beobachter: Es müssen Mechanismen existieren, die es erlauben Zustandsänderungen an Komponenten und Daten an andere Teile mitzuteilen, z. B. an Objekte.

Brücke: Es muß die Möglichkeit bestehen, Schnittstellen von ihrer Implementierung zu trennen. Dies wird z. B. für eine umfassende Plattformunabhängigkeit benötigt.

Memento: Zustände von Teilen der Komponente und Zustände der Daten sollen restauriert werden können.

Adapter: Es muß die Möglichkeit bestehen, neue Schnittstellen zu Klassen bzw. bestimmten Datentypen zu erzeugen.

Existieren bestimmte Möglichkeiten nicht implizit, so kann versucht werden, diese durch bestehende Mittel der eingesetzten Werkzeuge nachzubilden. Dies kann

in vielen Fällen zugunsten anderer Faktoren durchaus sinnvoll sein.

3.3.2. Modulare Mehrschicht

Bei einer Architektur, wie der im Konzept beschriebenen, sind Teile durch einen „Kleber“ (engl.: „glue“) miteinander verbunden. Dieser „Kleber“ besteht aus Skripten.

Da die entwickelten Komponenten auch noch dem Anwender *alle* Möglichkeiten bereitstellen, Synchronisation und Parallelität für seine Prozesse bzw. Aufrufe zu nutzen, ist eine kurze Beschreibung dieser Möglichkeiten sinnvoll.

Der Anwender kann in seinen Skripten, Daten, Bibliotheken etc. jeden dieser Wege mit relativ geringem Aufwand nutzen und kombinieren.

Der Aufbau von Abhängigkeiten zwischen Modulen kann als Schichtung zweier Schichten gesehen werden. Für eine Schicht, aus der weitere Aktionen außerhalb dieser Schicht ausgelöst werden sollen, existieren folgende Möglichkeiten.

1) Kindprozeß und `exec`.
2) Kindprozeß und `fileevent`.
3) Ladbare Erweiterungen.
4) Angepaßtes `main()` gelinkt mit `libtcl`.

Diese Möglichkeiten haben, kurz umrissen, bestimmte typische Anwendungsgebiete, basierend auf ihren Vorteilen und Nachteilen. Es wurden kurze und prägnante Beispiele in Tcl gewählt, um die Verfahren zu illustrieren.

3.3.3. Beispiele zur modularen Mehrschicht

1a) Kindprozeß und `exec` (synchron)

Beispiel (Abbildung 3.2):

```
set myresult \
 [exec cmd args 2>@ stderr]
```

Abb. 3.2: Quellentextbeispiel: Kindprozeß und `exec` (synchron)

Liefert insbesondere ein schnelles kurzes Ergebnis (Sperren (engl.: „blocking“), Speicher (engl.: „memory“)).

Pro:

- sehr übersichtlich
- verbreitet unter Unix

Contra:

- synchron, Sperren (engl.: „blocking“)
- keine engen Schleifen („Unkosten“ durch `fork()` (engl.: „fork overhead“))
- nicht alle Betriebssysteme

1b) Kindprozeß und `exec` (asynchron)

Beispiel (Abbildung 3.3):

```
set mypid \
 [exec cmd args 2>@ stderr &]
```

Abb. 3.3: Quellentextbeispiel: Kindprozeß und `exec` (asynchron)

Dies kann insbesondere bei einmaligen Aktionen eingesetzt werden, bei denen der weitere Status nicht weiter relevant ist, beispielsweise in Bezug auf MIME verknüpfte Anwendungen.

Pro:

- asynchron, Ereignisse, z. B. Signale

Contra:

- kein `exit` Status, kein Rückgabewert
- kein Terminierungssignal

2) Kindprozeß und `fileevent` (channel)

Beispiel (Abbildung 3.4):

```
proc was {arg} {
  global jobFinished
  puts "Still at $arg"
    if {![eof $arg]} {
        gets $arg data
      if [eof $arg] {
          set jobFinished 1
          catch {close $arg}
          puts "EOF reached"
          return
    }}}

set f [open "|calc " r]
fconfigure $f
 -buffering none -blocking no
fileevent $f readable "was $f"
vwait jobFinished
exit
```

Abb. 3.4: Quellentextbeispiel: Kindprozeß und `fileevent` (channel)

Fileevents sind sehr universell einsetzbar, z. B. bei verteilten Architekturen.

Pro:

- asynchron
- volle Eingabe/Ausgabe (engl.: „full I/O“) mit `"r+"` `open`
- Kindprozeß lebt nur solange benötigt
- Signal bei `stdout close`
- volle `exit` Rückgabewerte (z. B. via `catch {close}`)
- flexibel (Sockets, Befehls-Weiterleitungen (engl.: „named pipes“), `ptys`)
- Integration durch `vwait` mit `after` Handlern und GUI

Contra:

- IPC und Unkosten bei der Syntaxanalyse (engl.: „parsing overhead“)
- keine Befehls-Weiterleitungen (engl.: „pipes“) auf bestimmten Betriebssystemen

3) Ladbare Erweiterungen

Beispiel (Abbildung 3.5):

```
load my.so
myCall myargs ...
```

Abb. 3.5: Quellentextbeispiel: Ladbare Erweiterungen

Die betrifft vor allem Erweiterungen, die z. B. über IPC und vergleichbare Mechanismen hinausgehen.

Pro:

- sehr schnell
- kristallklare API
- auch nicht-Tcl Erweiterungen auf Win32

Contra:

- nicht asynchron
- Pflege (Verzeichnisse, Versionen, Namensgebung, . . .)

4) Angepaßtes `main()` . . .

Beispiel (Abbildung 3.6):

```
<main.c>:
...
Tcl_CreateCommand(...) ...

cc ... -lmylib -ltcl
```

Abb. 3.6: Quellentextbeispiel: Angepaßtes `main()`

Statische Ergänzungen sind nur selten sinnvoll, werden aber gelegentlich aufgrund von Performanz, Entwurf und Vermarktung gewählt.

Pro:

- einzige Lösung für statische firmeneigene Teile (z. B. engl.: „legacy code“)
- einfaches Packen in Binärform

Contra:

- Komplexität steigt stark bei orthogonalen Erweiterungen
- Tcl/Tk muß ebenfalls verfügbar sein

3.4. Softwareentwicklung

Die folgenden Abschnitte zur Softwareentwicklung sollen kurz auf relevante Zusammenhänge bezüglich des Konzepts und der geplanten Entwicklung hinweisen.

3.4.1. Vorgehensmodell

Das Vorgehensmodell ist architektur- und komponentenzentriert, anwendungsfallgetrieben, iterativ und inkrementell.

3.4.2. Anforderungsanalyse

Einige Aspekte der konventionellen Anforderungsanalyse im Rahmen eines Vorgehensmodells treten hier aufgrund der Entwicklungssituation in den Hintergrund.

Vorstudie: Die Zielsetzung war durch das Ziel dieser Arbeit gegeben. Anforderungen und Problemfeld sind diesbezüglich geklärt. Eine Anwendungsfallübersicht wird an späterer Stelle durch die behandelten Fallstudien geliefert. Lösungsalternativen werden anhand verschiedener Konzepte und Werkzeuge angesprochen.

RAD: Ein sogenannter *RAD-Workshop* (RAD, *Rapid Application Development*) konnte aufgrund der Entwicklungssituation nicht erfolgen.

Geschäftsprozeßanalyse: Dieser Vorgang (auch Analysemodell) entfiel, da keine konkreten Abhängigkeiten von Geschäftsprozessen vorlagen.

Anwendungsfallanalyse: Die softwarerelevanten Arbeitsabläufe sind während des Entwicklungsprozesses mit den fachspezifischen Anforderungen abzugleichen.

Evaluierung: Die Evaluierung von Architektur und Entwicklungsumfeld erfolgte durch verschiedene Tests mit Entwicklungsumgebungen und dadurch z. T. prädestinierten Architekturmodellen.

3.4.3. Zusammenfassung der Forderungen an einen Prototyp

Mit den aufgeführten Anforderungen müssen für ein entstehendes Gesamtsystem folgende Punkte umgesetzt werden:

Technisch:

- Keine Verwendung proprietärer Produkte für alle Kernentwicklungen.
- Ausschließlich Verwendung portabler, offener Entwicklungswerkzeuge.
- Entwicklung oder Verfügbarkeit des Hauptanteils der GIS-Applikationen in einer systemnahen Programmiersprache zur Gewährleistung möglichst hoher Effizienz.
- Entwicklung der Oberfläche in einer geeigneten objektorientierten, portablen Skriptsprache, die möglichst rasche Entwicklung ermöglicht.

Logistisch-organisatorisch:

- Trennung in verschiedene Entwicklungsebenen.
- Verbindung aller wichtigen Applikationen unter einer gemeinsamen Oberfläche.
- Entwicklung geeigneter Programme, die in die Oberfläche integriert werden können, die aber auch ohne Oberfläche im Stapelverarbeitungsmodus lauffähig sind.
- Unterstützung flexibler Erweiterbarkeit des Gesamtsystems.
- Minimierung der Betriebssystemabhängigkeiten und gleichzeitige Ausnutzung spezifischer Eigenschaften wichtiger Betriebssysteme.
- Keine Verwendung einer visuellen Entwicklungsumgebung für die Konstruktion der übergreifenden Oberfläche.

Funktionell:

- Entwicklung und Bereitstellung von Programmen zum Import und Export von Daten.
- Entwicklung und Bereitstellung von Datenbankfunktionen.
- Entwicklung und Bereitstellung von Funktionen zur Bearbeitung unterschiedlicher Daten, wie Satellitenbildern und vergleichbarem Material.
- Entwicklung von Eingabemasken zu einzelnen Programmen.
- Entwicklung von Verfahren zur Erstellung netzwerkfähiger systemunabhängiger Abbilder darzustellenden Datenmaterials.

- Erstellung von Beispielroutinen zur Stapelverarbeitung.
- Erstellung verschiedener Beispiele zur Integration weiterer Applikationen in das Gesamtsystem.

Thematisch:

- Speicherung, Verarbeitung und Transport geographischer oder verwandter Daten.
- Interaktives Abfragen von Informationen zu Lokationen.
- Bearbeitung von aufwendigen Aufgaben im Stapelverarbeitungsmodus, z. B. „Processing“.
- Visualisierung und Darstellung geographischer oder verwandter Daten in Form von Listen/ Reports, Graphiken und Diagrammen, virtuellen Graphiken, Animationen, Filmen oder interaktiven Abbildungszuweisungen (engl.: „mappings“).

3.4.4. Reale Rahmenbedingungen

Die Rahmenbedingungen bei vielen realen Anforderungen in diesem Bereich erfordern nicht nur Datenbankfunktionen, Darstellung räumlicher Informationen und Netzwerkfunktionen, sondern vielfältige externe Programmkomponenten zur Auswertung und Darstellung.

Ein solcher Komplex ist selbst bei einem kleineren Netzwerk mittelfristig mit nicht unerheblichen Systemkonfigurationen und weiteren Entwicklungen verbunden, sondern auch längerfristig ohne umfangreiche variierende administrative Aufgaben nicht praktikabel.

Ein komplexes System ist daher nie ein *Produkt*, sondern immer ein *Prozeß*.

Ziel muß es daher sein, die Komponenten zur Erfüllung der Teilaufgaben so flexibel wie möglich zu gestalten.

Es lassen sich daraus folgende Schlüsse ziehen:

- Die strikte plattformunabhängige Trennung von Datenbank, GUI und Komponenten ist unbedingt notwendig. Für den Echteinsatz muß neben der Ereignisdatenbank eine Datenbank für beliebige andere Daten eingesetzt werden.
- Eine offene und flexible Lösung kommt dem realen Anwendungsfall sehr zugute. Das bedeutet, es sollten für alle Basiskomponenten frei verfügbare Produkte eingesetzt werden können. Nicht alle Teile des resultierenden Gesamtsystems müssen aber im Quellentext frei zugänglich sein.
- Die Flexibilität bezüglich der einzelnen Komponenten macht unabhängig von einem bestimmten Produkt, wie einer speziellen Datenbank oder dem Betriebssystem und erhöht die Chancen für ein breiteres Einsatzfeld.
- Die zumindest teilweise Anwendung auf einem Server im Internet ist inzwischen fast immer wünschenswert. Dies kann auf verschiedene Weise realisierbar sein, mit Schwerpunkt auf einem speziellen Server und mit herkömmlichen Mitteln des Zugriffs oder mit Zugriff über ein Plugin oder über ein spezielles Protokoll usw.
- Damit verbunden muß für alle kritischen Funktionalitäten ein Sicherheitskonzept verfügbar sein. Dies ist in Form ähnlich einer sicheren virtuellen Maschine bereits verfügbar und muß in Zukunft für weitere Funktionalitäten ebenso von Anfang an mitentwickelt werden.
- Für spezielle Anforderungen müssen in jedem Fall Auswertungsprogramme, Komponenten oder zumindest Filter zur Unterstützung graphischer Darstellungen und dergleichen erstellt werden.
- Laufende Betreuung und umfangreiche Konfiguration sind bei derart komplexen Anwendungen immer erforderlich.
- Der Einsatz wichtiger Komponenten im Klienten- und Serverbereich unter Linux/Unix eröffnet zahlreiche Einsatzmöglichkeiten und vereinfacht viele Abläufe.

3.4.5. Aufwandschätzung für den Prototyp

Nach der Abschätzung des voraussichtlichen Umfangs des Prototyps, der notwendig ist, um die Anwendbarkeit des entwickelten Konzepts zu demonstrieren und der kurz zusammengefaßten Systemplanung war eine Entwicklung im Rahmen dieser Arbeit zwar aufgrund der Kapazitäten nicht immer einfach, aber möglich.

Der Zeitraum von etwa 3 Jahren für eine Entwicklung zu einem facettenreichen Umfeld, wie dem einer Komponente für ein GIS, ist gerade für die Entwicklung durch eine Einzelperson sehr kurz.

Die eigenen fachlichen Erfahrungen mit der Problemstellung und die beruflichen Erfahrungen mit der Umsetzung notwendigerweise aufwendiger Lösungen trugen zu einer positiven Einschätzung bei.

4. Wahl der Entwicklungsumgebung

4.1. GUI Entwicklungsumgebung für den neuen Prototyp

4.1.1. Tcl

Tcl („Tool Command Language“) ist zum einen eine Programmiersprache und zum anderen eine Bibliothek. Tcl/Tk (Tk, „Toolkit“) eignet sich aufgrund seiner extrem hochsprachlichen Konzeption (engl.: „Very High Level Language“, VHLL) und seiner graphischen Fähigkeiten hervorragend zur schnellen und portablen [Zim1997] Entwicklung graphischer Oberflächen [Joh1997a] und Visualisierungen [HSAS1996], insbesondere im Klienten-Server Bereich für Intra- und Internetlösungen [TZ1998]. Aufgrund des Entwurfs eignet sich Tcl/Tk für schnelle Entwicklung, graphische Benutzerschnittstellen und plattformübergreifende Entwicklung. Tcl ist auf verschiedenen Ebenen erweiterbar. Dank der Verfügbarkeit des Quellentexts sind Tcl/Tk Interpreter für alle wichtigen Plattformen einsetzbar. Im Gegensatz zu den meisten Hochsprachen, beispielsweise C, muß der Entwickler bei Tcl keine kompletten Codebäume liefern, um Entwicklungen z. B. an bestimmte Plattformen anzupassen.

Tcl/Tk und X11 sind besonders prädestiniert für Ereignis-Programmierung (comp.lang.tcl[1]). Auch wenn in den letzten Jahren viele sehr gute Sprachen und Werkzeuge für eine effiziente Entwicklung und die Erstellung von Oberflächen entwickelt wurden und einen breiten Einsatz gefunden haben, wie z. B. Python und GTK+, so ist Tcl auf dem Gebiet der Ereignis-Programmierung selbst bei kritischer Betrachtung immer noch die erste Wahl [Sch2000a].

Für diesen Zweck stehen in Tcl eine Vielzahl von Ereignistypen und Modifizierer sowie die Möglichkeit zur Verfügung neue Ereignistypen und Ereignisse, wie virtuelle und synthetische, d. h. zusammengesetzte Ereignisse zu definieren [RT1999].

Insbesondere interaktive X11 Anwendungen bewegen sich während des größten Teils der Laufzeit in einer Ereignis-Schleife (engl.: „event loop“).

Tcl/Tk Programme laufen im wesentlichen in zwei Phasen ab. In der ersten wird die Anwendung initialisiert, die Oberfläche aufgebaut und die Datenstrukturen geladen. In der zweiten Phase begibt sich das Programm in eine Ereignis-Schleife, die auf entsprechende Ereignisse wartet. Die Ereignisse ihrerseits rufen eine Routine zur Handhabung der Ereignisse (engl.: „event handler“) auf, der veranlaßt, die mit dem Ereignis verbundenen Instruktionen auszuführen.

Während der Aufbau der Tool Command Language (Tcl) eher prozeduralen Charakter hat, enthält das entsprechende Toolkit Tk eine Fülle objektorientierter Elemente.

Als Toolkit („Werkzeugkasten“) werden im allgemeinen gesammelte Hilfsmittel für besondere Aufgaben bezeichnet. Bei Tcl/Tk ist das Toolkit Tk primär eine Zusammenstellung von Funktionen zur Erstellung graphischer Fensterelemente.

Dabei handelt es sich z. B. um Elemente (engl.: „widgets“), die helfen, den Aufbau einer graphischen Oberfläche besser zu strukturieren [Joh1994].

Sehr mächtige Funktionalitäten stellt Tcl im Bereich der Prozeß-Kommunikation (IPC) über seine `fileevent` und `send` Fähigkeiten zur Verfügung. Die dadurch erreichten Möglichkeiten gehen weit über das hinaus, was man diesbezüglich von allen anderen modernen Shells, z. B. der hervorragenden Bourne Shell kennt. Die Basis dieser Fähigkeiten ist es, ein Skript auszuführen, wenn ein Kanal (engl.: „channel“) lesbar oder schreibbar wird. Auf diese Weise können Verbindungen (engl.: „file event handler“) zwischen einem Kanal und einem Skript bzw. Ereignis hergestellt werden.

Eine wesentliche Eigenschaft von Tcl ist, daß Kommandos in Tcl Listen als Argumente empfangen und sie als Kommandos einlesen können. So entstehen konventionelle Kontrollstrukturen als Kommandos und nicht syntaktisch.

[1] `news:comp.lang.tcl` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

Diese und weitere Eigenschaften haben zu einer großen Vielzahl verschiedener Produkte geführt, die auf Tcl/Tk basieren oder dieses nutzen [Lai1995] [Vir1996] und in letzter Zeit auch zunehmend mit zahlreichen Beispielen in den Bereichen Umwelt, Kommunikation und Informationssysteme [HLS2000] [Zer2000].

4.1.2. Tcl und Java

Zu Unrecht hat die Konzentration auf Java in den letzten Jahren die Aufmerksamkeit in vielen Fällen von Tcl/Tk abgelenkt. Dies ist jedoch nicht unbedingt als Nachteil zu sehen. Große Synergieeffekte lassen sich so beispielsweise bei der Kombination von Tcl und Java (z. B. Jacl) und damit auch von den Entwicklungen im Java Sektor erzielen [Joh1998]. Diese Kombination hat nicht nur Bedeutung für die vollständige Steuerung von Java Applikationen über Tcl und für den Einsatz im Internet. Über die Verbindung von Komponenten mit Tcl statt mit Java kann neben der einfachen Handhabung ein erheblicher Geschwindigkeitsvorteil erzielt werden. Eine andere wichtige Entwicklung zur Kombination von Tcl und Java ist TclJava[2], der Vorläufer von TclBlend[3].

4.1.3. Tcl und Informationssysteme

Im Bereich GIS existieren viele Einsatzmöglichkeiten. Beispielsweise hat das OGDI eine ANSI C Schnittstelle (API) und eine Tcl/Tk Schnittstelle (API), d. h. über die Tcl/Tk Schnittstelle ist auch die C Schnittstelle nutzbar und damit die OGDI Treiber [CLGM1998]. Die OGDI Schnittstelle wird teilweise bereits in mehrere plattformunabhängige GIS eingebaut, wie GRASSLAND[4], insbesondere basierend auf dem freien und plattformunabhängigen GRASS[5]. GRASS verfügt seit einiger Zeit über eine eigene in Tcl/Tk entwickelte Oberfläche.

Andererseits wird Tcl/Tk nicht nur für Schnittstellen und graphische Oberflächen, sondern in vielen Informationssystemen in Kombination mit C/C++ für die Entwicklung von Software für die Datenbearbeitung eingesetzt.

Die Eignung für Web- und B2B-Aufgaben [Pat2000] haben Tcl/Tk in den letzten Jahren zudem zu einer breiten Unterstützung bei Anwendern und in der Industrie [Pat1998] und verstärkt auch zu einer kommerziellen Nutzung verholfen [Hil2000].

4.1.4. Tcl und Objektorientierung

Es sind spezielle frei verfügbare objektorientierte Erweiterungen entwickelt worden, wie `[incr Tcl]`/`[incr Tk]` [McL1993] [McL1994] [McL1995] [Zer2001a], OTcl [Wet1995] und STOOOP [Fon1997]. Der größte Teil dieser Erweiterungen kann mit der „Object Modeling Technique“ (OMT) [RBP+1991] beschrieben werden.

4.1.5. Tcl und freie Erweiterungen

Als Skriptsprachen werden Programmiersprachen bezeichnet, die nicht für das jeweilige Betriebssystem kompiliert, sondern bei ihrer Ausführung interpretiert und ausgeführt werden.

Dies geschieht meist durch einen sogenannten *Interpreter* (engl.: „interpreter“).

Diese Programme oder Skripten können während ihrer Erstellung ausprobiert und geändert werden und eignen sich gut für verschiedene Zwecke, beispielsweise zur Automatisierung von Aufgaben in der Datenverarbeitung.

Die Verwendung einer Skriptsprache wie Tcl/Tk hat den angenehmen Nebeneffekt, daß der Code in der endgültigen Version weitestgehend offen gehalten werden kann, und daher dem Anwender beliebige Modifikationen und Anpassungen erlaubt, falls solches gewünscht wird. So stehen beispielsweise handliche Möglichkeiten zum Klonen von Prozeduren [Scr1999], Schnittstellen zu fast allen gängigen Programmiersprachen, Datenbanken und Protokollen etc. sowie eine Fülle von Erweiterungen zur Verfügung, von denen einige bereits weite Verbreitung gefunden haben [Har1997] [RT1999].

Erweiterungen existieren außerdem in fast allen Bereichen [Tcl2000b], komplexe Schnittstellen (Tix, `tixwish`), Graphen und Tabellen (BLT), Visualisierung und 3D-Graphik (VTK – OpenGL, TSIPP), Oracle und Sybase Datenbanken (Oratcl, Sybtcl), Distributed Programming, Netzwerk und Remote Procedure Calls (RPC) (Tcl-DP), Mehrbenutzer Umgebungen (GroupKit), Automatisierung von Programmen (expect), komplexe Datenstrukturen, Unix Systemaufrufe (TclX), in C/C++ eingebettete Tk Aufrufe (ET), Baumstrukturen (engl.: „trees, tree structures“), Aufzeichnung und Wiedergabe von Interaktionen (TKReplay), Unterstützung

[2] `http://ptolemy.eecs.berkeley.edu/~cxh/ptpub/tcljava.html` ..... [V: k. A.] [Ä: k. A.] [Z: 31.01.2001]
[3] `http://ptolemy.eecs.berkeley.edu/~cxh/java/tclblend/index.html` ... [V: k. A.] [Ä: k. A.] [Z: 31.01.2001]
[4] `http://www.las.com/grassland/` ..... [V: 1997] [Ä: k. A.] [Z: 25.01.2001]
[5] `http://www.geog.uni-hannover.de/grass/index2.html` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

von Unicode (seit Tcl Version 8), Audio-Unterstützung (Snack), XML (TclXML), CORBA (TclMico) und viele weitere.

4.2. Skripting mit Perl

4.2.1. Perl

Für zahllose Anwendungen, z. B. zur Verarbeitung von Textdateien, Konverter, Filter, Datenbanken, Nutzung von Internet Protokollen etc., werden Sprachmittel verwendet, die auf Finiten Automaten basieren. Perl stellt aufgrund seiner besonderen Architektur und seiner ausgereiften Methoden die beste Wahl für derartige Aufgaben dar.

Bei den beschriebenen Entwicklungen wurde Perl für die Konvertierung und Aufbereitung von Daten eingesetzt, aber auch für Datenbankzugriffe und zur Automatisierung von wiederkehrenden Aufgaben, z. B. im Bereich der Administration.

Folgende Eigenschaften verleihen Perl kurzgefaßt seine besondere Eignung:

RegEx: Der hochentwickelte RegEx-Maschine genannte Mechanismus zur Auswertung (engl.: „matching“, engl.: „pattern matching“) von regulären Ausdrücken (engl.: „regular expressions“) [Fri1997]. Reguläre Ausdrücke sind syntaktische Hilfsmittel, die Klassen von Zeichenketten beschreiben.

NFA: Die Basis der RegEx-Maschine bildet ein nichtdeterministischer finiter Automat (NFA).

Backtracking: Der implementierte NFA erlaubt Rückbezüge (engl.: „backtracking“). Bei einem deterministischen finiten Automaten (DFA) ist dagegen während eines Suchprozesses keine Speicherung von Treffern möglich.

Look-ahead: Die RegEx-Maschine erlaubt weiterhin ein Vorausschauen (engl.: „look-ahead“) auf Zeichenketten in Kombination mit konjunktiven Ausdrücken. Dies ist nur durch das Backtracking möglich. Zusätzlich sind seit der Version 5.005 zurückschauende Funktionen (engl.: „look-behind“) realisiert.

Skriptsprache: Perl wird in einer Skriptsprache geschrieben. Der Zugriff auf diese Programme und die Generierung von Programmteilen sind daher aus anderen Applikationen heraus möglich.

Vorkompilation: Die Implementierung des Perl-Compilers erlaubt eine Vorkompilation (engl.: „precompiling“) in speziellen Bytecode vor der Ausführung, was die Geschwindigkeit bei größeren Aufgaben deutlich erhöht. (Eine Umsetzung von Perl in eine höhere Programmiersprache und damit eine Nutzung von Maschinensprache ist ebenso möglich.)

Ein Perl Skript wird zur Laufzeit nicht lediglich interpretiert, sondern in einen Zwischencode kompiliert. In neueren Implementierungen kann auch dieser Zwischencode, ein Bytecode, gespeichert werden.

In vielen Fällen können Programme, die in Perl geschrieben sind, schneller abgearbeitet werden, als vergleichbare, die in Java geschrieben sind. Solche Programme sind aber bezüglich der Ausführung meist immer noch etwas langsamer, als solche in C. Java Bytecode emuliert eine virtuelle Maschine auf tiefster Prozessorebene, Perl Bytecode besteht aus Aufrufen optimierter hochsprachlicher C-Funktionen.

Perl Skripten können sehr portabel und multiplattformfähig programmiert werden. Die Aufteilung in Module fördert ein effizientes Arbeiten in größeren Projekten.

4.2.2. Perl und Objektorientierung

Perl unterstützt objektorientierte Programmierung. Mit Perl kann die objektorientierte Entwicklungsdisziplin eingehalten werden durch:

1. Ein Objekt wird gebildet durch eine Referenz, die weiß, welcher Klasse sie angehört.

2. Eine Klasse ist ein Paket, das Methoden zur Verfügung stellt, um mit Referenzen umzugehen.

3. Eine Methode ist eine Unterroutine, die eine Objektreferenz oder einen Paketnamen für statische Methoden als erstes Argument erwartet.

Pakete werden in Dateien abgelegt, die über die Verzeichnisstruktur ihre Klassenhierarchie erhalten.

Während C++ statische und virtuelle Methoden kennt, verfügt Perl ausschließlich über Klassenmethoden und Objektmethoden, die durch ihre Verwendung unterschieden werden. Aus dieser Sicht sind alle Perl Methoden virtuell. Perl Klassen können dagegen nicht nur als Typdefinition, sondern auch als Metaobjekt betrachtet werden, da sie Ähnlichkeit mit einem Objekt haben.

4.3. Einfluß von Skriptsprachen

Die in den letzten Jahren rasante Entwicklung auf dem Kleinrechnermarkt macht einen Einsatz derartiger Sprachen zudem zunehmend interessanter [Ous1997b] [Tcl2000a]. Skriptsprachen (engl.: „scripting languages“) und Sprachen für die Systemprogrammierung, wie z. B. Fortran77, Fortran90, C [KR1988], C++ [Str1987], Pascal, Java [AG1996] haben vollständig unterschiedliche Ausrichtungen und Schwerpunkte.

Die Verwendung von Skriptsprachen [Moz2000] [Inc1997] [OG1988] [WCS1996] bietet einen wesentlichen komplementären Teil zu den höheren Programmiersprachen, nicht zuletzt, weil ein Großteil der Aufgaben, die bei der Entwicklung graphischer Oberflächen anfallen, über Skriptsprachen, wie Tcl/Tk prägnanter und schneller zu lösen sind, als z. B. mit Java oder C++. Die Verwendung von Skriptsprachen gewinnt zudem immer größeren Einfluß [Fly1999] auf die Entwicklung neuer Programmierumgebungen [Pat1997].

Es bestehen bereits vielversprechende Entwicklungen der Integration weiterer Skriptsprachen in zukünftige Browsertechnologien und WebServer. Kandidaten hierfür sind derzeit neben Tcl/Tk vor allem auch Perl und Python[6] [Sch2000b] (Mozilla[7], ActiveState[8], Apache/mod_perl[9], Apache/mod_dtcl[10]).

Nach Erfahrungen verschiedener Gruppen von Softwareentwicklern ist die Implementierung einer Applikation unter Zuhilfenahme einer Skriptsprache, z. B. Tcl/Tk, um einen Faktor 5 bis 10 schneller zu realisieren, als z. B. allein unter Einsatz einer Programmiersprache, wie Java oder C++ [Ous1997a] [Bro1975].

Diesen Erfahrungen kann ich mich bezüglich der meisten Entwicklungsaufgaben, insbesondere der Ereignissteuerung in Kombination mit graphischen Oberflächen, nur anschließen.

Hingegen ist die Effizienz bei den meisten höheren Programmiersprachen ohne eine spezielle Unterstützung durch die Hardware größer, da der Code bei der Erstellung eines Programms in Maschinensprache übersetzt und nicht während der Ausführung interpretiert wird. Tcl-Applikationen werden in aktuellen Implementierungen gegenüber äquivalenten kompilierten Programmen um einen Faktor 5 bis 20 langsamer ausgeführt.

Um die Vorteile der höheren Programmiersprachen *und* der Skriptsprachen nutzen zu können, kann im Fall einer komplexen Applikation die graphische Oberfläche zum größten Teil mit Tcl/Tk sehr effizient entwickelt werden. Der GUI Teil einer Applikation ist in der Regel sehr zeit-unkritisch. Zeitkritische Komponenten, wie Funktionen und Routinen mit explizit numerischen Aufgaben können in einer geeigneten höheren Programmiersprache implementiert werden.

4.4. Verschiedene Entwicklungsumgebungen

Visuelle Entwicklungsumgebungen existieren für Skriptsprachen, wie auch für viele höhere Programmiersprachen. Ebenso existieren für Tcl/Tk inzwischen sehr ausgereifte visuelle Entwicklungsumgebungen und verschiedene Arten von Compilern. Zahlreiche dieser Entwicklungsumgebungen sind proprietär, so daß sie für diese Aufgabe ohnehin nicht prädestiniert sind, da sie in der Regel nur die Software eines bestimmten Herstellers gezielt unterstützen und in fast allen Fällen keiner Standardisierung unterliegen. Kaum eines dieser gängigen Werkzeuge unterstützt zudem das Konzept offener Systeme konsequent.

Von visuellen Entwicklungsumgebungen, die seit geraumer Zeit verschiedene Hersteller im Angebot haben, wird aus folgenden Gründen Abstand genommen.

- Die zu entwickelnde Applikation muß einen hohen Grad an Integrationsfähigkeit haben und ebenso viele verschiedene bestehenden Verfahren und Komponenten integrieren können, die mit den unterschiedlichsten Programmiersprachen und Entwicklungswerkzeugen erstellt worden sind. Eine visuelle Entwicklungsumgebung zu verwenden hieße, eine weitere Entwicklungsumgebung zu den bestehenden hinzuzufügen.
- Eine komplexere Applikation, deren verschiedene Aspekte über jeweils geeignete Verfahren entwickelt werden sollen, kann nicht über *eine* visuelle Entwicklungsumgebung entwickelt werden.
- Visuelle Entwicklungsumgebungen decken nur einen Teil des jeweils bestehenden Sprachumfangs ab.

[6] `http://www.python.org` ..... [V: k. A.] [Ä: k. A.] [Z: 20.02.2001]
[7] `http://www.mozilla.org` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[8] `http://www.activestate.com` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[9] `http://www.apache.org` ..... [V: k. A.] [Ä: k. A.] [Z: 06.03.2001]
[10] `http://tcl.apache.org` ..... [V: k. A.] [Ä: k. A.] [Z: 06.03.2001]

- Die Implementierung zahlreicher Elemente erfordert es, Systemabhängigkeiten zu berücksichtigen, die über visuelle Entwicklungsumgebungen nicht zur Verfügung stehen.
- Der Wiederverwertbarkeitsgrad von Quellentext aus visuellen Entwicklungsumgebungen ist gering.
- Visuelle Entwicklungsumgebungen sind in der Regel nicht frei verfügbar und unterliegen nicht normierbaren und normierten Veränderungen.
- Bezüge des Entwicklers zu den Interna einzelner Applikationen gehen verloren. Dies macht in der Regel den Einsatz weiterer Werkzeuge, wie z. B. CASE-Tools erforderlich.
- Derzeit ist der Quellentext, der mit visuellen Entwicklungsumgebungen erzeugt wird, in jedem Fall ineffizienter als ein nicht-visuell sorgfältig entwickelter Quellentext.

Der aus diesen Argumenten zu schließende Nachteil einer oder mehrerer visueller Entwicklungsumgebungen ist gegenüber dem vermeintlichen Nutzen für diese Systementwicklung nicht tragbar.

Dies schränkt aber z. B. die Verwendung von sogenannten *GUI-Buildern* für eigenständige Programme, die in ein neues System integriert werden sollen nicht ein. Zudem bietet Tcl/Tk eine Vielzahl von Integrationsmöglichkeiten und Schnittstellen zu anderen aktuellen Sprachen bzw. Sprachmodellen, wie z. B. Java, C++, CORBA, CGI und SQL.

Seit einiger Zeit wird von der Firma SUN an einer Integration von Tcl/Tk und Java gearbeitet, die die Verbindung von Tcl und Java Mini-Programmen (engl.: „applets“) vereinfacht und Tcl als Sprache für zukünftige Entwicklungen vorsieht.

Der Einsatz von *GUI-Buildern* ist jedoch trotz aller Überlegungen Geschmacksache und erhöht bei objektiver Betrachtung weder nachweislich die Geschwindigkeit und noch weniger die Portabilität der Entwicklung.

Hingegen sind einige Entwicklungsumgebungen modular aufgebaut, so daß Funktionen genutzt werden können, welche die Vielfältigkeit der Einsatzmöglichkeiten bei der Entwicklung erweitern.

Als Beispiel sei die hervorragende Entwicklungsumgebung TclPro der Firma Ajuba Solutions[11] (früher Scriptics Corporation[12]) genannt, die für Tcl wichtige Werkzeuge, wie Debugger, Syntaxprüfung, Compiler, Tcl Wrapper und zahlreiche Erweiterungen[13] integriert [Scr1999].

Seit der Übernahme durch Interwoven[14] im Jahr 2001 ist diese Entwicklungsumgebung auch frei verfügbar. Die Umgebung stellt eine Laufzeitverpackung (engl.: „wrapping“) mit weitergehenden Funktionalitäten bereit, z. B. dem Zugriff innerhalb der Verpackung (engl.: „file shadowing“) und außerhalb der Verpackung (engl.: „fall-through“) sowie die Erstellung von portablem Bytecode, der in einer erweiterten „Windowing Shell“ (`wish`) genutzt werden kann.

In den meisten Implementierungen, die eine Laufzeitverpackung ermöglichen, werden Archive mit Kompression erstellt, denen ein ausführbarer Teil vorangestellt wird, der die Nutzung des Inhalts zur Laufzeit ermöglicht [Nij2000].

Als Compiler und für die Portabilitäts- und Syntaxprüfung hat sich ebenso der Tcl Compiler[15] von ICEM[16] sehr bewährt [ICE1997] [RC1997].

Für die Einhaltung aller wichtigen Konventionen, einen möglichst einheitlichen Programmierstil sowie zur Vermeidung von Fehlern ist ein „Style Guide“ empfehlenswert [Joh1997b]. Bei größeren Applikationen können weiterführende konzeptionelle Aspekte des Entwurfs mit Tcl hilfreich sein [Fer1999].

4.5. Einsatz von Compilern

Für die Entwicklungen zu dieser Dissertation können verschiedene Compiler verwendet werden.

Für die meisten Änderungen und Erweiterungen ist kein Compiler notwendig, da viele Teile, z. B. die Oberfläche, interpretiert werden. Dennoch kann der Einsatz eines Compilers sinnvoll sein.

Gründe für den Einsatz von Compilern im Zusammenhang mit der Entwicklung komplexer Softwarekomponenten sind:

1. Vereinfachung des Supports.
2. Schutz von geistigem Eigentum.
3. Unterstützung von Lizensierungen.

[11] `http://www.ajubasolutions.com` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[12] `http://www.scriptics.com` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[13] `http://dev.scriptics.com` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[14] `http://www.interwoven.com` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[15] `http://icemcfd.com/tcl/ice.html` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[16] `http://icemcfd.com/tcl/` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

4. Modularisierung verschiedener Komponenten (z. B. Shell/Tcl/C).
5. Verpackung von Komponenten.
6. Erhöhung der Ausführungsgeschwindigkeit/ Performanz.
7. Erhöhung der Sicherheit auf Servern mit speziellen Aufgaben.

Diese Argumente sind insbesondere auch für die Entwicklungen von Bedeutung, die hier beschrieben werden.

Die Erhöhung der Performanz ist angesichts der Implementierung der Compiler noch von untergeordneter Bedeutung. Modularisierung, Verpackung und Support stehen im Vordergrund.

5. Wahl des BS

Linux has been considered like the biggest effort made in Informatics during the last years. In Europe and The United States, it is not only used at schools and colleges but also in universities and researching labs. A great number of enterprises and industries begin to install Linux in their informatics projects.
In a few words, Linux is the tool which permits to reduce the technological gap between the countries. Linux is a new way to make informatics, where the most important thing is the technical quality and people solidarity.

The UNESCO, 1998

5.1. GNU/Linux als Betriebssystem für die Entwicklung des Prototyps

5.1.1. Verwendung für das vorliegende Projekt

Für das vorliegende Projekt und die Entwicklung eines Prototyps wurde ein GNU/Linux System als Basissystem für *alle* notwendigen Planungen, Entwicklungen, Anwendungen und Tests verwendet.

Der Aufbau des Systems und die weite Verbreitung von umfangreichen GNU/Linux Distributionen schafft die Voraussetzung für das Verständnis, warum Software, die für Einzelaufgaben spezialisierte Werkzeuge einsetzt, eine effiziente und einfach zu handhabende Lösung für umfangreichere Probleme darstellt.

Folgende Gründe waren für die Wahl ausschlaggebend.

Bezüglich der Zukunftssicherheit:

1. Das System liegt offen und frei verfügbar vor und wird kontinuierlich weiterentwickelt.
2. Alle wichtigen Entwicklungswerkzeuge und Applikationen haben eine hohe Qualität und sind frei verfügbar.
3. Es stehen sehr mächtige freie graphische Oberflächen und Applikationen zur Verfügung.

Bezüglich technischer Entwicklungsaspekte:

1. Das Betriebssystem ist nicht zuletzt aufgrund des Entwicklungsmodells und der eingesetzten Verfahren sehr stabil und auf jede wichtige Hardware portiert.
2. Das System ist nach einem Schichtenmodell aufgebaut und ermöglicht einen Zugriff für alle denkbaren Entwicklungsaspekte.
3. Die Verwendung dynamischer Bibliotheken wird mit Versionsdifferenzierung unterstützt.
4. Alle wichtigen Netzwerkprotokolle werden abgedeckt.
5. Die minimalen Hardwareanforderungen sind gering. Bei Bedarf kann aber auch Hardware für spezielle Anforderungen eingesetzt werden.

Bezüglich des Einsatzes beim Endanwender:

1. Der Umfang an frei verfügbarer Systemsoftware und Anwendungen für fast alle Einsatzbereiche ist sehr hoch.
2. Alle wesentlichen Belange zur Anwendung, Nutzung und Erweiterung sind gut und frei zugänglich dokumentiert.
3. Der Verbreitungsgrad von umfangreichen Distributionen ist international bereits sehr hoch.
4. Heutige Distributionen enthalten alle wichtigen Werkzeuge und umfangreiche Anwendungen und unterstützen alle Funktionalitäten, die für komplexe Entwicklungen sowie tägliche Anwendung benötigt werden. Dies ist insbesondere wichtig für den unkomplizierten kombinierten Einsatz eines zu entwickelnden Prototyps mit vielfältigen spezialisierten Applikationen.

Der folgende kurze Exkurs soll diese Wahl durch zusätzliche Informationen abrunden und einen kompakten Überblick über das Umfeld freier Software und Linux vermitteln, da diese in Zukunft mit geringem Aufwand für Erweiterungen des Prototyps zum Einsatz kommen kann.

Neben dem, für das Verständnis der Planung und Entwicklung notwendigen Überblick über das X Window System (Abbildung 3.1 von Seite 25), soll damit auch der Bezug zum Aufbau des verwendeten Systems hergestellt werden.

Mit dem Stand dieser Arbeit ist GNU/Linux das Basissystem sowohl für die Entwicklung als auch für die Anwendung des entwickelten Prototyps.

5.1.2. Der Kernel

Linux ist eine komplette und freie Entwicklung des Kerns eines modernen Unix-Betriebssystems, der sowohl im Quellentext als auch in kompilierter Form verfügbar ist.

Das Copyright liegt bei Linus Benedict Torvalds und ist frei weitervertreibbar unter den Bedingungen der GNU-General-Public-License (GPL/LGPL) [Aut1999a]/[Aut1999b].

Damit ist Linux freie Software (Freeware) ohne Lizenzgebühren, aber weder Public Domain noch Shareware [Wol1999].

Alle Quellentexte sind frei verfügbar. Ursprünglich von seinem Schöpfer Torvalds 1991 als Alternative zum Betriebssystem Minix für Intel-PCs entwickelt, ist Linux mittlerweile auch auf die meisten existierenden Systeme portiert worden, bzw. in Entwicklung.

Im Jahr 1994 war die Kernel-Version 1.0 fertiggestellt. Damit wurde Linux seitdem als Betriebssystemkern für das GNU Projekt verwendet, da sich der grundlegend modulare Kernel GNU/Hurd[1] (HURD, HIRD) noch in der Entwicklung befindet.

Seit dieser Zeit wurde der Linux Kernel in rasantem Tempo um eine Fülle von Funktionalitäten erweitert [Han1999b] [Rai1999].

Die Kernel-Entwicklung findet zweigleisig statt. Varianten mit gerader zweiter Release-Nummer (z. B. 2.0.x) sind die Anwender-Kernel, bei denen der Schwerpunkt auf Stabilität liegt. Die zweite Version (z. B. 2.1.x) sind die Entwickler-Kernel, bei denen die Aktualität der unterstützten Hardware Vorrang hat, beziehungsweise in die neue Konzepte integriert werden.

Den Aufbau des Linux Kernels zeigt Abbildung 5.1 in einer sehr vereinfachten schematischen Darstellung.

Dabei besteht enge Verwandtschaft zum Unix Schichtenmodell (Abbildung 5.2).

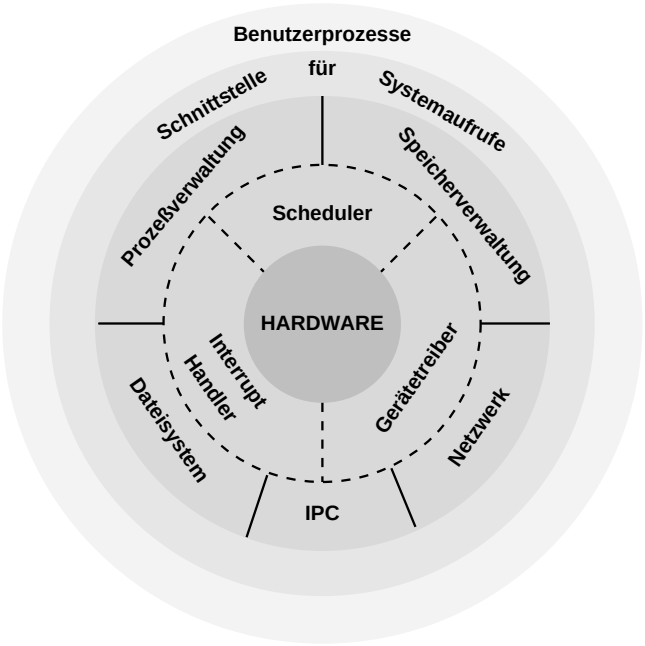

Abb. 5.1: Linux Kernel

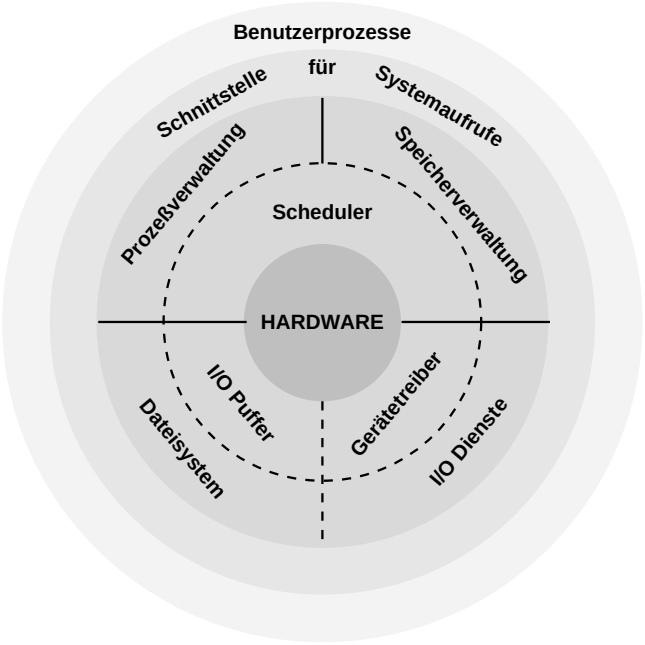

Abb. 5.2: Unix Schichtenmodell

Die aktuellen Entwicklungen gehen inzwischen weit über die angedeuteten Eigenschaften hinaus und schließen nicht nur weitere Plattformen ein (Kernel 2.4[2], Kernel call graph[3]).

Insbesondere zählen dazu auch eine weiter verbesserte Unterstützung großer Datenmengen, wie sie z. B.

[1] `http://www.gnu.org/software/hurd/hurd.html` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[2] `ftp://ftp.kernel.org/pub/linux/kernel/v2.4` ..... [V: k. A.] [Ä: k. A.] [Z: 30.03.2001]
[3] `http://lgp.linuxcare.com.au` ..... [V: k. A.] [Ä: k. A.] [Z: 30.03.2001]

bei der Verarbeitung umfangreicher Rasterdaten anfallen und einen effizienten Zugriff auf blockbasierte Geräte, z. B. für Datenbanken („*raw i/o devices*“) sowie DRI[4] Unterstützung zur Hardwarebeschleunigung von dreidimensionaler Graphik (OpenGL) im Kernel [Eng2001].

5.1.3. Werkzeuge und Anwendungen

Zum System gehören Freeware Werkzeuge, z. B. alle Werkzeuge nach den Unix de-facto Standard, XFree (X Window System), Compiler (GNU gcc C/C++) und hunderte weiterer Werkzeuge, Applikationen, Emulatoren und Softwarepakete.

Zu mehr als nur einem einfachen Betriebssystem werden Linux Distributionen durch tausende von Programmen und Programmpaketen, die fast jedes erdenkliche Einsatzgebiet abdecken.

Mit der zunehmenden Verbreitung nahm auch die Anzahl der für Linux verfügbaren Anwendungen zu. So steht Linux immer häufiger auf den Listen der unterstützten Betriebssysteme, auch bei renommierten Herstellern.

5.1.4. Eigenschaften

Linux hat folgende Eigenschaften zur Unterstützung relevanter Verfahren:

- Unterstützung für ein echtes Multitasking/ Multithreading und echten Mehrbenutzerbetrieb (ohne prinzipielle Begrenzung auf eine Benutzeranzahl).
- 64-Bit Unterstützung, ebenso volle 64-Bit und 32-Bit Unterstützung virtuellen Speichers.
- Multi-Plattform (x86, DEC Alpha, MIPS, PowerPC, Sparc, S/390, Mac, Acorn, 68K etc.) Quellentext Kompatibilität.
- SMP (Intel und Sun CPU’s)
- Läuft im 386-Protected-Mode mit Speicherschutzmechanismen für die einzelnen Prozesse. Damit kann ein einzelner Prozeß das System nicht zum Absturz bringen.
- Fortgeschrittene Speicherausnutzung durch *Paging* („*shared copy-on-write pages*“ etc.), bei Bedarf ladbaren Programmen (engl.: „load on demand“), gemeinsamen Speicherraum für Programme und Festplatten Cache, und dynamisch gelinkte Bibliotheken (engl.: „dynamically linked shared libraries“). Damit können sich beispielsweise mehrere Prozesse denselben Speicher teilen. Diese Funktionen führen zu einer höheren Geschwindigkeit und geringerem Speicherbedarf. Außerdem können sehr große effiziente virtuelle Speicher verwendet werden.
- Freie Speicherverfügbarkeit für Programme und Zwischenspeicher (engl.: „cache“): Freier Speicher wird solange als Zwischenspeicher genutzt, bis ein Anwendungsprogramm mehr Speicher anfordert, in diesem Fall wird der zuvor durch den Zwischenspeicher belegte Speicher sofort wieder freigegeben.
- Funktionen für Speicherabzüge (engl.: „dump“) zum Beseitigen von Fehlern (engl.: „debugging“).
- Vollständige Quellentext-Kompatibilität zu POSIX, System V und BSD/FreeBSD[5], System V und BSD Erweiterungen.
- Vollständige Kompatibilität zu IPv4 und IPv6.
- Volle Verfügbarkeit des System-Kernels, aller Treiber, Entwickler-Programme und Endbenutzer-Programme.
- Paralleles Arbeiten auf mehreren virtuellen Konsolen ist möglich. Pseudoterminals (pty’s).
- XFree86, graphische Oberfläche und Schnittstelle.
- Dynamische Bibliotheken (engl.: „shared libraries“) und statische Bibliotheken.
- Unterstützung zahlreicher nationaler und anwenderdefinierter Tastaturen.
- Optimale Unterstützung verschiedener moderner Prozessoren. Für verschiedene Prozessoren verfügt der Kernel auch über eine Coprozessor-Emulation (z. B. 387-Emulation im System-Kernel).
- Alle verbreiteten Dateisysteme werden unterstützt, Der Zugriff auf MS-DOS (FAT, VFAT, FAT32), Minix-1, Xenix, System V und OS/2-Partitionen und viele andere Dateisysteme und Erweiterungen (RockRidge, Joliet etc.) ist implizit möglich. Linux kann sogar auf DOS-Partitionen installiert werden.
- CD-ROM Unterstützung (SCSI/IDE).

[4] `http://dri.sourceforge.net` ..... [V: k. A.] [Ä: k. A.] [Z: 30.03.2001]
[5] `http://www.freebsd.org` ..... [V: 1997] [Ä: k. A.] [Z: 25.02.2001]

- Hochperformante Netzwerkunterstützung (engl.: „high performance networking“) (NFS/SMB/ IPX/Appletalk networking), schnellste Stapelverarbeitung unter Unix-Vergleichstests.
- Festplatten-Management (engl.: „disk management“), z. B. Striping, Mirroring, RAID 0,1,5 usw.
- Volle Netzwerk-Unterstützung. Einschließlich: SSH, TCP/IP (Internet RFC 1006), FTP, (Internet RFC 959), telnet, SLIP, PPP, PLIP etc. Es ist bereits jedes weltweit existierende offene Protokoll nach Linux portiert worden.
- Peripherie-Unterstützung (SCSI, PCI, PCMIA, EIDE, USB, IEEE-1394 etc.).

Seit der Veröffentlichung von Linux 1.0 am 14.3.1994 wuchs die Zahl der Linux-Nutzer, nach seriösen Schätzungen[6], von einigen wenigen, auf derzeit (Stand: Mitte 2001) 18 Millionen.

5.1.5. Hardwareanforderungen

Die *minimalen* Hardwareanforderungen sind sehr gering:

Linux läuft auf Rechnern die mindestens einen Intel 386 Prozessor und einen ISA-, EISA-, Local- oder PCIBus besitzen. Die kleinste vorstellbare System-Konfiguration, auf der Linux arbeitet, ist die folgende: Ein 386SX/16, 2 MB RAM, 1.44 MB or 1.2 MB Diskettenlaufwerk, eine der unterstützten Graphikkarten und der Rest, der zu einem System gehört (Tastatur, Monitor etc.).

Um mit einem Linux-System vernünftig zu arbeiten wird ein Rechner mit einem schnellen Prozessor (ab 386DX/33 aufwärts) und mindestens 4 MB RAM benötigt. Wenn auch X11 auf dem Rechner eingesetzt werden soll, dann sollten mindestens 8 MB RAM zur Verfügung stehen.

Der Speicherbedarf auf der Festplatte hängt davon ab, welche Software installiert werden soll. Das Minimum liegt hier bei ungefähr 12 MB (17 MB mit einem Minimal-X).

Bei einer Installation ohne X11 ist mit ca. 60 MB benötigtem Speicherplatz zu rechnen. Wird ein komplettes System mit X installiert, sind etwa 80–100 MB nötig.

In allen Fällen ist noch der Speicherplatz zum Arbeiten zuzurechnen. Nach oben hin gibt es kaum eine Grenze.

Aufgrund der Entwicklung sehr umfangreicher Software in den letzten Jahren, wachsen mittlere Installationen in der Regel auf die Größe von mehreren GB.

Derzeit unterstützte Intel-kompatible Hardware:

- CPU: Alle modernen 80x86 Prozessoren ab dem 80386 (der 80286 wird nicht unterstützt), RISC Prozessoren usw., aber auch zahlreiche portable und Mikrosysteme.
- Architektur: ISA, EISA, Local-Bus, PCI usw.
- RAM: Theoretisch derzeit auch auf Intel Plattformen bis zu 64 GB.
- Festplattensysteme bis in den Terabyte-Bereich.
- Externe Speichermedien: IDE/EIDE-Controller, XT-Controller, SCSI, QIC-Streamer, MO-Laufwerke, ZIP/JAZ Laufwerke etc.
- Graphikkarten: VGA, EGA, CGA und Hercules arbeiten im Text-Modus. Für X11 wird mindestens eine EGA-Karte benötigt. Für VGA gilt: Da Linux den XFree86 X-Server benutzt, wird u. U. ein spezieller X-Server benötigt. (Mittlerweile sind aber Treiber für die meisten VGA-Karten vorhanden.)
- Weitere Hardware: Fast alle seriellen Mäuse, viele Bus-Mäuse; Digitalisiergeräte; Soundkarten: AdLib, SoundBlaster, ProAudio Spectrum 16 und andere; AST Fourport Karten (mit 4 seriellen Ports), diverse Modelle der *Boca serial boards*, die Usenet Serial Card III und viele andere . . .

Da dies nur ein winziger Teil der unterstützten Hardware ist und ständig neue Hardware unterstützt wird, sei auch hier wieder auf das (vollständigere) Hardware-Compatibility-Howto verwiesen.

Für Entwicklungen und Tests, wie den hier beschriebenen, ist eine gute Ausstattung des Arbeitsspeichers von höherer Priorität, als die Rechenleistung des Systems.

Dies trifft insbesondere zu, da zum einen die Einzelteile einer Komponente insbesondere während der Testphase nicht zwingend kompiliert werden müssen und daher keine jedesmal wiederkehrenden Kompilierzyklen benötigen.

Zum anderen sollen für anspruchsvollere Tests größere Einzeldaten auch ohne spezielles Speichermanagement verwendet werden können.

[6] `http://counter.li.org` ..... [V: 1994] [Ä: k. A.] [Z: 01.07.2001]

Für den Endanwender mag je nach Einsatzgebiet die Rechenleistung oder die Graphikleistung im Vordergrund stehen.

Da die eingesetzte Technik die Verwendung vielfältiger Aspekte ermöglicht, beispielsweise multimedialen Einsatz, sollte bei Entwicklungen auch auf dieser Ebene auf Hardware und Software geachtet werden, die möglichst plattformunabhängige Verwendung ermöglicht.

5.1.6. Verfügbare und portierte Software

Die folgende Zusammenstellung führt einige Beispiele für Software auf, die sich vom Anwender und Entwickler einfach im Prototyp anbinden oder zweckmäßig für die Entwicklung einsetzen lassen. Eine Anbindung kann z. B. über Skripten erfolgen.

- Sprachen: C/C++, Perl, Tcl/Tk, Python, Modula, Lisp, Fortran, SML, ...
- Entwickler-Werkzeuge: gcc, `gdb`, `make`, bison, `flex`, `perl`, rcs, cvs, gprof, ...
- Editoren: GNU Emacs, Lucid Emacs, MicroEmacs, jove, epoch, elvis, joe, pico, jed, vim.
- Shells: Bash (Posix sh-kompatibel), zsh (inklusive *ksh compatiblity mode*), tcsh, csh, rc, ash (nahezu sh-kompatibel) und viele andere.
- Grundlegende Unix Kommandos: `ls`, `tr`, `sed`, `awk` und so weiter ...
- Graphische Oberflächen: X11R6 (XFree86), WINE (Windows-Emulator) ...
- Telekommunikation: Taylor (BNU-kompatibel) UUCP, kermit, szrz, minicom, pcomm, xcomm, term/slap und Seyon.
- News, Mail, WWW: C-news, trn, nn, tin, pine, smail, elm, mh, Xrn, Netscape, Konqueror.
- Informationssysteme: X-Mosaic, Gopher.
- Graphik, Textverarbeitung und Textsatz: Gimp, netpbm, TEX/LATEX, groff, doc, LyX, kLyx, koffice, StarOffice.
- Kommerzielle Software: Maple, Mathematica, Word Perfect, ...

5.2. Die Philosophie freier Software

5.2.1. Freie Verfügbarkeit

Seit den Anfängen von Linux sind alle Quellen frei verfügbar. Die freie Verfügbarkeit hat dazu geführt, daß immer mehr Programmierer die Entwicklung zu einem leistungsfähigen und stabilen Betriebssystem vorangetrieben haben. Hierzu leistet das Internet einen entscheidenden Beitrag, da die auf der ganzen Welt verteilten Entwickler ihre Kommunikation über dieses Medium abwickeln (z. B. die Usenet Linux Gruppen zu den Themen System[7], Applikationen[8], Hardware[9], Netzwerk[10], X11[11], Antworten[12] u. v. a.). Linux eröffnet damit im Vergleich zu anderen Systemen unzählbare Möglichkeiten, nicht zuletzt aufgrund seiner Entwicklungsphilosophie ([Ray1999]).

5.2.2. Entwicklungsmodell

Das Entwicklungsmodell kann folgendermaßen zusammengefaßt werden:

Linux wird im Gegensatz zu vieler anderer Software unter einem offenen und verteilten Modell entwickelt. Das bedeutet z. B., daß die aktuelle Entwicklungsversion immer öffentlich zur Verfügung steht. Neben der effizienten Weiterentwicklung ist so auch eine Beseitigung von Fehlerquellen z. B. in einem neuen System-Kernel dank der weltweiten Vernetzung der Entwickler meist schon innerhalb weniger Stunden möglich.

Mit seinem Entwicklungsmodell leistet freie Software einen wichtigen Beitrag zur gesellschaftlichen Entwicklung [Gre1999].

Im Sinne einer positiven und interessenunabhängigen Entwicklung der zukünftigen Informationslandschaft ist eine weitreichende Ablehnung der Patentierbarkeit[13] von Software wünschenswert.

[7] `news:comp.os.linux.development.system` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[8] `news:comp.os.linux.development.apps` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[9] `news:comp.os.linux.hardware` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[10] `news:comp.os.linux.networking` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[11] `news:comp.os.linux.x` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[12] `news:comp.os.linux.answers` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[13] `http://petition.eurolinux.org/index.html` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

5.2.3. Dokumentation

Es liegen tausende von Dokumentationen für alle Belange kostenfrei als Online Dokumentation vor (LDP[14], FAQ, HOWTO, . . .). Die meisten Distributionen enthalten einen Teil der wichtigsten Dokumentationen, neben Hinweisen zu zahlreichen Erscheinungen in gedruckter Form, als Bücher und Magazine.

5.2.4. GNU Namensgebung

Da inzwischen eine Vielzahl von Linux Distributionen erhältlich ist, die zum Teil mit verschiedensten speziellen Funktionen und Erweiterungen ausgestattet sind, ist es notwendig sicherzustellen, daß hier nicht irgendein System unter Linux zu verstehen ist, sondern strenggenommen nur ein System, welches keine essentiellen proprietären Ergänzungen enthält. Ein solches System wird gemeinhin als GNU/Linux bezeichnet, nicht zuletzt, um der Verwässerung des Marktes entgegenzuwirken. Im folgenden ist überall dort, wo kurz Linux genannt wird, ein GNU/Linux gemeint.

[14] `http://sunsite.unc.edu/LDP` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

6. Ausrichtung der Realisierungen zu dieser Dissertation

6.1. Entwicklungsplattform für die neuen Komponenten

6.1.1. Die konzipierte Entwicklungsebene

Als Entwicklungsebene werden die konzeptionellen Teile des Prototyps bezeichnet, für die explizit Entwicklungen entweder durch die Entwickler von Kernfunktionen oder an der endgültigen Komponente auch durch versierte Anwender, z. B. zur Handhabung spezieller Effekte für einen Datensatz, vorgenommen werden. Zu diesen Teilen zählen:

1. Benutzeroberfläche (GUI).
2. Shell.
3. Funktionen hinter der graphischen Benutzeroberfläche (engl.: „backend“).
4. Nutzung von Skriptsprachen („Skripting“).

Das Zusammenspiel der wesentlichen Funktionen der Entwicklungsplattform und der Umgebung, bestehend aus Betriebssystem, Programmiersprachen und Schnittstellen sowie Applikationen, kann vereinfacht durch ein Plattform-Diagramm (Abbildung 6.1 auf Seite 46) dargestellt werden.

Diese Übersicht ist in der Vertikalen in zwei größere Teile geteilt. Links finden sich die primär lokal ausgerichteten Umgebungen, rechts die überwiegend WWW-basierten Umgebungen.

Der Übergang ist aufgrund der schnellen Entwicklungen in diesem Sektor nicht festgefügt. Dies zeigen z. B. zahlreiche netzbasierte Applikationen.

Die oberste Tcl-basierte Schicht ist hellgrau dargestellt. Darunter folgen in hellerem Grau die Umgebungen bzw. Interpreter, innerhalb derer die aufliegende Schicht genutzt werden kann. Perl, VTK und Java haben dabei eine Sonderstellung, weil sie nur Teile nutzbar machen dafür aber auch einfach nur per Aufruf oder Schnittstelle genutzt werden können.

Insbesondere die lokal ausgerichteten Umgebungen bieten sich derzeit zur Integration höherer Programmiersprachen, zum Aufruf von Applikationen und Shell-Funktionen an. Theoretisch ist dies auch noch für Java gültig, kann aber für die hier folgenden Entwicklungen vernachlässigt werden.

Die unterste Schicht ist dunkelgrau hinterlegt. Hier können beispielsweise Systemfunktionen oder Kernel-Module genutzt oder entwickelt werden.

Dieser Ansatz mit der zentralen Nutzung einer vollständigen Skriptsprache eröffnet auch für den nicht spezialisierten Anwender Möglichkeiten, Applikationen zu nutzen, wie sie bei einer flachen Entwicklung mit höheren Programmiersprachen, z. B. C/C++/Java, nicht in dieser Form intuitiv realisierbar wären.

Wie bei vergleichbaren Entwicklungen anderer Softwareprojekte, würde dies einen vielfach erhöhten Aufwand bedeuten, selbst für eine näherungsweise Implementierung [Lin2000].

Als anwendungsnahes Beispiel aus dem Plattform-Diagramm (Abbildung 6.1) kann folgendes Szenario dienen.

Ein Tclet kann durch Plugin-Unterstützung in einem Browser oder in einem Browser mit nativer Unterstützung genutzt werden.

Es kann aber auch in einer Java-basierten Umgebung laufen, die wiederum über einen Adapter in einem Browser oder auch auf einem Java-basierten Betriebssystem laufen kann.

Ein solches Tclet kann, ausgebaut zur kompletten Tcl/Tk Anwendung, in einer Windowing Shell (`wish`) ausgeführt werden, beispielsweise als Tcl (`.tcl`), Tcl Bytecode (`.tbc`) oder „Portable Tcl“ (`.ptcl`).

Es kann durch die ausführende Umgebung Hochsprachen nutzen, d. h. beispielsweise Bibliotheken, aber auch Applikationen und Skripten auf Shell-Ebene aufrufen.

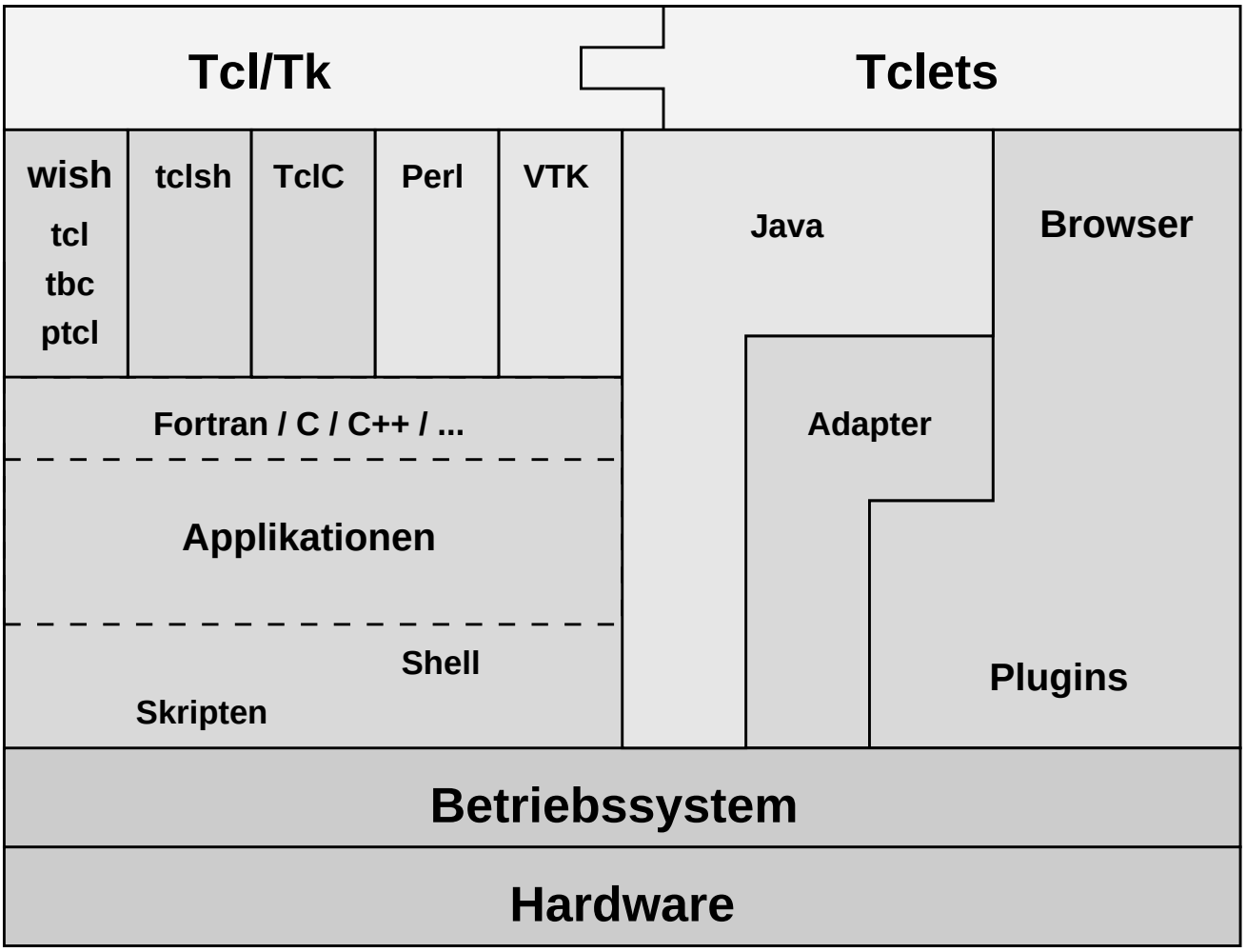

Abb. 6.1: Struktur der entworfenen Entwicklungsplattform für die neuen Komponenten

6.1.2. Die konzipierte Anwendungsebene

Durch den Aufbau der Entwicklungsebene lassen sich auf Anwendungsebene verhältnismäßig leicht verschiedene Anwendungen erstellen oder modifizieren. Zu den Kategorien solcher Anwendungen zählen insbesondere folgende:

- Eine eigentliche Applikation, bzw. Modifizierungen der Applikation.
- Applikationsmodule, die zur Erweiterung der Applikation allgemein oder nur bei Bedarf hinzugeladen werden können.
- Skriptmodule, die durch Skripting auf beliebige Systemprogramme oder externe Applikation zugreifen können.

Weitere und tiefergehende Möglichkeiten bestehen durch die Erweiterung des verwendeten Interpreters, in der Regel auf Basis von C oder C++.

6.1.3. Übergeordnete Verbindung von Komponenten am Beispiel einer exemplarischen Oberfläche: GISIG

Bei den anfänglichen Entwicklungen zu dieser Dissertation wurde zunächst eine Komponente (GISIG) entwickelt, um Möglichkeiten zu untersuchen, verschiedene externe Applikationen miteinander zu verbinden.

Diese Komponente kann z. B. in der Shell einer fremden GIS-Anwendung (z. B. GRASS) laufen, um Arbeitsvorgänge für diese Anwendung und weitere externe Anwendungen zu automatisieren und mit einer benutzerdefinierten graphischen Oberfläche zu versehen.

Weitere Komponenten können mit dieser Komponente auf einfache Weise kommunizieren und erhalten so einen weiteren Zugriff auf externe Anwendungen.

Diese übergeordnete Oberfläche kann damit als Schnittstelle zwischen einem GIS und separaten anwendungsdefinierten Benutzeroberflächen verschiedener Komponenten angesehen werden.

Aus diesem Grund kann es sinnvoll sein, verschiedene übergeordnete Oberflächen zu haben, beispielsweise für spezielle externe Applikationen, für spezifische Anwendungsfälle oder bestimmte Nutzergruppen.

Der weitere Ausbau ist mit den nötigen Detailkenntnissen über die eingesetzten Applikationen sehr leicht durchführbar, kann aber für die Unterstützung eines größeren Funktionsumfangs und aufgrund von versionsbedingten Änderungen der Bedienung fremder Applikationen langfristig sehr zeitaufwendig werden.

Aufgrund der offenen Entwicklungsumgebung ist prinzipiell die *Integration* von Programmen und Programmelementen jeder Programmiersprache sowie die *Verwendung* jeder Programmiersprache und Klassenbibliothek für die Fortentwicklung möglich.

Die Entwicklung war lediglich als erster Test der Handhabbarkeit der Entwicklungswerkzeuge für die Umsetzung des Konzepts nützlich.

Dieser Teil hat für das vorliegende Projekt nur prinzipiellen Charakter, daher erfolgt keine weiterführende Beschreibung.

Die prinzipielle Möglichkeit, Komponenten ihrerseits wieder durch eine Oberfläche zusammenzufügen, ist vor allem für den Endanwender von Bedeutung.

Eine Weiterentwicklung wird daher erst wichtig, wenn weitere Komponenten hinzukommen und z. B. bestimmte Werkzeuge oder externe Fremdanwendungen optional integriert und zu einem Ganzen verbunden werden sollen.

Die wesentliche Ausrichtung dieser Dissertation ist hingegen, zugunsten der Details der Ereignissteuerung und Visualisierung, vielmehr auf die Kernkomponente konzentriert.

6.2. Schwierigkeiten mit Bilddaten im WWW

Aufgrund der realen Rahmenbedingungen steht in Netzen mit teilweise hoher, wachsender Auslastung der konventionellen Verwendung umfangreicherer Bilddaten neben weiteren Problemen ein Performanzproblem entgegen. Oft sollen zudem Bezüge auf kleine Bereiche eines Bildes beschränkt werden, z. B. um Verweise daran zu knüpfen. Zusatzinformationen, sogenannte *imagemaps*, ermöglichen Verweise ohne Zerstückelung des Bildes. Bei einem Kacheln hingegen wird ein Bild in Stücke zerlegt, die auch separat verwendet werden können. Bei der Umwandlung von Quellentext-Daten in reines HTML sollte anwendungsbedingt die jeweils geeignetste Form gewählt werden.

6.2.1. Beispiel für Imagemap

Folgender Quellentext zeigt ein einfaches Beispiel für die Verwendung einer Imagemap (Abbildung 6.2):

```
<html>
<head><title>
Beispiel: imagemap
</title></head>
<body>
  imagemap auf image:

  <p>
  <img src="schvier.gif"
  width="200" height="70"
  align="right"
  usemap="#dialog">

  <p>
  <map name="dialog">
  <area shape="rect"
    alt="Bild nicht erreichbar!"
  coords="51,27,84,115"
    href="knoten02.html">
  <area shape="default" nohref>
  </map>
</body>
</html>
```

Abb. 6.2: Quellentextbeispiel: Imagemap

Imagemaps haben ihren Ursprung in der oberen linken Ecke der Graphik. Eine definierte „map“ kann beliebig viele Teilflächen (engl.: „area“) mit eigenen Koordinaten enthalten. Zudem lassen sich die „maps“ in separate Dateien auslagern.

Die vorhandene Funktionalität genügt z. B. für die einfache Handhabung von Karten.

Allerdings ist derzeit der Zugriff der vorhandenen WWW-Klienten Software (engl.: „browser“) auch auf schnelle Festplattensysteme insbesondere bei großen Datensätzen noch etwas langsam.

Dieser Nachteil kann durch eine mosaikartige Aufteilung und Trennung vorskalierter Daten meist zufriedenstellend ausgeglichen werden.

6.2.2. Beispiel für Kacheln

Die mosaikartige Teilung des Bildmaterials, ein sogenanntes *Kacheln* (engl.: „tiling“), hat den impliziten Vorteil, daß gewünschte Ausschnitte gezielt schneller zu bearbeiten und darzustellen sind.

Über HTML ist die Visualisierung derart gekachelten Bildmaterials z. B. folgendermaßen möglich (Abbildung 6.3):

```
<html
<head><title>
Tiled map created by GISIG gisht
</title></head>
<body>
<table border=0 cellspacing=0
       cellpading=0>
<tbody>
<tr><td nowrap>
<a href="schvcut-50-0.html">
<img src="schvcut-50-0.gif"></a>
<a href="schvcut-81-0.html">
<img src="schvcut-81-0.gif"></a>
<a href="schvcut-112-0.html">
<img src="schvcut-112-0.gif"></a>
</td></tr><tr><td nowrap>
<a href="schvcut-50-30.html">
<img src="schvcut-50-30.gif"></a>
<a href="schvcut-81-30.html">
<img src="schvcut-81-30.gif"></a>
<a href="schvcut-112-30.html">
<img src="schvcut-112-30.gif"></a>
</td></tr><tr><td nowrap>
</td></tr>
</tbody>
</table>
</body>
</html>
```

Abb. 6.3: Quellentextbeispiel: Kacheln

6.2.3. Beispiel für Einbetten in HTML

Die meisten HTML-fähigen WWW-Klienten erlauben die Verwendung von Erweiterungen, sogenannten *Plugins* („plug-in“). Plugins wurden zuerst von der Firma Netscape unterstützt. Diese Technik wird inzwischen für verschiedene Klienten eingesetzt. Für Tcl/Tk existiert beispielsweise ein Plugin, das die Verwendung derartiger Skripten innerhalb des Klienten ermöglicht. Aus Sicherheitsgründen müssen die verwendeten Skripten ohne spezielle Vorkehrungen einigen Anforderungen entsprechen. Beispielsweise dürfen diese Skripten keine externen Programme aufrufen.

Der Inhalt eines minimalen Skripts ist z. B. folgender Abbildung 6.4 zu entnehmen:

```
button .b -text "Hello, world!"
pack .b
```

Abb. 6.4: Quellentextbeispiel: Minimales Skript

Das Einbetten von Skripten (engl.: „embedding“) in HTML kann z. B. wie folgt geschehen (Abbildung 6.5):

```
<html>
<embed src="testdatei.tcl"
       width=140 height=30>
</html>
```

Abb. 6.5: Quellentextbeispiel: Einbetten von Skripten

6.2.4. Schnittstelle für Bearbeitung auf Server-Seite

Eine Schnittstelle (CGI) erlaubt Interaktivität zwischen einem Klienten und einem entfernten Server über ein Netz. Im folgenden Beispiel wird das HyperText Transfer Protocol (HTTP, Internet RFC 1945) verwendet, das ein weit verbreitetes Protokoll für externe „Gateway“ Programme ist, um mit Informations-Servern zu kommunizieren. Gateway Programme oder auch kurz Gateways sind Programme, die Anfragen verarbeiten und spezielle Dokumente dazu generieren. Das HTTP Protokoll adressiert Dateien über sogenannte *Uniform Resource Locators* (URL, Internet RFC 1738).

Der Kontext der einzelnen Komponenten läßt sich durch ein kleines Schema (Abbildung 6.6 auf Seite 49) veranschaulichen.

1. Der Server übermittelt das betreffende Dokument über die Kommunikationsverbindung.

2. Der WWW-Klient (engl.: „browser“) stellt das Dokument dar.

3. Der Anwender reagiert mit Bearbeitung des Dokuments und Übermittlung (engl.: „submit“).

4. Die entsprechenden Daten (z. B. FORM Name/Werte, bzw. engl.: „name/value“ Paare) werden an den Server zurückübertragen.

5. Der Server startet die spezifischen Prozesse, die im Vorspann (engl.: „header“) angegeben sind (und weist der FORM Name/Werte-Paare zu).

6. Es werden die notwendigen Schritte ausgeführt und ein formatiertes HTML-Dokument generiert.

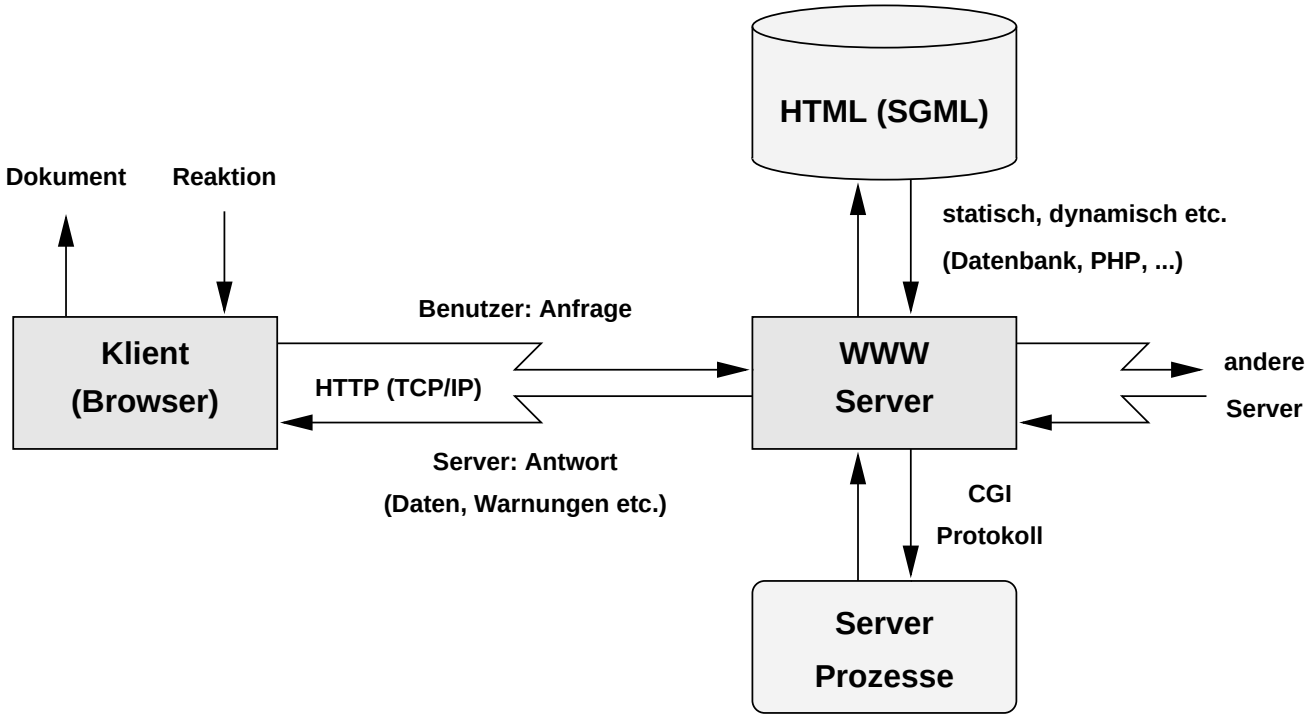

Abb. 6.6: Kontext der Klienten-Server Komponenten

Dieses Dokument kann selbst Strukturen beinhalten, die zu weiterer Kommunikation auffordern.

7. Das HTML-Dokument wird als Antwort an den WWW-Klienten zurückübertragen und dargestellt.

Das folgende kleine HTML-Dokument ist statisch und wird vom Server geliefert (Abbildung 6.7).

```
<html>
<a href="hello.cgi">hello.cgi</a>
</html>
```

Abb. 6.7: Quellentextbeispiel: HTML und CGI

Das assoziierte CGI Programm `hello.cgi`, ein minimales Perl Programm, wird bei Auslösung der Referenz in Echtzeit ausgeführt und kann aus diese Weise verwendet werden, um dynamische Informationen zu verarbeiten (Abbildung 6.8).

Im Kopf der übermittelten Daten können außer der erforderlichen Angabe `Content-type:` noch verschiedene weitere Angaben stehen.

```
#!/usr/bin/perl
$t      = "Hello!";
print   <<EOT;
Content-type: text/html

<title> $t </title>
<h1>    $t </h1>
EOT
```

Abb. 6.8: Quellentextbeispiel: CGI Programm

Mit Hilfe derartiger Programme kann der Server daher Informationen liefern, die ansonsten vom Klienten (engl.: „client“), z. B. einer Datenbank, nicht lesbar wären. Diese Programme können daher als vielseitiger Vermittler zwischen Klient und Server verwendet werden.

Eine sehr verbreitete Anwendung ist der Einsatz für die Handhabung von FORM Requests für HTTP. CGI ist eine Konvention zur Integration von Gateway Programmen, die unter den Herstellern von HTTP Servern vereinbart wurde.

Als Programmiersprache für CGI Programme kommt jede Sprache in Frage, die ein ausführbares Programm liefert. Beispiele sind daher z. B. Perl, Tcl, Shells, C/C++, Python, Fortran. Erstere können direkt interpretiert werden, letztere müssen erst kompiliert werden. Beide Methoden haben in bestimmten Fällen

Vorteile. Für Perl und Tcl existieren zahlreiche Werkzeuge zur CGI/Web-Programmierung [Ivl1997].

Ein Beispiel für eine Anwendung im Bereich Kartenserver über das Internet ist der MapServer [Koo2000], der als freie Software im Rahmen des freeGIS Projekts [Rei2000] erst seit kurzer Zeit entwickelt wird, trotzdem aber bereits für vielfältige Aufgaben zum Einsatz kommen kann.

Die Basisfunktionen des MapServers sind Navigation (Vergrößern, Verkleinern, Verschieben) und Abfragen (Punkte, Regionen, Felder, Werte).

Der Kern des MapServers ist ein CGI-basiertes Programm.

Die Erstellung einer Anwendung unter Nutzung des MapServers umfaßt die Erstellung von Dateien zur Definition von Eigenschaften und Darstellungen und für die Vorlagen der Benutzerschnittstelle. Eine reine Nutzung der Möglichkeiten in diesem Bereich durch den Anwender erfordert *immer*, daß die betreffende Anwendung bereits erstellt wurde. Für viele Anwender sind daher erst spezielle Anwendungsimplementierungen interessant, die auf Basis eines MapServers entwickelt wurden.

Die Anwendung wird über die Definition von Objekten und deren Eigenschaften konfiguriert.

Die Funktionen der so erstellten Anwendungen können über Skriptsprachen, wie Perl und Tcl/Tk, genutzt werden.

6.3. Geokognostik

6.3.1. Kognitive Ansätze

Die Integration kognitiver und geometrischer Ansätze führt zum Begriff der Geokognostik [Edw1996] (ICA Dokumente[1]).

Modelle in der Geokognostik müssen vielfach höhere Strukturen haben, als rein geometrische Modelle. In vielen Fällen kann dies auch zu kollageähnlichen thematischen Überlagerungen (engl.: „overlays“) für verschiedene Abbildungen (engl.: „views“) dieser höheren Strukturen führen [Tve1993].

Aus Sicht der Geokognostik ist Raum und Zeit untrennbar.

Datenqualität ist definiert als „Eignung für einen bestimmten Verwendungszweck“ (engl.: „fit for use“) [Goo1989].

Zahlreiche Applikationen zwingen den Benutzer, Übergangszonen durch scharfe Grenzlinien zu kennzeichnen. In allen Fällen, in denen die Antwort heißt „*Es hängt von der Anwendung ab . . .*“, muß die Lösung eigentlich in der kognitiven Welt (engl.: „cognitive domain“) gesucht werden.

Eine übergreifende Theorie, die geometrische Eigenschaften und kognitive Aspekte umfaßt, muß folgendes beinhalten.

1. Fähigkeiten zur Vereinigung geometrischer und kognitiver Aspekte mit geometrischen und topologischen Ansätzen für den Raum sowie kulturelle, linguistische und soziale Modelle zur räumlichen Beschreibung.

2. Theoretische Struktur zur Verbindung von räumlichen Modellen und Datenstrukturen und der Eingrenzung von Datenstrukturen.

3. Kompakte Theorie für lokale Statistik. Beziehungen zur globalen Statistik.

4. Fehlertheorie und Theorie zur Datenqualität.

5. Begriffe zum Verständnis der Raum-Zeit Beziehungen.

6. Verbindungen zur Interaktivität der Mensch/ Computer Schnittstelle.

6.3.2. Geokognitive Räume

Der kognitive Raum wird in zwei Arten von Räumen strukturiert, den direkt manipulierbaren Raum und den Großraum (engl.: „large-scale space“), der auch als geokognitiver Raum bezeichnet wird [Mon1993].

Geokognitive Räume sind nichtlineare Räume, die aus mehreren sogenannten *Views* aufgebaut sind. Diese Views werden durch reale oder virtuelle menschliche Trajektorien verbunden. Geokognitive Räume können nicht direkt abgebildet werden. Sie können aber transformiert, projiziert oder teilweise im lokalen geometrischen Raum dargestellt werden.

Views sind Sammlungen von Raum-Zeit Ereignissen, die ebenfalls nur teilweise im lokalen Raum dargestellt werden können.

Trajektorien sind in diesem Zusammenhang geordnete Folgen von Raum-Zeit Ereignissen, die mit einem Empfänger der Wahrnehmungen assoziiert sind, der als zeitlich konsistent angenommen wird.

Objekte sind Konzepte, die aus multiplen Views mit inkrementellen Änderungen auf kurzen Trajektorien abgeleitet werden.

[1] `http://www.geo.unizh.ch/ICA/Documents/Workshop97/paper_titles.html` [V: k. A.] [Ä: k. A.] [Z: 30.01.2001]

Über die reale Welt wird keinerlei Aussage gemacht. Alle unsere Beschäftigungen beziehen sich auf die Wahrnehmungen der realen Welt, nicht auf die reale Welt selbst. Innerhalb des geokognitiven Paradigmas muß der im Maßstab große Raum notwendigerweise kognitiv sein.

Der lokale Raum wird auch als geometrischer Raum bezeichnet. Der lokale Raum ist flach, euklidisch und kartesisch.

Ein GIS kann als kognitives Modell des Raumes dienen, da verschiedene Repräsentationen der gleichen Daten möglich sind und das System fehlertolerant ist [Hir1994].

Vektordaten simulieren Trajektorien, mißachten aber die zeitlichen Bezüge der Trajektorien. Sie können daher dort verwendet werden, wo Zeitabhängigkeiten nicht erwünscht sind und wo der kognitive Inhalt nicht von Bedeutung ist.

6.4. Relationen der Entwicklungen zu dieser Dissertation zu ausgewählten GIS-Komponenten

6.4.1. Vectaport

Bereits Ende 1998 ist im Rahmen dieser Arbeit die Firma Vectaport Inc.[2] in Redwood City (Kalifornien) kontaktiert worden. Vectaport entwickelt freie offene Software auf einem technisch hochwertigen Niveau, darunter verschiedene Komponenten für Darstellung, Handhabung und Bearbeitung von geographischen Daten, darunter auch Klienten- und Server-Anwendungen für 3D-Tracking. Keine dieser Applikationen verfügte über eine Kopplung von Ereignissen an externe Komponenten.

Beispielhaft wurde der Firma Vectaport ein einfaches Modell vorgeschlagen, um die bereits im Quellentext vorhandene Identifizierung von Objekten ohne allzu großen Aufwand durch eine *Schnittstelle zu externen Komponenten* zu erweitern.

Vectaport implementierte die Anregungen im Anschluß bereits im Januar 1999.

Daraufhin wurde für diese neue Funktion *eine einfache Komponente mit eigener Ereignis-Datenbank entwickelt* (`geoevent.tcl`) und der Firma Vectaport für die Allgemeinheit unter der GPL als Prototyp zur Verfügung gestellt, um das Prinzip zu demonstrieren.

Eine grundlegende konzeptionelle Unterstützung derartiger Funktionen kann aber von Vectaport aufgrund personeller Kapazität in absehbarer Zeit nicht entwickelt werden.

Vectaport hat für die Zukunft aber eine Integration derartiger Konzepte in die eigentlichen Komponenten in Aussicht gestellt.

Wie inzwischen Entwicklungen in anderen Bereichen gezeigt haben, ist eine solche einfache Schnittstelle auch für anspruchsvolle Aufgaben einsetzbar. Eine vergleichbare Schnittstelle wird inzwischen z. B. auch von dem frei verfügbaren XEphem[3] in den aktuellen Versionen sogar zur automatischen Steuerung von Roboter-Teleskopen verwendet.

6.4.2. GRASS

Die graphische Oberfläche von GRASS basiert inzwischen auf TclTkGRASS. Wie der Name bereits andeutet, wird für die Entwicklung Tcl/Tk als zukünftiger Standard verwendet, weil dies eine komfortable und einfache Entwicklung erlaubt und eine systemunabhängige Oberfläche ermöglicht.

Ein Eingreifen in die Oberfläche ist somit auf sehr individuelle Weise ebenso ermöglicht, wie eine direkte Kommunikation mit neuen entwickelten Komponenten, z. B. dem hier entwickelten Prototyp.

6.4.3. Grassland

In vergleichbarer Weise, nur auf stärker integriertem Niveau, verwendet das kommerzielle Grassland, das auf GRASS basiert, inzwischen eine Oberfläche, die fast vollständig in Tcl/Tk implementiert ist.

Prinzipiell gelten hier bezogen auf Tcl/Tk für einen Eingriff und Erweiterungen die gleichen technischen Möglichkeiten, die auch für die Oberfläche von GRASS gelten.

Zu beachten sind die möglichen lizenzrechtlichen Belange. Eine Erweiterung der Kernfunktionen, die zusätzlich zu GRASS entwickelt wurden, ist aufgrund der rechtlichen Aspekte nicht gleichermaßen einfach möglich, wie dies bei GRASS der Fall ist.

6.4.4. OGDI

Da OGDI über eine Tcl/Tk- und eine ANSI C-API Schnittstelle erreichbar ist, ergeben sich in Zukunft Möglichkeiten, OGDI auf flexible Weise für den entwickelten Prototyp zu nutzen.

[2] `http://www.vectaport.com` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[3] `http://www.ClearSkyInstitute.com/xephem/xephem.html` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

Beispielsweise kann eine Nutzung direkt aus der Oberfläche oder der Objektgraphik erfolgen, zum anderen ist eine Integration in den Interpreter bzw. dynamisch ladbare Bibliotheken möglich.

6.4.5. Andere Werkzeuge

Es existieren viele weitere Werkzeuge, die in diesem Zusammenhang eingesetzt werden können. Zahlreiche Oberflächen und Werkzeuge sind inzwischen mit Tcl/Tk in fortgeschrittener Entwicklung. Gerade in Bereich wissenschaftlicher Anwendungen und Geschäftsanwendungen ist der Anteil besonders hoch.

Der in Abschnitt 6.2.4 (Seite 50) bereits angeführte MapServer ist ein Beispiel für eine sehr nützliche Komponente in diesem Rahmen, die über Tcl/Tk und Perl gesteuert werden kann. Damit ist einer Verbindung eigener Erweiterungen und Datensätze mit dieser Technik auf einfache Weise möglich.

Weitere wichtige Werkzeuge, die insbesondere für die Nutzung von Projektionen und zur Visualisierung eingesetzt werden können und per Skripting zugänglich sind, stehen mit Generic Mapping Tools[4] (GMT, unter GPL), dem Cartographic Data Visualizer[5] (CDV), und tkGeoMap[6] frei zur Verfügung.

[4] `http://imina.soest.hawaii.edu/gmt` ..... [V: k. A.] [Ä: k. A.] [Z: 20.02.2001]
[5] `http://www.geog.le.ac.uk/argus/ICA/K.Stynes` ..... [V: k. A.] [Ä: k. A.] [Z: 07.01.2000]
[6] `http://www.tkgeomap.org` ..... [V: k. A.] [Ä: k. A.] [Z: 20.02.2001]

7. Objektgraphik und Ereignisdaten

7.1. Konkretisierung des neuen Datentyps Objektgraphik: Ereignisdaten

Wie bereits beschrieben, wird in dieser Dissertation mit Ereignisdaten eine Konkretisierung des *neu eingeführten Datentyps* der Objektgraphik bezüglich *ereignisorientierter Daten und ereignisaktiver Objekte* bezeichnet.

7.1.1. Vergleich mit konventionellen Daten

Graphische Ausgabegeräte können in zwei Klassen unterteilt werden, rasterbasierte und vektorbasierte Geräte.

Rasterbasierte Geräte (Matrixdrucker, Laserdrucker, CRT Terminals etc.) bauen ein Bild durch gleichartige Bildelemente (engl.: „pixel“, „Picture Element“) auf, in der Regel zeilenweise.

Vektorbasierte Geräte (Pen Plotter etc.) bauen ein Bild hingegen auf, indem sie Linien (elementare geometrische Formen, Polygonzüge, Schattierungen etc.) zeichnen. Solche Geräte sind in der Ausgabe in der Regel langsamer als zeitgleiche rasterbasierte Geräte.

Rasterbasierte Geräte können durch Vektorbefehle gesteuert werden, die interpretiert und in eine Rasterdarstellung umgewandelt werden. Derzeit sind alle gängigen interaktiven graphischen Anzeigegeräte für GIS rasterbasiert. Hingegen sind Pen Plotter immer noch die verbreitetsten Ausgabegeräte, um Karten herzustellen (s. auch [Aro1989]).

Formate, z. B. das im professionellen Bereich sehr weit verbreitete PostScript und Varianten wie EPS, basieren auf einer Programmiersprache und können Text-, Vektor- und Pixelobjekte enthalten.

Für eine Erweiterung zum Einbetten von beliebigem Code für Funktionen, graphische Oberflächen oder insbesondere Ereignissteuerung sind Sprachen wie PostScript jedoch nicht gedacht und auch nicht prädestiniert. Hier bietet sich eine Sprache an, die auf Quellentext basiert und auf Ereignis-Programmierung und graphische Oberflächen ausgelegt ist.

Die wichtigsten Aspekte im Vergleich von Ereignisdaten mit konventionellen Datenformaten sind in Tabelle 7.1 auf Seite 54 zusammengestellt.

Bezüglich dieser Arbeit sind besonders hervorzuheben: die Vorteile der Ereignisdaten bei der Speicherung von Ereigniseigenschaften und Funktionen, die Einbettung von Raster- und Vektordaten und die Möglichkeit zum Einsatz zahlreicher leistungsfähiger Werkzeuge.

Gemeinsam haben alle diese Datenformate, daß sie mit jeweils spezifischen Informationsverlusten behaftet sind, bezogen auf die abzubildenden realen Situationen. Solche Verluste liegen beispielsweise im Bereich Auflösung, Präzision der Wiedergabe, Verluste durch Wiederverwendung bzw. Konvertierung von Daten und Aufnahme fremder Datenanteile in bestehende Daten.

Durch die Möglichkeit der flexiblen Integration fremder Daten in Ereignisdaten lassen sich solche Probleme je nach Einsatzgebiet umgehen.

Eine geeignete Form solcher Daten muß sich konzeptbezogen durch Objektgraphik umsetzen lassen, einen Schwerpunkt im Umgang mit Ereignissen haben und ausreichend geeignete graphische Werkzeuge für die Entwicklung von graphischen Oberflächen mitbringen.

Für die Realisierungen zu dieser Dissertation bot sich daher Tcl/Tk in besonderer Weise an.

7.1.2. Quellentext-Datenformat

Insbesondere bei Applikationen, die mit großen Datensätzen umgehen müssen, kann ein binäres Datenformat aus verschiedener Hinsicht eine erhebliche Erhöhung der Performanz bedeuten.

Für die entwickelte Komponente, die speziell zu dem Zweck der Anbindung von Ereignissen entwickelt worden ist, steht dieser Aspekt jedoch in keinem Verhältnis zu den Vorteilen der Ereignis-Datensätze auf Basis von Quellentext.

Die Anzahl der handhabbaren Objekte und die Größe der einzubindenden Rasterobjekte ist zudem maßgeblich durch die eingesetzte Hardware vorgegeben.

Vorteile		
Rasterdaten	*Vektordaten*	*Ereignisdaten*
• einfache Datenstruktur • Überlagerung mit einfachen Mitteln zu realisieren • effiziente Darstellung räumlicher Variabilität • breite Unterstützung durch Aufnahmegeräte • in den meisten Fällen notwendig für effiziente Manipulation und Verbesserung digitaler Aufnahmen • Unterstützung durch sehr viele Programme • weitgehend systemunabhängig	• kompaktere Datenstruktur als im Rastermodel • einzelne Objekte möglich, obwohl die Bedeutung meist nicht über bestimmte Applikationen hinausgeht • effiziente Umsetzung von Topologie, effiziente Implementierung entsprechender Operationen • bessere Annäherung an Handzeichnungen • effizient für technische bzw. geometrische Elemente, z. B. bei Vergrößerungen (engl.: „zoom“)	• flexible Speicherung von Ereigniseigenschaften und Funktionen • Einbettung von Raster- und Vektordaten • Überlagerung durch Raster- und Vektordaten • Topologie, räumliche Variabilität, Manipulation etc. wie bei Raster- und Vektordaten • weitgehend systemunabhängig • Bearbeitung: viele konventionelle Werkzeuge und Programme • *[desweiteren Vorteile des eingebetteten Datenmaterials]*

Nachteile		
Rasterdaten	*Vektordaten*	*Ereignisdaten*
○ keine Speicherung von Ereigniseigenschaften ○ weniger kompakte Datenstruktur ○ Problematik bei der Darstellung topologischer Relationen ○ keine Bildung von Objekten ○ Rasterung von Grenzen, unzureichende Auflösung bei Details ○ Bearbeitung: meist nur spezielle Werkzeuge und Programme	○ keine Speicherung von Ereigniseigenschaften ○ komplexere Datenstruktur ○ Überlagerungen schwerer zu implementieren ○ ineffizientere Darstellung räumlicher Variabilität ○ Manipulation und Verbesserung digitaler Bilder in der Vektordomäne nicht effizient ○ häufig anwendungsspezifisch	○ komplexere Datenrelationen ○ komplexere Relationen bzgl. Daten/Komponenten ○ komplexere Relationen bzgl. Daten/externe Werkzeuge ○ Erzeugung aus einfacher strukturierten Daten mit geringerem Informationsgehalt notwendig ○ *[desweiteren Nachteile des eingebetteten Datenmaterials]*

Tab. 7.1: Vergleich der neu eingeführten Ereignisdaten mit konventionellen Datenformaten

Die entwickelte Software ist aber nicht auf gängige PC-Hardware beschränkt, sondern kann auch auf leistungsfähigeren Rechnern eingesetzt werden.

Sind Komponenten der Applikation nach Bytecode kompiliert und z. B. als Wrapper-Applikation verpackt, so muß ein geeigneter Interpreter (engl.: „interpreter“) statisch oder dynamisch zur Verfügung gestellt werden. Dies impliziert, daß dann auch die Ereignis-Datensätze für Karten, Präsentationen usw. sowohl in offener als auch vorkompilierter Form geladen werden können.

Steht auch eine Lizenz des verwendeten Compilers oder eine frei verfügbare Version bereit, so können erstellte Ereignis-Datensätze auch selbst nach Bytecode kompiliert werden.

Dies hat auch für den Anwender den Vorteil, daß private Funktionen und Daten nicht offen zur Verfügung gestellt werden müssen.

7.2. Einsatzbereiche

7.2.1. Kategorien von Anwendungen

Es ist wichtig, zwei grundlegende Kategorien von Anwendungen zu unterscheiden, die lokalen und die WWW-basierten Anwendungen. Insbesondere die WWW-basierten Anwendungen haben seit einigen Jahren zunehmend Verbreitung gefunden. Anlässe, um WWW-basierte Anwendungen zu entwickeln, sind vielfältig. Einige der wichtigsten Gründe sind folgende:

- Nutzung dynamischer Elemente in elektronischen Dokumenten, beispielsweise die Verwendung von Mini-Programmen auf Benutzerseite mit *geringem* Aufwand und *ohne* weiterreichende Kenntnisse.
- Nutzung spezifischer interaktiver Funktionen in einem Intranet.
- Bereitstellung von (Demo-) Versionen im Internet.

Im Gegensatz zu Anwendungen, die lokal installiert werden, müssen WWW-basierte Anwendungen, Mini-Programme, sogenannte *Applets*, andere Anforderungen erfüllen.

7.2.2. Lokale Anwendungen

Lokal auf einem Computer installierte Anwendungen besitzen in der Regel einen vollen Zugriff auf das lokale System, unter Beschränkung durch die lokalen Rechte. Sie haben Zugriff auf den Speicher, können Dateien anlegen und löschen, andere Anwendungen starten und vieles mehr.

In der Regel wird eine solche Anwendung als vertrauenswürdig bezeichnet, da man den Urheber kennt und sie beispielsweise eigenständig installiert hat.

7.2.3. WWW-Anwendungen

Ein Applet hingegen wird in aller Regel während eines Aufenthalts im WWW von einem WWW-Klienten (engl.: „browser“) aus dem Netz geladen und ausgeführt. Dies kann für den lokalen Nutzer auch transparent geschehen. In der Regel wird der Nutzer also nicht immer wissen, wer für mögliche Vorgänge auf seinem Computer verantwortlich gemacht werden kann.

Aus diesem Grund müssen Mini-Programme in einer geschützten Umgebung ablaufen und über strenge Sicherheitskriterien kontrolliert werden. Sie können dann z. B. nicht lokale Informationen ausspionieren oder Dateien auf dem lokalen Computer löschen.

Es ist daher aufgrund der Architektur des Gesamtsystems nicht möglich, eine komplexe Anwendung mit allen Funktionen uneingeschränkt zu einer sicheren WWW-basierten Anwendung zu machen.

Bei einer solchen Umsetzung ist es sehr wichtig, aufgrund der jeweils erforderlichen Funktionalitäten zwischen verschiedenen Einsatzbereichen zu unterscheiden:

- Lokaler Einsatz.
- WWW-basierter Einsatz (geschützte Umgebung).
- WWW-basierter Einsatz (vertrauenswürdige Applikation).

Desweiteren können für einen WWW-basierten Einsatz auch durch zusätzliche Bedingungen, wie etwa die Nutzung in einem Intranet, Aspekte hinzukommen, die je nach Umgebung spezielle Überlegungen erfordern.

Für die Verwendung in den verschiedenen Einsatzbereichen ist eine weitreichende Modularisierung nach den einzelnen Funktionalitäten notwendig.

7.2.4. Schritte hin zu Applets

Es ist möglich eine Applikation so zu konzipieren, daß sie lokal in vollem Umfang einsatzfähig ist, daß sie WWW-basiert in einer vertrauenswürdigen Umgebung in vollem Umfang einsatzfähig ist und Teile davon WWW-basiert in einer geschützten Umgebung Verwendung finden können.

Um eine konventionelle lokale Applikation als WWW-Anwendung verwenden zu können, sind grundlegende Schritte erforderlich:

- Die Applikation muß in der Regel in geeigneter Weise umgeschrieben werden.
- Es muß eine WWW-Seite mit einer `embed` Marke erstellt werden.
- Applikation und WWW-Seite müssen auf einem WebServer zur Verfügung gestellt werden.
- Zur Nutzung muß ein WWW-Klient (engl.: „browser“) mit entsprechendem Plugin oder nativer Unterstützung verwendet werden.

Bei vielen Applikationen erfordert der erste Punkt eine vollständige Neuentwicklung der Applikation oder größerer Teile, was sehr aufwendig und langwierig sein kann. Die weiteren Punkte sind in der Regel kaum mit nennenswerten Aufwand verbunden.

7.2.5. Einschränkungen beim Einsatz von Applets

WWW-basierte Mini-Programme (engl.: „applets“) sind aus den bereits beschriebenen Gründen, insbesondere Sicherheit, mehreren Einschränkungen unterworfen:

- Die gesamte Applikation muß in einem einzigen Programm vorliegen.
- Graphiken und graphische Elemente der Applikation müssen in diesem Programm vollständig enthalten sein.
- Fenster-Manager Funktionen werden nicht benötigt.

Desweiteren sollten die Quellentexte kompakt gehalten und beispielsweise nicht benötigte Daten oder umfangreiche Kommentaranteile entfernt werden.

In einer sicheren Umgebung gelten für derartige Mini-Programme zusätzliche funktionelle Einschränkungen:

- Menüs können nicht verwendet werden. Anstelle von Menüs können Schaltflächen Verwendung finden.
- Veränderliche Menüs müssen durch passende Dialoge ersetzt werden.
- Es können keine Fenster oberster Hierarchie (engl.: „toplevel windows“) erzeugt werden. Anstelle von derartigen Fenstern können Rahmen (engl.: „frames“) verwendet werden.
- Neben `menu` und `toplevel` können auch Funktionen wie `grab`, `vwait`, `socket` und `open` *nicht* verwendet werden. Ein Mini-Programm kann daher in einer sicheren Umgebung auch keine Dateien öffnen. (Die hier genannten Funktionen sind für Tcl/Tk exemplarisch.)

7.2.6. Plugins: Tcl-Plugin

Eine grundlegende Form von Unterstützung für Applets sind, neben den in einen WWW-Klienten eingebauten Sprachunterstützungen, die Plugins.

Bei dem Tcl-Plugin (eigentlich Tcl/Tk-Plugin), handelt es sich um eine integrierbare Komponente für einige WWW-Klienten. Wenn der Klient (z. B. über einen MIME-Typ) ein eingebettetes Tcl/Tk Skript erkennt, wird das Tcl-Plugin zur Ausführung aufgerufen. Gegebenenfalls ist es die schnellste Lösung, die Dateinamenserweiterung des verwendeten Skripts für diesen Zweck auf `.tcl` zu ändern.

Obwohl es derzeit bereits mehrere hundert verschiedene Plugins für eine Vielzahl von Aufgaben gibt, ist das Tcl-Plugin das einzige, das aufgrund der Integration von Tcl vollständig programmierbar ist.

Das Tcl-Plugin liegt in den Quellen vor, darf modifiziert und kann wie Tcl/Tk auch für kommerzielle Anwendungen kostenfrei verwendet werden.

Um eine möglichst große Flexibilität zu gewährleisten, ermöglicht das Tcl-Plugin die separate Kontrolle über den Zugriff auf Klientenfunktionen, den Zugriff auf das verwendete Protokoll des Klienten (z. B. für die Nutzung eines Proxy-Servers), die Nutzung von Sockets, die vorübergehende Freigabe eines Speicherbereichs und die Nutzung vertrauenswürdiger Mini-Programme, die vollen Zugriff auf die lokalen Ressourcen erhalten. Letzteres ist z. B. in einem Intranet praktikabel.

8. Dynamische Kartographie mit Ereignisdaten

8.1. Ursprünge

Mit der Bedeutung des Internets für die Geoinformatik und Kartographie hat sich die dynamische Kartographie entwickelt.

Seit Jahren gewinnt die Nutzung weltweiter Netze zur Erlangung und Verbreitung von Datenmaterial zunehmend an Bedeutung. Das in den neunziger Jahren des 20. Jahrhunderts aufgebaute World Wide Web (WWW) unterscheidet sich von älteren Netzen für den Benutzer insbesondere durch Bandbreite bzw. Geschwindigkeit, Kommerzialisierung, Interaktivität und eine Vielzahl unterschiedlicher Umgebungen beim Endanwender.

Bei Veröffentlichung von kartographischem Material über das heute bestehende World Wide Web treten zwei grundlegende Hindernisse in den Vordergrund, die gegenüber den meisten anderen Einschränkungen softwaretechnisch gelöst werden können. Zum einen existiert eine breite Vielzahl unterschiedlicher Umgebungen und Konfigurationen beim Endanwender, zum anderen fehlen grundlegende interaktive Funktionen, wie sie für dieses Material benötigt werden.

8.2. Visualisierung

8.2.1. Bedeutung und Weiterentwicklungen

Die Visualisierung liefert zusätzliche Einblicke in Ergebnisse, die andernfalls als Text oder Zahlen dargestellt würden [LHCP1992].

Noch vor wenigen Jahren war die Visualisierung eine zusätzliche Möglichkeit für den Prozeß der Entscheidungsfindung, nicht ein essentieller Teil. Die Bedeutung ist inzwischen eher substantiell, denn dekorativ. Ein Ziel muß in Zukunft die Verifizierbarkeit, Verläßlichkeit und Genauigkeit der Datenbestände sein, die den Visualisierungen zugrunde liegen.

Techniken der Photomontage verwenden eine Kombination aus photographischen Aufnahmen, bestimmten Formen der Wiedergabe (engl.: „rendering“) und künstlerischer Gestaltung. Bezüglich Daten in GIS kann Photomontage als Komposition aus photographischen Bilddaten eines Gebietes und im Computer umgesetzten Bildern der zu untersuchenden Objekte aufgefaßt werden [KM1988]. Die Realität ist dabei niemals erreichbar und wie in vielen Fällen erzielt man die besten Näherungen unter Ausnutzung der Gesetze der Physik. In den meisten Fällen erlauben perspektivische Darstellungen, z. B. von Höhendaten keine weitreichende Analyse. Räumliche Daten variieren mit der Zeit, daher wird immer eine Anpassung und Rekalibrierung notwendig sein. Der Realismus liefert den visuellen Kontext innerhalb dessen Zwänge und Äußerlichkeiten berücksichtigt werden können [BG1990]. Methoden der computergestützten Bildbearbeitung können realistischere Ergebnisse erzielen, als Bearbeitungen von Hand und sie sind kosteneffizienter, als Photo-Retusche und tragen damit wesentlich zu der visuellen Erfahrung der Zieldarstellung bei. Auch wenn Realismus und Transparenz des Mediums auch für Personen außerhalb einer Expertengruppe höher werden, so ist Präzision und Gültigkeit schwerer zu vermitteln [OD1995].

Für die Suche nach dem geeigneten Maß an Realismus muß desweiteren ein Kompromiß zwischen Realität und Kosten gefunden werden [ZK1995].

Für Überlagerungen, Ecken und Kanten können Anti-Aliasing Techniken eingesetzt werden [KM1988].

Bei Montage-Verfahren müssen andere Techniken eingesetzt werden, da die Rasterelemente bzw. Hintergrundszenen auf Pixeln basieren, die pixelweise variieren und überlagerten Elementen, die nicht auf Pixeln basieren [NHIN1986].

Eine genaue Darstellung aller Details in einer komplexen Szenerie ist nicht möglich, in der Regel nicht notwendig und in vielen Fällen auch nicht wünschenswert [Gro1991]. In dem betrachteten Bereich, z. B. im Vordergrund, sollte daher die Auflösung höher sein, als in peripheren Bereichen, z. B. im Hintergrund [GSH+1994]. Um Satellitenbilder und Luftbildaufnah-

men zu perspektivischen Darstellungen zu nutzen, müssen sie sich auf das gleiche geometrische Referenzsystem beziehen wie das Höhenmodell (DEM) oder das Geländemodell (DTM).

In traditionelle Photomontage-Techniken werden Objekte über ein Hintergrundbild gelegt. Dies hat den Nachteil, daß das Ergebnis sehr von den Fähigkeiten des Bearbeiters abhängt [GSH+1994]. Positionierung und Genauigkeit sind dadurch zusätzlich begrenzt.

8.2.2. Techniken

Bei Techniken zur Visualisierung finden sich vier unterschiedliche Kategorien implementiert in verschiedene Software [McG1997]:

1. Die geometrische Modellierung (engl.: „geometric modelling“).
2. Die Video Erstellung, Bearbeitung und Nutzung (engl.: „video imaging“).
3. Die geometrische Video Erstellung, Bearbeitung, Nutzung (engl.: „geometric video imaging“).
4. Die Bild-Drapierung, dreidimensionale Drapierung (engl.: „image draping“).

Als Beispiele für die wichtigen Kategorien der geometrischen Modellierung und verschiedene Aspekte der Video Erstellung bzw. Bearbeitung stehen Persistence Of Vision Raytracer (POV-Ray[1]) und das GNU Image Manipulation Program (GIMP[2]) [BUB1998].

8.3. Karten

8.3.1. Konventionelle und computergestützte Visualisierung

Karten sind graphische Darstellungen bestimmter Teile eines Raumes oder einer Oberfläche. Karten liefern in der Praxis meist zweidimensionale Darstellungen mehrdimensionaler, meist dreidimensionaler Objekte.

Die Struktur einer Karte erlaubt eine sinngemäße Vermittlung der dargestellten Informationen.

Vor dem Einsatz von Computern war das Papier der Speicher für die kartographischen Daten und die dargestellte Karte die zentrale Grundlage für die meisten Einsatzgebiete. Diese Möglichkeiten reichen aber vielfach nicht mehr aus. Das Kartendesign wird zunehmend wichtiger und die konventionellen Techniken zur Kartenerstellung sind nicht mehr geeignet.

- Konventionelle Karten sind beschränkt auf zwei Dimensionen. Die reale Welt ist nicht zweidimensional. Projektionen helfen dieses Problem zu reduzieren.
- Konventionelle Karten sind statisch. Zeitliche Veränderungen oder Animationen können nicht ausschließlich mittels konventioneller Karten vermittelt werden.
- Probleme ergeben sich mit der Darstellung von Wechselwirkungen oder Austauschvorgängen zwischen verschiedenen Punkten.
- Material und Herstellungmittel begrenzen den Einsatz.
- Unschärfen sind schlecht darzustellen. Dies erweckt im allgemeinen einen überzogenen Eindruck von Genauigkeit.

Eine Visualisierung mittels der Unterstützung von Computern behebt zahlreiche Probleme, die sich aus diesen Punkten ergeben und hat neben der vereinfachten Wiederverwendbarkeit auch noch folgende Eigenschaften.

- Computergestützte Ausgabegeräte können auf Rasterbasis und Vektorbasis arbeiten.
- Computergestützte Ausgabegeräte unterstützen Interaktivität und Animationen und ermöglichen damit z. B. die Darstellung bzw. Integration einer weiteren Dimension.
- Computergestützte Ausgabegeräte können flexibel nahezu kontinuierliche Abstufungen von Farben und Farbtönen, Texturen usw. darstellen.
- Daten in drei Dimensionen können z. B. über stereographische Anzeigegeräte dargestellt werden.

Alle verwendeten Methoden bleiben jedoch immer nur Hilfsmittel und eine jede Methode hat ihre Berechtigung insbesondere, wenn sie sich auf die Kernpunkte reduzieren läßt, für deren Vermittlung sie eingesetzt wird.

Aus Sicht der Computerkartographie kann Visualisierung kartographischer Zusammenhänge als ein Prozeß aufgefaßt werden, mit dessen Hilfe räumliche Zusammenhänge dem menschliche Bewußtsein vermittelt werden.

Wichtige Schritte für anliegende Forschungen sind daher:

[1] `http://www.povray.org` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]
[2] `http://www.gimp.org` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

- Die Schaffung engerer und intuitiver Verbindungen von Rasterdaten und Vektordaten.
- Die Integration multimedialer Daten und interaktiver Prozesse zur Vermittlung zusätzlicher Informationen.
- Die flexible und dynamische Handhabung von Unschärfen im Datenmaterial.
- Die Ausweitung der Informationsvermittlung auf drei Dimensionen.

Der letzte Punkt ist mit dem Konzept mit auf Quellentext basierenden Daten derzeit nicht auf jeder Hardware praktisch realisierbar und wurde daher für mögliche Demonstrationen ausgeklammert. Ansatzweise kann VTK über seine Tcl/Tk Schnittstelle als Ausgangspunkt für Weiterentwicklungen verwendet werden.

Die Grundlagen der ersten Punkte sind jedoch auch mit einer zweidimensionalen Implementierung zu realisieren.

Rasterdaten und Vektordaten können über Skripting verbunden werden. Häufig wird bei solchem Vorgehen für verwandte Anwendungen von „Kleber“ (engl.: „glue“) gesprochen. Multimediale Informationen können so auf diese Weise angebunden und vermittelt werden.

Hier spielen Ereignisse eine besondere Rolle für die Erhöhung der Interaktivität. Letztendlich können Unschärfen, wie auch andere Eigenschaften von Objekten dynamisch behandelt werden. Gerade diese Eigenschaften sind in besonderer Weise durch Tcl/Tk abgedeckt.

8.3.2. Informationsgehalt und Eigenschaften von Karten

Eine Karte kann Schichten (engl.: „layers“) oder Überdeckungen (engl.: „coverages“) beinhalten, die häufig zum Zweck der Ausgabe kombiniert werden.

Eine Karte kann außerdem beschreibende Informationen enthalten, die helfen, die Informationen der Karte zu interpretieren.

Die Bestandteile einer Karte können im allgemeinen in zwei Gruppen eingeteilt werden, kartographische Elemente und geographische Eigenschaften.

Unter kartographischen Elementen versteht man Interpretationshilfen, also z. B. Titel, beschreibenden und erklärenden Text, Legenden, Richtungspfeile, Skalenangaben.

Mit geographischen Eigenschaften sind Flächen, Linien und Punkte gemeint, die unter Verwendung der geographischen Datenbasis erstellt werden.

Polygonzüge für Grenzlinien sind in der Regel über Attribute mit Farben oder Mustern oder beidem ausfüllbar.

Alle geographischen Eigenschaften können mit Beschriftungen (engl.: „labels“) oder Symbolen verknüpft werden.

Eine Nutzung von Ereignisanbindungen mit den Daten, die der Handhabung der konventionellen Elemente des Datenmaterials gleichwertig wäre, ist in keinem bekannten Fall verfügbar. Wenn eine solche Möglichkeit benötigt wird, so kann in einigen Fällen lediglich eine Ersatzlösung durch Einsatz einer speziellen Applikation oder eine Erweiterung diese Funktionalität *emulieren*. Die dynamischen Daten sind aber in konventionellen Anwendungen *nicht* grundlegender Bestandteil des Datenmaterials.

8.4. Jenseits statischer Repräsentationen mittels Ereignisdaten

8.4.1. Erhöhte Dynamik

Neue Werkzeuge, wie Tcl/Tk und Java, ermöglichen eine wesentlich breitere Kontrolle über die Interaktivität bei der Visualisierung kartographischen Datenmaterials [SWJ+1998].

So kann zum einen ein weites Feld an Anwendungsumgebungen realisiert, zum anderen können aber auch verstärkt interaktive Prozesse transparent und flexibel genutzt werden.

Dynamische Kartographie geht über statische Repräsentationen räumlicher Datensätze hinaus und konzentriert sich auf die Umsetzung von zeitlichen Veränderungen und Ereignissen. Diese Methoden sind daher insbesondere nützlich im Forschungsstadium zur Erweiterung der Möglichkeiten zur Darstellung und zum Austausch von Daten.

Eine Typologie dynamischer Visualisierung muß daher auf Änderungen in den Darstellungen von Kartenmaterial eingehen, das sogenannte *Kartenverhalten* [She1995]. Dieses Verhalten umfaßt u. a. Kamera- bzw. Betrachterbewegung, Objektbewegungen, dynamische Vergleiche und dynamische Relationen.

Ein weiterer wichtiger Aspekt dynamischer Visualisierung ist der Einsatz für kommunikative Prozesse.

Umfangreichere Untersuchungen sind in diesem

Zusammenhang allerdings noch nicht durchgeführt worden.

Tcl/Tk ist aufgrund seiner Eigenschaften prädestiniert für die effiziente Umsetzung dynamischer Kartographie [Dyk1996] [Dyk1999] [Dyk1998].

Tcl bietet mächtige Fähigkeiten zur Handhabung von Dateien sowie zur Manipulation von Daten, mit deren Hilfe Datenmaterial in dynamische Anwendungen integriert werden kann. Eine eingehende Diskussion zeigt, daß die Modellierung räumlicher Daten in Listenform insbesondere Vorteile für diese Zwecke hat [BC1996]. Tcl/Tk stellt implizit Farbsymbolismus, Textzuweisungen und die Kommunikation zwischen Objekten sowie die Basis für eine Dynamik bereit, die für eine Umgebung zur Abbildung von Visualisierungen benötigt wird. Das Canvas Element (engl.: „canvas widget“), eine virtuelle Leinwand, liefert zudem die benötigte Kontrolle über die Anordnung von Objekten. Auf diese Weise können die grundlegenden Klassen graphischer Symbole (Linie, Polygon, Oval, Rechteck, Kreisbogen, Bitmap und Image) angeordnet und über eine Vielzahl von Options-Wertepaaren angesprochen werden.

Das Canvas stellt damit eine geeignete Basis zur Abbildung dar, mit Hilfe derer dynamische Objekte räumlichen Lokationen zugewiesen werden können und bietet einen Symbolismus, der Datenwerte und Eigenschaften wiedergibt. Die Kommandos des Canvas sind daher der wichtigste Zugang zu einem dynamischen Abbilden in dieser Umgebung.

8.4.2. Anbindung von Ereignissen

Die Möglichkeiten zum Einsatz von Ereignissen mit Tcl/Tk sind sehr vielseitig und flexibel [RT1999] [Har1997] [HM1998]. Aus diesem Grund läßt sich hier nur ein kleiner Überblick vermitteln.

Die Anbindung von Ereignissen an Objekte auf dem Canvas funktioniert ähnlich der Anbindung von Ereignissen an Elemente (engl.: „widgets“) mit dem Tk-Befehl `bind`.

An Objekte des Canvas können nur Ereignisse angebunden werden, die Maus oder Tastatur zugeordnet sind sowie virtuelle Ereignisse (engl.: „virtual events“).

Besondere Standardattribute (engl.: „tag“, Attribut) sind `current` und `all`. Bestimmte Ereignisse, z. B. das Betreten und Verlassen eines Objektes, werden dadurch gesteuert, daß ein Objekt das aktuelle (`current`) Objekt wird. Ereignisse, die der Maus zugeordnet sind, werden auf dieses Objekt bezogen. Ereignisse, die der Tastatur zugeordnet sind, werden auf das Objekt bezogen, das den Tastaturfokus (`focus`) hat, der z. B. mit der `focus`-Methode gesetzt werden kann. Wird `all` verwendet, bezieht sich die spezielle Bindung auf alle erreichbaren Objekte, z. B. im Fenster mit einem angegebenen Pfadnamen.

Ein virtuelles Ereignis kann nur durch ein reales Ereignis (Maus, Tastatur) ausgelöst werden, das hinter dem virtuellen Ereignis verborgen ist.

Treffen mehrere Bindungen auf ein bestimmtes Ereignis zu, so werden alle Bindungen ausgelöst. Die Reihenfolge der Auslösung von Ereignissen ist folgendermaßen.

Ein Ereignis, das an das Attribut `all` gebunden ist, wird als erstes ausgelöst. Danach werden Ereignisse ausgelöst, die an je ein beliebiges Attribut gebunden sind und anschließend Ereignisse, die sich auf die Identitätsnummer eines Objektes (ID) beziehen.

Sind mehrere Bindungen für ein bestimmtes Attribut vorhanden, dann wird das Ereignis ausgelöst, das hinter der Bindung steht, die am wenigsten allgemein ist. Mit dem Befehl `break` werden Sequenzen von Befehlen abgebrochen, die über Bindungen zugewiesen sind. Alle Bindungen an das Canvas Element (engl.: „canvas widget“) mit dem Befehl `bind` werden nach möglichen passenden Bindungen an Objekte ausgelöst.

Eine detailierte Übersicht der wichtigsten grundlegenden Funktionen ist in allen Basiswerken zu Tcl/Tk vorhanden ([RT1999] S. 57–70).

Als konkretes Beispiel für die Handhabung von Symbolismus und Ereignissen soll hier die Manipulation einer Eigenschaft eines Elementes des Canvas über ein Attribut gegeben werden (Abbildung 8.1):

```
nameofcanvas itemconfigure \
  tag -option value \
  [-option value ...]
```

Abb. 8.1: Quellentextbeispiel: Eigenschaften eines Elementes des Canvas über Attribute

Desweiteren ist die Anbindung von Ereignissen an solche Elemente (Abbildung 8.2) von besonderer Bedeutung. Auf diese Weise können z. B. Kommandos an Elemente mit bestimmtem Attribut gebunden werden.

```
nameofcanvas bind \
  tag <action> {
        Tcl Skript
      }
```

Abb. 8.2: Quellentextbeispiel: Anbindung von Ereignissen an Attribute

Es können aber auch Aktionen an beliebige Lokationen auf einem Canvas gebunden werden (Abbildung 8.3).

```
bind nameofcanvas \
  <action> {
    Tcl Skript
  }
```

Abb. 8.3: Quellentextbeispiel: Anbindung von Ereignissen an beliebige Lokationen

Für `nameofcanvas`, `tag`, `<action>` und `Tcl Skript` müssen in diesem Beispiel der Name des Canvas, das betreffende Attribut, das gewünschte Ereignismuster und auszuführende Tcl-Befehle eingesetzt werden.

8.4.3. Ereignismuster

Es gibt drei grundlegende Möglichkeiten Sequenzen von Ereignismustern z. B. an Attribute von Objekten anzubinden.

Die einfachste Form ist ein einzelnes Zeichen, das die entsprechende Zuordnung erhalten soll, mit Ausnahme des Leerzeichens und der spitzen Klammer.

Die zweite Form von Ereignismustern findet Verwendung zur anwenderdefinierten Erzeugung benannter virtueller Ereignisse. Es hat die Form `<<name>>`, also der Name des virtuellen Ereignisses eingeschlossen in doppelte, statt einfache spitze Klammern.

Die dritte Form hat die Syntax (Abbildung 8.4)

```
<modifier-modifier-type-detail>
```

Abb. 8.4: Quellentextbeispiel: Syntax Ereignismuster

Beispiele für gültige und bereits vordefinierte Modifizierer (engl.: „modifier“) für Aktionen sind `Control`, `Shift`, `Lock`, `Button1` oder `B1`, `Button2` oder `B2`, `Button3` oder `B3`, `Button4` oder `B4`, `Mod1` oder `M1`, `Mod2` oder `M2`, `Mod3` oder `M3`, `Mod4` oder `M4`, `Mod5` oder `M5`, `Meta` oder `M`, `Alt`, `Double`, `Triple`.

Typ-Elemente (engl.: „type“) entsprechen dem Standard der Ereignistypen des X Window Systems: `ButtonPress` oder `Button`, `ButtonRelease`, `Circulate`, `Colormap`, `Configure`, `Destroy`, `Enter`, `Deactivate`, `Expose`, `FocusIn`, `FocusOut`, `Gravity`, `KeyPress` oder `Key`, `KeyRelease`, `Leave`, `Map`, `Motion`, `Property`, `Reparent`, `Unmap`, `Visibility`, `Activate`.

Die erlaubten Werte der „Details“ (engl.: „detail“) hängen von dem jeweiligen Typ-Element ab.

In der angegebenen Syntax lassen sich die Ereignismuster also weitergehend mit Bezeichnern, Typ-Elementen und Details kombinieren, z. B. zu (Abbildung 8.5):

```
<Control-F5>
<Control-Meta-Down>
<Control-B3-Motion>
<Control-ButtonRelease-3>
<Shift-Button-2>
```

Abb. 8.5: Quellentextbeispiel: Beispiele für Kombinationen von Ereignismustern

Für einige Sequenzen existieren Abkürzungen. Es kommt auf den Einzelfall an, ob man solche weniger „sprechenden“ Abkürzungen verwenden sollte.

Nach einem der obigen Beispiele zur Anbindung von Ereignissen stellt sich ein reales Beispiel eines `bind` wie in Abbildung 8.6 dar.

```
$w bind \
  all <Meta-ButtonRelease-3> {
        puts "Button Released"
      }
```

Abb. 8.6: Quellentextbeispiel: Anbindung von Ereignissen mittels Ereignismuster

In diesem Beispiel wird allen Objekten (`all`) im Pfad `$w` (z. B. mit `set w .f.sub`) eine kleine Textausgabe zugeordnet, die ausgelöst wird, wenn `Meta` gedrückt ist und gleichzeitig die Maustaste 3 losgelassen wird.

Ereignisse lassen sich auch auf einfache Weise kombinieren. Es ist leicht möglich, beliebige Befehle an Sequenzen von mehreren Ereignissen anzubinden (Abbildung 8.7).

```
bind . \
  <Meta-x><Meta-y> \
  { puts "Ereignis: M-x M-y" }
bind . \
  <KeyPress-s><o><w><a><s> \
  { puts "s o w a s gibt’s" }
```

Abb. 8.7: Quellentextbeispiel: Anbindung an Kombination von Ereignissen

Die Ausführung der Befehle, die an diese Ereignisse angebunden sind, beginnt, wenn die entsprechenden

Ereignisse nacheinander ausgelöst worden sind.

Es lassen sich aber auch vom Anwender neue Ereignisgruppen erzeugen. Virtuelle Ereignisse können durch `event` Operationen erzeugt und gehandhabt werden (Abbildung 8.8):

```
event operation [arg1 arg2 ...]
```

Abb. 8.8: Quellentextbeispiel: Syntax zur Handhabung virtueller Ereignisse

Operationen sind in diesem Zusammenhang `add`, `delete`, `generate` und `info`. Diese Operationen haben spezifische Optionen und verarbeiten eine Reihe von Argumenten.

Das folgende Beispiel (Abbildung 8.9) demonstriert die praktische Vorgehensweise:

```
event add <<myvirtual>> \
  <Control-v> <Control-t>
bind . <<myvirtual>> {
  puts "- Virtuelles Ereignis! -"
}
event info <<myvirtual>>
```

Abb. 8.9: Quellentextbeispiel: Beispiel virtuelles Ereignis

Der erste Befehl definiert ein neues virtuelles Ereignis unter dem Namen `myvirtual` mittels der Operation `add` und den zugeordneten Ereignismustern.

Die Anbindung bindet eine Ausgabe mittels `puts` an das virtuelle Ereignis.

Der letzte Befehl liefert die Informationen über das betreffende virtuelle Ereignis. Ohne Angabe eines bestimmten Ereignisses liefert dieser Aufruf eine Liste aller aktuell definierten virtuellen Ereignisse.

Vordefiniert sind in der Regel z. B. die virtuellen Ereignisse `<<Copy>>`, `<<Paste>>` und `<<Cut>>`.

Virtuelle Ereignisse helfen sowohl Systemunterschiede zu berücksichtigen als auch spezielle Gruppen von Ereignissen zu bilden.

Eine nützliche Funktion bei der Verwendung von Ereignissen ist die Ersetzung in Ereignisfeldern. Ähnlich wie Formate in Befehlen vieler Programmiersprachen lassen sich Ersetzungen mit Ereignissen verwenden, z. B. in `bind` Befehlen. Auf diese Art und Weise ist es möglich, bei der Ereignisbehandlung Informationen über das aktuelle Ereignis zu parametrisieren.

Eine Auswahl der wichtigsten Ersetzungen mit einer Kurzbeschreibung zeigt folgende Tabelle (Tabelle 8.1).

Ausdruck	*Kurzbeschreibung*
`%%`	einzelnes Prozentzeichen
`%W`	Fenstername, ereignisempfangend
`%x`	x-Koord. (Widget) des Mauszeigers
`%y`	y-Koord. (Widget) des Mauszeigers
`%X`	x-Koord. (Bildschirm) des Mauszeigers
`%Y`	y-Koord. (Bildschirm) des Mauszeigers
`%b`	Nummer der bedienten Maustaste
`%A`	ASCII-Zeichen einer Taste
`%K`	symbolischer Name einer Taste
`%h`	Höhe eines Widgets
`%w`	Breite eines Widgets

Tab. 8.1: Auswahl wichtiger `bind` Ersetzungen

Folgendes Beispiel (Abbildung 8.10) demonstriert eine einfache Anwendung:

```
bind . <B1-Motion> {
  puts "Bewegung Mauszeiger %x %y"
}
```

Abb. 8.10: Quellentextbeispiel: Einfaches Anwendungsbeispiel einer Ersetzung

Bei jeder Bewegung der Maus mit gehaltener erster Maustaste wird eine Zeichenkette mit den aktuellen Koordinaten des Mauszeigers ausgegeben.

8.4.4. Allgemeines zu Ereignissen

Ereignisbehandlung für komplexere Ereignisse, Informationen zu Anbindungsattributen, Ereignisschleifen usw. sowie Hinweise zu speziellen Anwendungen finden sich in der gängigen Literatur [HM1998].

Allgemein gelten z. B. spezielle Regeln für Fehlerausgaben bei Ereignissen, Mehrfachzuweisungen und ignorierten Ereignissen.

Ereignisse auf einer Arbeitsoberfläche können auf modernen Systemen über verschiedene Mechanismen sehr flexibel zugewiesen werden. Es ist gegebenenfalls darauf zu achten, daß sich Ereignisse, die in einer Anwendung zugewiesen worden sind, mit denen überlagern können, die bereits durch die Oberfläche vorbelegt sind.

Weitere Ereigniszuweisungen sind in der Regel über die X Window Ressourcen des X Window Systems definiert.

Für die jeweiligen Details sind die betreffenden Dokumentationen des verwendeten Fenster-Managers und zum X Window System hilfreich, die in der Regel auf dem lokalen Linux/Unix System vorhanden sind.

9. Prototyp einer Komponente zum Konzept

9.1. Ein Prototyp zum Konzept dieser Dissertation: `actmap`

Der Prototyp, an dessen Entwicklung und Eigenschaften diese Dissertation zahlreiche Funktionalitäten des vorgestellten neuen Konzeptes diskutiert und demonstriert, ist mit diversen Beispielen und Daten im Internet verfügbar [Rüc2001]. Alle Programmteile und Daten, die zur Verfügung gestellt werden, bieten sich für weitergehende eigene Experimente an.

Die zentrale Komponente wurde `actmap` genannt und umfaßt eine ganze Reihe von Erweiterungen (s. auch *Verfügbarkeit*, Seite 113 und *Liste ausgewählter Teile des Prototyps*, Seite 115). Der Aufbau hält sich an die in dieser Dissertation konzipierte Vorgehensweise. Für die Nutzung und das Verständnis der Zusammenhänge der Implementierung sind Grundlagen in Tcl/Tk sowie in Perl bzw. C/C++ hilfreich.

Es handelt sich bei den verfügbaren Teilen um eine Umsetzung des Konzepts für die Flexibilisierung und Erweiterung von verschiedenartigen räumlichen und ereignisorientierten Daten und nicht um eine abgeschlossene Applikation. Zur Verifikation wurden die Prototypen verschiedener modularer Softwarekomponenten entwickelt.

Eine umfassende Beschreibung und Diskussion der exemplarisch entwickelten Prototypen und ihrer umfangreichen Funktionen wäre zu komplex und würde den Rahmen dieser Arbeit sprengen. Aus diesem Grund kann für die meisten Betrachtungen die Komponente `actmap` verwendet werden.

Um den Blick auf die konzeptionell und thematisch relevanten Aspekte nicht zu verstellen, werden die aus den Entwicklungen gewonnenen Einsichten gegenüber den Details der Implementierung in den Vordergrund gestellt, wo dies möglich ist.

Stellvertretend sollen folgende Entwicklungen und Eigenschaften dieser speziellen Komponente zugunsten einer kompakten Darstellung hier ohne weitere Erläuterungen aufgelistet werden.

Oberfläche und Benutzerführung:

- Kontextsensitive Hilfe durch Ereignisse und Hyperlinks.
- Kontextsensitive integrierte und konfigurierbare Ballonhilfe (engl.: „balloon help“).
- Objektspezifische Menüs auf dem Canvas (engl.: „popups“).
- Interaktiv konfigurierbare Anzeige- und Eingabefelder.
- Abtrennbare Menüs.
- Verschiedene Bedienelemente (Schaltflächen, Schiebebalken, Radioknöpfe, Karteikarten usw.).
- Visuelle Koordinaten.

Datennutzung:

- Lade- und Speicherfunktionen.
- Funktionen mit Nutzung eines sicheren Interpreters.
- Stapelverwaltung geladener Daten.

Spezielle Funktionen:

- Verschiedene Vergrößerungsfunktionen, z. B. bezogen auf den Ursprung oder das Zentrum des Canvas.
- Rudimentäre Funktionen zur Manipulation von Objekten.
- Funktionen zur Handhabung von Transparenz und Hervorhebung von aktiven Objekten.
- Unterstützung einer Ereignisdatenbank.

- Funktionen zum Import von Rastergraphiken, Vektordaten, Textdateien usw. und zum interaktiven Nachladen von Ereignisanbindungen und Definitionen.
- Verschiedene Farbeditoren.
- Konfigurierbare PostScript Ausgabe.

Weitergehende Nutzung:

- Integrierte Shell: Navigation mit Maus-, Tastatur *und* Shell-Befehlen.
- Ladbare zusätzliche Tcl-basierte Bibliotheken für verschiedene Aufgaben, z. B. Texteffekte.
- Zentrale und benutzerdefinierte Konfiguration.
- Konfiguration über X Window Ressourcen.
- Übergabe von Datensätzen auf der Kommandozeile.
- (Demo-) Datensätze.

Dies sind bei weitem nicht alle zusätzlichen Funktionen, aber für einen besseren Eindruck über die Möglichkeiten im Rahmen einer verhältnismäßig schnellen Implementierung mag diese Zusammenstellung ausreichen.

Bei der entwickelten Applikation handelt es sich in der Regel um Teile, die in eine stetige Entwicklung eingebettet sind [Rüc2000]. Dies ist nicht zuletzt durch die vielfältigen und flexiblen Funktionen der Komponenten bedingt.

Aufgrund des Wegfalls von Entwicklungsphasen, die für den Einsatz bei reinen Anwendern oder gar einen kommerziellen Einsatz unabdingbar sind, ist weder eine hohe Stabilität noch eine Ergonomie zu erreichen, wie sie der reine Endanwender bei gelegentlicher Nutzung gewohnt ist.

Viele Einzelheiten sind auf die Entwicklung und diese Dissertation und die damit zusammenhängende Demonstration ausgerichtet. Dies betrifft z. B. Besonderheiten bestimmter Funktionen, Anordnung von Menüs und Bedienelementen, Eigenheiten von Skalierungen sowie die Anwesenheit von nicht weiter beschriebenen Funktionen und Elementen, die über die grundlegende Darstellung hinausgehen oder zu Testzwecken implementiert wurden.

Für die Demonstration und das eigene Studium ist der Zustand des Prototyps gut geeignet, demonstriert er doch neben zahlreichen Aspekten von Modularisierung bis Sicherheit auch die einfache Nutzung der vielfältigen Einsatzmöglichkeiten ebenso, wie er den Aufwand einer möglichen umfangreichen Applikation für die Unterstützung reiner Endanwender verdeutlicht.

Auf die Darstellung der Anforderungsanalyse wird verzichtet, da in diesem Fall keine realen Anwender existieren und es sich um Tests für Prototypen handelt.

9.2. Modell ereignisaktiver Objekte

9.2.1. Verknüpfung von Informationen

Eines der wichtigsten Wesensmerkmale eines GIS ist die Verknüpfung von Informationen über die Lage und Größe raumbezogener Objekte mit Sachdaten, die thematische Eigenschaften dieser Objekte beschreiben.

Im Sachdatenbereich existiert diesbezüglich keine Programmlogik. Der Aufwand mit diesen Daten ist daher naturgemäß hoch.

In Abbildung 9.1 (Seite 65) sind die grundlegenden Relationen verschiedener Funktionalitäten aufgezeigt, die für die entwickelte Kernkomponente (`actmap`) des vorliegenden Projekts von Bedeutung sind.

Beliebige Ereignisse können ebenso, wie die mit ihnen verknüpften Daten, in Datenbanken abgelegt werden. Dabei kann und sollte die Datenbank nach den individuellen Bedürfnissen ausgesucht und angepaßt werden. Der Einsatz einer separaten Datenbank ist aber nicht zwingend.

Für große binäre Datenmengen, z. B. Multimediadaten (Audio-, Video-, Kartendaten), können Datenbanken mit einer Spezialisierung auf Binary Large Objects (BLOBs), z. B. Empress, geeignet sein [Dic1999]. Dies bietet zusätzlich eine Unterstützung für Tcl/Tk, Perl, C und SQL.

Für diese Betrachtungen ist innerhalb des Projektes von geringer Bedeutung:

1. Erfassung, Speicherung, Verifikation von Daten.
2. Bezug zu Geodaten. Georeferenzierung.
3. Bezug zu Schnittstellen.
4. Konvertierung von Datenformaten.
5. Einbindung von Datenbanken.
6. Einbindung von speziellen Werkzeugen.
7. Kompakte Kapselung der Komponenten.

Von höherer Bedeutung sind folgende Aspekte und stehen vielmehr im Vordergrund:

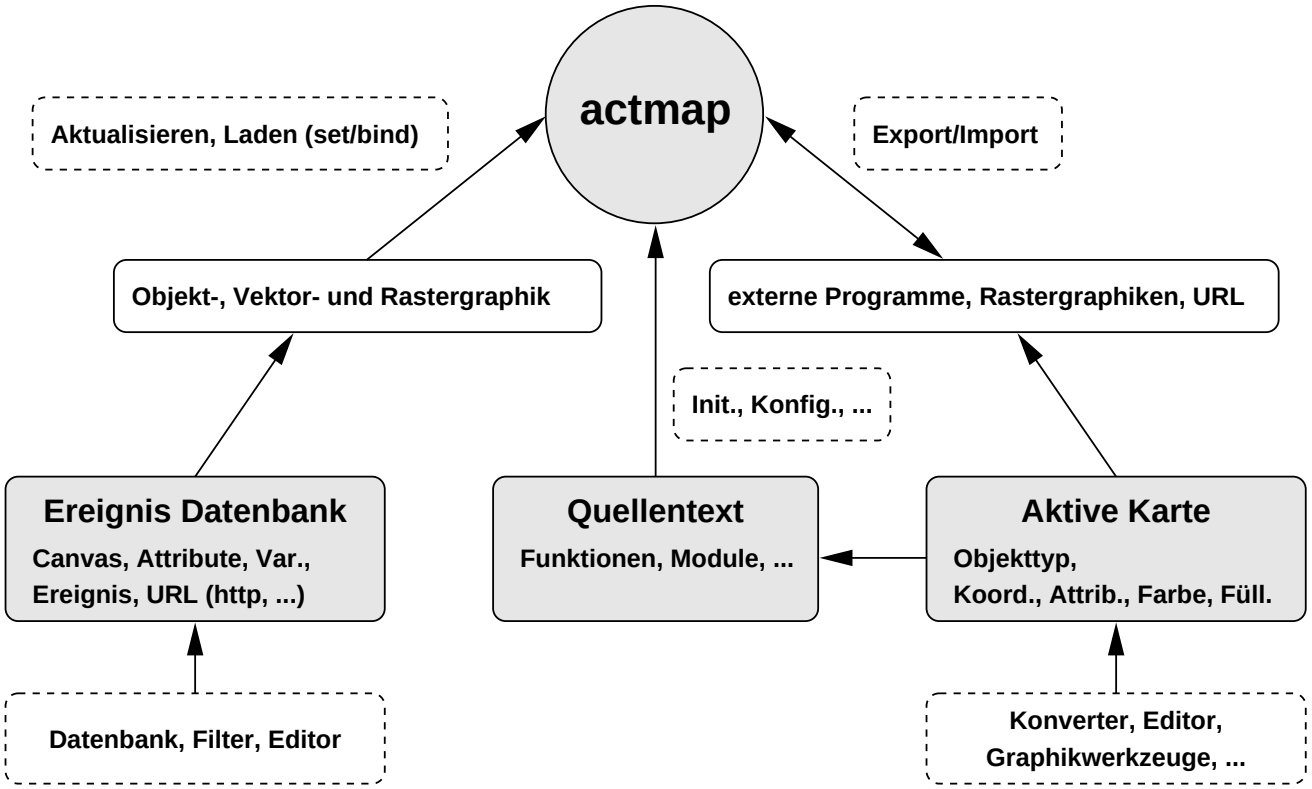

Abb. 9.1: Relationen verschiedener Funktionalitäten für die zu entwickelnde Kernkomponente `actmap`

1. Erfüllung der geforderten Funktionalitäten (dynamische Visualisierung, Ereignissteuerung).
2. Modularität.
3. Konfigurierbarkeit.
4. Integration in Datenverarbeitung.
5. Vollständig freie Entwicklungswerkzeuge, offene Quellen.
6. Portabilität.
7. Wiederverwendung bestehender Entwicklungen.

Das Format auf Basis von Quellentext kann beliebig erweitert und mit dem notwendigen logischen und technischen Aufwand in fast jede Form überführt werden. Attribute und Ereignisse müssen in der Regel ergänzt oder im Einzelfall aus bestehenden Altdaten zusammengeführt werden, da die bestehenden Formate solche Eigenschaften nicht beinhalten.

Die einzelnen Teile dazu können aber für größere Aufgaben auch in separaten Datenbanken verwaltet werden.

Es können alle wichtigen Arten von Objekten, wie persistente und transiente bzw. passive und aktive Objekte, auf einfache Weise gehandhabt werden.

1. Persistente Objekte bleiben auch nach Beenden des Programms erhalten und sind so wiederverwendbar. Transiente Objekte sind nur während der Laufzeit vorhanden, z. B. für bestimmte Anzeigefunktionen. Sie werden beim Beenden nicht erhalten.

2. Passive Objekte bleiben nach Beendigung des Programms erhalten, sind aber im aktiven Programm nicht enthalten. Aktive Objekte sind die während der Laufzeit vorhandenen Exemplare.

Mit den Vorteilen bezüglich sehr hoher Flexibilität geht jedoch der Nachteil einher, daß eine solche Aufbereitung einen hohen Aufwand oder zumindest genaue Kenntnis der Daten erfordert, mit denen umgegangen wird.

Die Daten in einem solchen Quellentext-Format können auf einfache Weise zur Aufbewahrung aller Arten von Objekten verwendet werden. In vielen Fällen sind die relevanten Daten aber persistente aktive Objekte.

Es ist aber z. B. auch möglich passive Objekte durch entsprechende Markierung bzw. Attribute zu speichern. Ebenso können auch transiente Objekte wie in einem Puffer zwischengelagert werden, obwohl sie dadurch temporär zu persistenten Objekten werden können.

Ein Vorteil des Quellentext Konzeptes ist für den einfachsten Fall die Verwendung eines generischen Canvas. Dies impliziert auch die leichte Umsetzung von Objektgraphik in autarke Applikationen oder für die Verwendung in Plugins. Ein Einsatz als Transferformat ist ebenfalls denkbar.

9.2.2. Projektstruktur

Das Projekt nutzt verschiedenartige Verfahren und Werkzeuge, die naturgemäß einen unterschiedlich weitreichenden Bezug zu der eingesetzten Objektgraphik und den entwickelten Komponenten haben. Die wichtigsten sind folgende:

- Tcl/Tk, Perl, C.
- Werkzeuge. Unix „Toolbox“-Philosophie, daher vorwiegend Reihe von unabhängigen Werkzeugen.
- Tcl/Tk, C, Perl ... Erweiterungen. Die Quellentext-Implementierung ist, nicht zuletzt aufgrund der großen Zahl möglicher Erweiterungen, eine der flexibelsten und portabelsten Wege einen Map-Viewer und zugehörige Ereignis-Bindungen (engl.: „event bindings“) zu realisieren.
- Dynamische Bibliotheken, Datenbanken, Wrapper Werkzeuge, Wrapper für technische oder mathematische Zusammenhänge.
- Graphische Benutzeroberfläche, Modifikation der Oberfläche.
- Shell, Ausführung von interaktiven Befehlen und Skripten.
- Makrofunktionen können in der nativen Sprache der Komponente entwickelt werden.

Alle Tcl/Tk-basierten Verfahren und Werkzeuge sind in diesem Rahmen eng mit der Objektgraphik und den eigentlichen Komponenten gekoppelt.

Externe Werkzeuge können sehr geeignet für den jeweiligen speziellen Zweck gewählt werden. Eine engere Verbindung zu den Komponenten ist nicht notwendig. Dies hat gegebenenfalls Einfluß auf die Portabilität. Aufgrund der Forderung beliebige Anwendungen an Ereignisse anbinden zu können, ist dies aber indirekt gewollt.

Dynamische Bibliotheken, Datenbanken usw. sind spezielle Werkzeuge und in der Regel plattformabhängig aber prinzipiell unabhängig von einer Komponente.

Graphische Benutzeroberflächen und Modifikationen können mit verschiedenen Mitteln erstellt werden. Der Einsatz von Tcl/Tk ist in diesem Fall zu bevorzugen. Dies gilt ebenso für die Nutzung der Shell, da diese für jede Tcl/Tk basierte Komponente verfügbar ist.

Damit sind Daten, Oberfläche, Skripten, Makrofunktionen und Funktionssammlungen der Komponenten in hohem Grad portabel und bestehen aus Quellentext der gleichen Sprache.

9.2.3. Datenmodell

GIS bezeichnet ein Softwaresystem zur Bearbeitung (engl.: „processing“) räumlicher Daten (engl.: „spatial data“).

Aufgrund der benötigten Eigenschaften ist ein adäquates Datenmodell realisierbar, das eine Möglichkeit zur Verfügung stellt, räumliche Daten und insbesondere Ereignisdaten (engl.: „event data“) in einem Computerspeicher zu handhaben.

Dabei kann der Grad der Abstraktion bestimmter Vorgänge auf verschiedenen Stufen gewünscht sein.

Die Mehrzahl moderner GIS, insbesondere die vektorbasierten, versuchen eher die Wiedergabe einer Abbildung räumlicher Phänomene, als die Wiedergabe der räumlichen Phänomene selbst. Dieses führt zu einer Überkomplizierung von Speicherformaten und Algorithmen zur Bearbeitung, was mit dazu führt, daß sich der Anwender mit technischen Dingen, z. B. der Polygon-Topologie, auseinandersetzen muß. Dies ist aber oft irrelevant für sein eigentliches Problem im Bereich Geologie, Geophysik, Geographie etc., genauso, wie das Erzeugen von Abständen von Zeichen für eine Proportionalschrift (engl.: „kerning“) irrelevant für den Inhalt der meisten Artikel ist.

Aus diesem Grund ist es auch leicht verständlich, daß Kartenmaterial jeglicher Art auch nur ein weitverbreitetes Werkzeug zur Analyse räumlicher Daten darstellt, aber eben nicht mehr ist, als ein Werkzeug.

Dies gilt jedoch nicht, wenn, verbunden mit der Darstellung des Datenmaterials, Funktionen benötigt werden, die über die einer statischen Karte hinausgehen, egal mittels welchen Mediums diese vorliegt.

Die folgende Zusammenstellung erläutert Begriffe, die für Nutzung und Entwicklung von Komponenten von Bedeutung sind, wie sie in dieser Dissertation vorgestellt werden.

Objekte: Objekte sind Elemente auf einem Canvas, die bestimmte Koordinaten, Attribute, (physikalische und mathematische) Eigenschaften und Zustände haben.

Schichten: Eine Schicht ist eine Menge von Objekten, die Gemeinsamkeiten haben. Bei dem entwickelten Konzept muß es sich dabei nicht notwendigerweise um räumliche Daten handeln.

Eine Schicht besteht aus Objekten, die semantische Informationen für Koordinaten beinhalten. Diese Information kann z. B. abgefragt oder für Darstellung, Ereignisbindung oder andere Aufgaben verwendet werden.

In Erweiterung des Objektkonzeptes können funktionelle Eigenschaften durch Prozeduren ausgenutzt und mathematisch beschrieben werden. Schichten können damit auch durch Funktionen definiert werden.

Schichtklassifikation:

Schichten können durch ihren Definitionsbereich und ihre Werte klassifiziert werden.

Definitionsbereich:

Zweidimensionale Schichten:

Zweidimensionale Schichten sind definiert in einem bestimmten zusammenhängenden Bereich. Dieser Bereich ist in der Regel endlich.

Eindimensionale Schichten:

Eindimensionale Schichten sind definiert über einen Satz von Linien im Untersuchungsgebiet.

Nulldimensionale Schichten:

Nulldimensionale Schichten sind definiert über einzelne Punkte.

Werte:

Numerische Schichten:

Die Werte numerischer Schichten gehören zu einem kontinuierlichen Intervall auf einer numerischen Achse und können beliebige Werte zwischen einem minimalen und maximalen Wert annehmen.

Klassifizierte Schichten:

Klassifizierte Schichten haben eine finite Anzahl von Werten. Als Werte sind auch Zeichenketten möglich.

Diese Klassifikation deckt die meisten theoretisch relevanten Schichten ab. Implementierungstechnisch sind weitere Detailierungen notwendig, z. B. bezüglich der Herkunft thematischer Daten.

Es existieren viele verschiedene Arten von Schichten, die in den häufigsten Fällen auf ihrer Herkunft aus verschiedenen Datenquellen beruhen.

1. Rasterschicht (engl.: „raster layer“)
   - Rasterdaten (engl.: „raster data“), statische Daten, im Regelfall Datensätze mit tabellarischer Legende.
   - DEM, statische Daten, Werte im Regelfall errechnet über mathematische Funktionen aus Datensatz-Klassen.
   - Darstellungsdaten (engl.: „chart“), dynamische Daten, im Regelfall über Datenbankabfragen.
2. Punktschicht (engl.: „point layer“, engl.: „point data“).
   - Attribute (engl.: „tag“), Satz von Punkten beliebiger Werte.
   - Diagramm (engl.: „diagram“), Satz von Punkten, mit einem Vektor numerischer Werte.
   - Observation (engl.: „observation“), Satz von Punkten, im Regelfall mit Werten aus Datenbankabfragen.
3. Vektorschicht (engl.: „vector layer“).

Schichtobjekte (engl.: „layer objects“) können Methoden und Eigenschaften haben. In vielen konventionellen Fällen werden einzelne Layer als Ganzes behandelt. Aufgrund der Eigenschaften und Attributbindungen können aus Objekten verschiedener Schichten neue Schichten oder Gruppen gebildet werden.

Durch ihre Eigenschaften und Attribute können auch Objekte verschiedener Schichten gruppiert werden. Dadurch können Schichten entstehen, die Objekte aus konventionellen Schichten enthalten, ohne diese zu zerstören.

Regionen: Sammlung von Datenmaterial mit gleicher Überdeckung und Projektion, aber möglicherweise unterschiedlicher räumlicher Auflösung.

Schicht-Visualisierungsmodi: Die verwendeten Vektorobjekte erlauben Füllfarben-Modus (engl.: „colour mode“), Füllmuster-Modus (engl.: „pattern mode“) und Symbol-Modus (engl.: „symbol mode“).

Datenbankanbindung: Aufgrund der verwendeten Entwicklungsumgebungen stehen eine Reihe von Datenbankanbindungen (RDBMS, ODBC, DBI (Perl) etc.) zur Verfügung. Aufgrund der Attribut-Ereignis Relationen können spezielle Datenbanklösungen integriert werden.

Objekt-Attribut-Ereignis Relationen: Aufgrund flexibler Relationen können aus einem Rasterdatensatz viele Rasterobjekte mit unterschiedlicher Semantik gebildet werden.

Erweiterbarkeit: Da es sich bei GIS-Anwendungen in der Regel um ein komplexes Zusammenspiel vieler Einzelkomponenten handelt, steht die Erweiterbarkeit und Skalierbarkeit an herausgehobener Stelle. Durch die ausschließliche Verwendung von Komponenten, Basisprozeduren, Elementen (engl.: „widgets“) und anderem wiederverwendbaren offenen Code kann im Gegensatz zu den meisten kommerziellen Applikationen eine Erweiterung in alle Richtungen und auf Basis unterschiedlichster Komponenten erreicht werden.

9.2.4. Implementierung

Räumliche Geodaten können nur in den seltensten Fällen durch mathematische Gleichungen adäquat beschrieben werden. Selbst wenn dies möglich ist, so ist dies das Ziel einer Analyse, aber nie der Ausgangspunkt.

Projektionen, z. B. zu kartographischen Zwecken, wurden im vorliegenden Fall nicht implementiert, da sie für das Konzept weitestgehend irrelevant sind. Ebenso ist für die Darstellung die Entwicklung eines Mechanismus für beliebige Genauigkeiten nicht notwendig.

Um das vorgestellte Konzept und die in dieser Arbeit beschriebenen Techniken zur dynamischen Kartographie und Visualisierung sowie die Eignung der diskutierten Entwicklungsumgebung zu testen, wurde in den letzten Jahren eine entsprechende Software (Active Map, `actmap`) entwickelt. Zur besseren Veranschaulichung wurde geeignetes Datenmaterial für unterschiedliche Anwendungsgebiete geschaffen und entsprechend aufbereitet. Die exemplarisch bereitgestellte Software kann zur Demonstration verschiedener Eigenschaften des aufbereiteten Datenmaterials dienen. Alle relevanten Teile sowie das Datenmaterial liegen im Quellentext vor, so daß eine Entwicklung eigener Funktionen und eine Nutzung und Aufbereitung von Datenmaterial für eigene Anwendungen mit geringem Aufwand möglich sein sollte. Neue Versionen der Tcl/Tk Bibliotheken und der Interpreter können bei Bedarf ebenso wie zusätzliche eigene Erweiterungen mit dem inzwischen frei verfügbaren TclPro[1] kompiliert und mit bestehenden Teilen beliebig neu gepackt werden.

Die hier vorgestellte Software ist die vollständigste existierende Applikation, welche die Techniken zur Ereignissteuerung und dynamischen Visualisierung verwendet, die in dieser Dissertation konzipiert wurden.

9.2.5. Komponentendesign

Schichten können sich wie Objekte in objektorientierten Programmiersprachen verhalten.

Schichten haben folgende Eigenschaften:

- Schichten sind vom Typ „Objekt“. Sie beinhalten damit Linien, Bogenzüge, Polygonzüge, Textbausteine etc. und haben Vektorcharakter.
- Schichten können Werte zu Koordinaten liefern.
- Schichten können mehrere Möglichkeiten für ihre Darstellung beinhalten.
- Schichten haben eine hinterlegte Datenquelle, z. B. Legende oder Abbildungsfunktion.
- Schichten haben Visualisierungsparameter.
- Schichten beinhalten Metadaten.

Geeignete Vektorobjekte können im Farb-, Muster- und Symbolmodus verwendet werden. Ihre Darstellung kann damit teilweise durch Rasterdaten ergänzt werden.

Rasterobjekte hingegen beinhalten nur aufgrund ihrer Objekteigenschaften Vektorfunktionen.

9.3. Verteilte Objekte

9.3.1. Dezentrale Datenhaltung

Eine Erweiterung des Datenmodells auf verteilte Objekte in einem Netzwerk ist mit dem entwickelten Konzept auf verschiedene Weise möglich. Eine implizit vorhandene Nutzung von Objektgraphik mit dezentralen

[1] `http://sourceforge.net/projects/tclpro` ..... [V: 2000] [Ä: k. A.] [Z: 25.01.2001]

Daten, mittels der Kernkomponente `actmap`, ist in Abbildung 9.2 veranschaulicht. Dies bringt nicht nur den Vorteil der Möglichkeiten für eine einfache dezentrale Datenhaltung, sondern im Idealfall auch einen Echtzeitzugriff auf benötigte Daten, die nicht im eigenen Netz verwaltet werden. Damit einher gehen allerdings einige in letzter Konsequenz nicht zufriedenstellend lösbare Probleme bezüglich Sicherheit und Performanz.

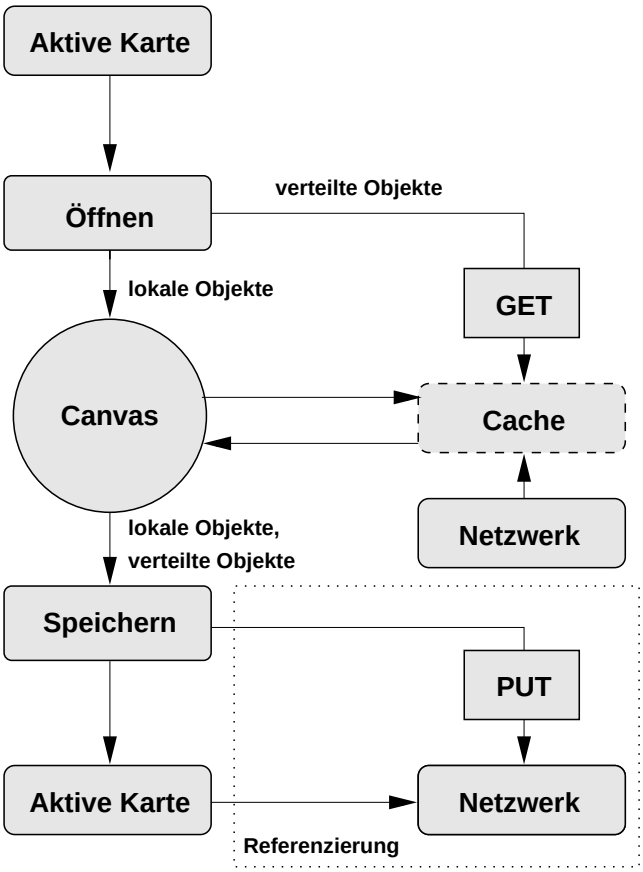

Abb. 9.2: Modell verteilter Objekte, unter Verwendung von Objektgraphik mit dezentralen Daten mittels der Kernkomponente `actmap`

9.3.2. Zyklus des Datenzugriffs

Der Zyklus des Datenzugriffs bei lokalen und verteilten Objekten kann daher im einfachsten Fall folgendermaßen beschrieben werden.

Der geöffnete Datensatz dient als Datenbasis für die Auflösung von Referenzen auf die nötigen lokalen und nicht lokalen Objekte.

GET und PUT Methoden setzen die Kommunikation mit einen WebServer voraus.

Nach dem Holen der verteilten Objekte mit der GET Methode von einem WebServer werden diese in einem Zwischenspeicher (engl.: „cache“) gespeichert und liegen dann in einer Instanz lokal vor.

Beim Speichern eines Datensatzes werden alle lokalen Objekte und die Referenzen auf verteilte Objekte gespeichert. Parallel dazu werden die Instanzen der verteilten Objekte mit der PUT Methode auf die Netzwerkrechner zurückgeschrieben.

Für diese Aufgaben können sehr effizient Tcl/Tk oder Perl Skripten eingesetzt werden.

9.3.3. Sicherheit und Performanz

Der in Abbildung 9.2 zusammengefaßte Bereich von PUT, Referenzierung und Netzwerk umschließt eine Fülle möglicher Sicherheitsprobleme, die sich auf Server- und auf Klientenseite auswirken. Dies wird mit einer Erweiterung von verteilten Objekten auf verteilte Ereignisse (engl.: „distributed events“) weiter drastisch verschärft, wenn am Einsatzort nicht geeignete Vorkehrungen getroffen werden. In diesem Zusammenhang ist mit dem heutigen Stand der technischen Entwicklungen der physikalische Aufbau des verwendeten Netzwerkes und dessen Betreuung sowie die Verwendung von vertrauenswürdigen Skripten (engl.: „trusted scripts“) und die Definition von vertrauenswürdigen Nutzern (engl.: „trusted users“) in vielen Fällen eine gangbare Lösung. Methoden, die die Sicherheit weiter verbessern, werden in den nächsten Jahren im Internet erarbeitet [SYT1998] [Zua1996].

Aus Sicht der Performanz liegt der derzeit hinderlichste Grund in der Übertragungsdauer der verteilten Objekte. Dieses ließe sich z. B. durch Optionen beim Öffnen eines Datensatzes mindern, die eine Übertragung verteilter Objekte erst bei Bedarf ermöglichen. Damit ist aber *keine* Echtdarstellung (z. B. WYSIWYG) zur gleichen Zeit möglich, eher ein abstraktes WYSIWYM.

Zu diesen Problemen tritt bei der Verwendung von verteilten Objektgruppen das zusätzliche Problem rekursiver Inklusionen auf, das mit ähnlichen Erscheinungen verbunden ist, wie die technisch im Internet bisher nicht zufriedenstellend gelösten Probleme mit endlosen weitergeleiteten (engl.: „forwarded“) Mail-Rekursionen.

Desweiteren müssen sensible Daten, insbesondere außerhalb eines Intranets, auf speziell eingerichteten Leitungen transportiert werden. Derzeit kann für die Realisierung z. B. Secure Sockets Layer (SSL) und eine offene Implementierung von Pretty Good Pricacy (PGP) verwendet werden.

9.3.4. Administration

Aufgrund der umfangreichen funktionellen Möglichkeiten ist eine Administration und Konfiguration erforderlich.

Das folgende Kontextdiagramm (Abbildung 9.3) zeigt die Wechselwirkungen zwischen dem System und den Nutzern auf einem sehr hohen Niveau.

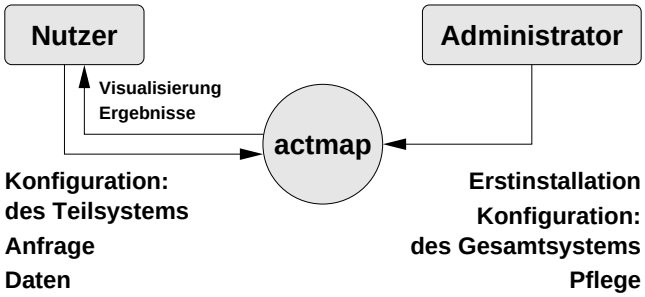

Abb. 9.3: Kontextdiagramm der entwickelten Kernkomponente `actmap`

Bei einer Netzwerkinstallation übernimmt der Administrator spezifische Aufgaben, wie Erstinstallation, Konfiguration und Pflege des Systems.

Für den Nutzer der Installation ist zusätzlich eine weitreichende Konfiguration möglich, mit der für Projekte, Datensätze oder eigene Erweiterungen sowohl Daten und Anfragen als auch Visualisierung und andere Aufgaben individuell gestaltet werden können.

Bei dem derzeitigen Stand ist dazu aber ein Verständnis grundlegender Programmierung in der eingesetzten Sprache (Tcl/Tk) Voraussetzung.

9.3.5. Implementationsstruktur

Das Diagramm in Abbildung 9.4 (Seite 71) zeigt die Implementationsstruktur anhand verschiedener exemplarischer Teile im Zusammenspiel mit der Kernkomponente `actmap`. Dargestellt sind lediglich Teile, die in Tcl/Tk und C/C++ implementiert sind. Zwischen der Kernkomponente mit ihren ladbaren Bibliotheken und externen Applikationen existiert zudem eine Tcl Schnittstelle („Kleber“, engl.: „glue“). Für die nicht dargestellten Teile, wie z. B. in Perl implementierte Erweiterungen, existieren vergleichbare Zusammenhänge.

Diese Darstellung veranschaulicht die Möglichkeiten, die implizit bestehen, um dieses System zu erweitern oder über eine Konfiguration hinausgehend speziell anzupassen.

Die Kernkomponente kann verschiedene Bibliotheken und Erweiterungen laden. Diese sind in einem Interpreter (z. B. `prowish`) lauffähig. Diese Teile der Applikation können in Tcl/Tk bzw. Bytecode beschrieben sein.

Der Interpreter kann mit verschiedenen Erweiterungen ausgestattet sein und läuft auf dem verwendeten System. Diese Interpreter sind in der Regel in einer Hochsprache geschrieben und können durch geeignete Eigenentwicklungen erweitert werden.

Aufgrund der dynamischen Zugriffsmöglichkeiten der meisten verwendeten Interpreter können diese dynamisch ladbare Bibliotheken zur Laufzeit nutzen. Auf deren Funktionen kann über den Interpreter auch von der Kernkomponente und ihren Erweiterungen aus zugegriffen werden.

Für die meisten konkreten Anwendungen wird auf diese Weise eine flexible Zusammenarbeit der spezifischen funktionalen Einheiten ermöglicht.

Die beiden dunkleren Kästen geben die minimalen Voraussetzungen für eine lauffähige Kernkomponente an. Minimal wird also eine reine Tcl/Tk Bytecode Version der Kernkomponente `actmap` und ein zugehöriges lauffähiges Programm, der Interpreter (`prowish`), auf dem System benötigt.

Auf diese minimalistischen Anforderungen wurde konsequent hingearbeitet, um die Kernkomponente möglichst flexibel und portabel zu halten und keine zukünftigen Erweiterungen auszuschließen.

9.4. Entwicklung und Support

9.4.1. Verwendete Versionen

Innerhalb des Entwicklungszeitraums konnten verschiedene Tcl/Tk Versionen eingesetzt werden (`wish4.2`/`tclsh7.6`, `wish8.0`/`tclsh8.0`, `wish8.2`/`tclsh8.2`, `wish8.3`/`tclsh8.3`). Aufgrund der Evolution von Tcl/Tk und möglicher Erweiterungen war insbesondere der Übergang auf die 8.x Versionen (speziell 8.3) zweckmäßig.

Die laufende Entwicklung wurde zusätzlich z. T. unter aktuellen Versionen erweiterter Varianten getestet. Dazu zählen `tixwish`, `wishx`, `vtk`, Tcl-Plugin, ICE `tclsh`/`wish` (kommerziell), Scriptics Corporation `prowish`/`protclsh` (kommerziell, seit kurzer Zeit frei verfügbar) und andere.

Da viele der grundlegenden Funktionen in reinem Tcl/Tk implementiert werden konnten, sind weite Teile unter allen, auch den älteren Versionen dieser Werkzeuge verwendbar. Neuere Interpreter stellen aber insbesondere umfangreichere und verbesserte Funktionen

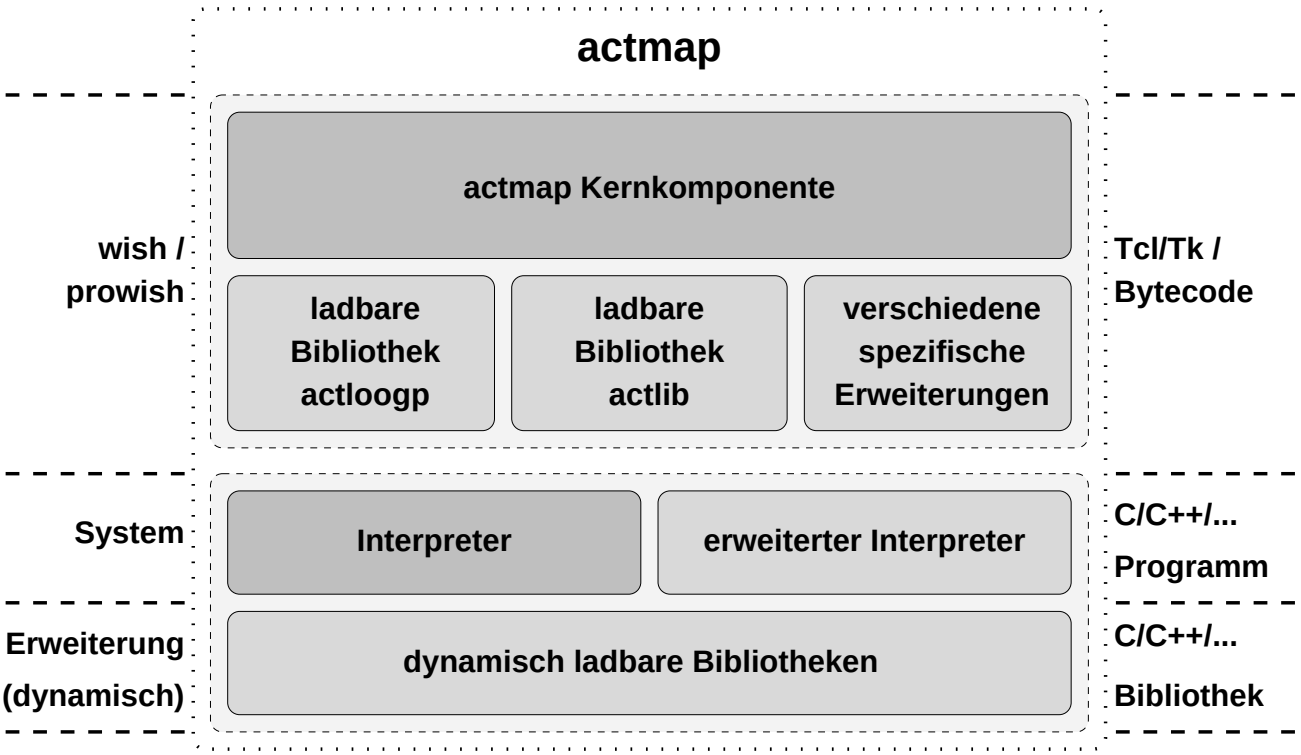

Abb. 9.4: Implementationsstruktur im Zusammenspiel mit der entwickelten Kernkomponente `actmap`

für die Nutzung und Erweiterung derartiger Applikationen bezüglich Konfiguration und spezieller Anwendungsfälle zur Verfügung.

Einige Erweiterungen der Interpreter wurden erst im Laufe des Projektes verfügbar, einige über längere Zeit nur als Patches und in letzter Zeit auch integriert in aktuelle Versionen. Viele Erweiterungen von Tcl/Tk sind für die Zukunft sehr wahrscheinlich [Zer2001b]. Darunter fällt u. a. eine stärkere Ausrichtung auf XML.

Dies betrifft z. B. verbesserte Handhabung von Ereignissen, Erweiterung des Canvas Elements (engl.: „canvas widget“), Verbesserung und Erweiterung des PostScript Exports sowie weitere Funktionen. Nützliche Erweiterungen des Tk, z. B. gestrichelte und unterbrochene Außenlinien von Objekten und implizite PostScript Ausgabe für Graphiken und Fenster, sind erst ab Tk Version 8.3 fest integriert.

Tcl 8.x ist vollständig ereignisgesteuert. Das bedeutet: Sobald ein Skript beendet ist, startet die Applikation eine Ereignisschleife, die kompatibel zur Tk Ereignisschleife ist. Beispielsweise kann so Tk dynamisch in eine Tcl Shell geladen werden. Diese entspricht dann dem Aufruf von `wish`.

Tcl/Tk kann weiterhin in den neueren Versionen mit jedem ANSI und K&R kompatiblen C oder C++ Compiler kompiliert werden.

Für die endgültige Implementierung wurden alle für die Demonstration und Nutzung relevanten Teile als offener Quellentext belassen, die restlichen Teile der Maschinerie als verpackter Bytecode mit dem ursprünglich kommerziellen und inzwischen frei verfügbaren TclPro erzeugt.

Das Augenmerk kann so auf die hier interessanten Aspekte konzentriert werden.

Dieses Vorgehen hat u. a. den angenehmen Nebeneffekt einer unkomplizierten Installation der relevanten Teile auf einem Zielsystem.

Auch die Teile in Bytecode können gegebenenfalls entpackt und auf anderen Plattformen genutzt werden, auf denen ein geeigneter Bytecode Interpreter zur Verfügung steht. Der TclPro Bytecode Interpreter stand die ersten Jahre nicht in den Quellentexten zur Verfügung.

Dies war aber für die Entwicklung prinzipiell keine Einschränkung, da der Bytecode *nur* unter den hier gegebenen Randbedingungen genutzt wurde.

Der TclPro Bytecode Interpreter steht mit seinen Erweiterungen seit 2001 im Quellentext auf SourceForge[2] im CVS bereit.

[2] `http://sourceforge.net` ..... [V: k. A.] [Ä: k. A.] [Z: 01.07.2001]

9.4.2. Unterschiedliche Plattformen

Die Entwicklungen mittels Tcl/Tk sind prinzipiell portabel. Es gibt jedoch Fälle, in denen Teile der entwickelten Software nicht portabel sein *können*. Dies ist beispielsweise der Fall, wenn grundlegende Unterschiede zwischen Plattformen bestehen, also etwa Funktionen auf einer Plattform nicht vorhanden sind oder anders realisiert werden müssen.

Desweiteren sind kommerzielle Teile denkbar, die nur in bestimmten Versionen vorhanden sein sollen.

Solche Systemspezifika und kommerzielle Teile werden beispielsweise

- in Funktionen oder Klassen gekapselt oder
- über Präprozessor- oder Laufzeitanweisungen behandelt.

Für Tcl/Tk ist das bei TclPro enthaltene Werkzeug `procheck` ein gutes Hilfsmittel mögliche grundlegende Portabilitätsprobleme zu identifizieren.

9.4.3. Wiederverwendbarkeit

Als grundlegende Basis existieren folgende Konzepte für die Verfügbarkeit eines Softwareproduktes.

„weiß“ (engl.: „white boxes“): Alle Teile liegen als Quellentext vor. Eine Beschäftigung der Nutzer mit den Quellentexten ist in vielen Fällen sinnvoll für das Verständnis und Anpassungen an die speziellen Bedürfnisse.

„schwarz“ (engl.: „black boxes“): Nur Binärversionen sind verfügbar. Quellentexte sind nicht für die Öffentlichkeit gedacht und können nicht modifiziert werden. Eine feste Programmierschnittstelle (API) bindet die Entwickler untereinander und isoliert sie von Details der Quellenseite.

Die Wahl eines Konzept hat nicht nur erheblichen Einfluß auf die Vermarktung, sondern auch auf den möglichen Anwenderkreis. Eine Entwicklung, bei der beliebige Kombinationen vorliegen und der Anwender selbst, je nach Projekt, den Anteil an offenen und binären Anteilen wählen kann, ist mit einem Modell sichergestellt, wie es für den entwickelten Prototyp entworfen wurde.

9.4.4. Weitere Aspekte

Für eine n-Schicht Architektur spielen Aspekte, wie Entwicklungsmodell (weiß/schwarz), Größe der Entwicklergruppe, Erfahrungen der Entwickler/ Entwicklergruppe, Zeitvorgaben, Consulting/Support, API etc. eine nicht zu unterschätzende Rolle bei der Planung und Umsetzung eines Entwicklungsvorhabens.

Hinsichtlich der Konzeptionierung sind besonders die Anforderungen an die gewünschte Komponente/ Gesamtsystem und der Aufwand für das Consulting zur Zusammenstellung von Komponenten zu speziellen Gesamtsystemen zu erwähnen.

9.4.5. Prinzipielle Grenzen

Die Einschränkungen des heutigen WWW bezüglich Zugriffszeiten, Speicherbeschränkungen, Inkompatibilitäten sowie die Problematik des Austauschs von Datenmaterial bezogen auf Copyright, Verläßlichkeit und der Wert von Investitionen beschränken gerade im Bereich der Forschung die Nutzung interaktiver und dynamischer Graphiken in öffentlichen Bereichen. Die Bedeutung dieser Probleme wird jedoch erheblich reduziert, wenn der Zugriff auf eine kleinere Gruppe von Nutzern beschränkt wird.

Abgesehen von diesen Hindernissen existieren aber auf der Seite der Entwicklung weitere Probleme:

- Es sind nicht nur Komponenten integriert bzw. integrierbar. Die Verfügbarkeit externer Anwendungen ist beispielsweise nicht auf allen Systemen sichergestellt.
- Wartung und Dokumentation erfordern einen nicht zu unterschätzenden Aufwand.
- Verschiedene Teile, z. B. Komponenten, Module oder eigenständige Softwareprodukte, werden von Herstellern entwickelt, die nicht auf geeigneter Ebene zusammenarbeiten.
- Es existieren unterschiedliche Konzeptionen der Datenmodelle.
- Die Verfügbarkeit von geeigneten Komponenten ist nicht immer sichergestellt.

Die wichtigste Einschränkung für das laufende Projekt ergibt sich aus dem Umfang und den Zeitvorgaben.

9.5. Grundeigenschaften des Prototyps

9.5.1. Vorteile der Komponentenebenen

Durch die vorgenommene Aufteilung in Komponentenebenen besteht die Möglichkeit für folgende Gesichtspunkte.

- Erhöhung von Performanz und Stabilität.
- Möglichkeiten für verteilte Entwicklung.
- Es ist jedes Datenmodell implementierbar.
- Das GUI kann vollständig untergeordnet werden.
- Es stehen sehr leistungsfähige Programmiersprachen zur Verfügung.
- Integration externer Entwicklungen.
- Nutzung von Skripting-Funktionen.
- Effizientere Umsetzung fachlicher Funktionen.
- Vereinfachung des Datenaustausches und der Datenaufbereitung.
- Vereinfachter Direktzugriff auf Daten.

9.5.2. Funktionalität

- Ereignissteuerung!
- Einbindung bestehender Komponenten.
- Max. Anzahl darstellbarer Punkte $>$100000, max. Anzahl verwendbarer Objekte $>$20000.
- Raster-, Vektor-, Objekt- *und* Funktionen-Daten.
- Verknüpfung Objektdaten, Ereignisse . . .
- Netzwerkfunktionalitäten.
- Makrosprache und zusätzliche Shell-Steuerung.
- Performanz/Eignung (Mehrschicht).

9.5.3. Grobaufbau

- Schichtung der Komponenten.
- Graphische Oberfläche.
- Shell.
- Backend.
- Skripting.
- Technische Erweiterbarkeit.
- Kapselung von Code.

Viele dieser Punkte sind direkte Umsetzungen aus den Forderungen des vorgestellten Konzepts.

Durch die Verwendung eines Interpreters sind die gesamte Oberfläche und alle wichtigen Funktionen durch Skripten steuerbar und erweiterbar.

Die Integration eines Interpreters in eine Applikation vervielfacht zudem in aller Regel die Funktionalität und Flexibilität [Sch2000a]. In den meisten Fällen entstehen dadurch Synergieeffekte, die neue Möglichkeiten eröffnen. Durch die Schichtung, d. h. die strikte Trennung von Vordergrundprozessen, wie der Oberfläche von Hintergrundprozessen, dem sogenannten *Backend*, ist eine Klient-Server Architektur vorhanden, die sehr leicht durch den Anwender für eigene Aufgaben genutzt werden kann.

Mit Architektur ist die Spezifikation der grundlegenden Struktur eines System gemeint. Ein Architektur-Prototyp ist ein Prototyp, der dazu beiträgt, die prinzipielle Einsatz- und Funktionsfähigkeit einer technischen und fachlichen Architektur nachzuweisen. Bei einem Entwicklungsprozeß, der bereits bei der Entwicklung die speziellen Gegebenheiten der Architektur, wie Konzepte, Abstraktionen und Artefakte berücksichtigt, spricht man von einem architekturzentrierten Entwicklungsprozeß. Dies trifft vor allem auf die realisierten Prototypen zu.

9.6. Datensprache als Teil einer Komponente

9.6.1. Quellentext und Datensprache

Das Speichern im Quellentext als Datensprache wurde in dem entwickelten Prototyp in der Tcl-Syntax realisiert. Zum Speichern wird vom Canvas Element (engl.: „canvas widget“) der aktuelle Inhalt an Objekten abgefragt und dann eine Beschreibung der Objekte in einer Datei abgelegt.

Die Beschreibung könnte bezüglich Form und Formatierung auf viele individuelle und proprietäre Arten vorgenommen werden. Die konventionellen Formate sind aber in der Regel meist nicht nur proprietär sondern auch nicht intuitiv. So ist nicht immer ohne weiteres erkennbar, welcher Teil einer Beschreibung eines Objektes mit welcher Eigenschaft in Beziehung steht.

Beispiele für konventionelle Daten wurden bereits in Abschnitt 1.3 (Seite 10) unter Punktdaten, Liniendaten und Polygondaten exemplarisch dargestellt.

Es ist deutlich ersichtlich, daß Funktionen zum Laden solcher Daten nicht intuitiv sein können.

Ein Datensatz auf Basis von Quellentext kann im einfachsten Fall hingegen mit einem `source` Befehl oder mit einer Funktion geladen werden, wie sie in Abbildung 9.5 dargestellt ist.

```
proc myopenSource {FileName} {
global w
  source $FileName
}
```

Abb. 9.5: Anwendungsbeispiel: Benutzerdefinierte Funktion zum Laden von Quellentext-Daten in einer Komponente

Für bestimmte Funktionalitäten, z. B. Handhabung global vereinbarter Definitionen oder der temporären Speicherung der Namen geladener Datensätze, können zusätzliche einfache Mittel in einer solchen Funktion zur Verfügung gestellt werden. Globale Variablen, wie hier `$w` bzw. `w` *können* deklariert werden, wenn die geladenen Datensätze globale Variablen enthalten, z. B. für die Angabe eines Canvas, die nicht als Funktionsargument übergeben werden.

Verschiedene Funktionen können für verschiedene Aufgaben parallel bereitgestellt und bei Bedarf auch dynamisch nachgeladen oder ausgewechselt werden. In diesem Beispiel beginnen die Funktionsnamen mit `my` damit keine bereits vorhandenen, systemeigenen Funktionsnamen umdefiniert werden.

Eine Variante dieser Funktion, zur Nutzung eines sicheren Interpreters, kann ebenso einfach erzeugt werden (Abbildung 9.6).

```
proc mysopenSource {FileName} {
if { [catch {
  mySafeOpenSource $FileName
  } tmpcatch ] } {
  puts "WARNING:     $tmpcatch"
  puts "DIAGNOSTICS: We have NOT\
        executed some commands\
        as these might be unsafe."
  puts "TIP:         You might\
        inspect $FileName\
        for further action."
  } else {
  }
}
```

Abb. 9.6: Anwendungsbeispiel: Benutzerdefinierte Funktion zum Laden von Quellentext-Daten über sicheren Interpreter in einer Komponente

Es wird der Name eines Datensatzes übergeben und an eine zugehörige Funktion weitergegeben, deren Warnungen angezeigt werden.

Dabei wird über die folgende zugehörige Funktion ein einfacher sicherer Interpreter verwendet (Abbildung 9.7), der die Anweisungen in den Datensätzen auswertet und über einen `alias` Mechanismus vereinbarte zulässige Elemente erkennen kann.

```
proc mySafeOpenSource {FileName} {
global w

set canvParserSource \
  [interp create -safe]

proc canvas_parser_cmd {w args} {
  eval $w $args
}

$canvParserSource alias \
  .f.sub.c canvas_parser_cmd $w

set F [open $FileName r+]
set ToBeSafeScript [read $F]
close $F

$canvParserSource eval \
  set w .f.sub.c
$canvParserSource eval \
  $ToBeSafeScript
interp delete $canvParserSource
}
```

Abb. 9.7: Anwendungsbeispiel: Benutzerdefinierte Nutzung eines sicheren Interpreters in einer Komponente

Enthält ein geöffneter Datensatz Anweisungen wie `source` oder `exec`, so werden diese nicht ausgeführt und stattdessen wird eine Warnung erzeugt, die von der aufrufenden Funktion ausgegeben wird. Alle kritischen Sprachelemente sind durch den sicheren Interpreter abgedeckt. Eine Erweiterung um individuelle Befehle, die beispielsweise benutzerdefiniert als Tcl/Tk Funktion implementiert oder als Routine in den verwendeten Interpreter integriert ist, wäre auf Basis eines sicheren Interpreters ebenso einfach möglich.

In ähnlicher Weise kann der Benutzer Funktionen zum Speichern von Daten sehr einfach für spezifische Gegebenheiten selbst gestalten. Eine nützliche Funktionalität ist z. B. die Auswertung eines Attributs bestimmter Objekte, z. B. `canvas_save_ignore`, welches bewirkt, daß das Speichern dieser Objekte unterbunden wird. Dies kann z. B. für Objekte zum Einsatz kommen, die nur temporär in einer interaktiven

Darstellung vorhanden sein sollen.

Daten, die durch solche Funktionen hantiert werden können, wurden für den Prototyp als „GIS Active Source“ (GAS, `.gas`) bezeichnet.

Die Nutzung eines teilweise nativen Formates ist ebenso möglich, erfordert aber etwas mehr Aufwand, da möglicherweise einige Elemente erst durch Funktionen zusammengebaut werden müssen (dies wird auf Seite 78 näher erläutert).

Eine weitere Möglichkeit ist ein Speicherabzug (engl.: „dump“) des aktuellen Canvas. Dabei werden alle notwendigen Informationen über Objekte, Eigenschaften usw. abgegriffen. Diese Informationen sind dann beliebig zu verarbeiten, also z. B. in eine Datei schreibbar oder in einer Variable oder einem Datenfeld speicherbar. Auf diese Art läßt sich beispielsweise ein beliebiger Zustand des Canvas später wiederherstellen. Dies kann bei Funktionalitäten wie „Rückgängig machen“ (engl.: „undo“), aber auch flexibler für Schnappschüsse von Bearbeitungsschritten oder Überlagerungen von Zuständen Anwendung finden.

Für den Prototyp wurde dieser Typ von Daten als „Canvas Daten“ (CAN, `can`) bezeichnet.

9.6.2. Für und Wider

Der Inhalt einer jeden Darstellung läßt sich als eine Folge von Tcl/Tk Befehlen speichern. Dies steht in Analogie zu der Nutzung der Sprache PostScript in den meisten Druckern, die jeden Druckauftrag als Programm erhalten und dieses zur Darstellung ausführen.

Zum Laden der Daten in einem solchen Datensatz ist im einfachsten Fall lediglich eine Funktion zum Ausführen der Datei und zum Auswerten der enthaltenen Objekte notwendig.

Die Verwendung von Programm-Quellentext zur Speicherung räumlicher und anderer Daten hat eine Reihe herausragender Vorteile:

Ausdrucksstärke: Das Format ist sehr ausdrucksstark und auch für Anwender weitestgehend intuitiv.

Format: Das Format ist „offen“.

Erweiterbarkeit: Das Format ist flexibel und sehr leicht erweiterbar, z. B. durch beliebige Funktionen und Ereignisse.

Portabilität: Daten in einem solchen Format weisen einen hohen Grad an Portabilität auf.

Struktur: Es handelt sich um eine Datensprache.

Übersichtlichkeit: Bei der Nutzung eines solchen Formats muß für Applikationen deutlich weniger Quellentext geschrieben werden.

Wiederverwendbarkeit: Alle Teile in dieser Datensprache sind hervorragend wiederverwendbar.

Datenhaltung: Teile von Daten, Ereignisse oder Funktionen sind leicht in Datenbanken zu verwalten.

Entwicklungsumfeld: Es existieren sehr viele frei verfügbare Werkzeuge und Anwendungen, welche die Sprache des Formats nutzen. Ein großer Personenkreis ist bereits mit der Sprache vertraut.

Dokumentation: Das Format ist gut dokumentiert.

Als Nachteil ist zu nennen, daß diese Daten interpretiert werden. Dies läßt sich für umfangreichere Datensätze derzeit aber dadurch lösen, daß entweder Datenbanken die gerade benötigen Daten liefern und rechenintensive Funktionen z. B. in C implementiert werden.

Bei den durchgeführten Anwendungen war dies jedoch selbst bei Einsatz derzeitig durchschnittlicher Hardware selbst bei Datensätzen mit hunderttausenden von Vektor-Datenpunkten und auch bei großen Rasterdaten nicht notwendig.

9.6.3. Sicherheitsaspekte

Eine derartig flexible und universelle Datensprache eröffnet alle Möglichkeiten einer modernen Programmiersprache. Damit ergeben sich aber auch Gefahren, z. B. bei der Nutzung von Daten aus unsicheren Quellen, denn diese Daten sind nichts anderes als Programme.

Es muß also möglich sein, solche Programme in einer sicheren Umgebung auszuführen, z. B. einer speziellen virtuellen Maschine oder einem sicheren Interpreter (engl.: „interpreter“). Tcl stellt ein einfach zu handhabendes aber sehr flexibles Konzept für die Verwendung sicherer Interpreter implizit bereit. Dies unterstreicht die Verwendung von Tcl für einen derartigen Zweck. Es ist auf diese Weise leichter zu verhindern, daß alle Befehle, die eine mögliche Gefahr für das laufende System oder Teile davon darstellen *könnten*, ausgeführt werden. Stattdessen ist eine beliebige Aktion auslösbar, wie z. B. die Ausgabe einer Warnung.

In diesem Zusammenhang ist es sinnvoll darauf hinzuweisen, auch andere mögliche Sicherheitsvorkehrungen zu nutzen, wie bezüglich X11 (`xhost`, `xauth` etc.) sowie TCP Restriktionen und Filter (engl.: „TCP

wrapper“) und dergleichen [Har1997] [RT1999]. Es gelten die üblichen Empfehlungen zur Systemsicherheit, angefangen bei den Hinweisen sichere Passworte zu verwenden und nicht benötigte Dienste abzuschalten.

9.6.4. Erweiterbarkeit der Oberfläche

Die Oberfläche einer Komponente wie `actmap` ist auf vielfältige Weise modifizierbar und erweiterbar.

Beispielhaft kann eine datenbezogene Erweiterbarkeit der Oberfläche bezüglich Information, Steuerung u. ä. über entsprechende benutzerdefinierte Arbeitsflächen vorgenommen werden. Ein einfaches Beispiel dafür ist `textover`, das für beliebige Daten überlagernd einblendbar ist.

Vergleichbare Erweiterungen sind vom Anwender auf einfache Weise erstellbar und für bestimmte Datensätze leicht zu modifizieren.

Daten von diesem Typ werden in Verwendung mit dem Prototyp als „GIS Text Overlay“ (GTO, `.gto`) bezeichnet.

Vorteile einer solchen Vorgehensweise sind eine sehr einfach handhabbare Modularisierung und Erweiterbarkeit. Für spezielle Fälle ist dies hilfreich, weil Daten auch Teil der Applikation sein können und umgekehrt.

Die große Flexibilität *kann* andererseits auch ein Nachteil sein. Die Auflösung der Trennung verschiedener Daten von Applikationsteilen kann bei einer sehr extensiven Nutzung unübersichtlich sein, wenn keine weiteren Verfahren zur Verwaltung der relevanten Fragmente zum Einsatz kommen.

9.6.5. Kommunikation mit verschiedenen Modulen

Mit einer Komponente wie `actmap` können fast beliebige Verfahren zur Kommunikation mit anderen Programmen auf einfache Weise genutzt werden.

Bei einem synchronen Kommunikationsschema wird von einer Komponente einer Applikation eine Anfrage an eine andere Komponente gesendet und *solange nichts weiteres* ausgeführt, bis diese andere Komponente eine Antwort zurücksendet.

Bei einem asynchronen Kommunikationsschema wird von einer Komponente einer Applikation eine Anfrage an eine andere Komponente gesendet und eine Reihe *bestimmter Befehle ausgeführt, während* auf die Antwort dieser anderen Applikation gewartet wird.

Wie bei einer Datensprache kann es auch für solche Zwecke notwendig sein, einen sicheren Interpreter zu verwenden, der solche Antworten auswertet.

Einige Beispiele für die unterschiedliche Anwendung solcher Kommunikationsschema im Rahmen des entwickelten Prototyps sind im folgenden aufgeführt.

`asrv` kann als laufendes Tcl Skript in der Anwendung zum Einsatz kommen, um einen sehr einfachen Server zu simulieren. Diese Vorgehensweise kostet zwar erheblich Ressourcen, ist aber sehr generisch und anschaulich.

Das Modul `colodial` zur Auswahl von Farben auf einer Wählscheibe läuft synchron und nutzt die gleiche Shell interaktiv mittels einer `refresh` Funktion.

Aufgerufene Anwendungen, z. B. aus Datensätzen heraus, lassen sich über `exec` synchron oder asynchron in einer separaten Shell benutzen.

Dies ist erforderlich, weil für bestimmte externe Aufgaben, z. B. in bestimmten Datensätzen, eine parallele Nutzung der Kernkomponente wünschenswert ist, wie z. B. bei der Betrachtung oder Bearbeitung bestimmter Rastergraphiken. So kann beispielsweise ein interaktiver Aufruf von `gimp` zur Bearbeitung einer Graphik erfolgen, ohne die Kernanwendung für den Zeitraum der Bearbeitung stillzulegen.

Andererseits sollen sicherlich nicht alle Aufgaben asynchron mit der Anwendung ausgeführt werden *müssen*.

Einige Datenbankzugriffe, z. B. mittels Perl Skripten, erfordern aus logischen Gründen einen synchronen Zugriff.

So kann z. B. erzwungen werden, daß zunächst eine Rückgabe des Bearbeitungsfokus an die Kernkomponente stattfindet, bevor dieser wieder verwendbar ist.

9.6.6. Prozeßkommunikation

Die Komponente `actmap` unterstützt Kommunikation zwischen Prozessen (IPC). Unter X11 ist IPC über das Tcl/Tk Kommunikationsprotokoll implementiert. Unter Windows nutzt Tcl/Tk seit Version 8 als Ersatz DDE [Zer1999].

Ein Beispiel für eine Fernsteuerung von Objekten in laufenden Applikation `actmap` und `actsea` zeigt das folgende Quellentextfragment (Abbildung 9.8).

```
send {actmap} $w move germany \
       50 50
send {actsea} .text insert \
       1.0 CPR
send {actsea} {.text insert \
       5.6 \
       "some linebreaks,\n\ntoo"}
send {actmap #2} { \
       $w move germany 150 50 ;\
       $w move france 50 50}
```

Abb. 9.8: Anwendungsbeispiel: Beispiele für Benutzerbefehle mit der entwickelten Kernkomponente `actmap` und IPC

Das unter dem entsprechenden Namen registrierte Programm erhält auf diese Weise Daten (`send`). Über diese Daten können Tcl/Tk Befehle oder auch applikationsspezifische Anweisungen übertragen werden.

Dieses Verfahren erlaubt damit u. a. eine Fernsteuerung der Applikationen.

9.6.7. Dynamische Visualisierung

Neben der vollständigen Ereignissteuerung aller Teile der Komponente sind über die Nutzung eigener Programmierung auch beliebige dynamische Visualisierungen realisierbar.

Es lassen sich prinzipiell beliebige Darstellungen in effizienter Weise umsetzen, also auch Diagramme oder komplexe Visualisierungen mit beliebigen Abhängigkeiten.

Beispielsweise können Datenpunkte, Symbole, Kurven oder andere Teile von Darstellungen spezielle Eigenschaften erhalten und so auf Interaktionen reagieren.

Da die erzeugten Visualisierungen bei gezielter Erstellung plattformunabhängig und anwendungsunabhängig sind, sind sie sehr flexibel einsetzbar. Sie lassen sich, je nach Eigenschaften, auch in andere Komponenten integrieren.

9.6.8. Objektorientierte Datensprache

Zum Vergleich mit einer nativen Datensprache, die auf Tcl/Tk basiert, wurde eine objektorientierte (OO) Datensprache entworfen und ist über eine STOOOP Implementierung (Tcl) nutzbar. Diese Umgebung wurde gewählt, da sie mit geringem Aufwand portiert und praktisch mit jedem Tcl/Tk basierten Interpreter genutzt werden kann.

In dieser objektorientierten Datensprache sind die wichtigsten graphischen Primitive (Linie, Oval, Polygon etc.) sowie der Zugriff auf Rastergraphiken in Klassen gekapselt.

Objekte haben Eigenschaften und Methoden (Abbildung 9.9). Die Eigenschaften sind Werte, die das Objekt beschreiben. Die Methoden sind Funktionen des Objekts zur Manipulation von Eigenschaften und anderen Werten.

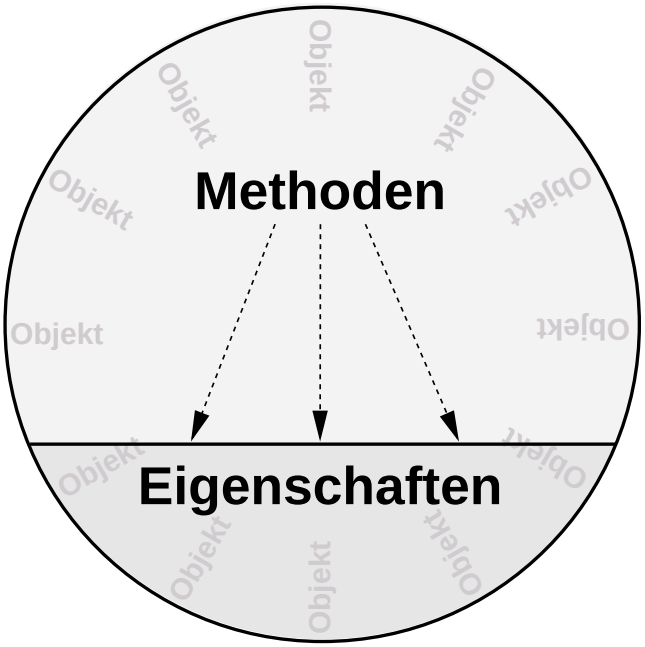

Abb. 9.9: Objektgraphik: Ein Objekt in objektorientierter Datensprache besteht aus Eigenschaften und Methoden.

Jede Klasse enthält mehrere Methoden zur Behandlung von Objekten.

Die zu betonenden Vorteile der objektorientierten Datensprache sind:

- Die Objektorientierung erlaubt eine Kapselung in Klassen.
- Das Format im Quellentext ermöglicht eine einfache syntax- und implementierungskonsistente Erweiterung, um neue Objektklassen und Objektmethoden.
- Durch die Bildung von Klassen ist ein einfacher transparenter Zugriff auf zusammengesetzte Objekte möglich.
- Aufgrund der Kapselung ist in den Daten ein höherer Grad an Abstraktion und Strukturierung erreichbar.

Diese Vorteile werden begleitet von folgendem Mehraufwand bzw. Nachteilen:

- Die Definition von Klassen und Methoden ist notwendig.
- Es gibt keine native Unterstützung benutzerdefinierter Klassen und Methoden. Das kann sich z. B. in graduell geringerer Geschwindigkeit bemerkbar machen.
- Daten enthalten einen geringerer Grad an selbsterklärenden Elementen durch die Definition neuer Elemente.
- Daten sind nicht mehr nativ ausgedrückt durch Basiselemente einer Sprache, sondern die Nutzbarkeit ist implementierungsabhängig.

Der wichtigste Vorteil der Abstraktion und Erweiterbarkeit steht damit einer implementierungsabhängigen Nutzbarkeit entgegen.

Aufgrund der Implementierung können bei Bedarf Elemente der objektorientierten Datensprache mit der konventionellen Tcl/Tk basierten Datensprache kombiniert werden. Darunter können konventionelle Objekte fallen, beispielsweise aber auch eine beliebige Anzahl von Prozeduren.

Eine Bewertung wird je nach Anwendungsfall unterschiedlich ausfallen.

9.6.9. Beispiel: Teilweise native Datensprache

Wenn Daten basierend auf einer nativen Datensprache aufgebaut werden, kann der Anwender beliebige Aufgaben der Komponente überlassen, die mit diesen Daten genutzt wird.

Auf diese Weise ist es möglich, Daten zu erzeugen, die teilweise oder vollständig frei strukturiert und formatiert sind. Für die Komponente müssen dann aber geeignete Funktionen bereitgestellt werden, die mit diesem Format umgehen können. In dieser Hinsicht besteht, bei nur teilweiser Datensprache, eine Ähnlichkeit mit der Handhabung konventioneller Daten.

Ein Beispielausschnitt aus einem solchen einfachen und gekürzten Datensatz, basierend auf der in dieser Dissertation eingeführten teilweise nativen Datensprache, ist in Abbildung 9.10 (Seite 79) dargestellt.

Derartige Daten wurden für den Prototyp als „GIS Active Map“ (GAM, `.gam`) bezeichnet.

In dem angegebenen Quellentext dargestellt ist

- . . . die Angabe eines automatisch generierten Kopfbereichs, der Informationen zum Datensatz und System sowie beliebige Kommentare enthalten kann.
- . . . die Angabe von vier Vektor-Gitterlinien.
- . . . die Angabe eines minimalen Polygonzugs.
- . . . die Angabe von drei kleinen gefüllten Kreisen mit verschiedenen Attributen.

Dieser Datensatz ist ein gültiger Datensatz, jedoch sehr stark verkürzt und vereinfacht.

Beispielsweise können auch Elemente definiert oder weitere definierte Elemente verwendet werden, wie beispielsweise in Abbildung 9.11 (Seite 79) dargestellt.

Die Pünktchen (`...`) stehen in diesem Fragment für die Auslassung von Teilen eines realen Datensatzes.

9.6.10. Beispiel: Vollständig native Datensprache

Ein Beispielausschnitt aus einem Datensatz in Objektgraphik mit vollständig nativer Datensprache ist in Abbildung 9.12 (Seite 80) dargestellt.

Derartige Daten wurden für den Prototyp als „GIS Active Source“ (GAS, `.gas`) bezeichnet.

In dem Quellentext dargestellt ist

- . . . die Angabe eines gefüllten Polygonobjekts mit Attributen (`-tags`) auf einem Canvas `$w`.
- . . . die Anbindung zweier Ereignisse an ein Attribut (`germany`) dieses Objekts. Die erste Anbindung nutzt eine interne Funktion des Prototyps, die zweite eine andere implementierte Komponente, die extern aufgerufen wird.
- . . . die Angabe eines gefüllten Ovals mit Attributen.
- . . . die Anbindung zweier Ereignisse an ein Attribut (`mess1`) dieses Objekts. Die erste Anbindung nutzt eine interne Funktion des Prototyps, die zweite eine andere implementierte Komponente, die extern aufgerufen wird.
- . . . die Skalierung aller Objekte bzw. der Koordinaten aller Punkte.

```
#====================================================================
# GIS Active Map layer -- (c) Claus-Peter R"uckemann, 1995--2001
#
# MODUL: actmap V 2.0a
# USER:  cpr
# HOME:  /home/cpr
# PWD:   /home/cpr/gisig
# TERM:  xterm
# SHELL: /bin/bash
# HOST:  toxutat
# DISP:  toxutat:0.0
# DATE:  Wed Feb 28 21:34:07 2001 MET
#====================================================================
line 0.0 0.0 0.0 10000.0 -tags {itemshape gridline} -fill grey -width 1
line 0.0 0.0 10000.0 0.0 -tags {itemshape gridline} -fill grey -width 1
line 20.0 0.0 20.0 10000.0 -tags {itemshape gridline} -fill grey -width 1
line 0.0 20.0 10000.0 20.0 -tags {itemshape gridline} -fill grey -width 1
polygon 91.012 145.236 82.368 131.592 91.012 145.236 -tags {itemshape}
oval 384.0 204.0 388.0 208.0 -tags {itemshape city muenster} -fill yellow
oval 404.0 196.0 408.0 200.0 -tags {itemshape city minden} -fill yellow
oval 372.0 224.0 376.0 228.0 -tags {itemshape city koeln} -fill yellow
```

Abb. 9.10: Datenbeispiel: `actmap` Datensatz, basierend auf der eingeführten Objektgraphik, mit teilweise nativer Datensprache

```
...
bitmap 432.0 232.0 -bitmap "@/home/cpr/gisig/images/letters.xbm" ...
...
copycut::/home/cpr/.../earth.gif copy [image create photo -file ...
...
copycut:0101-zoom:/home/cpr/.../earth.gif copy [image create photo ...
...
image 180.0 400.0 -image [image create photo "/home/.../smilee.gif" ...
...
```

Abb. 9.11: Datenbeispiel: `actmap` Datensatzfragment mit teilweise nativer Datensprache

Einige häufig wiederkehrende Teile sind vordefiniert und daher über Variablen erreichbar.

Ein Zugriff auf einzelne Elemente oder Gruppen von Elementen ist sowohl manuell als auch über ein Skript, z. B. aus der Shell beispielsweise folgendermaßen möglich (Abbildung 9.13):

```
$w move germany -150 -50
update; after 100
$w itemconfigure germany -fill red
```

Abb. 9.13: Anwendungsbeispiel: `actmap` Benutzerzugriff mittels Datensprache auf Elemente

9.6.11. Beispiel: Objektorientierte Datensprache

Das gleiche Datenfragment kann unter Nutzung des „objektorientierten“ Formates der nativen Datensprache als Objektgraphik geschrieben werden, wie in Abbildung 9.14 (Seite 81) dargestellt.

Dabei ist insbesondere zu beachten, daß die beiden Objekte als Instanzen der jeweiligen Klasse entstehen und hier in den Variablen `c` und `d` gespeichert werden. Für umfangreichere Datensätze sollten längere Namen verwendet werden.

```
$w create polygon \
1.33039 0.57027 1.36029 0.59123 1.34591 0.50223 1.34591 0.44262 \
1.33314 0.41707 1.32797 0.37687 1.30972 0.35746 1.27992 0.35320 \
1.25650 0.33617 1.24296 0.32417 1.21179 0.32979 1.19902 0.34469 \
1.17347 0.34469 1.14416 0.36309 1.12876 0.34895 1.14792 0.34043 \
1.14154 0.31488 1.11386 0.31488 1.10109 0.29998 1.07054 0.29423 \
1.03341 0.29185 1.02019 0.32766 1.02019 0.34895 1.02861 0.38514 \
1.02861 0.41158 1.00742 0.39152 0.98251 0.38164 0.98251 0.40430 \
0.96484 0.39152 0.94359 0.41876 0.93879 0.48524 0.88614 0.51518 \
0.88614 0.60021 0.88852 0.64209 0.91374 0.65977 0.89988 0.68344 \
0.99688 0.72716 0.96694 0.81694 1.07471 0.82592 1.25673 0.82354 \
1.30943 0.73852 1.21301 0.62593 1.33039 0.57027 \
-fill gold -width 1 -tags {itemshape country germany}
$w bind germany <Button-1> {showName "$text_country_name_germany"}
$w bind germany <Shift-Button-3> {exec wish actsel$t_suff}

$w create oval \
0.97 0.54 0.98 0.55 \
-fill blue -width 1 -tags {itemshape pointdata mess1}
$w bind mess1 <Button-1> {showName "Meßpunkt 1"}
$w bind mess1 <Shift-Button-3> {exec browedit$t_suff}

$w scale all 0 0 400 400
##EOF:
```

Abb. 9.12: Datenbeispiel: `actmap` Datensatz, basierend auf der eingeführten Objektgraphik, mit nativer Datensprache

Daten von diesem Typ werden in Verwendung mit dem Prototyp als „GIS Object Source“ (GOS, `.gos`) bezeichnet.

Dies ist lediglich *ein* umgesetztes Beispiel für eine objektorientierte Form derartiger Daten. Die Implementierung der zugehörigen Klassen zu solchen Daten kann auf verschiedene Weise vorgenommen werden.

In der folgenden Implementierung wurden Funktionen verwendet, die ausschließlich auf Tcl/Tk basieren (STOOOP), um die größtmögliche Flexibilität beim Einsatz von Interpretern und Werkzeugen zu gewährleisten.

Die zugehörige nötige Klasse `polygon` mit einigen Methoden für die Handhabung der Objekte zeigt der Quellentext, der in Abbildung 9.15 (Seite 81) angegeben ist.

Diese und ähnliche Klassen sind jederzeit in gleicher Weise ladbar oder nachladbar, wie alle anderen Funktionen und Daten in dieser Implementierung.

Abgebildet ist hier lediglich *ein Teil einer* Klasse `polygon` mit einem nachgebildeten Destruktor und zwei Methoden `move` und `fill`. In dieser Art von Klasse wird das Objekt direkt in das (durch ein Argument) übergebene Canvas erzeugt.

Obwohl hier nur ein kleiner Teil einer realen Klasse abgebildet ist, ist dieser Teil mit seinen Methoden bereits autark und funktionsfähig.

Diese Klassen mit spezifischen Methoden können vom Anwender nach Belieben durch eigene Klassen ersetzt oder ergänzt werden. Für verschiedene Zwecke sind so nicht nur verschiedene Sammlungen von Klassen, sondern auch verschiedene Implementierungen denkbar.

```
set c [new polygon $w \
1.33039 0.57027 1.36029 0.59123 1.34591 0.50223 1.34591 0.44262 \
1.33314 0.41707 1.32797 0.37687 1.30972 0.35746 1.27992 0.35320 \
1.25650 0.33617 1.24296 0.32417 1.21179 0.32979 1.19902 0.34469 \
1.17347 0.34469 1.14416 0.36309 1.12876 0.34895 1.14792 0.34043 \
1.14154 0.31488 1.11386 0.31488 1.10109 0.29998 1.07054 0.29423 \
1.03341 0.29185 1.02019 0.32766 1.02019 0.34895 1.02861 0.38514 \
1.02861 0.41158 1.00742 0.39152 0.98251 0.38164 0.98251 0.40430 \
0.96484 0.39152 0.94359 0.41876 0.93879 0.48524 0.88614 0.51518 \
0.88614 0.60021 0.88852 0.64209 0.91374 0.65977 0.89988 0.68344 \
0.99688 0.72716 0.96694 0.81694 1.07471 0.82592 1.25673 0.82354 \
1.30943 0.73852 1.21301 0.62593 1.33039 0.57027 \
-fill gold -width 1 -tags {itemshape country germany} ]
$w bind germany <Button-1> {showName "$text_country_name_germany"}
$w bind germany <Shift-Button-3> {exec wish actsel$t_suff}

set d [new oval $w \
0.97 0.54 0.98 0.55 \
-fill blue -width 1 -tags {itemshape pointdata mess1} ]
$w bind mess1 <Button-1> {showName "Meßpunkt 1"}
$w bind mess1 <Shift-Button-3> {exec browedit$t_suff}

$w scale all 0 0 400 400
##EOF:
```

Abb. 9.14: Datenbeispiel: `actmap` Datensatz, basierend auf der eingeführten Objektgraphik, mit objektorientierter Datensprache

```
class polygon {
  proc polygon {this canvas args} {
    set polygonpoints $args
    set polygon::($this,polygonpoints) $polygonpoints
    set polygon::($this,canvas) $canvas
    set tmppoly "$canvas create polygon $polygonpoints"
    set polygon::($this,id) [eval $tmppoly]
  }
  proc ~polygon {this} {
    $polygon::($this,canvas) delete $polygon::($this,id)
  }
  proc move {this x y} {
    $polygon::($this,canvas) move $polygon::($this,id) $x $y
  }
  proc fill {this fill} {
    $polygon::($this,canvas) itemconfigure $polygon::($this,id) \
                             -fill $fill
  }
}
```

Abb. 9.15: Anwendungsbeispiel: (ladbare) `actmap` Klasse mit Methoden

Damit kann aus einem Datensatz oder manuell bzw. über ein Skript aus der Shell z. B. folgendermaßen auf ein Objekt zugegriffen werden (Abbildung 9.16):

```
polygon::move $c -150 -50
update; after 100
polygon::fill $c red
```

Abb. 9.16: Anwendungsbeispiel: `actmap` Benutzerzugriff auf Objekt in objektorientierter Datensprache

Der Zugriff über Methoden auf das Objekt, hier das Polygonobjekt in `c`, erfolgt über die betreffende Variable, in der das Objekt abgelegt ist.

Eine Verbindung zu bestimmten Vordefinitionen oder Anbindungen ist in beiden Fällen möglich, nativ *und* objektorientiert.

Entsprechende Daten wurden in Zusammenhang mit dem entwickelten Prototyp in den meisten Fällen in separaten Dateien abgelegt.

Die beiden Datentypen wurden „Bind Daten“ (BND) und „Settings Daten“ (SET) benannt.

Abbildung 9.17 (Seite 83) zeigt einen kleinen Ausschnitt aus dem zugehörigen Klassendiagramm der graphischen Primitive der entwickelten objektorientierten Datensprache.

Das Klassendiagramm (Abbildung 9.17) zeigt einige als Generalisierungshierarchien modellierte taxonomische Zusammenhänge.

- Linie, Polygon, ... sind konkrete Formen des abstrakten Sammelbegriffs *Basiselemente*. Das Auslassungszeichen „...“ deutet an, daß noch weitere Varianten existieren, die nicht explizit aufgeführt sind.
- Die Superklasse *Basiselemente* (*kursiv*) ist abstrakt. Es kann keine direkten Instanzen von der Superklasse *Basiselemente* geben. Jedes zugehörige Objekt muß einer der Unterklassen angehören.

Die Handhabung der unterschiedlichen Kategorien von Daten zeigt Abbildung 9.18 (Seite 83).

Über die Kernkomponente `actmap` hat der Anwender Zugriff auf Daten in nativen Formaten, wie sie in dieser Dissertation beschrieben werden (z. B. native, teilweise native, objektorientierte Daten basierend auf Quellentext).

Beispielsweise über Filter oder durch das Klonen von Funktionen können autarke Daten hergestellt werden.

In gleicher Weise können Daten für Klienten, z. B. die Nutzung innerhalb Plugins aufbereitet werden.

Neben der prinzipiellen Möglichkeit der Integration von Verfahren zur Nutzung konventioneller Daten, stehen flexible Erweiterungsmöglichkeiten für zukünftige Entwicklungen zur Verfügung.

Wie diese konkret aussehen, entscheiden die Entwicklungen der nächsten Jahre.

9.6.12. Verwendung mit einem Plugin

Das Einbetten von Programmteilen und Daten auf der Basis von Quellentext in `HTML` zur Verwendung mit einem Tcl-Plugin funktioniert vergleichbar, wie dies auch bei anderen Arten von eingebetteten Skripten und ähnlichem Material üblich ist (Abbildung 9.19, Seite 82).

```
<html>
<center>
<embed src="noname.tcl"
       width=250 height=240>
</center>
</html>
```

Abb. 9.19: Quellentextbeispiel: Einbetten von Objektgraphik (Quellentext, Daten, Ereignisse) für die Verwendung mit einem Plugin

Zu beachten sind insbesondere folgende Punkte:

- Die Quellentexte, die über das Netz in einem Plugin verwendbar sind, haben ohne spezielle Vorkehrungen immer die Voraussetzung eines sicheren Interpreters.
- Alle Programmteile sowie Daten müssen im einfachsten Fall in einer Datei vorliegen. Diese kann im Einzelfall manuell oder durch ein individuelles Skript automatisch erzeugt werden. Bei diesem Vorgang können weitere Elemente hinzugefügt werden, also z. B. weitere Bedienelemente, Makrofunktionen und dergleichen.
- Der sichtbare Ausschnitt sollte so gewählt werden, daß das gesamte Canvas und alle Bedienelemente zu sehen sind.

9.6.13. Weiterführende Gesichtspunkte

Eine Behandlung ist prinzipiell für alle Kategorien von Daten möglich. Implementiert wurden aber im wesentlichen die Unterstützung nativer Daten und geklonter Daten.

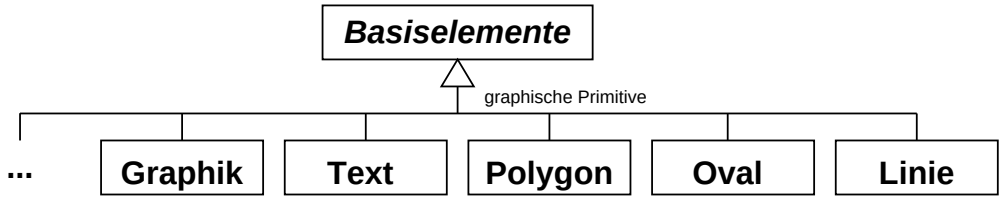

Abb. 9.17: Klassendiagramm zu graphischen Primitiven in OO-Daten der entwickelten Datensprache

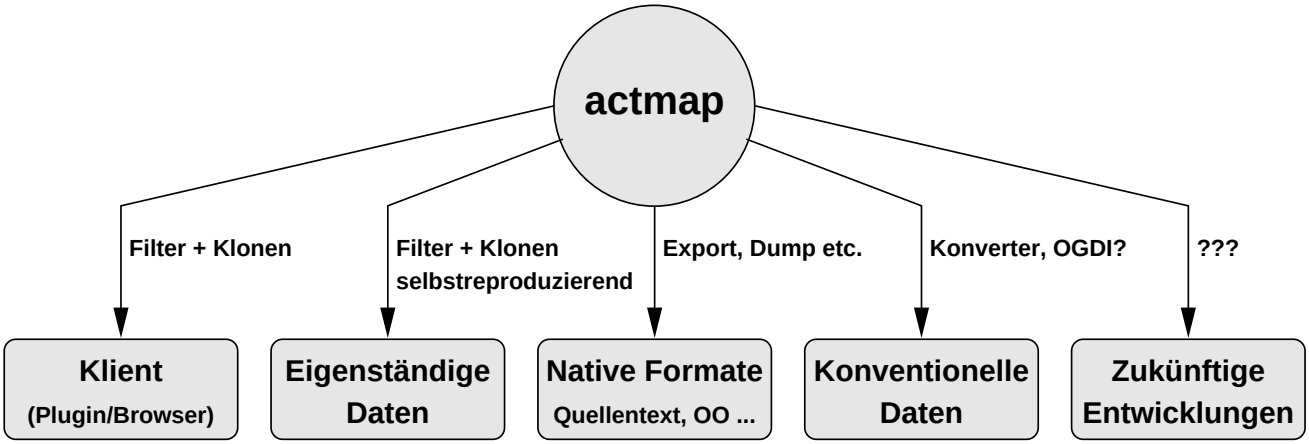

Abb. 9.18: Handhabung unterschiedlicher Kategorien von Daten

Die Umwandlung der verschiedenen Kategorien von Daten ist möglich. Dabei ist für bestimmte Anwendungsgebiete und Aufgaben eine Umwandlung nur von der logisch höheren Kategorie zu einer Kategorie ähnlichen oder niedrigeren Niveaus automatisch möglich.

Beispielsweise können konventionelle Daten zwar automatisch in ereignisunterstützende native Quellentext Formate umgewandelt werden, die möglicherweise gewünschten Informationen für bestimmte Ereignisse sind aber in den konventionellen Daten nicht enthalten und müssen daher auf anderem Wege ergänzt werden.

Bei geklonten Daten handelt es sich um eine Art nativer Daten, die zusätzlich mit Teilen der zentralen oder anderer Applikationen angereichert werden.

Die so entstandenen Daten können daher ohne weitere Unterstützung in einem Plugin dargestellt werden oder sind im Extremfall sogar autark und können sich selbst darstellen.

Eine Nutzung von Namensräumen innerhalb der Daten ist für die verschiedensten Zwecke in gleicher Weise mit nativen Mitteln möglich, wie der Einsatz objektorientierter Daten.

10. Exemplarische Fallstudien anhand des Prototyps

10.1. Datenmaterial

Im Mittelpunkt dieser Arbeit steht die Funktionalität von Daten, die auf Quellentext basieren. Aus diesem Grund ist der Gehalt an Informationen in den für die Entwicklung und die Beispiele verwendeten Daten von sehr untergeordneter Bedeutung. Vielmehr ist der Aufbau der Daten primär an der Demonstration der technischen Zusammenhänge orientiert.

Die aufgezeigten Anwendungen sind Beispiele dafür, wie räumliche Daten für die Visualisierung und Analyse abgebildet werden können. Dies kann nur als Versuch gelten, einen Eindruck von den vielfältigen Möglichkeiten zu vermitteln, da jede Beschreibung statischer erscheinen muß, als dies für ereignisorientierte und dynamische Verfahren angemessen ist.

Diese und viele weitere Beispiele und Daten, z. B. zur Einbettung von Quellentext-Animationen in Objektgraphik, Bleicheffekten (engl.: „fading“), verschiedenen ereignisgesteuerten Texteffekten, automatisches Plotten von mehrdimensionalen vektorisierten Funktionen und Graphiken in das Canvas einer Komponente, z. B. aus Gnuplot sowie weitere Anwendungen werden zusammen mit dem Prototyp im Internet zur Verfügung gestellt (s. Anhang A, S. 113).

Aus Gründen der Einfachheit zur Vermittlung grundlegender Ideen und günstigerer Performanz wurden bestimmte generalisierte Fälle ausgewählt.

Es wurde thematisch und konzeptionell unterschiedliches Datenmaterial exemplarisch zusammengestellt und aufbereitet. Wo bisher keine geeigneten Daten verfügbar waren, wurden notwendige Teile synthetisch erzeugt. Andere Beispiele oder Teile von Beispielen basieren auf realen Daten, die in unterschiedlichen Graden überarbeitet wurden.

Da Prototyp und Daten zur Demonstration des Prinzips dienen, wurden verschiedene ursprünglich vorhandene Daten entfernt, die nicht für diesen Zweck notwendig sind oder aus anderen Gründen an dieser Stelle nicht veröffentlicht werden können.

Das für die gezeigten Ausschnitte des Stadtgebiets Münster angefertigte Datenmaterial (Vektordaten von Gewässerflächen und Straßenzüge sowie Attribute und Daten zur Gewässergüte) basiert auf Daten vor dem Jahr 1997, die mit freundlicher Genehmigung vom Umweltamt der Stadt Münster[1] für die Entwicklungen und Publikation dieser Arbeit und zur Demonstration im Internet und mit dem entwickelten Prototyp kostenfrei zur Verfügung gestellt wurden.

Das Material der verwendeten Luftbilder des Stadtgebiets Münster basiert auf Daten [Han1999a], die mit freundlicher Genehmigung von der Firma Hansa Luftbild Consulting International G. m. b. H., Münster[2], für die Fallstudien in dieser Dissertation und zur Publikation im Internet und mit dem entwickelten Prototyp kostenfrei überlassen wurden.

Das angefertigte Datenmaterial für die Entwicklung des Prototyps und die gezeigten Ausschnitte des Stadtgebiets Münster (Hintergrundkarte, DGK), basiert auf Daten vor dem Jahr 1997, die mit freundlicher Genehmigung vom Landesvermessungsamt Nordrhein-Westfalen[3] für die Entwicklungen zu dieser Arbeit, zur Publikation in dieser Dissertation und zur Bereitstellung im Internet und mit dem entwickelten Prototyp kostenfrei überlassen wurden.

Alle Datensätze wurden in der Komponente `actmap` jeweils als Vergrößerungsstufe Null geladen.

Die Schnappschüsse wurden für den äußeren Rand undekoriert aufgenommen, d. h. ohne Anteile des verwendeten Fenster-Managers, wie Rahmen und Bedienelemente, da diese unter dem X Window System prinzipiell beliebig auswechselbar sind.

[1] `http://www.muenster.de/stadt/umweltamt` ..... [V: k. A.] [Ä: k. A.] [Z: 05.03.2001]
[2] `http://www.hansaluftbild.de` ..... [V: k. A.] [Ä: k. A.] [Z: 05.03.2001]
[3] `http://www.lverma.nrw.de` ..... [V: k. A.] [Ä: k. A.] [Z: 14.03.2001]

10.2. Objektgraphik und Hintergrundkarte

Das Beispiel in Abbildung 10.1 (Seite 87) zeigt ereignisgekoppelte Vektordaten mit einem Teil einer Hintergrundkarte (engl.: „basemap“) im Zentrum.

Dargestellt sind Gewässerflächen und Straßennetz im Stadtgebiet Münster. Alle Einzelobjekte können über Ereignisse an weitere Daten zu dem betreffenden Objekt angebunden sein, z. B. Daten zur Gewässerqualität oder zu Sedimentablagerungen.

Neben der Anbindung von Ereignissen an beliebige Objekte, z. B. zur Verwendung externer Hilfs- und Anzeigeprogramme und der Manipulation der Eigenschaften der Objekte, kann für alle Objekte z. B. Grad der Transparenz, Füllmuster oder Farbe für aktive Objekte eingestellt werden.

In der Darstellung ist beispielsweise jeder der fast 18000 Teilstraßenzüge und jede der fast 700 Gewässerflächen ein eigenes Vektorobjekt. An die Gewässerflächen sind im Beispiel in Form von Attributen Daten zur Naturnähe angebunden, die z. B. innerhalb einer Komponente angezeigt werden können.

Bei einer derartig großen Zahl von Objekten ist die adäquate Geschwindigkeit bei der Handhabung, insbesondere in Kombination mit Rasterdaten und Vergrößerungsfunktionen, stark von der verwendeten Hardware abhängig.

Rasterdaten könnten für spezielle Beispiele alternativ in verschiedenen Vergrößerungsstufen bereitgestellt werden. Für den Fall, daß Rasterdaten beispielsweise für verschiedene Kartenblätter vorliegen, empfiehlt sich für eine flexiblere Handhabung die Erzeugung von Kacheln aus den einzelnen Blättern. Diese hat den Vorteil einer einfachen Attributvergabe für einzelne Kacheln und kann zum anderen für schnellere Ladezeiten bezüglich einzelner Ausschnitte ausgenutzt werden.

Vor allem für große Rasterdaten kann die Verwendung eines Graphikformates ohne Komprimierung, z. B. PPM, zusätzlich zu einer reduzierten Ladezeit auf schwächerer Hardware beitragen.

Über Ereignissteuerung und dynamische Visualisierung hinausgehend, wären hier im Rahmen allgemeinerer Tcl/Tk Entwicklungen z. B. die Modularisierung des Kerns und eine veränderte Speicherverwaltung, beispielsweise für die effizientere Handhabung von Rastergraphiken wünschenswert.

Ebenso können optimierte Algorithmen, längere und strukturierte Attribute zu einer besseren Anpassung und Skalierbarkeit beitragen. Die Erarbeitung von Kategorien derartiger Attribute sollte interdisziplinär erfolgen.

10.3. Objektgraphik und Hintergrundkarte in höherer Auflösung

Abbildung 10.2 (Seite 88) zeigt einen vergrößerten Ausschnitt aus dem vorangegangenen Beispiel. In diesem Beispiel ist das Vektorobjekt „Aasee“ aktiv, zu erkennen an der für diesen Datensatz eingestellten roten Füllfarbe und der Transparenz des aktiven Objekts. Die Hintergrundkarte ist durch das aktive Vektorobjekt zu sehen. Der schematische Verlauf der Straßenzüge ist in roten nichttransparenten Linienzügen dargestellt. Wie bei allen anderen Vektorobjekten können auch die Eigenschaften der Straßenzüge, z. B. Linienstärke, Farbe, Füllmuster oder Ereignisanbindung, interaktiv verändert werden.

Der Detailreichtum einer solchen Darstellung sollte dem betreffenden Verwendungszweck angemessen sein. Nach geokognostischen Überlegungen kann über die implizit vorhandene Nutzung beliebiger Funktionen gegebenenfalls auch die Detaildarstellung in verschiedenen Vergrößerungsstufen angepaßt werden. Ebenso können mehrere Hintergrundkarten verwendet und interaktiv bzw. ereignisgebunden ausgewechselt werden, z. B. für verschiedene thematische Aspekte.

10.4. Objektgraphik und thematische Karte

Das Beispiel in Abbildung 10.3 (Seite 89) zeigt die ereignisgekoppelten Vektordaten aus Abbildung 10.1 (von Seite 87) mit einer thematischen Karte zur Gewässergüte im Stadtgebiet Münster. Dargestellt sind die Gewässerflächen und Straßenzüge als Vektordaten sowie eine thematische Rasterkarte (`sampleGuete`, `F6` bzw. `Meta-g`) und eine zugehörige Legende.

Die Daten für die thematische Karte wurden mit freundlicher Genehmigung vom Umweltamt der Stadt Münster überlassen und stammen aus den veröffentlichten Daten zum Umweltbericht[4] für das Jahr 1996/1997.

Im Zentrum ist die kleine unterlegte Hintergrundkarte zu sehen, die auch in Abbildung 10.2 (von Seite 88) für den vergrößerten Ausschnitt verwendet wur-

[4] `http://www.muenster.de/stadt/umweltamt/uebericht/guete.pdf` ..... [V: k. A.] [Ä: k. A.] [Z: 04.03.2001]

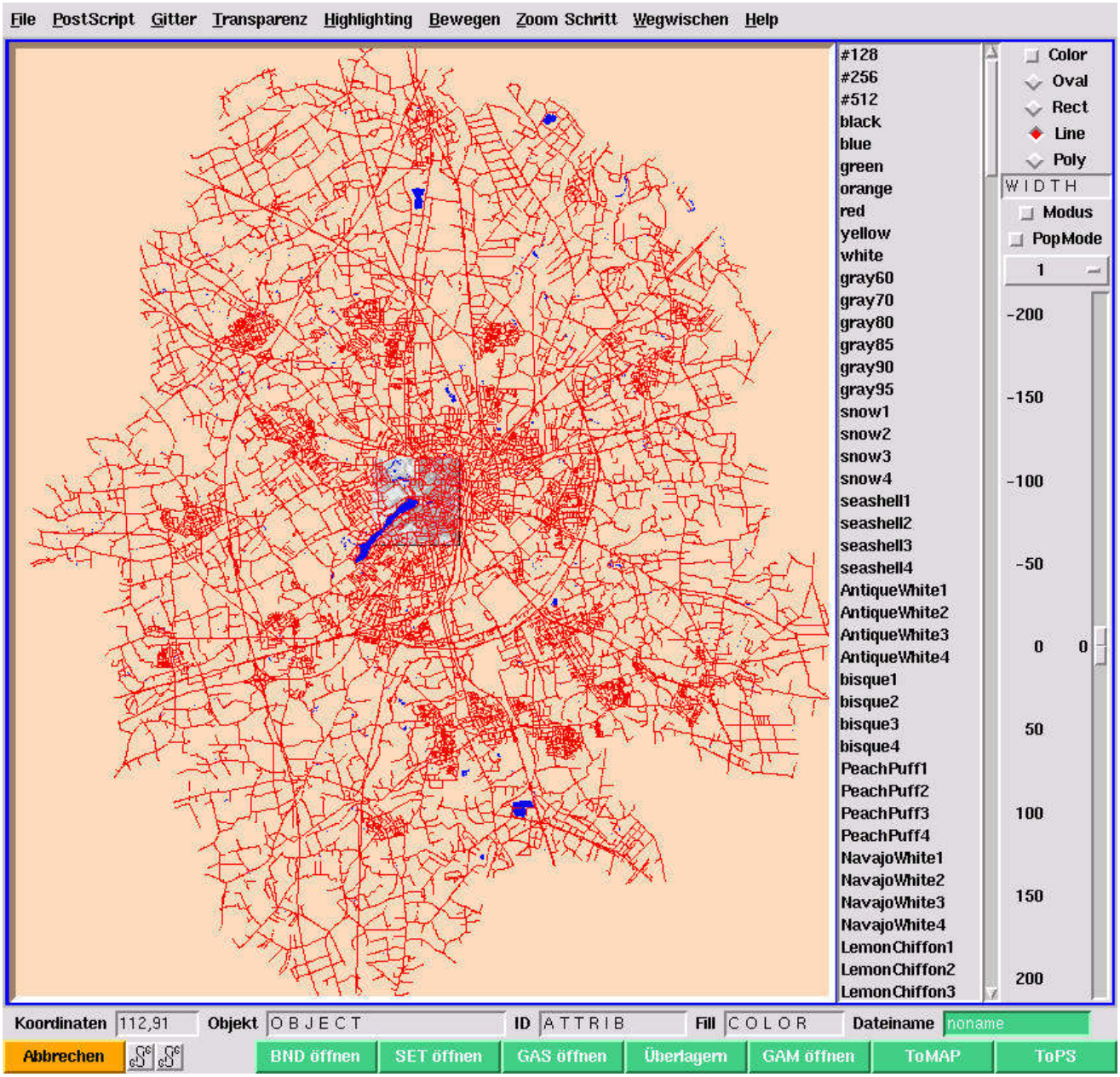

Abb. 10.1: Fallbeispiel: Ereignisaktive Vektordaten mit Hintergrundkarte (Münster, Gewässer und Straßennetz)

de. Diese Hintergrundkarte liegt in dem gezeigten Zustand als Schicht zwischen der thematischen Karte und der Schicht mit den Vektordaten.

Als Beispiel werden zu dieser Fallstudie hier einige konkrete Ereignisanbindungen aufgeführt. Zu den Objekten können über Ereignisanbindungen weitere Daten, z. B. Quellenangaben und vergleichbare Informationen, eingeblendet (`sampleGueteQuelle`, `F8` bzw. `Meta-q`) und wieder versteckt werden (`$w delete sampleguetequelle`, `F9` bzw. `Meta-r`). Dargestellt ist eine als Objekt eingeblendete Legende (`sampleGueteLegende`, `F7` bzw. `Meta-l`) zu der thematischen Karte. Über Ereignisanbindungen, z. B. Tastenbelegungen, kann die thematische Karte z. B. in den Vordergrund geholt (`$w raise sampleguete`, `F3` bzw. `Meta-f`) und wieder in den Hintergrund verlagert (`$w lower sampleguete`, `F4` bzw. `Meta-b`) oder für Vergrößerungen ausgeblendet werden. Eine Automatisierung ist ebenso möglich, wie auch die Einblendung von vergrößerten Ausschnitten mit eigenen Ereignisanbindungen realisierbar ist.

Die weiteren zugehörigen Ereignisse sind in dem entsprechenden Datensatz definiert. Dort können weitere benutzerdefinierte Ereignisanbindungen und Funktionen untergebracht werden.

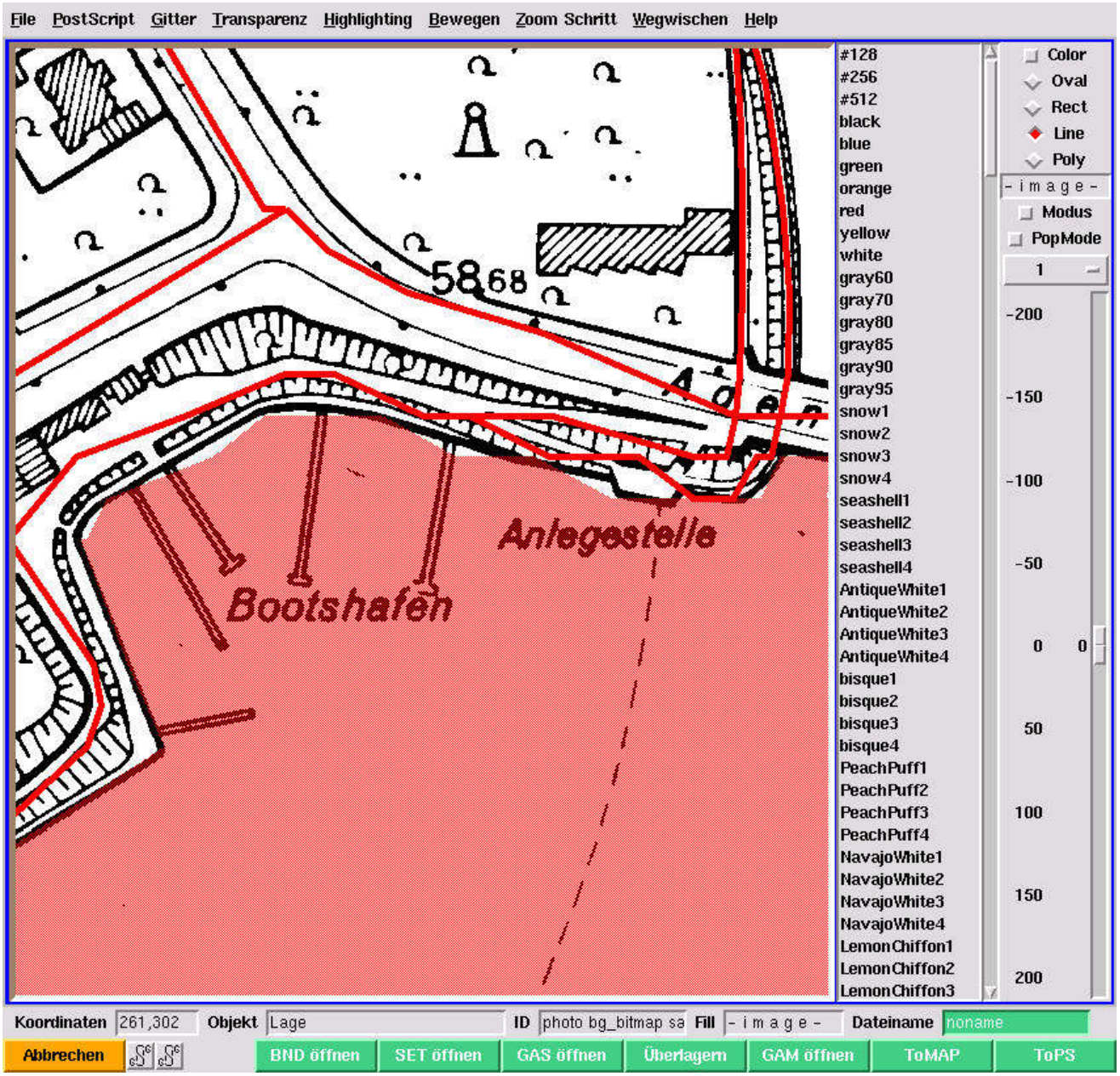

Abb. 10.2: Fallbeispiel: Ereignisaktive Vektordaten mit Hintergrundkarte (Münster, Aasee-Nord)

10.5. Objektgraphik und Luftbilddaten

Um das Verfahren auch mit konzeptbezogen sehr umfangreichem Datenmaterial zu testen, wurden spezielle Fälle untersucht. Das verwendete Datenmaterial stammt aus verschiedenen Jahren und unterschiedlichen Einsatzgebieten. Die Erfassung des verschiedenen Datenmaterials wird hier nicht explizit beschrieben.

Durch die Erstellung geeigneter Datensätze konnten auch bestehende Daten nach einiger Bearbeitung demonstrativ zu geeigneten Beispielen zusammengefügt werden.

Für das resultierende Extrembeispiel, das noch auf derzeit verhältnismäßig durchschnittlichen Computern mit akzeptabler Performanz handhabbar ist, wurden große Hintergrundkarten, Luftbilder in hoher Auflösung (ca. 1 m Bodenauflösung, etwa dreifach hochskaliert und nachbearbeitet) und Farbtiefe (Echtfarben) und Vektordaten mit über zehntausend Objekten und hunderttausenden von Punkten und Attributen in einem Datensatz kombiniert.

Datensätze in dieser Größenordnung erfordern in den getesteten Beispielen einen Arbeitsspeicher von minimal etwa 200 MB.

Die Handhabung nach dem Laden der Daten er-

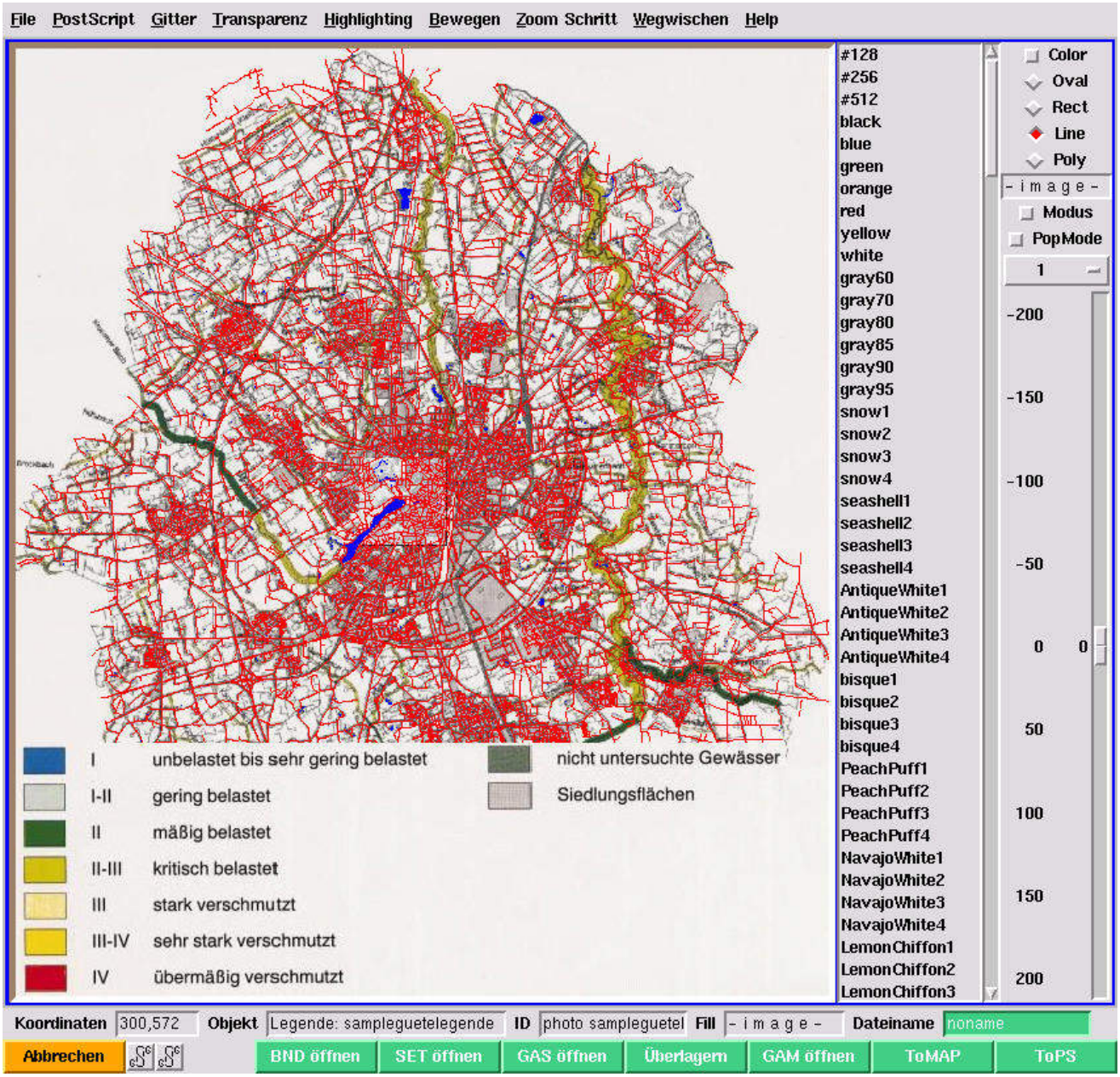

Abb. 10.3: Fallbeispiel: Ereignisaktive Vektordaten mit thematischer Karte (Münster, Gewässer, Straßennetz und Gewässergüte)

weist sich als erstaunlich flüssig für das eingesetzte Verfahren, trotz der großen Datenmenge.

Ein ausgewähltes fachbezogenes Beispiel ist die ereignisgestützte Aufbereitung und Visualisierung von Gewässerdaten in einem zu vermittelnden Kontext anhand von hydrologischen Meßdaten und Ergebnissen, potentiellen Ereignissen, Vektordaten sowie Luftbilddaten (Abbildung 10.4 auf Seite 90 und 10.5 auf Seite 91).

Abbildung 10.4 (Seite 90) zeigt einen Ausschnitt des Stadtgebietes in Münster mit einem Teil des Aasees (blaues Vektorobjekt). Wege am Ufer des Aasees sind in dieser Darstellung als rote Linien zu erkennen.

Durch die ereignisaktive Objektgraphik können leicht Strukturen und Elemente hervorgehoben werden.

Die Kombination von Vektordaten *und* Luftbildern ermöglicht die Vermittlung eines vollständigeren Eindrucks von der Lokalität. Beispielsweise kann einerseits auf dem Luftbild leicht die Umgebung erfaßt, andererseits kann beispielsweise durch die Vektordaten der Verlauf von Wegen unter dem Baumbestand angedeutet werden.

Abbildung 10.5 (Seite 91) zeigt dies in vergleichbarer Weise für den auf Luftbildaufnahmen weitgehend

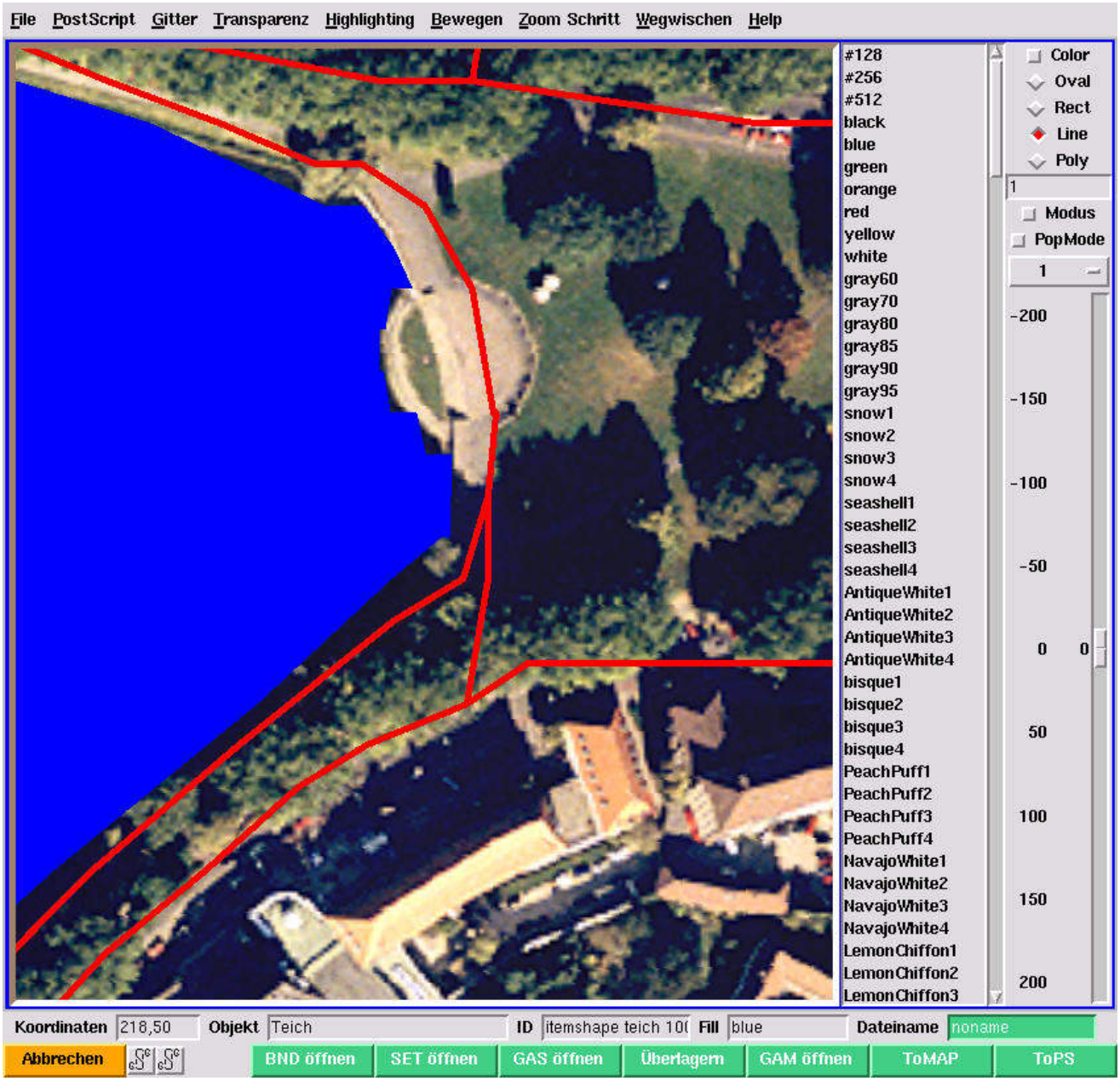

Abb. 10.4: Fallbeispiel: Ereignisaktive Vektordaten mit Luftbild (Münster, Aasee-Nord)

überwachsenen Schloßgraben des Schlosses in Münster.

Im Datenmaterial können Objekte über Ereignisse und Funktionen mit beliebigen externen Daten und Anwendungen verbunden werden.

Die Handhabung der Daten kann praktisch durch beliebige eigene Funktionen erweitert werden.

Die im Beispiel hauptsächlich enthaltenen Daten sind die weitestgehend vollständigen Gewässerflächen im inneren Stadtgebiet und die Straßenzüge des Straßennetzes.

Die Gewässerflächen sind über Ereignisse mit weiteren Daten zu den einzelnen Gewässern verknüpft. Die unterliegenden Luftbilder sind zum einen für die bessere Orientierung hilfreich und zum anderen für die Beurteilung der Lokalität unerläßlich, z. B. wenn Gewässer vollständig überwachsen sind.

Abbildung 10.6 (Seite 92) zeigt eine Kombination aus Vektordaten (Straßenzüge und Gewässer), Hintergrundkarte und Luftbild.

Das ursprüngliche Luftbild ist um einen Betrag zu der verwendeten Hintergrundkarte und den Vektordaten rotiert, um eine mögliche Überdeckung zu erreichen. Der Grad der Rotation ist für diesem Datensatz durch den dunklen Balken zwischen der Hintergrundkarte und Luftbild veranschaulicht.

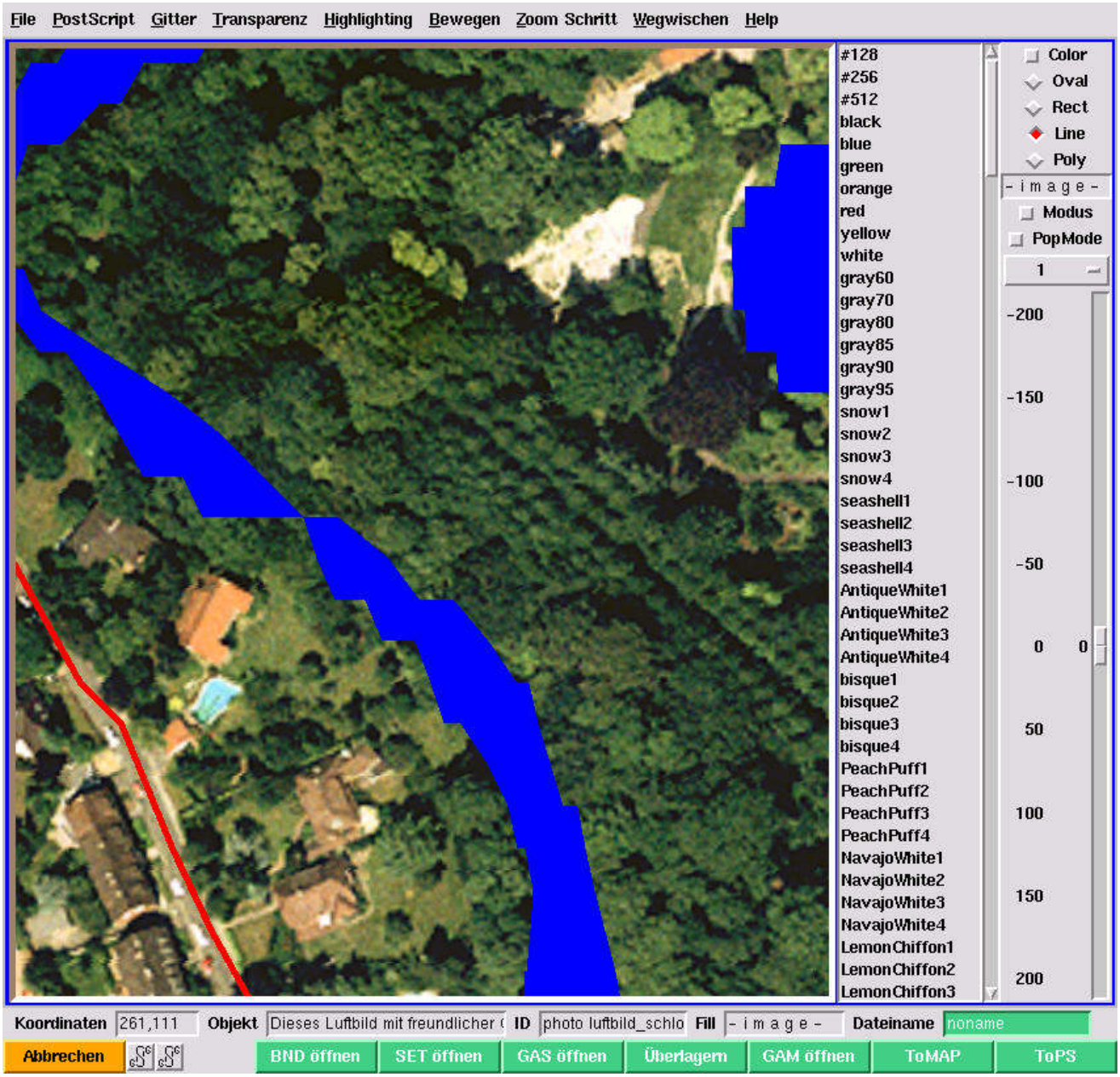

Abb. 10.5: Fallbeispiel: Ereignisaktive Vektordaten mit Luftbild (Münster, Schloßgraben)

Das Vektorelement „Aasee“ ist aktiv.

Auf dem Luftbild ist hier in weitestgehender Deckung der bereits aus der Hintergrundkarte in Abbildung 10.2 (von Seite 88) bekannte Bootshafen mit weiteren Details zu sehen.

10.6. Objektgraphik und Autoevents

Die Anbindung von Applikationen über Ereignisse kann beispielsweise dazu ausgenutzt werden, um Informationen oder Anwendungen automatisch, als quasi-selbsttätige Ereignisse zu starten. Derartig umgesetzte Ereignisse werden in dieser Arbeit als *Autoevents* bezeichnet. Ein Beispiel für die Anwendung dieser Art zeigt Abbildung 10.7 (Seite 93).

Dargestellt ist auf einer skizzenartigen Europakarte die Verteilung der Nationalparks in Deutschland. Die Lage der inländischen Nationalparks ist grün gekennzeichnet, die der küstennahen Parks blau. Der gerade ausgewählte aktive Nationalpark ist transparent rot.

Bei Betreten der gekennzeichneten Fläche des Nationalparks werden in diesem Fall automatisch zwei Teile einer Straßenkarte in Deckung eingeblendet. Diese werden beim Verlassen der gekennzeichneten Fläche automatisch wieder geschlossen.

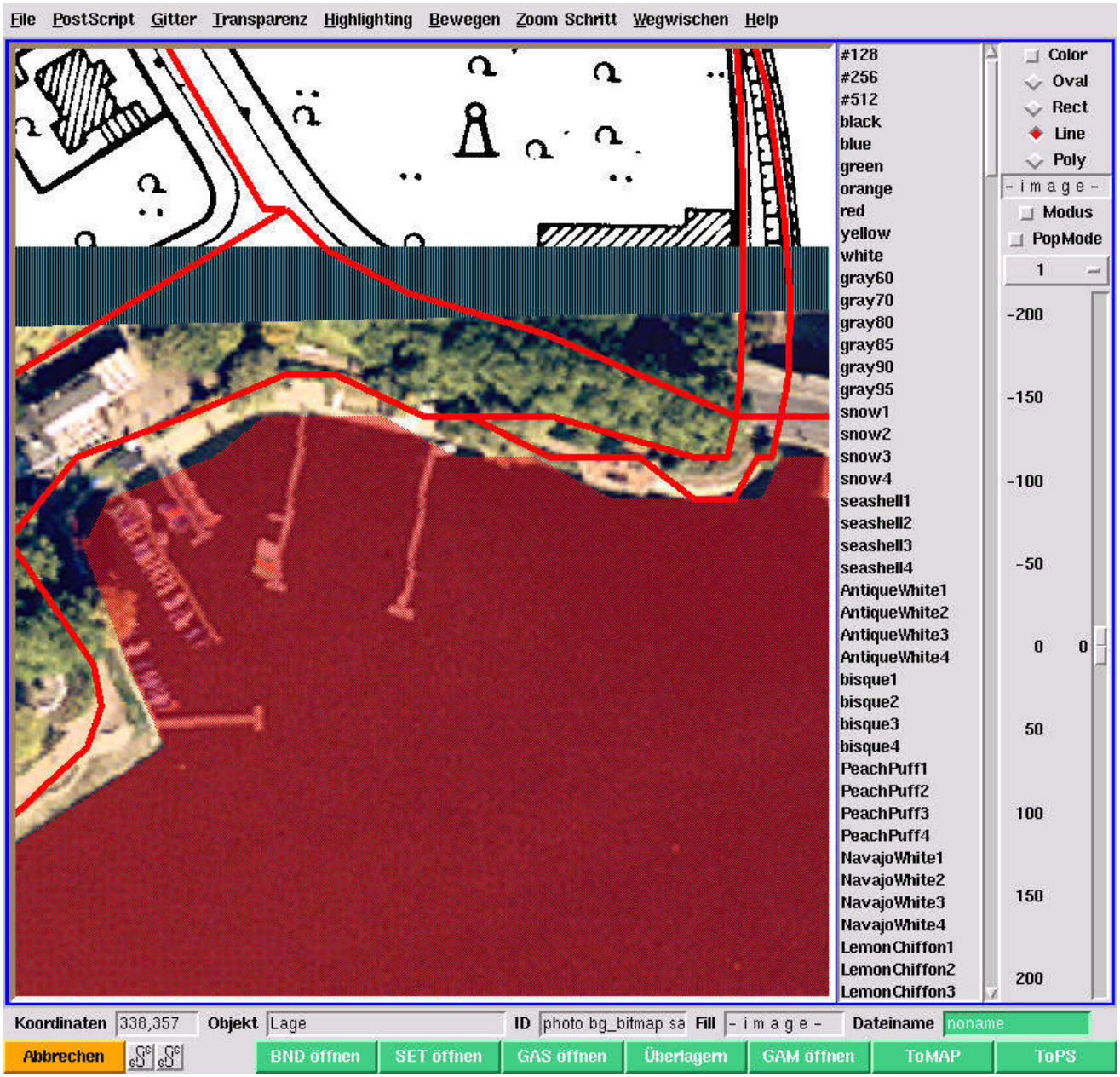

Abb. 10.6: Fallbeispiel: Ereignisaktive Vektordaten, Hintergrundkarte und Luftbild (Münster, Aasee-Nord)

Eine Deckung ist auf diese Weise nur möglich, wenn der aktuelle Bildschirm ausreichend groß für die absolute geometrische Positionierung der beiden Fenster ist, ansonsten werden diese z. B. automatisch in den sichtbaren Bereich verschoben.

Zu dem Nationalpark werden durch ein Ereignis exemplarisch Daten in der Komponente `actsea` angezeigt. Diese Komponente ist ein kleiner Editor mit speziellen Such- und automatischen Markierfunktionen und kann z. B. über eine andere Komponente oder mittels Objektgraphik programmiert und gesteuert werden. In der Realität sind die zugehörigen Daten für diesen Datensatz blinkend markiert.

Bei der Verwendung von Autoevents muß im einfachsten Fall keine Tastatur oder Maustaste betätigt werden, sondern lediglich ein Zeiger, also z. B. ein Mauszeiger. Auf Objekten können dann bei Betreten mit dem Zeiger beliebige Anwendungen automatisch gestartet und z. B. bei Verlassen des definierten Objekts wieder automatisch geschlossen werden. Für spezielle Anwendungen sind insbesondere die Größen und Positionierungen der sich öffnenden Fenster und die automatischen Ereignisse zu planen. Steht keine Tastatur zur Verfügung, sollten Funktionen alternativ z. B. auf Bedienelemente gelegt werden, die durch Betreten mit dem Zeiger aktiviert werden.

Abb. 10.7: Fallbeispiel: Autoevents (Nationalparks)

In diesem Kontext sind viele Anwendungen denkbar, beispielsweise der Einsatz auf Informationsterminals ohne Tastatur, aber mit berührungssensitivem Sensorbildschirm (engl.: „touch screen“).

10.7. Objektgraphik als Applet in einem Klienten

Objektgraphik mit Funktionen, Bedienelementen usw. können z. B. auf einem WebServer bereitgestellt und in einem Klienten (engl.: „browser“) mit einem Plugin über das Netz genutzt werden.

Ein einfaches Beispiel für die Nutzung von ereignisaktiver Objektgraphik mit Daten und Bedienoberfläche als Applet (Tclet) über ein Netz zeigt Abbildung 10.8 (Seite 94).

Die Darstellung zeigt im wesentlichen eine einfache Karte mit einem Teil Europas sowie einige mit aktiven Punkten markierte Lokationen. Für dieses Beispiel wurden der Objektgraphik im unteren Teil einige spezifische Bedienelemente hinzugefügt.

Dieser aufbereitete Datensatz ist prinzipiell autark und kann auch vollständig unabhängig von einer anderen Komponente verwendet werden. Er enthält selbst alle Informationen, Ereignisse und Funktionen, die für seine Darstellung und Handhabung vorgesehen

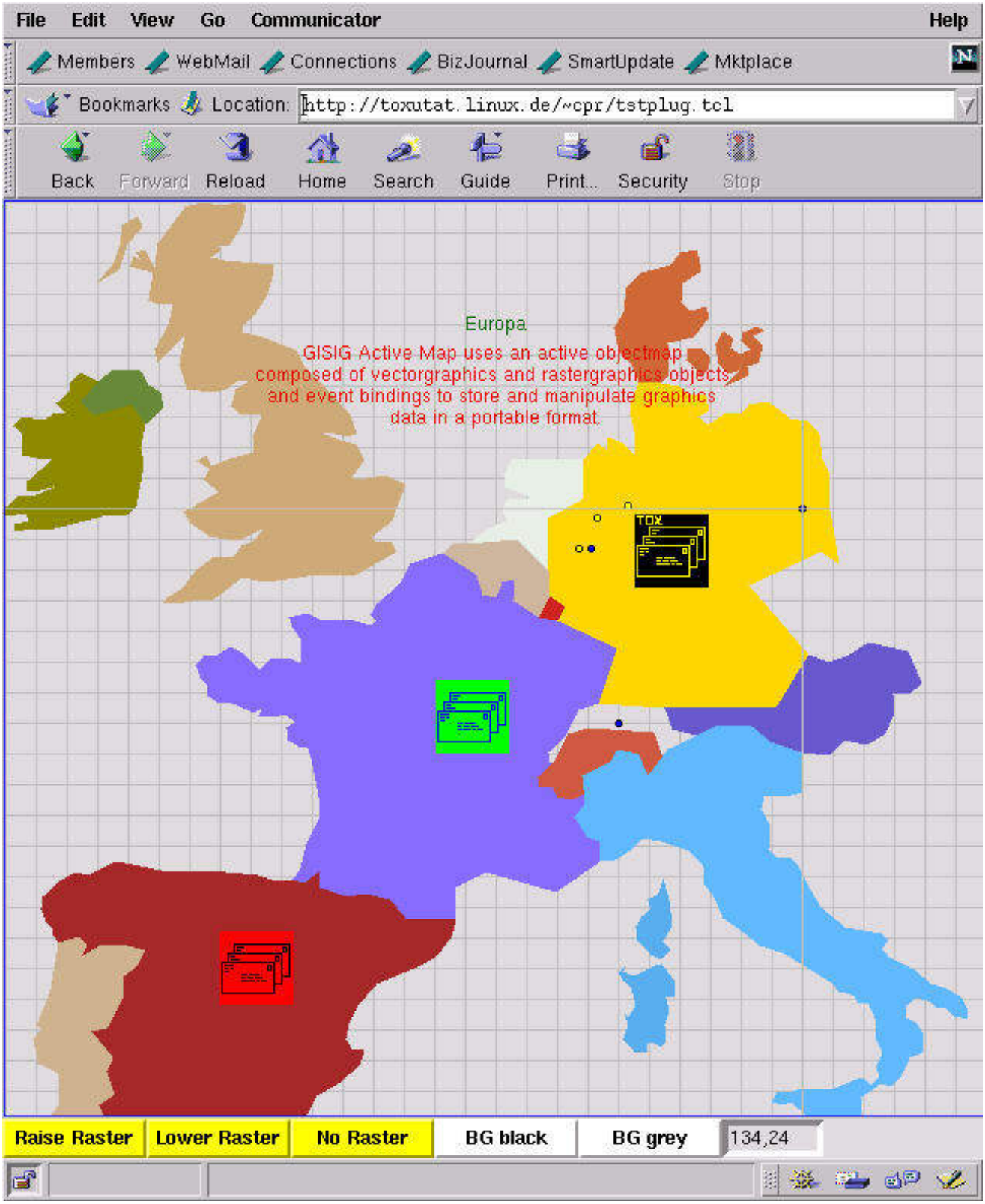

Abb. 10.8: Fallbeispiel: Ereignisdaten als Applet in einem Klienten (Europaskizze)

sind. Rastergraphiken können mit einfachen Mitteln als Quellentext in den Datensatz integriert werden.

Die Einbettung in eine Seite und die Nutzung über einen konventionellen Klienten bringt die üblichen Einschränkungen für diese Art von Anwendung bezüglich des praktisch möglichen Umfangs an Datenmaterial und der notwendigen Sicherheitsvorkehrungen.

Die Funktionalität kann für Spezialfälle individuell angepaßt werden. Die eingebettete Anwendung stellt selbstverständlich ein vollständiges Canvas zur Verfügung, auf dem z. B. Objekte manipuliert, verschoben oder gelöscht werden können.

Um anzudeuten, daß dies auch in einem Klienten funktioniert, wurden zwei Gitterlinien aus dem Hintergrund über einen Meßpunkt (innerhalb der Umrisse Deutschlands) in den Vordergrund bewegt.

Der schematische Umriß der Schweiz ist verschoben, um zu zeigen, daß Objekte in Schichten übereinanderliegen.

10.8. Verwendung einer Ereignisdatenbank mit Objektgraphik

Mit geringem Aufwand können vorhandene Ereignisse und andere Daten, beispielsweise Funktionen oder Objekte, in einer Datenbank abgelegt werden.

Ein Beispiel für die Nutzung einer Ereignisdatenbank zeigt Abbildung 10.9 (Seite 96).

Die von der Datenbank verwalteten Daten, beispielsweise Attribute, Tastenbelegungen und zugeordnete Ereignisse, können in der Datenbank gespeichert oder automatisch exportiert und zur Laufzeit z. B. über ein geeignetes Skript in eine Komponente übernommen werden, also z. B. für die Aktualisierung der Ereignisanbindungen eines bestimmten Datensatzes.

Dazu empfiehlt sich eine Datenbank, die dem jeweiligen Bedarf und der Art von Datenmaterial angemessen ist, insbesondere bezüglich Funktionalität, Performanz und Bedienung.

Aufgrund des verwendeten Konzepts können z. B. lokale oder zentrale Datenbanken verwendet werden. Definitionen und Ereignisanbindungen können separat nachgeladen werden.

Über die Ereignisanbindungen hinaus, können in Zukunft weitere aufgabenspezifische Entwicklungen hinsichtlich Georeferenzierung und einer integrierten Datenbankschnittstelle stattfinden.

Auf diese Weise kann die Nutzung einer Datenbank für den Anwender vereinfacht werden. Zu den Möglichkeiten, Teile der Daten in einer Datenbank abzulegen, können einzelne Objekte ihren globalen räumlichen Bezug erhalten.

10.9. Steuerung mittels IPC aus Objektgraphik

Ein einfaches Beispiel für die Nutzung einer Fernsteuerung mittels IPC, beispielsweise von externen Applikationen aus einer Objektgraphik, zeigt Abbildung 10.10 (Seite 97).

In die Objektgraphik sind Bedienelemente integriert, die mit verschiedenen Ereignissen für die Steuerung der Applikationen belegt sind. Die eingesetzten Applikationen müssen allerdings die verwendete IPC unterstützen.

Aufgrund der konzeptionellen Trennung der einzelnen Funktionalitäten können leicht alle externen Applikationen gemeinsam aus den Objektgraphiken per Mausbedienung gesteuert werden. Beispielsweise können aus der Kernkomponente die Moleküle in den externen Anwendungen gemeinsam räumlich rotiert oder vergrößert oder einzelne Atome abgefragt werden.

Da sich von allen Komponenten neue Instanzen erzeugen lassen (s. z. B. in Abbildung 10.9 auf Seite 96), d. h. sich unter automatischer Vergabe eines neuen eindeutigen Identifikationsnamens aufrufen und ansprechen lassen, kann IPC auch in Benutzeranwendungen leicht zur effizienten Kommunikation mit und unter den Komponenten verwendet werden.

Diese Eigenschaften sind integraler Bestandteil bei dem eingesetzten Verfahren und können daher auch aus beliebigen Datensätzen, z. B. geologischen Karten, eingesetzt werden.

Auf diese Weise kann der Benutzer beispielsweise über Ereignisse eine Kopplung zu anderen Komponenten und externen, spezialisierten Applikationen erreichen.

10.10. Nutzung und Erweiterung der Oberfläche mit Objektgraphik

Ein Beispiel für die Modifikation bzw. Erweiterung der Oberfläche der verwendeten Komponente mittels der benutzerdefinierten Funktionen zu einem spezifischen Datensatz ist in Abbildung 10.11 (Seite 98) dargestellt.

Der Datensatz Nationalparks enhält exemplarische Funktionen zur Ergänzung und Überlagerung des Kar-

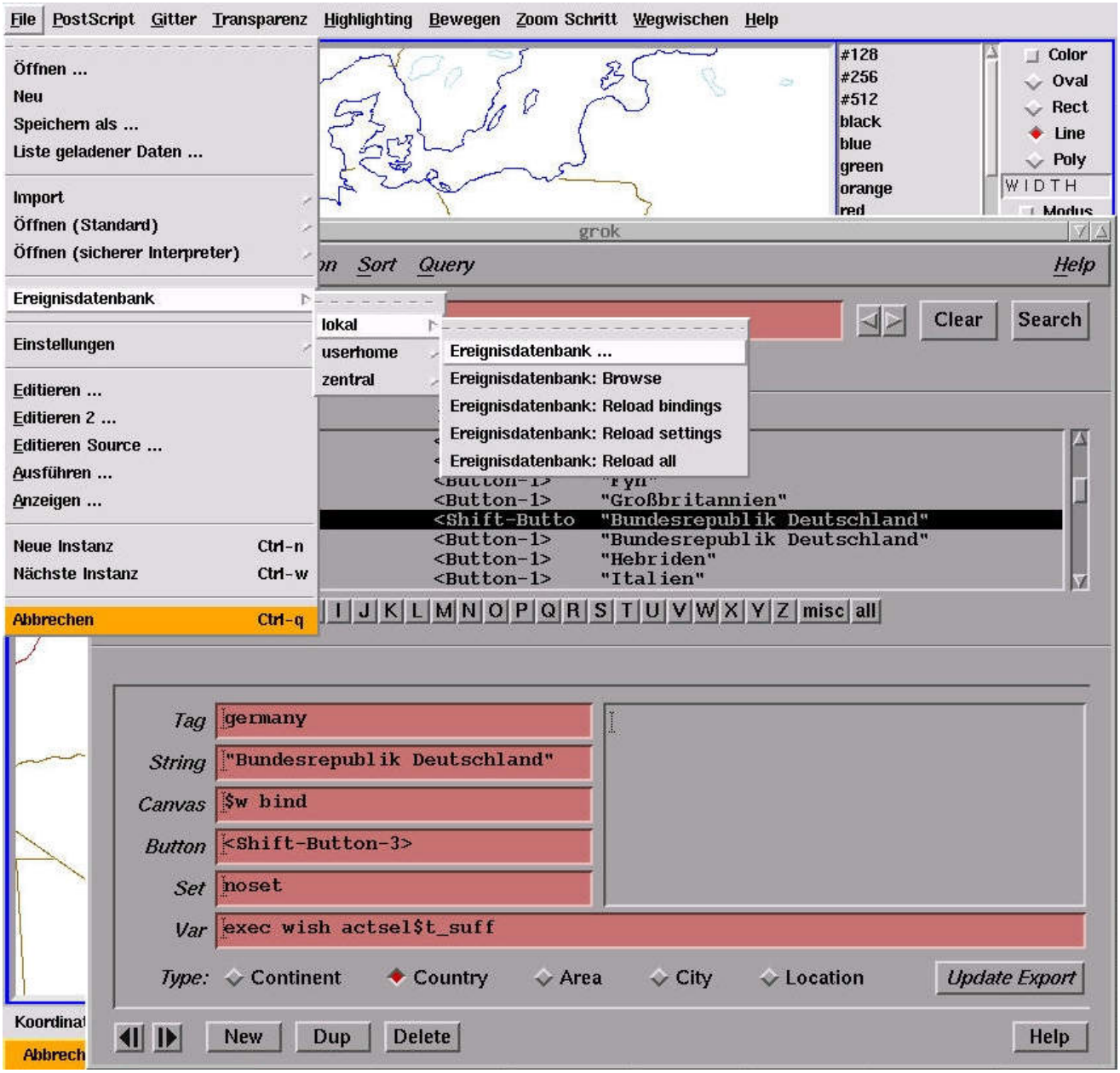

Abb. 10.9: Fallbeispiel: Nutzung einer Ereignisdatenbank (Weltkarte)

ten und Informationsmaterials durch graphische Objekte (`npark_de.gto`), die z. B. weitere Ereignisse und Verknüpfungen, wie Autoevents enthalten, die an beliebige Textobjekte bzw. Beschreibungen, Bedienelemente und Graphikobjekte angebunden werden können.

Diese Objekte sind ihrerseits ein Teil des Datensatzes und somit entsteht Zugriff in gleicher Weise, wie auf die eigentlichen Daten und Teile der Komponente.

Diese Form von Überlagerungen ist ein Beispiel dafür, daß funktionale Daten, wie portable Bedienelemente, in Objektgraphik integriert und z. B. für einen bestimmten Zeitraum ein echter Teil der Oberfläche einer Komponente werden können. Die Behandlung dieser funktionalen Daten ist ereignisgesteuert und dynamisch.

Über die Verknüpfungen sind prinzipiell alle Funktionen einer Komponente zu erreichen, d. h. es können auch integrierte Werkzeuge genutzt werden.

Auf der rechten Seite sind beispielsweise zwei in die Komponente integrierte Farbeditoren zu sehen, ein Standard-Farbeditor und ein selbst entwickelter Farbeditor.

Nicht nur sind alle Teile einer Applikation über funktionale Daten ansprechbar, diese können auch gleichzeitig direkten Zugriff auf die eigentlichen Daten haben.

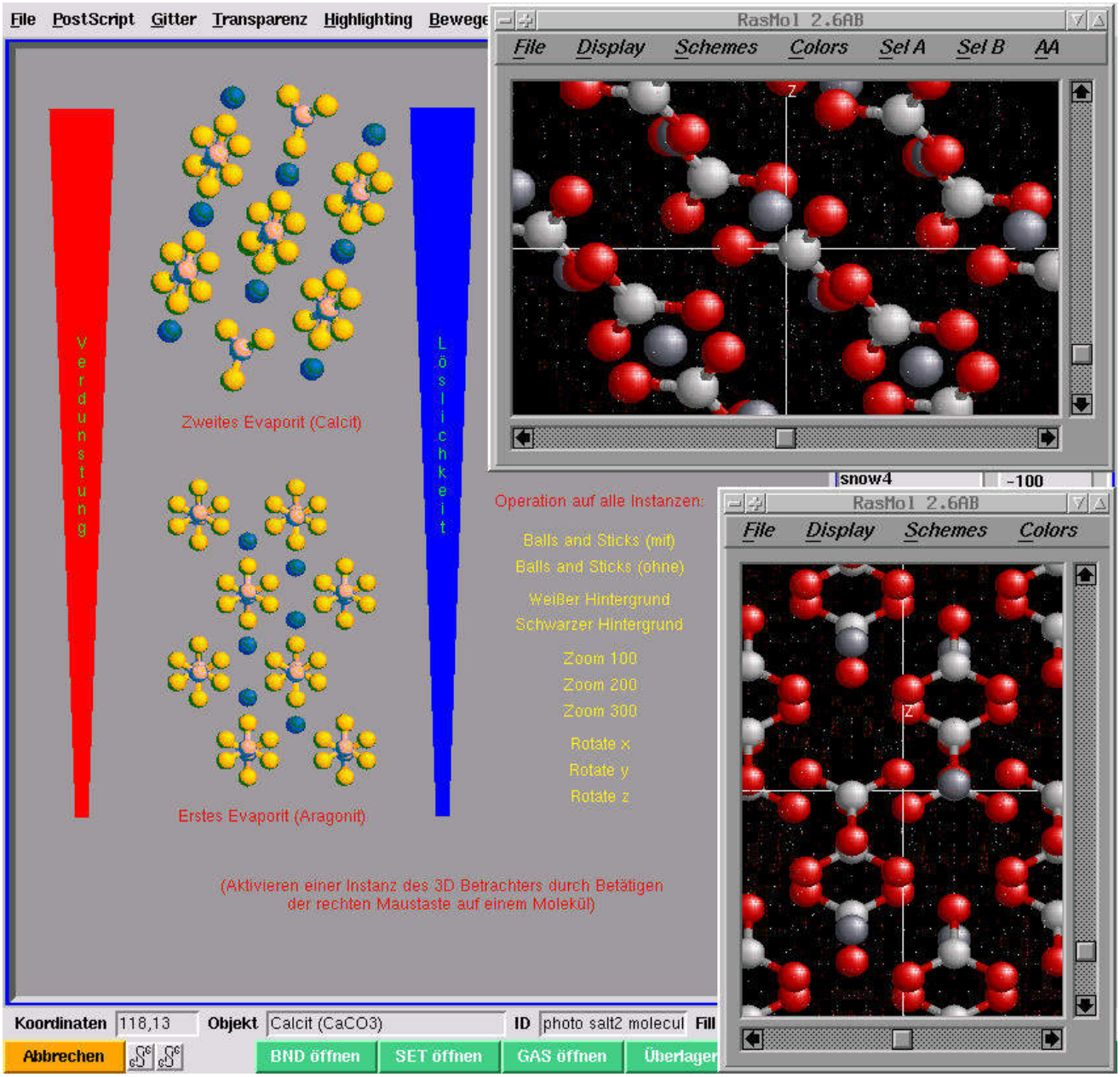

Abb. 10.10: Fallbeispiel: Objektgraphik und IPC (3D-Moleküle in externer Applikation)

Eine Steuerung der überlagerten Anteile über die integrierte Shell ist durch den konzeptionellen Aufbau implizit vorhanden.

Aufgrund der Trennung der Aufgaben durch die modulare Mehrschichtarchitektur und den Einsatz einer Skriptsprache können daher die Funktionen einer Komponente, durch den Benutzer auf einfache und flexible Weise modifiziert und erweitert werden. Beispielsweise lassen sich so das Aussehen von Teilen, wie Bedienelementen oder Meldungen, die Ereignisanbindungen und die dynamischen Elemente interaktiv modifizieren.

Aufgrund des eingesetzten Konzepts lassen sich daher auf intuitive Weise sowohl spezielle Benutzeroberflächen zu bestimmten Datensätzen konfigurieren als auch Oberflächen für bestimme eingesetzte Ausgabegeräte entwickeln, wie etwa spezielle Bildschirme portabler Geräte.

Die entstehenden Anwendungen können zusätzliche Funktionen beinhalten. Dies ist notwendig, wenn z. B. auf kleineren Bildschirmen Menüs oder andere Elemente zu viel Platz einnehmen und daher besser durch Elemente ersetzt werden, die besser an Daten und Gerät angepaßt sind.

Neben dem integrierten Interpreter sind die Shell und die Skriptfunktionen dabei ein wesentlicher Beitrag, diese Funktionalität zu ermöglichen.

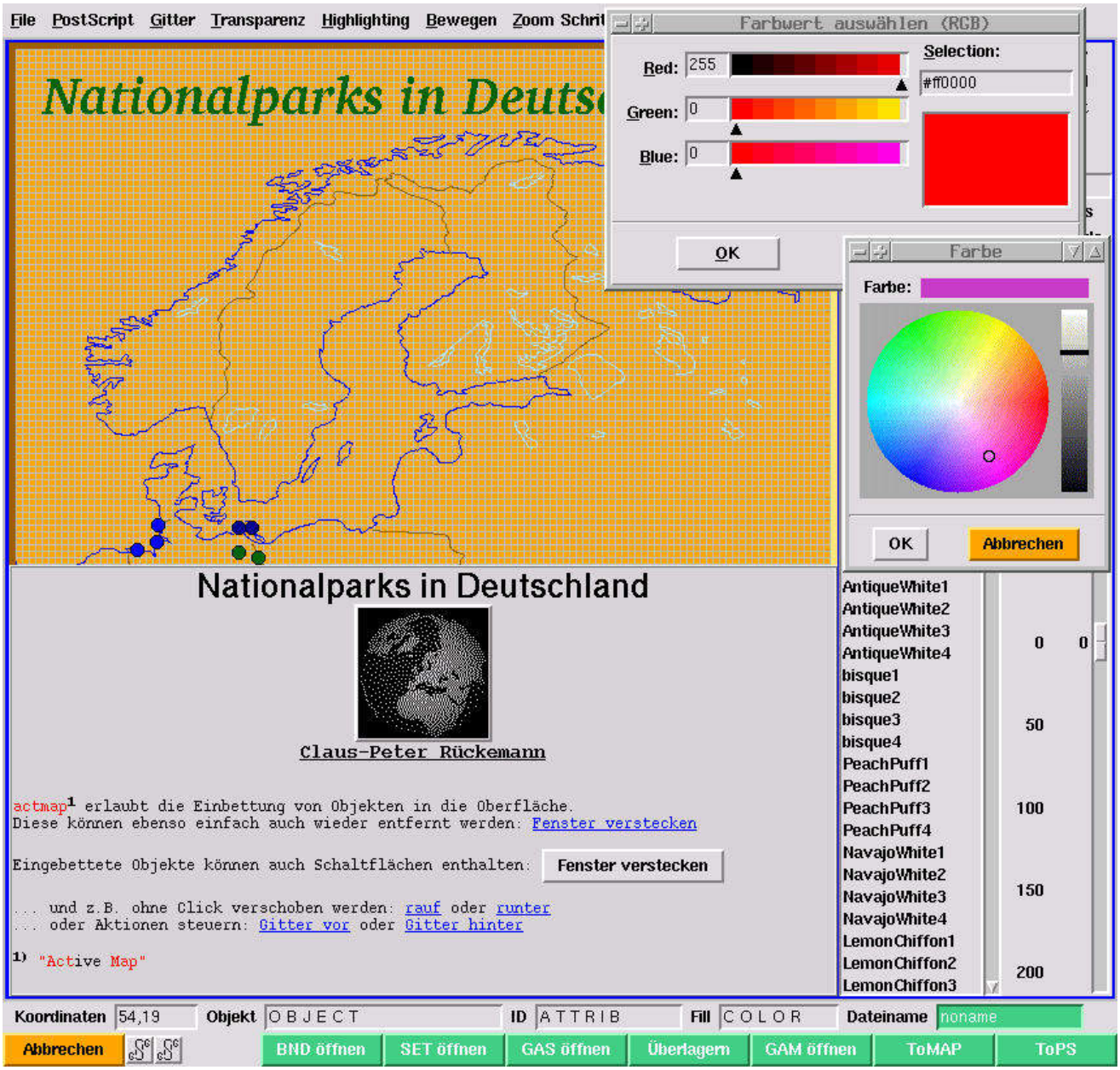

Abb. 10.11: Fallbeispiel: Modifikation der Oberfläche durch Objektgraphik (Nationalparks)

Eine minimale derartige benutzerdefinierte Anordnung der Elemente der Bedienoberfläche aus den bereitgestellten Konfigurationen ist in Abbildung 10.12 dargestellt. Es handelt sich um die Komponente `actmap` mit einer PDA ähnlichen Konfiguration der Oberfläche. Weitere Beispiele sind u. a. für eine platzsparendere Darstellung auf Notebook Bildschirmen in der Komponente `actmap` enthalten.

Dargestellt ist ein Ausschnitt des Canvas mit Luftbild und Vektordaten und einem Teil eines Gewässerobjekts.

Durch die interaktive Ausblendung bestimmter Elemente der Oberfläche und die Umkonfiguration und Einblendung einiger Anzeigefelder läßt sich so eine Anwendung für spezielle Aufgaben zusammenstellen.

Neue Bedienelemente, Felder, Funktionen oder andere Elemente können in die bestehende Komponente nach Belieben hinzugefügt und flexibel angeordnet werden, ohne daß Kernfunktionen modifiziert werden müssen.

Für spezielle Fälle ist auch das dynamische und beispielsweise kontextabhängige Einblenden mehrerer verschiedener Elemente an der gleichen Stelle denkbar.

Zu dem letzten aktiven Objekt sind in den Anzeigefeldern einige Daten angezeigt, wie Typ, Zeigerposition, Farbe des nicht aktiven Objekts und Attribute. Die Attribute beschreiben hier: Typ des Vektorobjekts, Art des Gewässers, Identifikationsnummer der Gewässers,

Einstufung der Gewässergüte, Kennzeichen des aktuell ausgewählten Objekts.

An bestimmten Koordinatenpunkten sind interaktiv Markierungen eingeblendet. Zur Verdeutlichung sind diese Markierungen nicht teiltransparent dargestellt. Wie bei allen Objekten von diesem Typ lassen sich auch hier die Transparenz des Hintergrunds und andere Eigenschaften interaktiv und dynamisch verändern.

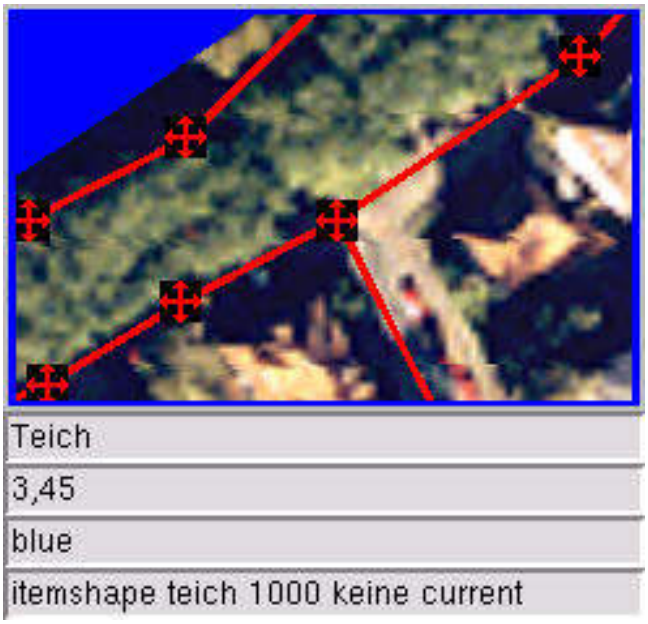

Abb. 10.12: Fallbeispiel: Konfiguration der Elemente der Bedienoberfläche in minimalem PDA Stil (Münster, Aasee)

In dieser benutzerdefinierten Oberfläche sind *alle* Funktionen der Komponente `actmap` verfügbar. Die Karte ist vollständig aktiv und z. B. auch ohne die Menüs verschiebbar und skalierbar. Objekte können beispielsweise neue Farbzuweisungen erhalten, bewegt oder gelöscht werden. Die Menüs lassen sich z. B. interaktiv ausblenden und wieder einblenden.

10.11. Dynamische Visualisierung eingebettet in Objektgraphik

Mittels dynamischer Visualisierung sind vielfältige Zusammenhänge je nach Interaktion darstellbar.

Die folgenden beiden Abbildungen (10.13 auf Seite 100, 10.14 auf Seite 100) zeigen einen Datensatz, der verschiedene Eigenschaften dynamischer Visualisierung demonstriert. Die dargestellten Informationen sind für dieses Beispiel inhaltlich belanglos.

Dargestellt ist in beiden Fällen ein Teil einer Europakarte, mit verschiedenen thematischen Daten, einmal mit einem Streudiagramm und einmal mit einem Verteilungsdiagramm größengewichteter Symbole. Beide Datensätze sind vollständig autark und selbstdarstellend, d. h. es wird keine spezielle Software benötigt, um sie darzustellen oder ihre Funktionen zu nutzen.

Alle Diagramme und Karten sind dynamisch und interaktiv. Diese Datensätze können direkt für eine interaktive dynamische Darstellung im Netz eingesetzt werden. Ereignisse werden unter anderem zur Steuerung eines transienten Symbolismus ausgenutzt, d. h. Farben, Helligkeit oder Füllmuster variieren mit der Interaktivität, Symbole können z. B. bei der Änderung des Betrachtungsfokus verändert werden.

Den Achsen können interaktiv verschiedene thematische Datenkombinationen zugeordnet werden. Es können verschiedene Objekte auf der Karte oder im Diagramm selektiert werden, die einen dynamischen wechselseitigen Einfluß auf die gesamte Darstellung haben. Für die räumlichen Relationen sind einige einfache räumliche kartographische Funktionen zur Unterstützung der dynamischen Visualisierung enthalten, z. B. die automatische Ermittlung und Auswahl der Nachbarobjekte. Da diese Daten *autark* und selbstdarstellend sind, vermutet man in der Regel keine weitere Integration in andere Anwendungen. Beide Datensätze sind jedoch aufeinander abgestimmt und enthalten gegenseitige Referenzen, die nur zum Einsatz kommen, wenn beide Datensätze in eine übergeordnete Komponente geladen werden. Daten von diesem Typ werden in Verwendung mit dem Prototyp als „GIS Dynamic Chart“ (GDC, `.gdc`) bezeichnet.

Abbildung 10.15 (Seite 101) zeigt die beiden autarken dynamischen Datensätze aus den Abbildungen 10.13 (von Seite 100) und 10.14, (von Seite 100) wenn sie in geeigneter Weise in der Komponente `actmap` zusammengeführt werden (z. B. `source`-Befehl).

Beide Datensätze sind vor dem Laden nicht verändert worden. Um ihre Verwendung mit anderen Datensätzen zu veranschaulichen, wurden diese Datensätze zusätzlich über einen anderen räumlichen Datensatz geladen, der seine Funktionen weiterhin vollständig behält. Die unterschiedliche Anordnung, die Einsparung einer Bedienoberfläche sowie weitere Funktionen, die nur innerhalb von `actmap` genutzt werden können (`chtpack`, `chtpackfill`, `chtpacknofill`, `Control-F5`, `Control-F6`), resultieren aus der Abstimmung der Datensätze untereinander und Anteilen, die nur innerhalb der übergeordneten Komponente ausgewertet werden.

Abb. 10.13: Fallbeispiel: Autarke dynamische Visualisierung (thematische Daten, Verteilungsdiagramm)

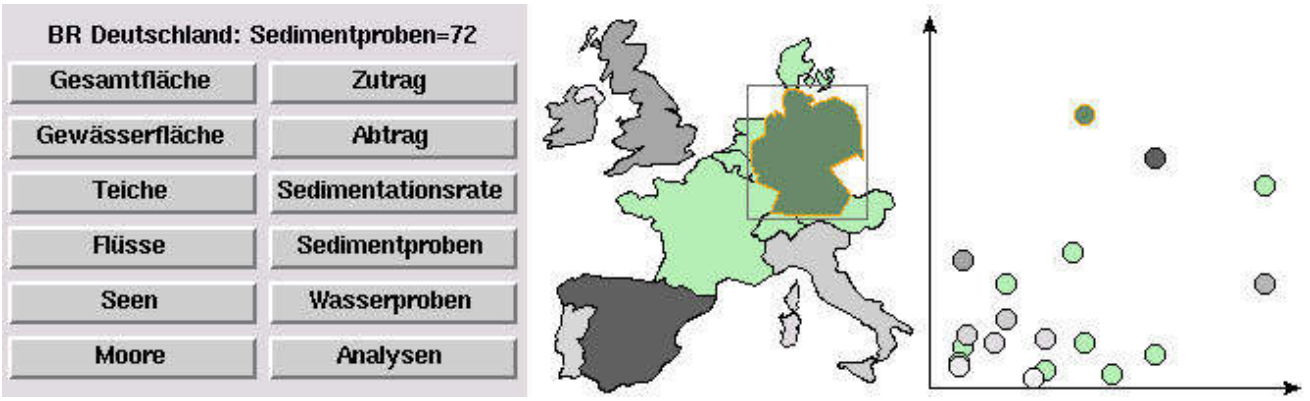

Abb. 10.14: Fallbeispiel: Autarke dynamische Visualisierung (thematische Daten, Streudiagramm)

Die Funktionen und Eigenschaften beider dynamischer Visualisierungen sind auch *innerhalb der übergeordneten Komponente* erhalten und *um eine gegenseitige Dynamik erweitert*, im Gegensatz zu der Verwendung der einzelnen Datensätze für sich.

Werden bestimmte Interaktionen in einem Datensatz ausgeführt, dann reagieren daher auch bestimmte Teile der dynamischen Darstellung des anderen. Dies wird durch die einbettende Komponente `actmap` ermöglicht. Durch die Einbettung sind alle Eigenschaften und Funktionen in gleicher Weise durch die `actmap` Shell erreichbar, wie dies auch bei beliebigen Objekten der Fall ist.

Durch eine konsequente Anwendung des Konzepts und der Ausnutzung der Eigenschaften des vorliegenden Prototyps können speziell angepaßte ereignisgesteuerte und dynamische Visualisierungen entstehen.

Für derartige dynamische Visualisierungen ist allerdings ein erheblicher Anteil nur durch manuelle Anpassungen herzustellen. Die gesamten Relationen, Dynamiken, die Aufbereitung und Strukturierung des Datenmaterials und andere Aspekte müssen in den meisten Einzelfällen logisch neu konzipiert und aufeinander abgestimmt werden und können auch mittelfristig sicherlich nicht automatisiert werden. Auf diese Weise sind aber nicht nur dynamische kartographische Funktionen, sondern auch ein weitergehender dynamischer Symbolismus möglich. Symbolismus ist somit implizit durch Ereignisse dynamisch veränderbar.

Gerade diese Fähigkeiten gehen weit über die Möglichkeiten konventioneller Softwarekonzepte hinaus und eröffnen eine gewaltige Fülle von Anwendungen dynamischer Visualisierungen.

Dies kann für zukünftige Entwicklungen, beispielsweise zur Erreichung eines hohen Grades an Internationalisierbarkeit und zur Erhöhung des kognostischen Informationsgehalts der Daten, ausgenutzt werden.

10.12. Nutzung einer Shell zur Handhabung von Objektgraphik

Neben der Steuerung über die erweiterbare graphische Oberfläche ist auch eine vollständige Steuerung mittels einer Shell, z. B. aus einem Terminal möglich. Diese Flexibilität ergibt sich aus dem Interpreter, der durch das Verfahren integriert ist.

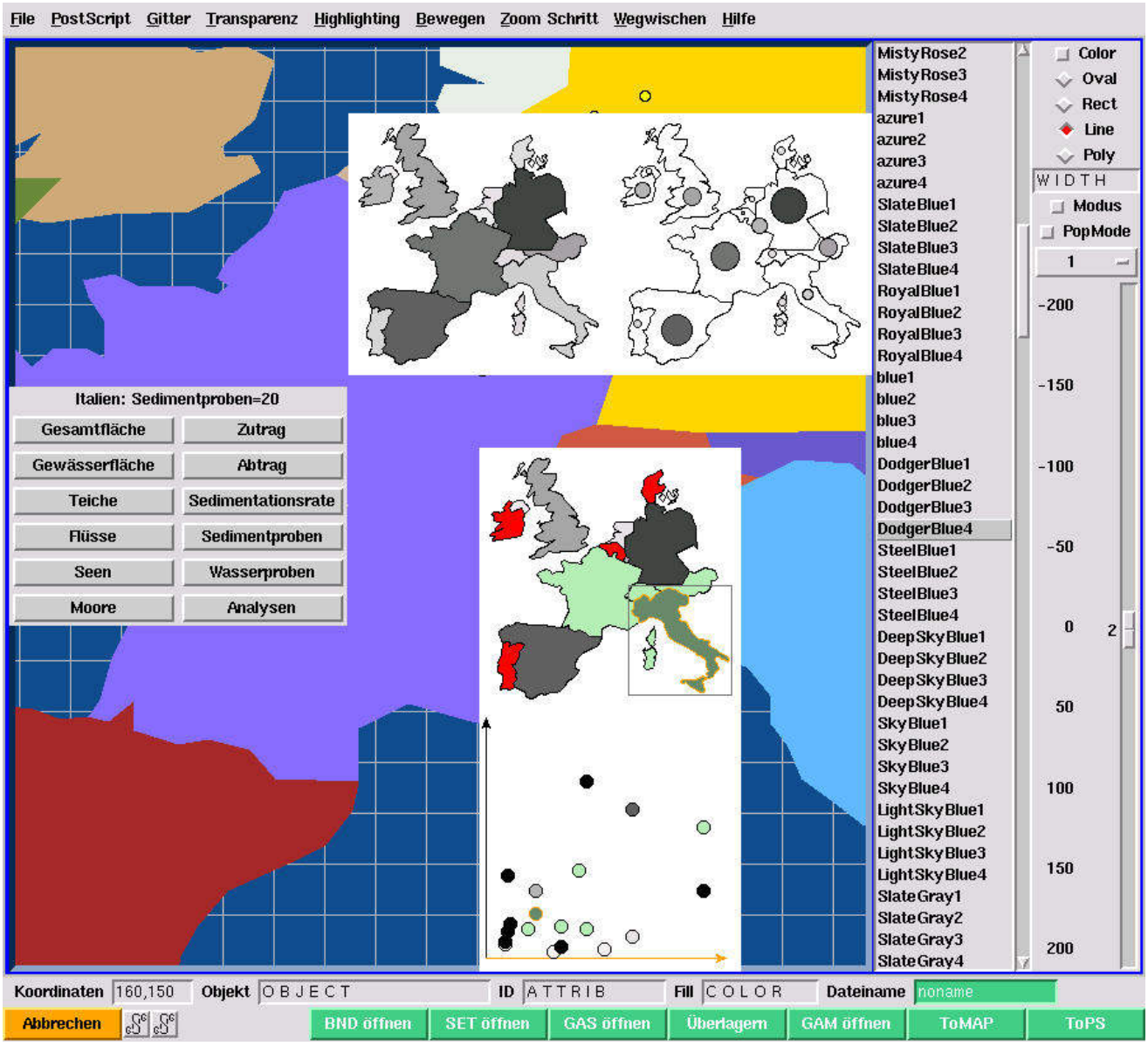

Abb. 10.15: Fallbeispiel: Dynamische Visualisierung eingebettet in Objektgraphik (thematische Daten, Diagramme)

Abbildung 10.16 (Seite 102) zeigt einen Auszug aus einer Sitzung mit der integrierten Shell mit einigen Befehlen. Manuell oder aus Skripten in der Shell ausgeführte Befehle können zur Steuerung aller Teile der Daten und der Oberfläche verwendet werden. Neben Befehlen zur Manipulation von Objekten und zur Darstellung werden auch Koordinaten eines Objekts abgefragt, Objekte in einem Ausschnitt zur Feststellung ihrer Attribute ermittelt, ein externes Programm im Hintergrund gestartet, alle aktiven Funktionsnamen beginnend mit `o` und `s` aufgelistet, eine PostScript Graphik der aktuellen Darstellung weggeschrieben, ein externes Skript ausgeführt, `Tcl` Makrofunktionen geladen und ein Befehlsprotokoll der Sitzung aufgelistet (nur der Anfang dieses Protokolls ist dargestellt).

Eine Beschreibung aller derzeit in den Prototyp implementierten Funktionen kann aufgrund des erreichten Umfangs an dieser Stelle nicht erfolgen. Die Erarbeitung einer eigenen Programmierschnittstelle (API) und der dafür notwendigen Dokumentation für Entwickler und Anwender war nicht Bestandteil dieser Dissertation. Es können aber zum weiteren Studium alle enthaltenen Funktionen in der Shell abgefragt und für eigene Tests beliebig eingesetzt werden. Funktionsnamen, die für eine Anwendung durch Benutzer gedacht sind, beginnen prinzipiell mit einem Kleinbuchstaben.

Aufgrund des Verfahrens kann Zugriff auf alle vorhandenen Funktionen der laufenden Komponente er-

```
actmap shell: (10) % deleteShape 4
actmap shell: (11) % $w move germany 120 60
actmap shell: (12) % $w raise germany
actmap shell: (13) % $w lower eire
actmap shell: (14) % $w delete gb
actmap shell: (15) % drawGrid raise 5
actmap shell: (16) % removeGrid
actmap shell: (17) % drawGrid lower 20
actmap shell: (18) % moveUp 20
actmap shell: (19) % showName testname
actmap shell: (20) % $w configure -background green
actmap shell: (21) % $w coords eire
91.012 145.236 82.368 131.592 57.96 138.772 48.62 129.668 59.88 116.74
48.62 111.228 42.156 117.98 40.876 124.392 33.18 130.804 38.548 137.088
30.616 138.496 25.488 142.344 18.436 141.06 10.1 140.42 4.54 146.188
11.024 151.32 11.024 164.632 21.0 166.708 30.416 168.224 13.944 182.096
12.928 188.584 19.076 188.584 34.008 184.992 19.716 191.076 8.176 191.716
0.0 198.64 0.0 204.54 12.664 203.256 14.616 207.98 6.464 209.668 6.464
215.412 18.436 212.876 28.052 213.516 62.676 205.18 78.704 203.896 84.532
198.64 90.248 171.196 87.684 159.656 91.012 145.236
actmap shell: (22) % listItemTags [getItemsEnclosed 0 0 200 200]
itemshape country hebrides
itemshape country nireland
actmap shell: (23) % exec xterm &
4721
actmap shell: (24) % o
ambiguous command name "o": open openPict openPictBind openPictSet
openPictSource openSource option overlayPict overlayPictSource
actmap shell: (25) % s
ambiguous command name "s": savePS savePSOverallSize savePSgray
savePSmono savePict scale scaleAllCanvas scaleNamedRaster scaleObjectCC
scalePhoto scaleRaster scan scrollButton scrollEnter scrollLeave
scrollbar seek selection send set setBorderWidthfromentry setCmsg
setCmsgWarning setColorfromentry setItemColor setObjectfromentry
setScaleFactorSignx setScaleFactorSigny setTagAttributefromentry setWidth
sfileBrowserImportCanvas sfileBrowserOpenPictBind sfileBrowserOpenPictSet
sfileBrowserOpenPictSource sfileBrowserOpenSource
sfileBrowserOverlayPictSource showCoord showName showUsage
simpleSystemCall simpleSystemCallBG socket solid sopenPictBind
sopenPictSet sopenPictSource sopenSource source soverlayPictSource split
startupFlash string subst switch
actmap shell: (26) % savePS noname.eps
actmap shell: (27) % exec myscript.pl
actmap shell: (28) % source myfuncs.tcl
actmap shell: (29) % history
     1  listItemTags [getItemsAll]
     2  clearCanvas $w
     3  history
```

Abb. 10.16: Fallbeispiel: Nutzung einer Shell in Verbindung mit Objektgraphik

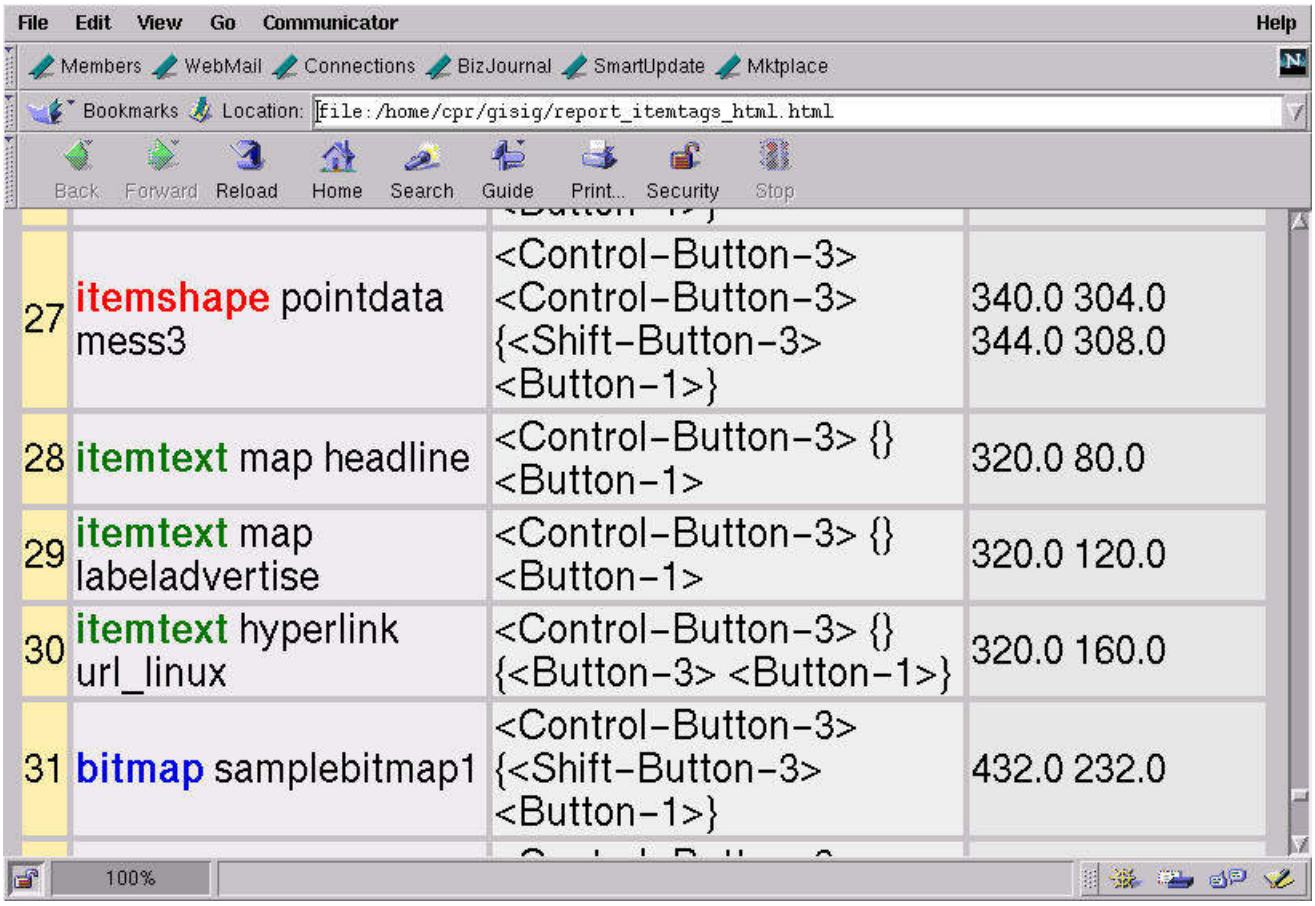

Abb. 10.17: Fallbeispiel: Fragment des dargestellten Reports

möglicht werden, egal ob diese z. B. in einer Skript-Sammlung von Tcl Prozeduren oder z. B. als C-Routine in einem eigenen Interpreter integriert sind. Ein grundlegendes Verständnis der Verwendung von Tcl/Tk ist bei dieser Implementierung minimale Voraussetzung.

Einzelne Komponenten sind damit auch auf diesem Wege beliebig erweiterbar und konfigurierbar, z. B. für die Handhabung spezieller Datensätze bzw. für benutzerdefinierte Anwendungen und eigene Funktionen.

Durch das Verfahren, je nach Anforderung mehr oder weniger Anteil an Skript oder Maschinensprache dynamisch zu nutzen, bieten sich für die Visualisierung vielfältige dynamische und interaktive Einsatzmöglichkeiten, die bei konventioneller Darstellung nur schwer genutzt werden könnten.

10.13. Nutzung eigener Funktionen über die Shell

Ein kleines aber nicht mehr triviales Beispiel soll dies veranschaulichen. Es sollen die Identifikationsnummern, Attribute, Ereignisse und Koordinaten eines Datensatzes in einer Liste in HTML ausgegeben werden. Ein Fragment der Ausgabe, dargestellt in einem Browser, zeigt Abbildung 10.17.

Zu jedem Attribut eines Objekts sind die aktuell vergebenen Ereignisse in einer Spalte in der Reihenfolge der angegebenen Attribute aufgeführt. Ist einem Attribut mehr als ein Ereignis zugeordnet, sind diese als Gruppe mit geschwungenen Klammern umgeben.

Durch ein kleines Skript, das in der Shell der Kernkomponente lauffähig ist, kann diese Ausgabe erzeugt werden (Abbildung 10.18 auf Seite 104).

Es werden die übergebenen Argumente übernommen, eine Datei geöffnet, die Identifikationsnummern der aktuell eingeschlossenen Objekte in eine Liste geschrieben und in einer Schleife die Attribute der Objekte und die zugehörigen Ereignisse und Koordinaten der Objekte in eine HTML-Tabelle geschrieben.

Dabei werden bestimmte Attribute in einem optionalen Block farblich und mit unterschiedlichen Zeichensätzen aufbereitet.

Anschließend werden die farblich hervorgehobenen Tabellenzeilen geschrieben und die Datei geschlossen.

```
#==========================================================================
# report_itemtags_html.tcl -- Example -- (c) Claus-Peter Rückemann, 2001
#
# Report of enclosed item id, tags, bind and coords in HTML format.
#
# usage from interactive actmap shell:
#   source report_itemtags_html.tcl
#   report_itemtags_html report_itemtags_html.html 0 0 1000 1000
#
# proc arg1   : file to output data          [report_itemtags_html.html]
# proc arg2-5 : x0, y0, x1, y1 for selected area   [0],[0],[1000],[1000]
#==========================================================================
proc report_itemtags_html { FileName x0 y0 x1 y1 } {
global w
set F [open $FileName w+]
set listone [getItemsEnclosed $x0 $y0 $x1 $y1]
puts $F "<table>"
foreach { a } $listone {
 puts $F "<tr>"
 set myitemtags [getItemTags $a]
 set myitembinds ""
 foreach { i } $myitemtags {
   catch {
     lappend myitembinds [$w bind $i]
   }
 }
 regsub -all "bg_bitmap " $myitemtags "<b>bg_bitmap</b> " myitemtags
 regsub -all "photo " $myitemtags \
        "<font color=\"#0000ff\"><b>photo</b></font> " myitemtags
 regsub -all "bitmap " $myitemtags \
        "<font color=\"#0000dd\"><b>bitmap</b></font> " myitemtags
 regsub -all "itemshape " $myitemtags \
        "<font color=\"\#ff0000\"><b>itemshape</b></font> " myitemtags
 regsub -all "itemtext " $myitemtags \
        "<font color=\"\#007700\"><b>itemtext</b></font> " myitemtags
 regsub -all "<" $myitembinds "\\&lt;" myitembinds
 regsub -all ">" $myitembinds "\\&gt;" myitembinds

 puts $F [ format "<td bgcolor=\"\#ffeeaa\" align=\"right\">%s</td>" $a ]
 puts $F [ format "<td bgcolor=\"\#eaeaea\">%s</td>" $myitemtags ]
 puts $F [ format "<td bgcolor=\"\#efefef\">%s</td>" $myitembinds ]
 puts $F [ format "<td bgcolor=\"\#e4e4e4\">%s</td>" [$w coords $a] ]
 puts $F "</tr>"
}
puts $F "</table>"
close $F
}
##EOF:
```

Abb. 10.18: Fallbeispiel: Funktion für benutzerdefinierten Report mit HTML Ausgabe

11. Evaluierung

11.1. Anlaß zur Evaluierung

Den Kern dieser Dissertation bildet das beschriebene Konzept. Dieses adressiert ein praktisches Problem und daher schließt diese Arbeit die Implementierung eines Prototyps als Nachweis des Konzepts (engl.: „proof of concepts“) mit ein. Bei diesem Prototyp handelt es sich wiederum um ein von Anwendern benutzbares und weiterentwickelbares System.

Aus verschiedener Hinsicht ist es also von Interesse welcher Aufwand erbracht werden mußte und welcher Zustand der Entwicklungen damit erreicht werden konnte.

11.2. Zeitlicher Aufwand

Neben den gewünschten Eigenschaften steht im Vordergrund der meisten Entwicklungen der zeitliche Aufwand, den eine Entwicklung mit sich bringt (s. Kapitel 3, Seite 21).

Dieser, vor allem hinsichtlich der Systemplanung und Entwicklung, nicht unerhebliche Aspekt ist hier vorangestellt, um anschließend eine geschlossenere Darstellung der wichtigsten aufgabenbezogenen Aspekte geben zu können.

Eine Einschätzung unterliegt in jedem Einzelfall vielen Kriterien, daher sollen hier die diesbezüglichen Erfahrungen aus den Entwicklungen zu dieser Dissertation ohne den Anspruch auf Allgemeingültigkeit dargestellt werden.

Die Daten beziehen sich im wesentlichen auf die Arbeiten zu der Kernkomponente `actmap` und ihrem Umfeld, wie der Konzipierung und Schaffung geeigneten Datenmaterials.

11.2.1. Prototyp

Die Implementierung umfaßt zu diesem Zeitpunkt cirka 20 000 Zeilen funktionalen hochsprachlichen Programmtext ohne Anteil von Interpretern oder verwendeten Bibliotheken und einige zusätzliche Bibliotheken, Filter und ähnliches Material mit einigen tausend effektiven Zeilen.

Der bisher geleistete zeitliche Gesamtaufwand für die Planung, Entwicklung und Tests des hier beschriebenen Prototyps mit seinen Beispielen beträgt über 8000 Arbeitsstunden, d. h. etwa 1000 Arbeitstage.

Davon entfallen cirka 10 Prozent auf Planung und Konzept, 20–30 Prozent auf Tests mit verschiedenem Datenmaterial und der Rest auf die Implementierung aller Einzelheiten.

Tabelle 11.1 (Seite 106) gibt einen Überblick über den zeitlichen Aufwand für den Prototyp der zentralen Komponente. Die angefallenen Teilsummen von zusammen über 8000 Stunden sind **fett** hervorgehoben.

Die Tabelle ist horizontal in Teilaufgaben geteilt, bezüglich der für diese Dissertation vorrangigen Bedeutung und geringeren Relevanz. Bei vorrangigen Aufgaben wurde bei der Planung nicht in Teilaufgaben unterschieden. Der einzelne Aufwand stellte sich erst im Laufe der Entwicklung heraus.

Bei den für die Arbeit weniger relevanten Aufgaben ist für ein mögliches Endprodukt der Aufwand deutlich höher, da diese Aspekte jenseits der eigentlichen Ziele dieser Arbeit liegen und zum großen Teil Anwenderbelange betreffen. Aufgrund des größeren Aufwandes wurde bei der Planung jeweils ein Teilaufwand geschätzt.

Außer acht blieb hierbei der Zeitaufwand für Recherche, Literatur oder Kommunikation. Nicht berücksichtigt wurden gleichermaßen die wenig zu kalkulierenden Zeiträume, wie zukünftige mögliche Alpha-Phase (ca. 6 Monate) oder Beta-Phase (ca. 3 Monate), da solche sehr stark von weiteren Faktoren abhängen.

Diese Werte sind wesentlich von dem Umfang der entwickelten Anwendung aber auch dem fachlichen und technischen Hintergrund der Anwender abhängig, können aber relativ zueinander eine Einschätzung vermitteln.

Ein sehr großer Teil des zeitlichen Aufwands, der bei einer solchen Neuentwicklung neben der Planung der Entwicklungsumgebung notwendig wurde, entfiel

Teilbereich	*Priorität*	*Arbeitsstunden*			
		Planung 199707	*bis 199809*	*bis 200101*	*noch*
Basisfunktionen und Tests	✓✓✓	ca. 6000	200	150	3000
Analyse, Entwurf, Vergleich usw.	✓✓✓	k. A.	>300	200	100
Basis GUI, Ereignisunterstützung, Graphik	✓✓✓	k. A.	2000	200	200
Werkzeuge, Automatisierung	✓✓✓	k. A.	300	>300	50
Objektgraphik, Objektdaten	✓✓✓	k. A.	–	200	?
Demos, Demodaten	✓✓	k. A.	–	600	?
Analyse/Test Formate	✓✓✓	k. A.	–	800	100
Test Plugin	✓✓✓	k. A.	–	400	100
Test Netzwerk	✓✓	k. A.	–	150	200
Test Konfiguration	✓✓✓	k. A.	–	250	100
Test Wrapper, Bytecode	✓✓✓	k. A.	–	200	200
Trennung, Bibliotheken etc.	✓✓✓	k. A.	–	150	50
spez. Funktionen (Skalierung, Animation etc.)	✓✓	k. A.	–	350	400
Modularisierung	✓✓✓	k. A.	–	200	200
Unterstützung von Dokumentation	✓✓✓	k. A.	200	150	?
		ca. 6000	>**3000**	>**4300**	?
Ausbau GUI	✗	>4000	–	>500	3500
Nachbildung spezieller Funktionen	✗	5000	–	–	5000
Portierung (1/2 a)	✗	1500	–	–	1500
API (1 a)	✗✗✗	3000	–	–	2800
Vorstudie API	✗✗	400	–	200	200
Internetfähigkeit (1 a)	✗✗✗	3000	–	–	3000
Plugin/Sicherheit (2 a)	✗✗✗	6000	–	500	5500
Dokumentation (derzeit mind. 300 S.)	✗✗	1000	–	–	1000
		>23900		>**1200**	>22500

✓: für die Arbeit von vorrangiger Bedeutung
✗: für die Arbeit wenig relevant, ✗✗: weniger relevant
k. A.: keine Angaben (nicht differenziert)
?: unbestimmt, aufgrund vieler Faktoren nicht abschätzbar

Tab. 11.1: Zeitlicher Aufwand für den zu dieser Dissertation entwickelten Prototyp der neuen Kernkomponente `actmap`

auf die Beschäftigung mit dem Datenmaterial.

Bei der Umsetzung wenig oder gar nicht dokumentierten Datenmaterials sind in aller Regel viele hundert Stunden notwendig, um das Datenmaterial für eine grundlegend neue Verwendung aufzubereiten. Hier wird in einem größeren Rahmen eine Einbindung geeigneter Verfahren eine erhebliche Erleichterung darstellen.

11.2.2. Mögliche Anwendungen

Unter der Voraussetzung der bestehenden Erfahrungen aus den genannten Beispielen können eigene Aufbereitungen und Anwendungen verhältnismäßig schnell implementiert werden.

Dennoch kann dies nur eine vorsichtige Schätzung sein, da keine realen Anwender hinzugezogen werden konnten.

Üblicherweise reduziert sich bei einer wachsenden Zahl umgesetzter Lösungen der Aufwand für zahlreiche Anteile neuer Anwendungen aufgrund synergetischer Effekte.

Dies gilt im Rahmen der Tests für die Lösungen mit dem vorliegenden Prototyp in besonderem Maße, da fast alle einmal entstandenen Teile einer Komponente oder Daten und Funktionen sehr flexibel erweiterbar, modifizierbar und sogar dynamisch umkonfigurierbar sind.

Der Grad der Wiederverwendbarkeit kann dadurch sehr hoch gehalten werden.

Dies gilt auch für Anwendungen und Lösungen, die potentielle Anwender erstellen. Wie bei den meisten Problemlösungen wächst dabei mit der Erfahrung nicht nur die Effizienz bei der Wiederverwendung, sondern vereinfacht sich auch der Umgang mit der Portabilität.

Der zeitliche Aufwand für einen Anwender mit grundlegenden Kenntnissen bezüglich Tcl/Tk und dem Umgang mit dem Prototyp kann für verschiedene Zielsetzungen etwa in folgenden Größenordnungen eingeschätzt werden (Tabelle 11.2).

Zielsetzung	*Arbeitsaufwand*
einfacher neuer Datensatz	5 h
Konvertierung fremder Datensätze	1–100 h
einfache Funktion für Anwendung	10 min
Aufbereitung für Plugin	5 min–1 h
Konfiguration der Oberfläche	1–20 h
Oberfläche zu Anwendung	>50 h
benutzerdefiniertes Skript	>2 h
kleine neue Bibliothek	>5 h
kleine neue Komponente	>5 h

Tab. 11.2: Zeitlicher Aufwand für benutzerdefinierte Anwendungen und Modifikationen der neuen Kernkomponente `actmap`

Eine Unterscheidung des zeitlichen Aufwands für native oder objektorientierte Daten erschien nicht sinnvoll, da der Einsatz sehr von den Erfahrungen des Nutzers und dem speziellen Zweck abhängt.

Die Konvertierung hängt stark von der Komplexität der Daten, dem Ziel der Aufbereitung und den zur Aufbereitung verwendeten Mitteln ab. Skripting kann dabei eine große Hilfe darstellen.

Der Aufwand für die Oberfläche wird entsprechend höher eingeschätzt, weil diese Tätigkeiten sich in den meisten Fällen kaum in gleicher Weise bei neuen Anwendungen wiederholen lassen und ein gewissermaßen höheres Maß an Planung erfordern.

11.3. Einsatz für die geforderten Aufgaben

Der Prototyp erfüllt alle Anforderungen, die bei der Planung in Betracht gezogen wurden. Darüber hinaus sind durch das verwendete Konzept viele weitere Eigenschaften entstanden, die teilweise auf Synergieeffekten beruhen.

Bezüglich der konzeptionellen Basis sind vor allem folgende Eigenschaften zu nennen:

- Die Anbindung von Ereignissen ist implizit.
- Dynamische Daten können sehr effizient behandelt werden.
- Alle benötigten Grundelemente, z. B. graphische Primitive, sind vorhanden.
- Die Daten sind in der grundlegenden Form durchgehend strukturiert aufgebaut, intuitiv und selbsterklärend.
- Die Basis für die eingesetzten Werkzeuge ist sehr gut dokumentiert.
- Daten und Komponenten sind mit verwandten Mitteln aufgebaut.
- Daten und Komponenten sind beliebig erweiterbar.
- Verfügbare und eigene Interpreter schaffen eine flexibel programmierbare Umgebung, die den Zugriff auf beliebige externe Werkzeuge ermöglicht.
- Daten und Funktionen können im Quellentext, als auch als Bytecode verwendet werden. In Zukunft werden hier weitere Möglichkeiten z. B. durch die Verbindung mit Java entstehen.

Bezüglich der Nutzung entstanden aus dem Zusammenspiel der verwendeten Verfahren zur Umsetzung des Konzepts folgende weiterführende Lösungen:

- Es lassen sich neue Konzepte, z. B. basierend auf objektorientierten Datenstrukturen erstellen.
- Es lassen sich mit geringem Aufwand eigene sichere Funktionen für die Interpretation *aller* Teile eigener Komponenten und Daten erstellen.
- Es existieren bereits viele Werkzeuge für die Bearbeitung von Quellentexten, wie beispielsweise Syntaxhervorhebung, Syntaxprüfung, Formatierhilfen und Teile kompletter Entwicklungsumgebungen.
- Es werden alle gängigen Techniken unterstützt.
- Es besteht die Möglichkeit für verschiedene Unterstützung durch eigene Interpreter, z. B. unter Nutzung eigener dynamischer Bibliotheken.

Die folgenden Punkte kann man hingegen aus der Sicht des nicht wissenschaftlichen Einsatzes als Nachteile auffassen.

Auf der Gegenseite zu der hohen Flexibilität sollte der Nutzer aber wissen, was er möchte und vor allem, wie seine Daten aufgebaut sind. Dies ist ohnehin für die meisten Einsatzbereiche zwingend erforderlich.

Derjenige aber, der die Verwendung eines Skripts oder eigener Makrofunktionen nicht als mittel- und langfristige Vereinfachung von Routineaufgaben ansieht und z. B. keinen Einblick in den Aufbau seiner Daten, Tcl/Tk, Perl oder HTML haben möchte, wird an einer flexiblen Lösung weniger Freude haben.

Für diese Zwecke können einfachere Hilfsmittel auf der vorgestellten Basis entwickelt werden, die spezielle Aufgabenbereiche abdecken.

Für komplexer aufgebaute Anwendungen kann bei weniger sauberer Trennung der einzelnen Teile im extremen Fall eine sehr individuelle Konstruktion aus Programm, Daten und externen Werkzeugen werden, die folgende Nachteile haben kann.

1. Die Werkzeuge zur Bearbeitung werden aufwendiger.

2. Die Funktionen in Komponenten müssen erweitert werden, z. B. beim Export für verschiedene Einsatzbereiche.

3. Durch individuellem Stil bei der Erstellung sind Austausch und Wiederverwendbarkeit für andere Anwendungen oder in anderen Komponenten nicht mehr gewährleistet.

Daher sollte in Zukunft die Spezifikation eines Modells zur Beschränkung und Trennung von Funktionen von einem größeren Anwender- und Entwicklerkreis erarbeitet werden.

Dies sollte einhergehen mit der Planung eines komponentenübergreifenden Modells zum Austausch von Daten. Mit dem Einsatz eines entsprechenden Konzepts, derzeit beispielsweise OGDI, sollten Erweiterungen zumindest bezüglich der Austauschbarkeit des Datenmaterials leichter handhabbar sein.

Die bereits angedeutete Erstellung einer umfassenden Programmierschnittstelle (API) für die verschiedenen Aufgabengebiete ist ein weiterer Schritt. Dies kann dazu beitragen, bei derartigen Komponenten eine unerwünschte Diversifikation, ein Auseinanderlaufen in verschiedene technische Entwicklungen, zu vermeiden.

11.4. Technische Randbedingungen

Wie bei allen realen programmtechnischen Umsetzungen existieren Randbedingungen, die eine Verwendbarkeit oder Eignung in gewisser Weise einschränken.

- Latenzen (Verzögerungen, Trägheit) aufgrund Dekodierung und Übertragung, z. B. durch nicht ausreichende Netzverbindungen.

- Hardwarebeschränkungen, z. B. bezüglich Arbeitsspeicher und Rechenkapazität. Dies wird insbesondere bei großen Datenmengen deutlich. In diesen Fällen können spezielle Interpreter entwickelt und die Hilfe einer schnellen Datenbank in Anspruch genommen werden.

Die Entwicklung im Bereich der Kleinrechner und der Ausbau der Netzwerke läßt diese Hindernisse für viele der Aufgaben aber zunehmend in den Hintergrund treten.

11.5. Konzept und Fallstudien

Die Umsetzung des Konzepts zu einem Prototyp der verschiedene Verfahren aufgabenbezogen in geeigneter Weise kombiniert und die Entwicklung einer geeigneten Grundlage für eine Objektgraphik wurde anhand einiger Fallstudien diskutiert.

Es bleibt noch die Frage, welche Kategorien geowissenschaftlicher Anwendungen für den Einsatz denkbar sind und welche Aufgaben prinzipiell erfüllt werden können.

Die Antwort hängt, bezogen auf den Prototyp, stark von Art und Umfang der zukünftigen Entwicklungen, dem weiteren Ausbau und der Integration neuer Komponenten ab und kann strenggenommen nur in dem Rahmen als vielversprechend beantwortet werden, in dem diese Arbeit Anwendungen behandelt und aufgezeigt hat.

Die Antwort, bezogen auf das Konzept für die Entwicklung von Komponenten für ereignisgesteuerte und dynamische Visualisierung geowissenschaftlicher Daten und Zusammenhänge, ist jedoch viel weitreichender, da die Erfahrungen aus der Implementierung mit eingehen können.

Zusammengefaßt ergeben sich folgende ausgewählte Kategorien und Anwendungen:

- Ereignisunterstützte Hydrogeologie und Umweltgeologie.
  Beispiele: Flächenhafte und punktuelle Darstellung der Gewässerbelastung, aktive punktuelle Darstellung der Orte der Probenentnahme, dynamische Visualisierung des lokalen Kontexts.

- Ereignisunterstützte Visualisierung von geowissenschaftlichen Informationen. Prinzipiell sind beliebige Informationen auf diese Weise visualisierbar.
  Beispiele: Meßdaten zur Kontamination, externe Hintergrundinformationen, Videos und Photos von Lokationen.

- Ereignisgesteuerte Präsentation geowissenschaftlicher Zusammenhänge. Dies kann auf verschiedene Arten erfolgen, wie mittels dynamischer Präsentationen in einer Kernkomponente oder (mit Einschränkungen) über das Internet.
  Beispiele: Geochemische Zusammenhänge und Auswertungen von Probenmaterial in dynamischer Anbindung an räumliche Daten.

- Dynamische Visualisierung und dynamische Kartographie.
  Beispiele: Aktive Anbindung dynamisch errechneter Auswertungen an räumliche Daten.

- WWW-basierte dynamische Visualisierung von geowissenschaftlichen Auswertungen und Zusammenhängen.
  Beispiele: Darstellung und Präsentation von interaktiven und dynamischen räumlichen Daten und Zusammenhängen über das Internet.

- Vermittlung von komplexen geowissenschaftlichen Zusammenhängen durch Nutzung von Objektgraphik zur effizienten Kombination verschiedenster Methoden, Datentypen und Anwendungen.
  Beispiele: An räumliche Daten gebundene dynamische Darstellung und Steuerung dreidimensionaler Moleküle in externen Applikationen.

- Effiziente Fernsteuerung von Applikationen zur Handhabung, Aufbereitung und Visualisierung geowissenschaftlichen Datenmaterials. Mittels IPC ist die Steuerung zwischen verschiedenen Applikationen auf dem gleichen Rechner ebenso möglich, wie die Steuerung beliebig entfernter Applikationen per Netzwerk oder Funkverbindung.
  Beispiele: Steuerung von Objekten und Manipulation von Ereignissen und Visualisierungen über das Internet.

11.6. Einsatzgebiete

Die Einsatzmöglichkeiten sind aufgrund der Erweiterbarkeit durch geeignete Methoden und Werkzeuge sehr vielfältig. Zu den naheliegenden Anwendungen zählen sicherlich ereignisgesteuerte und dynamische Visualisierung und Kartographie, interaktive Präsentation räumlicher Daten und geowissenschaftlicher bzw. geologischer und geophysikalischer Meßdaten, Applets und Backend Internet Anwendungen im Serverbereich zur Unterstützung vorhandener Werkzeuge.

Aufgrund der vorhandenen Schichtung und dem Aufbau der Umgebung ist ein flexibler Zugriff auf alle wichtigen Ressourcen möglich. Dies kann im Extremfall auch einen Zugriff auf die Ereignisdatenbank des X Window Systems oder sogar die Implementierung eines ladbaren Linux Kernel Moduls für sehr spezielle Anwendungen bedeuten.

Meiner Meinung nach würde der Ausbau des bestehenden Systems zu einem konventionellen Datenbanksystem der Konzeption modularer Komponenten widersprechen. Wenn dies für eine Anwendung erforderlich ist, sollte eine Schnittstelle zu einer bestehenden Datenbank erstellt werden, die auf den betreffenden Zweck zugeschnitten ist.

11.7. Zukünftige Entwicklungen

Quellentext-Daten haben bei ihrem Einsatz nur geringe Beschränkungen. Für spezifische Anwendungen können sie auch temporär in ressourcenschonendere Formate überführt werden. Es wird daher hinsichtlich vieler praktischer Anwendungen Ziel sein, die Funktionalität eher einzuschränken bzw. zu begrenzen, um Wildwuchs zu vermeiden. Dies kann durch z. B. ein dokumentiertes und offenes Metaformat erreicht werden. Die wichtigsten Eigenschaften für ein solches übergeordnetes Konzept sind:

- Datenbankeigenschaften.
- Hoher Abstraktionsgrad.
- Berücksichtigung geokognostischer Aspekte.
- Handhabung von Plattformspezifika.
- Berücksichtigung von Erweiterungen.

- Erweiterung auf drei Dimensionen.

Eine derartige Aufgabe kann aber nur unter interdisziplinären Gesichtspunkten zu einem sinnvollen Ergebnis führen.

Wie die Erfahrungen mit dem entwickelten Prototyp gezeigt haben, bestehen für zukünftige Arbeiten auch für die Entwicklung von Anwendungen zahlreiche Möglichkeiten.

- Erweiterung der Oberfläche für höhere Ergonomie und individuelle Anpassungen.
- Spezifikation einer dokumentierten Programmierschnittstelle (API).
- Überarbeitung und Implementierung des Sicherheits- und Netzwerkkonzepts.
- Import- und Exportfunktionen, z. B. Integration von OGDI.
- Verschiedene Canvas Implementierungen mit speziellen Eigenschaften, beispielsweise bezüglich Methoden zur Georeferenzierung.
- Entwicklung einer standardisierten Schnittstelle zu Datenbanken.

Diese Weiterentwicklungen sind nicht auf Funktionalität beschränkt, sondern betreffen auch grundlegendere Bereiche, wie die Entwicklung eines speziellen Interpreters, die die folgende Zusammenstellung zeigt:

- Spezieller Interpreter, basierend auf aktuellem Tcl/Tk Kern mit Erweiterungen, z. B. für erweiterte Unterstützung von Bildformaten, höherer numerischer Präzision und eigenes Speichermanagement für große Daten.
- Umgebung zur interaktiven Manipulation von Quellentext-Daten, z. B. Editor mit Schnittstelle zu Canvas.
- Kernel-Module für spezielle Aufgaben, z. B. für Netzwerkanwendungen.
- Einsatz spezieller Kernel, z. B. Echtzeit-Kernel (RT-Kernel).

Linux wird in immer stärkerem Maße für den Einsatz in eingebetteten Systemen (engl.: „embedded systems“) favorisiert. Tcl/Tk empfiehlt sich zunehmend, um ressourcensparend modulare, flexible und leicht konfigurierbare Applikationen aller Art auf solchen Systemen zu entwickeln.

Sehr erfolgreiche Ansätze in der Kombination von Linux und Skriptsprache liegen seit kurzer Zeit z. B. bei eingebetteten Systemen mit ETLinux[1] vor, bei dem sogar tiefgreifende Systembefehle über einen „Tcl-Motor“ realisiert wurden. Dadurch reduzieren sich die Anforderungen an die eingesetzte Hardware zum Teil erheblich und zugleich steht Tcl generell für Anwendungen zur Verfügung. Neben der dadurch beschleunigten Entwicklung ist die kurze Zeit für zukünftige Anpassungen ein besonders wichtiger Faktor für eingebettete Systeme.

Häufig sind bei eingebetteten Systemen speziell angepaßte Kernel mit andererseits adäquat reduziertem Umfang im Einsatz, die auf stabilen älteren Versionen des Linux Kernels basieren.

Durch eine Entwicklung des Prototyps für eine Reihe von Kerneln, die einige Jahre umfaßt und auch derzeitig verwendete Kernel für eingebettete Systeme einschließt, ist ein zukünftiger Einsatz auf diesen Plattformen vorbereitet worden.

Weiter begünstigend wirkt sich für die Entwicklungen die Beschränkung der Basiskomponenten auf reines Tcl/Tk aus, das auf diesen Systemen für die nächsten Jahre eine ressourcenschonende Basis bilden kann.

Eingebettete Systeme haben bereits heute eine vielfach höhere Verbreitung als alle übrigen Einsatzgebiete von Computern und ihre Zahl wird in Zukunft noch drastisch ansteigen. In naher Zukunft werden in fast allen denkbaren Bereichen des menschlichen Lebens Informationen verarbeitet werden können.

Die Entwicklung geht schon dazu über, mehr Informationen bei einem gleichzeitig effizienteren Einsatz von Software anzubieten. Skriptsprachen sind dabei die allererste Wahl, wenn es darum geht, Informationen bedarfsgerecht zu komplexen modernen Applikationen zu verbinden.

[1] `http://www.etlinux.org` ..... [V: k. A.] [Ä: k. A.] [Z: 25.01.2001]

12. Schlußfolgerungen

12.1. Evaluierung und Einsatzbereiche

Die Evaluierung hat in der Praxis und anhand von Fallstudien gezeigt, daß der Einsatz des vorgestellten Konzepts für weite Einsatzbereiche zur dynamischen Visualisierung in den Geowissenschaften eine vielseitige, flexibel erweiterbare und portable Lösung darstellt.

Zu diesen vielfältigen Einsatzbereichen zählen zusammengefaßt:

- Einfache Erweiterung räumlicher Daten um Ereignisse.
- Die Verwendung von Ereignisanbindungen, flexiblen Ereignismustern und virtuellen Ereignissen.
- Einfache Visualisierung dynamischer räumlicher Daten.
- Nutzung dynamischer Visualisierungen in lokalen Anwendungen und in Netzwerken.
- Nutzung von dynamischen geowissenschaftlichen Daten unter Verwendung von Plugins.
- Wiederverwendung von Daten für spezielle Aufgaben in den Geowissenschaften, insbesondere für einen offenen Datenaustausch.
- Integration dynamischer Daten in eigene Programme und Verwendung von Programmanteilen mit dynamischen Daten.
- Verbindung verschiedener Werkzeuge für die Auswertung, Visualisierung und Präsentation von geowissenschaftlichen Daten. Dies kann basierend auf Komponenten, portablen quelloffenen oder portablen binären Softwarebausteinen, erfolgen.

Mit dem vorgestellten Konzept ist eine definierbare Verknüpfung von Programmanteilen und Datenmaterial auf fast jede erdenkliche Weise realisierbar, um räumliche Daten flexibel, effizient und portabel mit Ereignisdaten bzw. Objektgraphik zu verbinden.

Die Komplexität einer Lösung wie der in dieser Arbeit beschriebenen, ist auf den ersten Blick sicherlich höher, als bei nicht weiterverwendbaren Lösungen. Ein wichtiger Grund dafür ist die geforderte Erhöhung der Relationen der einzelnen Objekte des Datenmaterials, die für erweiterte Funktionen notwendig ist.

Letztendlich kann aber durch die Verwendung in den genannten Bereichen die Vermittlung und Veranschaulichung von geowissenschaftlichen Zusammenhängen profitieren und damit derjenige, für den im geokognostischen Sinne diese Auswertungen gedacht sind.

12.2. Ausblick und zukünftige Technologien

Bei der derzeitigen rasanten Entwicklung im Bereich der Gebrauchselektronik, beispielsweise hinsichtlich eingebetteter Systeme bzw. Betriebssysteme (engl.: „embedded systems“), z. B. Embedded Linux, sind in den nächsten Jahren interessante Entwicklungen zu erwarten.

Einhergehend mit den Entwicklungen im Bereich der Hardware sind Softwarebeispiele, z. B. Weiterentwicklungen Unix-basierter WAP Shells oder zukünftig noch Bluetooth oder UMTS, mit denen die Steuerung von Rechnern mittels mobiler Kleinstgeräte bereits möglich ist.

Die Verfügbarkeit höherer Leistung in Kleinstgeräten, z. B. PDAs, für die heute im besten Fall nur einfache, konventionelle Routenplaner mit geringen Erweiterungsmöglichkeiten zur Verfügung stehen, wird Entwicklungen, die nach dem vorgestellten Konzept entworfen wurden, aufgrund ihrer weitergehenden Flexibilität zahlreiche neue Anwendungsfelder erschließen.

Die vorhandenen Mittel lassen den Einsatz von Konzepten, wie dem in dieser Arbeit entwickelten, auch auf Geräte übertragen, die nicht unbedingt konventionelle Computer bzw. Workstations sein müssen.

Ein wesentlich unterstützender Grund hierfür ist die freie und offene Verfügbarkeit der eingesetzten Werkzeuge. Essentiell ist die vorhandene Modularisierbarkeit und Skalierbarkeit, die durch die Möglichkeit zu flexibler Variation der eingesetzten Methoden gegeben ist: Die Gewichtung von Anteilen an Maschinensprache, Bytecode und Skriptsprache kann in höchstem Maß an die entsprechenden Bedürfnisse angepaßt werden.

Interpreter lassen sich beliebig klein, umfangreich, modular oder monolithisch aufbauen, um Bytecode oder auch vollständig offene dynamische Skripten zu verarbeiten.

Aktive Oberflächen können so, beispielsweise zur Erweiterung ereignisgestützter räumlicher Daten, nach dem jeweiligen Stand der Technik auf viele Formen mobiler Kleinstgeräte gebracht werden, ohne große konzeptionelle Änderungen zu erfordern.

Neben Weiterentwicklungen in diesem Bereich wären für eine Erhöhung der autarken Eigenschaften insbesondere Entwicklungen eines angepaßten Tcl/Tk Kerns wünschenswert, z. B. bezüglich der folgenden beiden Gruppen von Aspekten.

Die eine Seite betrifft allgemeinere Tcl/Tk Entwicklungen, beispielsweise die Modularisierung des Kerns, verändertes Speichermanagement (z. B. für Rastergraphiken), optimierte Algorithmen und längere strukturierte Attribute.

Die andere Seite solcher Entwicklungen ist streng aufgabenspezifisch und äußert sich z. B. in Entwicklungen für Module zur Georeferenzierung, in der Nutzung von Namensräumen in Datensätzen, durch eine integrierte Datenbankschnittstelle und spezielle Module mit GIS-Funktionen.

In jedem Fall existieren zahlreiche Möglichkeiten für zukünftige flexible Weiterentwicklungen und Anpassungen der so entwickelten Komponenten, die bereits jetzt für eine Fülle möglicher Anwendungen geeignet sind.

Beendend schließen möchte ich diese Arbeit mit der Hoffnung, daß offene und intuitiv interpretierbare Daten, basierend auf dem entwickelten Konzept, einen Einzug in möglichst viele Anwendungen finden und aus den hier untersuchten Möglichkeiten in Zukunft weitere Neuentwicklungen und Implementierungen resultieren.

A. Verfügbarkeit

Alle Teile des funktionsfähigen Prototyps, die in dieser Dissertation beschrieben sind, stehen mit Erweiterungen unter der Adresse

```
http://wwwmath.uni-muenster.de/cs/u/ruckema
```

oder alternativ unter:

```
http://www.unics.uni-hannover.de/cpr/gis
```

oder über:

```
http://www.RecommendedLinks.com/gis
```

in verpackter Form in verschiedenen Versionen (statisch/dynamisch) für Linux zur Verfügung. Mit Hilfe der enthaltenen Teile kann bei Bedarf auch der Anwender ohne weitergehende Programmierkenntnisse auf anderen Plattformen spezifische Versionen herstellen.

Da der Prototyp nicht Teil dieser Dissertation ist, sondern im Netz zur Verfügung gestellt wird, enthalten die Archive Kopierbedingungen, Kommentare und Hinweise zur Nutzung, viele weitere Programmteile, die in dieser Arbeit nicht behandelt werden konnten und verschiedenes weiteres Material. Hier sind auch farbige Schnappschüsse von verschiedenen Datensituationen und Teilen der Komponenten zu finden.

Neben den Kommentaren sind prinzipielle Hilfefunktionen in die zentrale Komponente integriert. Diese Dissertation bezieht sich auf eine Version des Prototyps, die, angesichts der beschriebenen Gründe, nicht für den reinen Endanwender gedacht ist. Ein grundlegendes Verständnis von Tcl/Tk vorausgesetzt, sollte die einfache Verwendung des Prototyps jedoch selbsterklärend sein.

Neben den in dieser Arbeit angesprochenen Daten und Demos stehen hier auch viele weitere exemplarische Demos offen zur Verfügung, die für das weitere Studium und den privaten Gebrauch genutzt werden können. Eine Verwendung des Prototyps oder der einzelnen Teile zu anderen Zwecken ist nicht erlaubt. Insbesondere eine Weiterverwendung bestimmter Teile des Datenmaterials ist aufgrund der rechtlichen Situation eventuell nicht möglich und bedarf im Einzelfall spezieller Klärung.

Für die Zukunft ist beabsichtigt, die vorhandenen Programme, Daten und Dokumentationen, auch im Rahmen neuer Versionen, weiterzuentwickeln und zu ergänzen, so daß verschiedene weitergehende Verwendungszwecke möglich werden.

B. Liste ausgewählter Teile des Prototyps

Der parallel zu dieser Dissertation entwickelte und frei verfügbare Prototyp besteht aus sehr vielen Einzelteilen, die unterschiedliche Aufgaben erfüllen. Dazu zählen die Teile der Kernkomponente, Konfigurationsdateien, Beispiele für Bibliotheken vorgefertigter Prozeduren, Beispieldaten und ladbare Funktionen und Demos.

Alle Bestandteile können mit ihren Funktionen und Anwendungs- und Konfigurationsmöglichkeiten an dieser Stelle nicht beschrieben werden, daher sind für den besseren Überblick und für ein weiteres Studium des Prototyps exemplarisch die wichtigsten Teile mit ihrer aktuellen Bezeichnung in Tabelle B.1 aufgeführt.

Bezeichnung	*[Typ] o. Erweiterung*	*Beschreibung*
`actmap`	tcl/tbc	Funktionen der Kernkomponente
`actmap.sfc`	[tcl/tbc]	Kernkomponente (verpackt, statisch oder dynamisch)
`actsfc`	[Skript]	Wrapperapplikation (zum individuellen Start der Kernkomponente)
`gisig.set`	[tcl]	Konfigurationsdatei (allgemeine Definitionen)
`actmap.ini`	[tcl]	Konfigurationsdatei (Initialisierungen)
`actmap.set`	[tcl]	Konfigurationsdatei (Startdefinitionen, auch explizit nachladbar)
`actmap.bnd`	[tcl]	Konfigurationsdatei (Startanbindungen, auch explizit nachladbar)
`actmap.cfg`	[tcl]	Konfigurationsdatei (individuelle Startkonfiguration)
`interp`	tcl/tbc	Funktionen für internen Interpreter
`dumpcanv`	tcl/tbc	Funktionen für „Speicherabzüge“ des Canvas
`actlib`	tcl/tbc	Beispielbibliothek mit einigen zusätzlichen Funktionen (allgemein)
`actloogp`	tcl/tbc	Beispielbibliothek mit einigen zusätzlichen Funktionen (OO)
`actltext`	tcl/tbc	Beispielbibliothek mit einigen zusätzlichen Funktionen (Texteffekte)
`textover`	tcl/tbc	Funktionen für Überlagerungen mit Text und Bedienelementen
`actmap.gas`	[tcl]	einfache Daten, GIS Active Source
`actloogp.gos`	[tcl]	einfache Daten, GIS Object Source
`npark_de.gas`	[tcl]	Demo Daten, GIS Active Source
`npark_de.gam`	[tcl]	Demo Daten, GIS Active Map
`npark_de.bnd`	[tcl]	Demo Daten, Anbindungen
`npark_de.dat`	[Text]	Demo Daten, Beispieltext für externe Anwendung
`npark_de.gto`	[tcl]	Demo Daten, Beispiel für Textoverlay
`gisig`	tcl/tbc	einfaches Beispiel für Oberfläche
`cpsplit`	tcl/tbc	einfaches Beispiel für Oberfläche
`cpsplit.bin`	[C, Binärdatei]	einfache externe Anwendung
`myWish.bin`	[C/C++, Binärdatei]	einfacher individueller Interpreter
`gbaseupd.sh`	[Skript]	Beispiel, Zusammenfassung von Aktionen für Datenbankexport
`gbase2b.pl`	[Perl]	minimaler Datenbankexport von Ereignisanbindungen
`gbase.gf`	[GROK]	einfaches Datenbankformular zur Behandlung von Ereignissen
`browedit`	tcl/tbc	einfache externe graphische Anwendung
`actsea`	tcl/tbc	einfache externe graphische Anwendung (Editor)

Tab. B.1: Ausgewählte Teile des zu dieser Dissertation entwickelten Prototyps

C. Erweiterungen der Dateinamen

Die wichtigsten eingeführten und verwendeten Erweiterungen der Dateinamen für die Demonstration des Prototyps sind in der Übersicht in Tabelle C.1 in alphabetischer Folge aufgeführt.

Erweiterung	*Quellen*	*Art*	*Beschreibung*
`.bin`	[C/C++]	Programm	Binary, ausführbare Binärdatei
`.bnd`	[Tcl/Tk]	Daten/Programm	Bind, Objektgraphik Ereignisanbindungen
`.can`	[Tcl/Tk]	Daten	„Canvas“, „Speicherabzüge“
`.cfg`	[Tcl/Tk]	Konfiguration	„Configuration“, Konfigurationsdatei
`.gam`	[Tcl/Tk]	Daten	GIS Active Map, Objektgraphik, teilweise nativ
`.gas`	[Tcl/Tk]	Daten/Programm	GIS Active Source, Objektgraphik, nativ
`.gdc`	[Tcl/Tk]	Daten/Programm	GIS Dynamic Chart, Objektgraphik, nativ
`.gos`	[Tcl/Tk]	Daten/Programm	GIS Object Source, OO-basierte Objektgraphik
`.gto`	[Tcl/Tk]	Daten/Programm	GIS Text Overlay
`.ini`	[Tcl/Tk]	Konfiguration	„Init“, Initialisierungsdatei
`.ptcl`	[Tcl/Tk]	Programm	Portable Tcl, Bytecode, ICEM
`.set`	[Tcl/Tk]	Konfiguration	„Set“, Objektgraphik Vordefinitionen
`.sfc`	[Tcl/Tk]	Programm	„Self-Contained“, ausführbares verpacktes Programm
`.sh`	[Shell]	Programm	Shell Skript
`.tbc`	[Tcl/Tk]	Programm	Tcl Bytecode, TclPro
`.tcl`	[Tcl/Tk]	Programm	Tcl/Tk

Tab. C.1: Wichtige eingeführte und verwendete Erweiterungen der Dateinamen

Literaturverzeichnis

WWW-Adressen ändern sich im Laufe der Zeit. Es wird geraten zur Hilfe Suchmaschinen zu verwenden, um ggf. aktuelle Informationen über die zitierte Literatur zu erhalten, falls die hier angegebenen WWW-Verweise und Referenzen zu einem Zeitpunkt nicht mehr aktuell sein sollten.

[AG1996] ARNOLD, K. and J. GOSLING: *The Java Programming Language*. Addison-Wesley, 1996, ISBN: 0-201-63455-4.

[Alb1993] ALBRECHT, J.: *GRASS Handbook*. GTZ, Eschborn/Vechta, 1993, URL: `ftp://ftp.informatik.uni-kiel.de/pub/packages/grass/tutorials/albrecht/` [V: 1993] [Ä: k. A.] [Z: 03.06.1997]

[Ama1997] AMANN, B.: *Integrating GIS components with mediators and corba*. Internet Draft, 1997.

[Aro1989] ARONOFF, S.: *Geographical Information Systems: A Management Perspective*. WDL Publications, Ottawa, 1989, ISBN: 0-921804-00-8.

[Aut1999a] AUTORENKOLLEKTIV: *GNU general public license*. [Internet], 1999, URL: `http://www.fsf.org/copyleft/gpl.html` [V: 1991] [Ä: k. A.] [Z: 25.09.1998].

[Aut1999b] AUTORENKOLLEKTIV: *GNU library general public license*. [Internet], 1999, URL: `http://www.gnu.org/copyleft/lgpl.html` [V: 1991] [Ä: k. A.] [Z: 24.03.1998].

[Aut1999c] AUTORENKOLLEKTIV: *Open source definition*. [Internet], 1999, URL: `http://www.opensource.org/osd.html` [V: 1999] [Ä: k. A.] [Z: 01.03.2001].

[BC1996] BRUNSDON, C. and M. CHARLTON: *Developing an exploratory spatial analysis system in Xlisp-Stat*. In: [Par1996], pages 135–145.

[BG1990] BUTTENFIELD, B. P. and J. H. GANTER: *Visualization and GIS: What should we see? What might we miss?* Proceedings of the 4th International Symposium on Spatial Data Handling, 1:307–316, 1990.

[Bla1995] BLAKE, M.: *The GIS glossary from ESRI*. Environmental Systems Research Institute, Inc. [Internet], 1995, URL: `http://www.geog.leeds.ac.uk/staff/m.blake/magis/glossary/esriglos.htm` [V: 29.04.1995] [Ä: 08.09.1995] [Z: 30.01.2001].

[BM1998] BUEHLER, K. and L. MCKEE (editors): *The OpenGIS(R) Guide; Introduction to Interoperable Geoprocessing and the OpenGIS Specification*. Open GIS Consortium Technical Committee, 3rd edition, 1998, ISBN: (ISBN pending), URL: `http://www.opengis.org/techno/guide.htm` [V: 1998] [Ä: k. A.] [Z: 03.01.1998]

[Bro1975] BROOKS, F.: *The Mythical Man-Month*. Addison-Wesley, 1975, ISBN: 0-201-00650-2.

[BUB1998] BUCKLEY, D. J., C. ULBRICHT, and J. BERRY: *The virtual forest: Advanced 3-D visualization techniques for forest management and research*. [Internet], 1998, URL: `http://www.innovativegis.com/products/vforest/contents/vfoverpaper.htm` [V: 1998] [Ä: k. A.] [Z: 11.01.1999]

[CLGM1998] CLÉMENT, G., C. LAROUCHE, D. GOUIN, and P. MORIN: *Open geospatial datastore interface*. OGDI: Scientifc Paper, 1998, URL: `http://www.las.com/ogdi/ogdilong.htm` [V: 1998] [Ä: k. A.] [Z: 11.07.2000].

[CPG1999] CARTWRIGHT, W., M. P. PETERSON, and G. GARTNER (editors): *Multimedia Cartography*. Springer Verlag, Berlin-Heidelberg, 1999, ISBN: 3-540-65818-1.

[Dic1999] DICKEN, H.: *Empress für Linux. Schnelle BLOBs*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 01:80–82, 1999

[Dri2001a] DRILLING, T.: *Glib-erig. Sinn, Zweck und Umgang mit Bibliotheken unter Linux*. Linux Intern, URL: `http://www.linuxintern.de` [V: k. A.] [Ä: k. A.] [Z: 05.05.2001], 02:93–98, 2001.

[Dri2001b] DRILLING, T.: *Quo Vadis Linux. Linux & OpenSource: Gestern, heute und morgen*. Linux Intern, URL: `http://www.linuxintern.de` [V: k. A.] [Ä: k. A.] [Z: 05.05.2001], 02:17–24, 2001.

[Dyk1996] DYKES, J. A.: *Exploring Spatial Data Representation with Dynamic Graphics*. [Internet], 1996, URL: `http://www.geog.le.ac.uk/argus/ICA/J.Dykes/` [V: 1996] [Ä: k. A.] [Z: 16.05.1997]

[Dyk1998] DYKES, J.: *The Tcl/Tk model for cartographic visualization*. [Internet], 1998, URL: `http://www.geog.le.ac.uk/argus/Research/CartoViz/model.html` [V: 1998] [Ä: k. A.] [Z: 17.06.1999].

[Dyk1999] DYKES, J.: *Scripting dynamic maps: Some examples and experiences with Tcl/Tk*. In: [CPG1999], Seite 343.

[Eck2000] ECKSTEIN, R.: *XML*. O'Reilly & Associates, 2000, ISBN: 3-89721-219-6.

[Edw1996] EDWARDS, G.: *Geocognostics – a new paradigm for spatial information?* In: *Proceedings of the AAAI Spring Symposium 1996*, 1996.

[Eng2001] ENGEL, M.: *Die neue Kernel-Version 2.4, Evolution*. Linux-Magazin, URL: `http://www.linux-magazin.de` [V: k. A.] [Ä: k. A.] [Z: 08.08.2001], 04:120–124, 2001.

[Ess2000] ESSER, T.: *teTeX-FAQ*. [Internet], 2000, URL: `ftp://sunsite.informatik.rwth-aachen.de:/pub/comp/tex/teTeX` [V: 2000] [Ä: k. A.] [Z: 16.03.2000].

[EUR1996] EUROGI: *European standardization strategies for geographic information*. [Internet], 1996, URL: `http://www2.echo.lu/oii/en/eurogi.html` [V: 1996] [Ä: k. A. – n. v.] [Z: 02.02.1999]

[FC1993] FRANK, A. U. and L. CAMPARI (editors): *Spatial Information Theory: A Theoretical Basis for GIS*, Lecture Notes in Computer Science No. 716. COSIT'93, Springer-Verlag, 1993.

[Fer1999] FERRIEUX, A.: *Concepts of Architectural Design for Tcl Applications*. [SuSE Linux Snapshot Januar 1999, CDROM 006], 1999.

[Fis1995] FISHER, P. F. (editor): *Innovations in GIS 2: Selected Papers from the Second National Conference on GIS Research UK*, volume 2. Taylor & Francis, London, 1995, ISBN: 0748402691.

[Fly1999] FLYNT, C.: *Tcl/Tk For Real Programmers*. Academic Press Professional, 1999, ISBN: 0-12-261205-1, URL: `http://www.msen.com/~clif/RealProgrammer.html` [V: 1999] [Ä: k. A.] [Z: 03.02.1999]

[Fon1997] FONTAINE, J.-L.: *stoop (Simple Tcl Only Object Oriented Programming)*. [Internet], 1997, URL: `http://www.sco.com/Technology/tcl/README/StoopDoc.html` [V: 1997] [Ä: k. A.] [Z: 23.09.1997]

[Fri1997] FRIEDL, J. F.: *Mastering Regular Expressions*. O'Reilly & Associates, Inc., Sebastopol, Calif., 1997, ISBN: 1-56592-257-3.

[Gar1993] GARDELS, K.: *What is open GIS?* GRASSCLIPINGS: The Journal of Open Geographic Information Systems, 7/1:40, 1993.

[GG1989] GOODCHILD, M. F. and S. GOPAL (editors): *The Accuracy of Spatial Databases*. Taylor & Francis, London, 1989, ISBN: 0-85066-847-6.

[GHJV1994] GAMMA, E., R. HELM, R. JOHNSON, and J. VLISSIDES: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, 1994, ISBN: 0-201-63361-2.

[Göb1999] GÖBEL, U.: *Erfahrungen im Benutzer-Support mit 3D-Visualisierung*. In: *Workshop 28.–29. September 1999, RRZN, Hannover*, 1999.

[Goo1989] GOODCHILD, M. F.: *Modelling error in objects and fields*. In: [GG1989].

[Gre1999] GREVE, G. C. F.: *Brave GNU World*. Linux-Magazin, URL: `http://www.linux-magazin.de` [V: k. A.] [Ä: k. A.] [Z: 08.08.2001], 11:28–31, 1999.

[Gro1991] GROSS, M.: *The analysis of visibility – environmental interactions between computer graphics, physics, and physiology*. Computers and Graphics, 15:407–415, 1991.

[GSH+1994] GRAF, K. CH., M. SUTER, J. HAGGER, E. MEIER, P. MEURET, and D. NUESCH: *Perspective terrain visualization - a fusion of remote sensing, GIS and computer graphics*. Computers and Graphics, 18:795–802, 1994.

[Han1999a] HANSA LUFTBILD: *Münster aus der Luft*. CDROM, Hansa Luftbild Consulting International G. m. b. H., Münster, 1999.

[Han1999b] HANTELMANN, F.: *Mechanismen und Aufbau des Linux-Kernels, Innenansichten*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 03:120–125, 1999

[Har1997] HARRISON, M. (editor): *Tcl/Tk Tools*. O'Reilly and Associates, 1997, ISBN: 1-56592-218-2.

[Hil2000] HILDEBRANDT, J.: *Erstes Europäisches Tcl/Tk-Treffen*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 08:20–20, 2000

[Hir1994] HIRTLE, S.: *Towards a cognitive GIS, proceedings of the advanced geographic data modelling workshop*. Journal of the Netherlands Geodetic Commission, 40:217–227, 1994.

[HLS2000] HEIDEMEIER, J., M. LÜTTGERT, and C. SCHOTT: *UDIS – a database application for waste water discharges and a concept for environmental database applications*. Paper presented at the First European Tcl/Tk User Meeting at TU Hamburg-Harburg, 15th and 16th June 2000, Deutschland, 2000, URL: `http://www.tu-harburg.de/skf/tcltk/papers2000/tcl-ws.pdf` [V: 2000] [Ä: k. A.] [Z: 04.08.2000]

[HM1998] HARRISON, M. und M. J. MCLENNAN: *Effektiv Tcl/Tk Programmieren*. Addison-Wesley, 1998, ISBN: 3-8273-1409-7.

[HSAS1996] HUDDLESTON, D. H., M. L. STOKES, J. D. ANDERSON, and S. SOTER: *An interactive exhibit for the smithsonian how things fly gallery*. [Internet], 1996, URL: `http://www.erc.msstate.edu/~hudd/HowWingsWork/html/ERC_hudd_template.html` [V: 1996] [Ä: k. A.] [Z: 10.07.1998]

[ICE1997] ICEM: *ICE Tcl/Tk User's Guide, ICE Tcl/Tk Version 2.0* ICEM CFD Engineering, 1997.

[Inc1997] INC., NETSCAPE: *JavaScript in Navigator 3.0*. [Internet], 1997, URL: `http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/atlas.html` [V: 1997] [Ä: k. A.] [Z: 29.11.1997]

[Ivl1997] IVLER, J. M.: *CGI Developer's Resource, Web Programming with Tcl and Perl*. Prentice-Hall (PTR/PH), 1997, ISBN: 0-13-727751-2.

[Joh1994] JOHNSON, S.: *Objecting to objects*. In: *Invited Talk*. USENIX Technical Conference, San Francisco, CA, January 1994, 1994.

[Joh1997a] JOHNSON, E. F.: *Graphical Applications with Tcl/Tk*. M&T Books, 1997, ISBN: 1-55851-569-0.

[Joh1997b] JOHNSON, R.: *Tcl style guide*. Sun Microsystems, Inc., 1997, URL: `http://sunscript.sun.com/techcorner/` [V: 1997] [Ä: k. A.] [Z: 30.09.1997] aktualisiert: `http://dev.scriptics.com/doc/styleGuide.pdf` [V: 1998] [Ä: k. A.] [Z: 20.02.2001]

[Joh1998] JOHNSON, R.: *Tcl and java integration*. Sun Microsystems Laboratories, Seite 13, 1998, URL: `http://dev.scriptics.com/software/java/tcljava.ps` [V: 1998] [Ä: k. A.] [Z: 14.02.2000]

[Kir2000] KIRSCH, C.: *Erste Beta von KDE 2.0, XML überall*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 07:110–111, 2000

[Kle1997] KLESPER, C.: *Analyse geowissenschaftlicher Modelle mittels VTK und XFORMS*. Linux-Magazin, URL: `http://www.linux-magazin.de` [V: k. A.] [Ä: k. A.] [Z: 08.08.2001], 06:37–40, 1997.

[KM1988] KENNIE, T. J. M. and R. A. MCLAREN: *Modelling for digital terrain and landscape visualisation*. Photogrammetric Record, 12:711–745, 1988.

[Kno1997] KNOPP, J.: *C++-Paradigmen und ihr Preis: zur Vereinbarkeit von Abstraktion und Effizienz*. Object Spektrum, 1/1997:68–77, 1997.

[Knu1984a] KNUTH, D. E.: *The TEXbook*. Addison Wesley, 1984, ISBN: 0-201-13447-0.

[Knu1984b] KNUTH, D. E.: *Literate programming*. The Computer Journal, 27(2):97–111, May 1984.

[Kob1991] KOBARA, S.: *Visual Design with OSF/Motif*. Addison-Wesley Publishing Company, 1991, ISBN: 0-201-56320-7.

[Koo2000] KOORMANN, F.: *FreeGIS – MapServer*. [Internet], 2000, URL: `http://www.linux-community.de/News/story?storyid=135` [V: 2000] [Ä: k. A.] [Z: 02.02.1999]

[Kop1994] KOPKA, H.: *LATEX Band 1: Einführung*. Addison-Wesley, 1994, ISBN: 3-89319-664-1.

[Kop1995] KOPKA, H.: *LATEX Band 2: Ergänzungen – mit einer Einführung in METAFONT*. Addison-Wesley, 1995, ISBN: 3-89319-665-X.

[Kop1997] KOPKA, H.: *LATEX Band 3: Erweiterungen*. Addison-Wesley, 1997, ISBN: 3-89319-666-8.

[KR1988] KERNIGHAN, B. and D. RITCHIE: *The C Programming Language*. Prentice-Hall, second edition, 1988, ISBN: 0-13-110362-8.

[Lai1995] LAIRD, C.: *Personal notes on commercial aspects of Tcl*. [Internet], 1995, URL: `http://starbase.neosoft.com/~claird/comp.lang.tcl/commercial-tcl.html` [V: 1995] [Ä: k. A.] [Z: 11.07.2000].

[Lam1985] LAMPORT, L.: *LATEX – A Document Preparation System – User's Guide and Reference Manual*. Addison-Wesley, 1985, ISBN: 0-201-15790-X.

[LHCP1992] LOH, D. K., D. R. HOLTFRERICH, Y. K. CHOO, and J. M. POWER: *Techniques for incorporating visualization in environmental assessment: an object-oriented perspective*. Landscape and Urban Planning, 21:305–307, 1992.

[Lin2000] LINGNAU, A.: *TkDVI: A Tcl/Tk-based TEX DVI previewer*. Paper presented at the First European Tcl/Tk User Meeting at TU Hamburg-Harburg, 15^{th} and 16^{th} June 2000, Deutschland, 2000, URL: `http://www.tu-harburg.de/skf/tcltk/papers2000/etclpaper.pdf` [V: 2000] [Ä: k. A.] [Z: 04.08.2000].

[LS1999] LIE, H. W. and J. SAARELA: *Multipurpose web publishing, using HTML, XML, and CSS*. Communications of the ACM, 42(10):95–101, 1999.

[McG1997] MCGAUGHEY, R. J.: *Techniques for visualizing the appearance of timber harvest operations.* In: *Forest operation for sustainable forests and health economies. 20^{th} annual meeting of the Council On Forest Engineering, 1997 July 28–31, Rapid City, SD*, Seite 10, 1997.

[McL1993] MCLENNAN, M. J.: *[incr Tcl]: Object-Oriented Programming in Tcl*. Proceedings of the Tcl/Tk Workshop, University of California at Berkeley, June 10–11 1993.

[McL1994] MCLENNAN, M. J.: *[incr Tk]: Building extensible widgets with [incr Tcl]*. Proceedings of the Tcl/Tk 1994 Workshop, New Orleans, LA, June 23–25 1994.

[McL1995] MCLENNAN, M. J.: *The new [incr Tcl]: Objects, mega-widgets, namespaces and more*. Proceedings of the Tcl/Tk 1995 Workshop, Toronto, Ontario, July 6–8 1995.

[Mon1993] MONTELLO, D. R.: *Scale and multiple psychologies of space*. In: FRANK, A. and CAMPARI I. [FC1993], pages 312–321.

[Moz2000] MOZILLA: *JavaScript*. [Internet], 2000, URL: `http://www.mozilla.org/js` [V: 2000] [Ä: k. A.] [Z: 25.01.2001]

[MP1999] MERKLE, B. und F. PILHOFER: *CORBA Mapping für Skriptsprachen*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 04:154–165, 1999

[NB1998] NETELER, M. und B. BYARS: *Geographic Resources Analysis Support System, Informationen zu GRASS 4.2.1*. [Internet], 1998, URL: `http://www.geog.uni-hannover.de/grass/faq/grass_info_deu.html` [V: 12.1998] [Ä: 06.1999] [Z: 30.01.2001]

[Net2000] NETELER, M.: *GRASS-Handbuch*. [Internet], 2000, URL: `http://www.geog.uni-hannover.de/grass/gdp/handbuch` [V: k. A.] [Ä: k. A.] [Z: 30.01.2001]

[NHIN1986] NAKAMAE, E., K. HARADA, T. ISHIZAKI, and T. NISHITA: *A montage method: the overlaying of the computer generated images onto a background photograph*. ACM Computer Graphics, 20:207–214, 1986.

[Nij2000] NIJTMANS, J.: *Standalone executables with wrap*. Paper presented at the First European Tcl/Tk User Meeting at TU Hamburg-Harburg, 15th and 16th June 2000, Deutschland, 2000, URL: `http://www.tu-harburg.de/skf/tcltk/papers2000/nijtmans.pdf` [V: 2000] [Ä: k. A.] [Z: 04.08.2000]

[NR1998] NEWHAM, C. and B. ROSENBLATT: *Learning the bash*. O'Reilly & Associates, Inc., Sebastopol, Calif., 2nd edition, 1998, ISBN: 1-56592-347-2.

[OD1995] ORLAND, B. and T. C. DANIEL: *Impact of proposed water withdrawls on the perceived scenic beauty of desert springs and wetlands: Image generation*. Imaging Systems Laboratory, Dept. of Landscape Architecture, 1995.

[OG1988] O'HARA, R. and D. GOMBERG: *Modern Programming Using REXX*. Prentice-Hall, 1988, ISBN: 0-13-597329-5.

[OII1998] OII: *Geographic data exchange standards*. [Internet], 1998, URL: `http://www2.echo.lu/oii/en/gis.html` [V: 1998] [Ä: k. A.] [Z: 02.02.1999]

[Ous1994] OUSTERHOUT, J. K.: *Tcl and the Tk Toolkit*. Addison-Wesley Publishing Company, Reading, Mass., 1994, ISBN: 0-201-63337-X.

[Ous1997a] OUSTERHOUT, J.: *Additional information for scripting white paper*. [Internet], 1997, URL: `http://www.sunlabs.com/people/john.ousterhout/scriptextra.html` [V: 1997] [Ä: k. A.] [Z: 23.09.1997].

[Ous1997b] OUSTERHOUT, J. K.: *Scripting: Higher level programming for the 21st century*. Internet Draft, 1997, URL: `http://www.sunlabs.com/~ouster/scripting.html` [V: 1997] [Ä: k. A.] [Z: 23.09.1997]

[Par1996] PARKER, D. (editor): *Innovations in GIS 3*, volume 3. Taylor & Francis, London, 1996, ISBN: 0748404589.

[Pat1997] PATZSCHKE, T. I.: *Tcl/Tk-Workshop in Boston mit Ausblick auf Tcl 8.0; Universell und frei*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 10:14–14, 1997

[Pat1998] PATZSCHKE, T. I.: *6. Tcl/Tk-Konferenz in San Diego*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 12:20–20, 1998

[Pat2000] PATZSCHKE, T. I.: *Tcl/Tk-Konferenz: Business und Web*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 04:48–48, 2000

[PR1996] PELOUX, J.-P. and P. RIGAUX: *A loosely coupled interface to an object-oriented geographic database*. Internet Draft, 1996.

[Rai1999] RAISON, A. V.: *Linux Unterstützung*. IX Magazin für professionelle Informationstechnik, CeBIT Special, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 04:24–25, 1999

[Ray1999] RAYMOND, E. S.: *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., 1999, ISBN: 1-56592-724-9.

[RBP+1991] RUMBAUGH, J., M. BLAHA, W. PREMERLANI, F. EDDY, and W. LORENSEN: *Object-Oriented Modeling and Design*. Prentice-Hall, 1991, ISBN: 0-13-629841-9.

[RC1997] ROUSE, F. and W. CHRISTOPHER: *A Typing System for an Optimizing Multiple-Backend Tcl Compiler*. ICEM CFD Engineering, 1997.

[Rei2000] REITER, B.: *GEO-Informationssysteme unter GNU/Linux*. [Internet], 2000, URL: `http://www.linux-community.de/News/story?storyid=48` [V: 2000] [Ä: k. A.] [Z: 02.02.1999]

[RM2000] ROBLES, V. and G. C. MELIA: *X11 desktop environments (KDE and GNOME) – part II*. CERN Computer Newsletter, January–April 2000, 1:14–17, 2000, URL: `http://ref.cern.ch/CERN/CNL` [V: 2000] [Ä: k. A.] [Z: 14.07.2000]

[Röh1994] RÖHRIG, O.: *Zwischen Realität und Simulation*. IX Multiuser Multitasking Magazin, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 08:42–50, 1994

[Ros1993] ROSENBLATT, B.: *Learning the Korn Shell*. O'Reilly & Associates, Inc., Sebastopol, Calif., 1st edition, 1993, ISBN: 1-56592-054-6.

[RT1999] RAINES, P. and J. TRANTER: *Tcl/Tk in a Nutshell*. O'Reilly & Associates, 1999, ISBN: 1-56592-433-9.

[Rüc2000] RÜCKEMANN, C.-P.: *Portable GIS Komponenten und OpenSource Werkzeuge: Entwicklung und Modularisierung*. Geoinformatik Forum, 08.02.2000, Westfälische Wilhelms-Universität Münster, 2000, URL: `http://wwwmath.uni-muenster.de/cs/u/ruckema` [V: 2000] [Ä: k. A.] [Z: 01.10.2001]

[Rüc2001] RÜCKEMANN, C.-P.: *Active Map Software*. [Internet], 2001, URL: `http://wwwmath.uni-muenster.de/cs/u/ruckema` [V: 2000] [Ä: k. A.] [Z: 01.10.2001]

[Sch1999] SCHLÜTER, K.: *Grundlagen der Extended Markup Language*. Gateway, 2:56–60, 1999.

[Sch2000a] SCHÖNWÄLDER, J.: *Married with Tcl*. Paper presented at the First European Tcl/Tk User Meeting at TU Hamburg-Harburg, 15th and 16th June 2000, Deutschland, 2000, URL: `http://www.tu-harburg.de/skf/tcltk/papers2000/maried.pdf` [V: 2000] [Ä: k. A.] [Z: 04.08.2000]

[Sch2000b] SCHWAN, B.: *Es gibt eine Wahl. Interview mit der Mozilla-Chefin Mitchell Baker*. c't Magazin für Computer und Technik, o. Jg., 25:24–25, 2000.

[Sch2001] SCHÜRMANN, T.: *Die Welt im Computer. Das Geoinformationssystem Grass*. Linux-Magazin, URL: `http://www.linux-magazin.de` [V: k. A.] [Ä: k. A.] [Z: 08.08.2001], 05:104–109, 2001.

[Scr1999] SCRIPTICS: *TclPro User's Guide, Version 1.3*. Scriptics Corporation, 2593 Coast Avenue, Mountain View, CA 94043 URL: `http://www.scriptics.com` [V: k. A.] [Ä: k. A.] [Z: 28.02.2001], 1999

[SGX1994] S. R., GORDON, C. W. H. GOODWIN, and D. XIONG: *Final report on status of spatial/map databases*. [Internet], 1994, URL: `http://itsdeployment.ed.ornl.gov/spatial/document/html/draft21.htm` [V: 1994] [Ä: k. A.] [Z: 03.02.1999]

[She1995] SHEPARD, I. D. H.: *Putting time on the map: Dynamic displays in data visualization and GIS*. In: [Fis1995].

[SM1998] SCHROEDER, W. and K. MARTIN: *The VTK User's Guide*. Kitware, 1998.

[SM2000] STALLMAN, R. M. and R. MCGRATH: *GNU Make – A Program for Directing Recompilation*. URL: `http://www.gnu.org/manual/make/index.html` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 2000

[Sma1996] SMART, J.: *User Manual for wxWindows 1.66: a portable C++ GUI toolkit*. AIAI, Univ. of Edinburgh, 1996.

[SML1998] SCHROEDER, W., K. MARTIN, and B. LORENSEN: *The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics*. Prentice-Hall, second edition, 1998, ISBN: 0-13-199837-4.

[Str1987] STROUSTRUP, B.: *The C++ Programming Language*. Addison-Wesley, 1987, ISBN: 0-201-12078-X.

[S.u1997] S.U.S.E.: *S.u.S.E. Linux 5.0*. S.u.S.E G. m. b. H., Gesellschaft für Software und Systementwicklung m. b. H., Fürth, 1997, ISBN: 3-930419-44-0, URL: `http://www.suse.de` [V: k. A.] [Ä: k. A.] [Z: 28.02.2000]

[S.u2001] S.U.S.E.: *S.u.S.E. Linux 7.2 Professional (mit 5 Handbüchern)*. S.u.S.E., Gesellschaft für Software und Systementwicklung m. b. H., Nürnberg, 2001, ISBN: 3-934678-84-X, URL: `http://www.suse.de` [V: k. A.] [Ä: k. A.] [Z: 01.07.2000]

[SWJ+1998] STYNES, K., J. WOOD, DYKES J., P. FISHER und D. UNWIN: *Publishing Cartography on the Web*. [Internet], 1998, URL: `http://www.geog.le.ac.uk/argus/ICA/K.Stynes` [V: 1998] [Ä: k. A.] [Z: 13.06.1999]

[SYT1998] SMITH, B., S. YEN, and S. TU: *Tcl-DP: a distributed programming extension to Tcl*. WWW, CODA Manual pages, 1998, URL: `http://simon.cs.cornell.edu/Info/Projects/zeno/Projects/Tcl-DP` [V: 1998] [Ä: k. A.] [Z: 29.08.1998].

[Tcl2000a] TCL DEVELOPER XCHANGE: *Tcl comparison chart*. [Internet], 2000, URL: `http://dev.scriptics.com:80/advocacy` [V: 2000] [Ä: k. A.] [Z: 30.01.2001].

[Tcl2000b] TCL DEVELOPER XCHANGE: *Tcl FAQs*. [Internet], 2000, URL: `http://dev.scriptics.com/resource/doc/faq` [V: k. A.] [Ä: k. A.] [Z: 30.01.2001]

[Tve1993] TVERSKY, B.: *Cognitive maps, cognitive collages, and spatial mental models*. Lecture Notes in Computer Science, 716:14–24, 1993.

[TZ1998] TRÄGER, S. und C. ZERBST: *Tcl und das Web, Feder im Netz*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 08:147–148, 1998

[Vir1996] VIRDEN, L. W.: *comp.lang.tcl FAQ*. [Internet], 1996, URL: `http://dev.scriptics.com/faq/part1.html` [V: k. A.] [Ä: k. A.] [Z: 30.01.2001]

[WCS1996] WALL, L., T. CHRISTIANSEN, and R. SCHWARTZ: *Programming Perl*. O'Reilly and Associates, second edition, 1996, ISBN: 1-56592-149-6.

[Wel1997] WELCH, B.: *Practical Programming in Tcl and Tk*. Prentice-Hall (PTR), Upper Saddle River, New Jersey, 1997, ISBN: 0-13-616830-2.

[Wes1991] WESTERVELT, J.: *Introduction to GRASS 4*. [PostScript from `moon.cecer.army.mil`], 1991.

[Wet1995] WETHERALL, D.: *OTcl – MIT object Tcl*. Internet FAQ, 1995, URL: `ftp://ftp.tns.lcs.mit.edu/pub/otcl/README.html` [V: 1995] [Ä: k. A.] [Z: 23.09.1997]

[WND1997] WOO, M., J. NEIDER, and T. DAVIS: *OpenGL Programming Guide*. Addison Wesley Developers Press, 2nd edition, 1997, ISBN: 0-201-46138-2.

[Wol1999] WOLF, U.: *Softwarelizenzen im Linuxumfeld*. Linux-Magazin, URL: `http://www.linux-magazin.de` [V: k. A.] [Ä: k. A.] [Z: 08.08.2001], 11:52–56, 1999.

[Zeh1993] ZEHNER, K.: *Das Geographische Informationssystem IDRISI auf dem PC*. Zeitschrift für Angewandte Geographie, 17(2):30-33, 1993.

[Zer1999] ZERBST, C.: *Version 8.1 von Tcl/Tk, Nach der Mauser*. IX Magazin für professionelle Informationstechnik, URL: `http://www.heise.de/ix` [V: k. A.] [Ä: k. A.] [Z: 04.05.2001], 06:64–66, 1999

[Zer2000] ZERBST, C.: *MARINET*. Paper presented at the First European Tcl/Tk User Meeting at TU Hamburg-Harburg, 15th and 16th June 2000, Deutschland, 2000, URL: `http://www.tu-harburg.de/skf/tcltk/papers2000/marinet-pp4.pdf` [V: 2000] [Ä: k. A.] [Z: 04.08.2000].

[Zer2001a] ZERBST, C.: *Objektorientierung mit Tcl, Klasse Federn*. Linux-Magazin, URL: `http://www.linux-magazin.de` [V: k. A.] [Ä: k. A.] [Z: 08.08.2001], 07:131–134, 2001.

[Zer2001b] ZERBST, C.: *Zweites Europäisches Tcl/Tk User Meeting, 7. und 8. Juni 2001, Hamburg, Deutschland*. Linux-Magazin, URL: `http://www.linux-magazin.de` [V: k. A.] [Ä: k. A.] [Z: 08.08.2001], 08:16, 2001.

[Zim1997] ZIMMER, J. A.: *Tcl/Tk for Programmers With Solved Exercises That Work With Unix and Windows*. IEEE Computer Society, 1997, ISBN: 0-8186-8515-8.

[ZK1995] ZEWE, R. and H.-J. KOGLIN: *A method for the visual assessment of overhead lines*. Computers and Graphics, 19:97–108, 1995.

[Zua1996] ZUAZAGA, H. O.: *The penguin model of secure distributed internet scripting*. WWW, 1996, URL: `http://momo.uthscsa.edu/~humberto/project.html` [V: 1996] [Ä: k. A. – n. v.] [Z: 22.09.1998]

Glossar

Verwendete Abkürzungen:

`actmap`	Active Map	68
AI	Artificial Intelligence	9
ANSI	American National Standards Institute	12, 34
API	Application Programming Interface	12, 34, 72, 101, 108, 110
ASCII	American Standard Code for Information Interchange	18
`awk`	Aho, Weinberger, Kernighan	24
B2B	Business to Business	34
BLOBs	Binary Large Objects	64
BLT	Bacon, Lettuce, and Tomato, Tk Elemente, sogenannte „Widgets“ für Tcl	34
BND	Bind Daten	82
BS	Betriebssystem	39
BSD	Berkeley System Distribution	41
CAD	Computer Aided Design	8
CAD-CAM	Computer Aided Drafting – Computer Aided Mapping	10
CAN	Canvas Daten	75
CDROM	Compact Disk Read Only Memory	9
CDV	Cartographic Data Visualizer	52
CERN	Centre Européen pour la Recherche Nucléaire, Schweiz	26
CGI	Common Gateway Interface	48, 49
CORBA	Common Object Request Broker Architecture	23, 35
CPU	Central Processing Unit	9
CSS	Cascading Style Sheets	26
CVS	Concurrent Versions System	24
DBI	Database Interface	68
DBMS	Database Management System	9
DCOP	Desktop Communication Protocol	25
DDE	Dynamic Data Exchange	76
DEM	Digital Elevation Model	58, 67
DFA	Deterministic Finite Automat	35
DGK	Deutsche Grundkarte	85
DRI	Direct Rendering Infrastructure	41
DTM	Digital Terrain Model	58
DVD	Digital Versatile Disk	9
EIDE	Enhanced Integrated Drive Electronics	42
EPS	Encapsulated PostScript	53
ET	Embedded Tcl	34
FAQ	Frequently Asked Questions	44
FAT	File Allocation Table	41
FRS	Functional Requirements Study	21
FTP	File Transfer Protocol	42
FVWM	Feebles Virtual Window Manager	23
g77	GNU Fortran Compiler	24

GAM	GIS Active Map	78
GAS	GIS Active Source	75, 78
gcc	GNU C Compiler	24
GDC	GIS Dynamic Chart	99
GDF	General Data Format	10
GIMP	GNU Image Manipulation Program	24, 58
GIS	Geographisches Informationssystem	1, 34
GISIG	GIS, reflexives Akronym	46
GL	Graphics Library	27
GLTP	gltp, geographic locator transfer protocol	12
GLTPD	gltpd, geographic locator transfer protocol daemon	12
GLU	Graphics Library Utility Library	27
GLUT	Graphics Library Utility Toolkit	27
GMT	Generic Mapping Tools	52
GNOME	GNU Network Object Model Environment	23, 25
GNU	GNU is Not Unix, gegr. 1984	23, 24
GOS	GIS Object Source	80
GPL	GNU General Public License	13, 23, 26, 40, 51
GRASS	Geographical Resources Analysis Support System	12, 24, 26, 34
GROK	Graphical Resource Organizer Kit	23
GTK+	GIMP Tool Kit	33
GTO	GIS Text Overlay	76
GUI	Graphical User Interface	19, 25, 45
HIRD	HURD of Interface Representing Depth	40
HTML	Hypertext Markup Language	26
HTTP	HyperText Transfer Protocol	48, 49
HURD	HIRD of Unix-Replacing DAEMONs	40
ID	Identity, Identity number	60
IDE	Integrated Drive Electronics	41
IPC	Inter-Process Communication	16, 30, 33, 76, 95
IPv4	Internet Protocol version 4	41
IPv6	Internet Protocol version 6	41
IPX	Internetwork Packet Exchange (Novell)	42
ISO	International Standardisation Organisation	26, 27
Jacl	Java Tcl, Java mit Tcl Skriptsprache	34
KDE	K-Desktop Environment	23, 25
KI	Künstliche Intelligenz	9
K&R	Kernighan & Ritchie	71
LDP	Linux Documentation Project	44
LGA	Local Government Areas	18
LGPL	Library/Lesser GPL	40
LIS	Land Information System	7
MIME	Multipurpose Internet Mail Extension	29, 56
MIT	Massachusetts Institute of Technology	25
NFA	Non-Deterministic Finite Automat	35
NFS	Network File System (Sun)	42
ODBC	Open Database Connectivity	68
OGDI	Open Geospatial Datastore Interface	2, 12, 34
OMT	Object Modeling Technique	34
OO	objektorientiert	77
OpenGL	Open Graphics Library	27, 34

ORB	Object Request Broker ... 23
OSD	Open Source Definition ... 22
OTcl	Object Tcl ... 34
PCI	Peripheral Components Interface ... 42
PCMIA	Personal Computer Memory Card Interface Association ... 42
PDAs	Personal Digital Assistants ... 111
Perl	Practical Extraction and Report Language ... 24, 27, 35, 49, 50, 66, 76
PEX	Phigs/phigs+ Extension to X ... 27
PGP	Pretty Good Pricacy ... 69
PHIGS	Programmer's Hierarchical Interactive Graphics System ... 27
PHP	PHP Hypertext Preprocessor ... 24
PLIP	Parallel Line Internet Protocol ... 42
POSIX	Portable Operating System Interface; Portable Operating System Interface for Unix ... 41
POV-Ray	Persistence Of Vision Raytracer ... 58
PPM	Portable Pixmap ... 86
PPP	Point-to-Point Protocol ... 42
RAD	Rapid Application Development ... 31
RAID	Redundant Array of Inexpensive Disks ... 42
RAM	Random Access Memory ... 24
RCS	Revision Control System ... 24
RDBMS	Relational Database Management System ... 68
RegEx	Regular Expression ... 35
RFC	Request For Comments ... 25, 42, 48
RFP	Request for Proposals ... 22
RPC	Remote Procedure Call ... 34
RS	Remote Sensing ... 10
RT	Real-Time ... 110
SCSI	Small Computer System Interface ... 41, 42
`sed`	Stream Editor ... 24
SET	Settings Daten ... 82
SGML	Structured General Markup Language ... 26
SLIP	Serial Line Interface Protocol ... 42
SMB	Server Message Blocks ... 42
SMP	Symmetrical Multi-Processing ... 41
SQL	Structured Query Language ... 37
SSH	Secure Shell ... 42
SSL	Secure Sockets Layer ... 69
STOOOP	Simple Tcl Only Object Oriented Programming ... 34, 77, 80
Tcl	Tool Command Language ... 24, 26, 27, 33, 34, 36–38, 48–50, 56, 60, 66, 75–77
Tcl-DP	Tcl Distributed Programming ... 34
TclX	Extended Tcl ... 34
TCP/IP	Transfer Control Protocol/Internet Protocol ... 42
Tix	Tk Interface Extension ... 34
Tk	Toolkit ... 24, 26, 27, 33, 34, 36, 37, 48, 60, 66, 75–77
TSIPP	Tcl SIPP, Tcl Simple Polygon Processor ... 34
UMTS	Universal Mobile Telecommunications System ... 111
UNESCO	United Nations Educational, Scientific, & Cultural Organization ... 39
URL	Uniform Resource Locator ... 48
USB	Universal Serial Bus ... 42
VDU	Visual Display Unit ... 9
VFAT	Virtual File Allocation Table ... 41
VHLL	Very High Level Language ... 33
`vim`	Vi Improved ... 24

VTK	Visualization Toolkit	27, 34
W3C	World Wide Web Consortium	26
WAP	Wireless Application Protocol	111
`wish`	Windowing Shell	27, 37
WWW	World Wide Web	57, 72
WYSIWYG	What You See Is What You Get	69
WYSIWYM	What You See Is What You Mean	69
XGL	X Graphics Language	27
XML	Extensible Markup Language	26, 35
XSSI	Extended Server Side Includes	26

Markennamen, Handelsnamen, Warenzeichen, Logos usw.:

Die verschiedenen aufgeführten und nicht aufgeführten Markennamen, Handelsnamen, Logos usw. sind Copyright der betreffenden Eigentümer.

Linux® ist ein eingetragenes Warenzeichen von Linus Benedict Torvalds.
XFree86™ ist ein Warenzeichen des The XFree86 Project, Inc.
X Window System™ und The Open Group™ sind Warenzeichen der The Open Group.
TclPro® und Scriptics® sind eingetragene Warenzeichen der Ajuba Solutions, früher Scriptics Corporation.
S.u.S.E.® ist ein eingetragenes Warenzeichen der S.u.S.E. G. m. b. H.
Red Hat ist ein Warenzeichen von Red Hat Software, Inc.
Netscape Navigator™ und Netscape Communicator™ sind Warenzeichen der Netscape Communications Corporation.
TEX™ ist ein Warenzeichen der American Mathematical Society.
PostScript™ ist ein Warenzeichen von Adobe Systems, Inc.
Qt® ist ein eingetragenes Warenzeichen von Trolltech AS.
AT&T® ist ein eingetragenes Warenzeichen von American Telephone & Telegraph Company.
Der Begriff Unix wird in dieser Schreibweise als generelle Bezeichnung für die Unix-ähnlichen Betriebssysteme verschiedener Hersteller benutzt, nicht als die Bezeichnung für das Trademark von X/Open.
UNIX® ist ein eingetragenes Warenzeichen in verschiedenen Ländern und exklusiv lizensiert von X/Open Company, Ltd.
OpenGL® ist ein eingetragenes Warenzeichen von Silicon Graphics, Inc.
Sun, Sun Microsystems, das Sun Logo, JavaOS, Java®, HotJava, JDBC, die Java Cup und das Steam Logo, Solaris sind Warenzeichen oder eingetragene Warenzeichen von Sun Microsystems, Inc., in verschiedenen Ländern.
Adobe® ist ein eingetragenes Warenzeichen von Adobe Systems, Inc.
Microsoft® ist ein eingetragenes Warenzeichen, und Windows™ ist ein Warenzeichen der Microsoft Corporation.
IBM® ist ein eingetragenes Warenzeichen der International Business Machines.

Index

Wichtigere Einträge sind **fett** gesetzt und Definitionen *kursiv* hervorgehoben.

A

B

C

D

E

F

G

L

M

N

O

S

T

Danksagung

Meinem Doktorvater, Herrn Prof. Dr. Thomas Kreuser, gilt mein herzlichster Dank für die Anregung zu dieser Arbeit, sein Interesse und die stete Diskussionsbereitschaft, die zu vielen Lösungen maßgeblich beigetragen haben sowie die Betreuung, die er mir über die gesamten Jahre zuteil werden ließ. Vor allem für das Vertrauen, das er mir bei der Planung, Vorbereitung und Durchführung von Anfang an entgegenbrachte, möchte ich mich bei ihm ganz besonders bedanken.

Herrn Prof. Dr. Guido Wirtz danke ich herzlichst für die Übernahme des zweiten Gutachtens, für sein Interesse am Fortgang dieser Arbeit und die unkompliziert entgegengebrachte, vielfältige Unterstützung, die insbesondere für den Abschluß der Arbeit von besonderer Bedeutung war.

Ich danke Herrn Dr. Panagiotis Aslanidis von der Firma G. A. S. (Köln) für Anregungen und logistische Unterstützung.

Mein Dank gilt allen Mitgliedern des Instituts für Geologie und Paläontologie der Universität Münster, die durch Diskussionen zur Lösung von Problemen beigetragen haben und dem Direktor, Herrn Prof. Dr. Wilhelm G. Coldewey, für die freundlichen und klärenden Gespräche.

Ich danke dem Direktor des Instituts für Geoinformatik der Universität Münster, Herrn Prof. Dr. Ulrich Streit, für die hilfreichen Gespräche im Umfeld dieser Arbeit.

Dem Umweltamt der Stadt Münster, stellvertretend Herrn Uwe Kock, gilt mein Dank für die kostenfreie Bereitstellung von realem Datenmaterial seit der Erprobungsphase bestimmter Verfahren und Techniken.

Der Firma Vectaport Inc. (Redwood City, Kalifornien), stellvertretend Herrn Scott Johnston, danke ich für die kostenfreie Bereitstellung ihrer räumlichen Softwarekomponenten und frei verfügbaren Daten zu Vergleichszwecken und die Diskussionen über räumliche Ereignisse.

Der Firma Hansa Luftbild Consulting International GmbH, Münster, gebührt mein Dank für die kostenfreie Überlassung der geeigneten Luftbilddaten.

Dem Landesvermessungsamt Nordrhein-Westfalen, stellvertretend Frau Karin Obermeit, gilt mein besonderer Dank für die kostenfreie Überlassung von benötigtem Datenmaterial.

Für die großzügige Bereitstellung der benötigten Kapazitäten und der URL zur Veröffentlichung des Prototyps und der aufbereiteten Daten bedanke ich mich nachdrücklich beim Institut für Informatik der Universität Münster.

Für das Korrekturlesen dieser Arbeit bedanke ich mich bei allen Beteiligten und bei meinen Arbeitskollegen, Herrn Dr.-Ing. Wolfgang Sander-Beuermann und Herrn Dr. Helmut Storr, vom Regionalen Rechenzentrum für Niedersachsen (RRZN), Universität Hannover.

Frau Christel Bartels im Dekanat der mathematisch-naturwissenschaftlichen Fakultät danke ich für die freundliche Unterstützung bei der notwendigen Abwicklung aller Formalitäten.

Meiner lieben Mutter gilt mein innigster Dank für ihre Unterstützung und ihren unermüdlichen Einsatz und Zuspruch auch in schweren Zeiten.

Meinem lieben Vater, der die Fertigstellung dieser Arbeit leider nicht mehr erleben konnte, danke ich in stillem Gedenken für all die aufopferungsvollen Jahre der Unterstützung.