UNIVERSITÄT DORTMUND
Projektgruppe 369: Rechnergestützte Planung von Nahverkehrsnetzen

Seminar zur Projektgruppe 369:
Rechnergestützte Planung von Nahverkehrsnetzen
Prof. Dr.-Ing. Heinz Beilner, Dr. Peter Kemper, Markus Völker
WS 2000/2001

"Geo-Informationssysteme in der Praxis: ArcInfo, ArcView und GRASS"

Verfasser:
Tim Schürmann
Dortmund, Oktober 2000
E-Mail: tischuer@yahoo.de
Homepage: http://www.timshome.purespace.de

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Diese Seminararbeit soll einen Überblick über die drei wichtigsten Geo-Informationssysteme (GIS) geben.
In Kapitel 2 wird zunächst das marktführende Informationssystem der Firma ESRI, “ArcInfo” vorgestellt. Kapitel 3 geht auf das, ursprünglich nur als Visualisierungshilfe zu ArcInfo gedachte, "ArcView GIS" ein.
Kapitel 4 stellt im Gegensatz zu den beiden vorangegangenen, kommerziellen GIS, die freie und somit auch kostenlose “Open-Source” Lösung "Geographic Resources Analysis Support System", kurz “GRASS” vor.
Zum Abschluss wird ein Fazit gezogen und eine kleine Zusammenfassung in Form eines Vergleichs durchgeführt.

Im Folgenden soll das Augenmerk weniger auf die detaillierte Beschreibung der Bedienung der hier vorgestellten Informationssysteme gerichtet werden, sondern es sollen vielmehr die Vor- und Nachteile bezüglich eines Einsatzes im Hinblick auf das Projektgruppenthema herausgearbeitet werden.

2. ArcInfo

2.1. Was ist ArcInfo

ArcInfo ist ein Geo-Informationssystem (GIS), das vom weltweiten Marktführer auf diesem Gebiet, Environmental Systems Research Institute, Inc, kurz ESRI angeboten wird. Es ist mit seinen über 100.000 verkauften Lizenzen das meist benutze GIS der Welt.
Der Hersteller beschreibt die Leistungsmerkmale von ArcInfo selbst [ESRI00]: "ArcInfo bietet die komplette Lösung für die Erfassung, Verwaltung, die Darstellung, die Analyse und die hochqualitative kartographische Ausgabe von geographischen Daten, verbunden mit zahlreichen weiteren, multimedialen Informationen. [...] ArcInfo bietet dem Nutzer die kompletten Werkzeuge zum Zugriff, zur Visualisierung und zur Abfrage geographischer und tabellarischer Daten." ArcInfo ist ein komplettes "GIS-Toolkit" mit "hunderten von integrierten Funktionen zur gemeinsamen Nutzung und Verarbeitung geographischer Daten und dazu optional voll integrierte Extensions für Spezialaufgaben. [...] ArcInfo besitzt leistungsstarke Tools zur Anpassung und Anwendungsentwicklung, so dass Sie mit ArcInfo genau das System erstellen und entwerfen, was Sie brauchen".
ArcInfo ist für die Betriebsysteme Unix und Windows NT erhältlich. Zusätzlich existiert eine "PC/ArcInfo"-Version, die im wesentlichen ein abgespecktes ArcInfo für Personal Computer auf MS-DOS Basis darstellt und "als GIS-Server für große und kleine Unternehmen" [ESRI00] dienen soll.
ArcInfo kann mit Hilfe der integrierten Makrosprache AML als Grundlage für weitere Zusatzmodule bzw. Produkte dienen. So bauen viele derzeit auf dem Markt erhältliche Produkte (die meistens ebenfalls mit dem Wort "Arc" beginnen) auf ArcInfo als Datenbasis auf.
Als zusätzliche Schnittstellen wurden erstmals in der aktuellen Version 8 neben AML und der "Inter Application Communication" (vgl. Kapitel 3.5), auch das von Microsoft lizenzierte Visual-Basic, sowie eine Erweiterungsmöglichkeit über COM-Komponenten integriert.

2.2. Konzepte und Arbeitsweisen

ArcInfo unterscheidet wie alle GIS, zwei verschiedene Typen von Daten: raumbezogene Daten und Sachdaten. Letztere werden unter ArcInfo als beschreibende Informationen ("descriptive Information") bezeichnet. Die raumbezogenen Daten können entweder durch Vektor- oder Rastergrafiken dargestellt werden. Bei den Rastergrafiken erfolgt eine Unterteilung in Bilder ("Images"), die nur zur Illustration oder als ein Sachdatum dienen können und in Rastergrafiken ("raster data"), bei denen es sich um raumbezogene Daten handelt. Im letzten Fall lassen sich jeder einzelnen Zelle Attributdaten zuordnen, wobei der "Basistyp" (das Raster) unter ArcInfo als "Grid" bezeichnet wird.
Bei den Vektordaten unterscheidet ArcInfo zwischen Punkten (geographische Objekte, die zu klein für die Karte sind, z.B. Denkmäler), Linien (z.B. Straßen), Arcs ("Arc", engl. Bogen) und Polygone (geschlossenes Vieleck, z.B. Regionen).
Punkte können zum einen geographische Objekte (wie z.B. Denkmäler), als auch Markierungspunkte ("label points", Punkte mit einer Beschriftung) repräsentieren. Arcs werden formal definiert als "continous string of x,y-coordinate pairs" (diese Punkte werden "Vertices" genannt), der an einem Ort beginnt ("From-Node"), an einem zweiten endet ("To-Node"), eine Länge, jedoch keine Fläche besitzt. Damit besitzt ein Arc eine Richtung und ist somit mehr als eine Linie (vgl.

[Zipf96]).

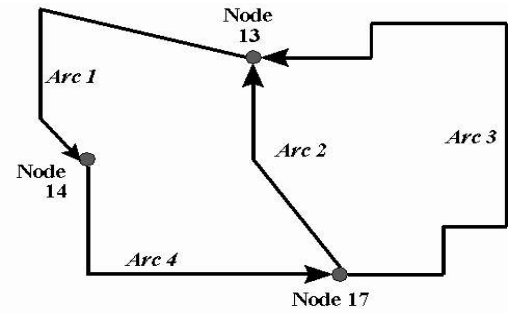

Abbildung 1: Arcs [Zeiler94]

Polygone sind eine geschlossene Abfolge von Arcs, d.h. Arcs, bei denen der Anfangspunkt gleich dem Endpunkt ist. Gleichzeitig tragen sie einen Markierungspunkt ("label point") mit einer Identifikationsnummer innerhalb Ihrer Grenzen. ArcInfo verwaltet die Topologie, d.h. die Richtungen der Arcs, in der Regel selbst.
ArcInfo arbeitet mit Layern (bildlich als Folien vorstellbar), die als "Coverages" bezeichnet werden. Ein Coverage dient als Basis-Speichereinheit für Vektordaten in ArcInfo und nimmt neben den raumbezogenen, auch die zugehörigen, beschreibenden Daten auf. Jedes Coverage beinhaltet Informationen zu einem bestimmten Thema, wie z.B. das Straßennetz einer Karte. Neben einem, vom Benutzer zugeordneten Namen, enthält ein Coverage eine Attributtabelle, die als "feature attribute table" bezeichnet wird, sowie weitere Daten, die die Positionen der Objekte festlegen (vgl. [Zeiler94]). Eine Attributtabelle wird dabei als eine einfache Datenbanktabelle realisiert.

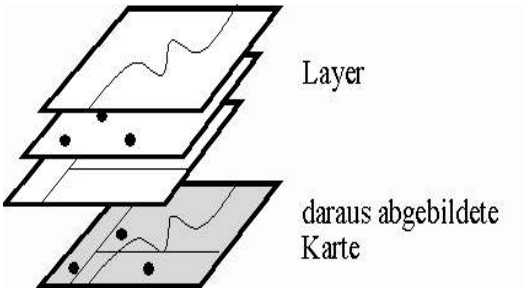

Abbildung 2: Coverages

Über die bereits angesprochenen, einfachen Objekte, wie z.B. Punkte oder Linien hinaus, können Coverages weitere, komplexere Daten aufnehmen, zu denen auch die Routen gehören. Eine Route ist ein Pfad entlang von verbundenen Arcs, wie z.B. eine Buslinie. Ein Routensystem ("route system") ist eine Sammlung von Routen, wobei zu einem Coverage mehrere Routensysteme gehören können. "Die Routensysteme sind eine Methode zur Darstellung von Abschnitten auf Linienstrukturen. Diese Abschnitte können dynamisch verändert werden (dynamic segmentation). Dabei muss nicht jedesmal die zugrunde liegende Arc-Node-Topologie geändert werden. Auf diese Weise können auf demselben Coverage verschiedene Routen (z.B. von Buslinien) definiert oder Verkehrsbelastungsszenarien durchgespielt werden, ohne dass eine Änderung an den Arc-Definitionen und damit der Grundtopologie notwendig ist." [Zipf96] Neben der Route kann als weiteres Element in einem Routensystem eine "Section" auftauchen. Dies ist entweder ein Teilstück oder auch eine komplette Arc (vgl. [Zeiler94]).
Sobald ein Routensystem besteht, können "Punkt Ereignisse" ("point events", z.B. der Standort

eines Verkehrsschildes) und Linienereignisse (“line events”, z.B. eine Geschwindigkeitsbegrenzung oder die Fahrtzeit eines Busses zwischen zwei Punkten) definiert werden. Die Programme unter ArcInfo bieten auf einem Routensystem spezielle Kommandos unter dem Stichwort “Dynamic Segmentation”, kurz “DynSeg” an (vgl. [Zeiler94]).
Weitere komplexe Daten bilden die "Annotations" (Anmerkungen) und die Regionen (zusammengefasste Polygone).

2.3. Interna

ArcInfo kennt folgende Exportformate für Karten: ArcInfo Graficsfile (.gra), EPS, Adobe Illustrator (.ai), Computer Graphics Metafile (.cgm)
Analog können über die Tool-Sammlung “Image-Integrator” folgende Formate importiert werden: ERDASS, GRASS, RLC, Grids, BIL, BIP, TIFF, Sun-Raster. Diese Formate können innerhalb von ArcInfo allerdings nur zu Illustrationszwecken oder als ein Sachdatum verwendet werden.
Als bekannte CAD-Formate unterstützt ArcInfo DXF, Intergraph und IGES, die nach dem Import jedoch nicht zu Analysezwecken herangezogen werden sollten.

"Die einzelnen Elemente der Coverages, [unter ArcInfo als “Coverage-Features” bezeichnet, also z.B. Polygone oder Linien,] besitzen zwei Kennummern. Die eine ist die vom Benutzer vergebene Feature ID, die andere eine fortlaufende, softwareinterne Nummer (sequence number). [...] Letztere kann sich nach Editiersitzungen ändern, was vom Benutzer nicht beeinflussbar ist. Beide Kennungen stellen ganzzahlige, numerische Werte dar." [Zipf96]
Die Coverages selbst sind in Workspaces (Arbeitsbereiche) organisiert (vgl. [Zeiler94]). Ein Workspace ist implementiert als eine Serie von Verzeichnissen, Unterverzeichnissen und Dateien auf der Festplatte. Es gibt zwei verschiedene Arten von Verzeichnissen in einem Workspace: ein INFO-Verzeichnis und coverage-Verzeichnisse. Ein Workspace kann immer nur ein INFO-, aber mehrere coverage-Verzeichnis besitzen.
Die Daten eines Coverage werden in zwei Dateien gespeichert: einem “Coverage file” (trägt im Dateiamen die Zeichenfolge ARC) und einer Datei, die die Attributtabelle aufnimmt (vgl. [Zeiler94]). Alle diese Dateien besitzen die gleiche Dateinamenerweiterung “.adf” (“Arc Data File”). Die Attributtabelle ist dabei abhängig vom Inhalt des Coverages: so wird z.B. die Attributtabelle der Polygone in einer Datei gespeichert, die in ihrem Namen die Buchstabenfolge PAT (“Polygon Attribute Table”) trägt. Einige Attributtabellen beinhalten fest vorgegeben Datenfelder, die aber um eigene erweitert werden können.

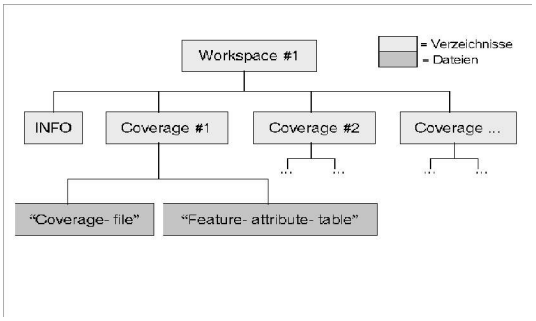

Abbildung 3: Aufbau eines workspace

Erweiterte Daten werden in zusätzlichen Dateien gespeichert: "Für jedes Routensystem eines Coverages finden sich die Daten über die Abschnitte in einer separaten SEC-Datei." [Zipf96]
Attributdaten können über die IDs (vgl. Abschnittsanfang) mit weiteren Sachdaten, z.B. aus einer externen Datenbank, verknüpft werden.
Die externen Daten können entweder per "Join" permanent an die "Feature Attribute Table" angehängt oder nur dynamisch mittels "Relate" verknüpft werden. ArcInfo kann mit drei Datenbankstrukturen umgehen: Feature Attribute Tables, INFO Datenfiles, relationale Datenbanken, sowie in INFO eingelesene ASCII-Dateien. ArcInfo verfügt darüber hinaus über Schnittstellen zu folgenden Datenbanken: ORACLE, INGRES, INFORMIX, SYSBASE, RdB (vgl. [Zipf96]).

2.4. Bedienung und Oberfläche

Nach dem Start befindet sich der Benutzer im ARC-Programm, einem nichtgrafischen System, das für den Umgang mit den GIS-Datenbanken zuständig ist. Neben Import, Export, Fehlerkontrolle, Verwaltung und Analyse der Karten ist es auch für das Management der "workspaces" zuständig. Sollte ein Makro namens "arc.aml" im aktuellen "Workspace" vorliegen, so wird dieses automatisch nach dem Start von ArcInfo ausgeführt.
ARC bildet zudem den Einsprungspunkt für alle weiteren Programme, wie "ARCPLOT" und "ARCEDIT". ARCPLOT ist ein Sub-System und dient zur Visualisierung und grafischen Ausgabe von GIS-Datenbanken. Unter ArcPlot können nur Datenbankeinträge verändert werden - für alle geographischen Elemente ist ARCEDIT, ein Werkzeug zur Bearbeitung von Coverages, zuständig.
ArcInfo ist ein Programmpaket, dessen Bedienung fast Ausnahmslos über die Eingabe von textuellen Befehlen (über 120 exklusive der optionalen Erweiterungen) an einem Eingabeprompt erfolgt. Andere Module bzw. Makros können vom Eingabeprompt direkt durch die Eingabe ihres Namens aufgerufen werden (z.B. "`Arc: ARCPLOT`").
"Info", als ein weiterer Bestandteil von ArcInfo, ist eine eingebaute relationale Datenbank. Sie wird in der Regel verwendet, um Attributdaten mit anderen relationalen Daten zu verknüpfen. Das Programm "TABLE" stellt im wesentlichen eine textuelle "Oberfläche" für Info dar.
"ArcTools" wird in der Literatur zwar oft als ein weiteres Modul genannt, es handelt sich hierbei aber um eine Sammlung von Makros, die ein grafisches Frontend für ArcInfo zur Verfügung stellen.
Zur Bearbeitung von Routensystemen dient das Modul "Networks".

2.5. Programmierung: AML

ArcInfo besitzt eine integrierte Makrosprache, die Arc Makro-Language (AML), mit der sich Kommandofolgen zu neuen, eigenständigen Kommandos zusammenfassen lassen (vgl. [Helm00]). Die AML wurde nach Prime Computer Inc's Command Procedure Language (CPL) modelliert und ist ähnlich wie die Shellprogrammierung unter UNIX aufgebaut (vgl. [ESRI93]). Im Gegensatz zu Avenue (vgl. Kapitel 3.5.) ist die AML eine prozedurale, interpretierte Sprache. Der Ablauf eines Makros ist über Schleifen und Bedingungen steuerbar.
Einfache AML-Programme lassen sich durch zeilenweises aneinander fügen von Kommandozeilen in einer normalen ASCII- Textdatei erzeugen. Diese Textdatei kennt vier verschiedenartige Zeilentypen: ArcInfo Kommandos (wie sie auch in Arc oder einem seiner Unterprogramme eingegeben werden), AML Direktiven, AML Funktionen und Kommentare (vgl. [Zeiler94]).
Als Hilfe oder als eine Alternative zur manuellen Eingabe kann auch eine Kommandomitschrift in ein "watch-file", erfolgen. Dabei handelt es sich um eine Aufzeichnung aller vom Benutzer vorgenommenen Handlungen, sowie die darauf als Reaktion folgenden, textuellen Bildschirmausgaben. Die so erzeugten "watch-files" können anschließend mit einem entsprechenden Kommando in ein AML-Skript umgewandelt werden (vgl. [Helm00]).
Die AML ist unter allen ArcInfo-Programmen bekannt. Es muss aber beachtet werden, dass ein AML-Programm, das für ARCPLOT geschrieben wurde, nicht unter ArcEdit läuft (vgl. [Zeiler94]).

```
Arc: |> ARCPLOT <|
Copyright (C) 1989,1990,1991,1992 Environmental Systems
Research Institute, Inc.
            All Rights Reserved Worldwide
ARCPLOT Version 6.1.1 (December 23, 1992)
Arcplot: |> MAPEXTENT INDEX <|
Arcplot: |> LINESYMBOL 5 <|
Arcplot: |> POLYGONS CENSUS <|
Arcplot: |> RESELECT ROADS LINE DESC = MAJOR <|
ROADS lines: 21 of 153 selected
Arcplot: |> LINESYMBOL 12 <|
Arcplot: |> ARCS ROADS <|
Arcplot: |> POINTMARKERS SCHOOLS 24 <|
```

Abbildung 4: Beispiel für ein "watch-file" [ESRI93]

Im Folgenden sollen kurz die syntaktischen Unterschiede zu anderen Sprache skizziert werden (vgl. [Helm00]). Kommentare und Sonderzeichen sind folgendermaßen vergeben:

- `/*` Kommentar,
- `&` leitet eine Direktive ein (z.B. `&DO`),
- `[]` umschließt Funktionen,
- `%` umschließt Variablennamen (zur Verwendung des Inhalts),
- `~` am Zeilenende ermöglicht die Fortsetzung diese Zeile auf der Nächsten und
- `'` (Apostroph) umschließt Zeichenketten.

Einer Variable kann mit dem Kommando `&setvar Variablenname` (abkürzbar durch `&sv`) ein Wert zugewiesen werden. Alternativ ist es auch möglich, einen der sonst üblichen Operatoren (= oder auch :=) dazwischen zu setzten, Bsp: `&SV Variablenname = Wert`. Anschließend kann durch `&LV` oder `&TYPE Inhalt` der `%Variablenname%` auf dem Bildschirm ausgegeben werden. Der Typ der Variable wird automatisch nach dem zugewiesenen Wert vergeben.

Numerische Variablen lassen sich in Berechnungen verwenden, die die folgende Form besitzen: `&SV HEKTAR [%Flaeche% /100]`, was in mathematischer Schreibweise hektar=Fläche/100 entspricht. Werden bei der Zuweisung mehrere Bezeichnungen hintereinander aufgeführt, so wird automatisch eine Liste erzeugt, z.B.: `&SV Blaetter 5034 5044 5143`. Diese Listennamen können in Schleifen abgearbeitet und mit den Funktionen [EXTRACT] und [TOKEN] ausgewertet werden.
Variablen können auch als Parameter beim Aufruf einer AML übergeben werden:
`&RUN Amlname Parameter1 Parameter2` übergibt die aufgeführten Werte in der gleichen Reihenfolge an die durch `&ARGS Variable1 Variable2` bezeichneten Variablen.
Ein AML-Programm kann selbstverständlich auch ein anderes AML-Skript starten: Dazu wird die Routine, die als Unterprogramm dienen soll mittels `&ROUTINE <name>` und einem abschließenden `&RETURN` definiert. Ein Aufruf erfolgt dann per `&CALL <name>`. Analog können "Label" im Skript definiert werden (`&Label <name>`), an die mit `&GOTO <name>` gesprungen werden kann (vgl. [Zeiler94]). Jedes AML-Programm sollte die `&RETURN`-Direktive als letzte Zeile beinhalten, damit immer ein korrekter Rücksprung erfolgen kann (vgl. [ESRI93]).
In der AML existiert weiterhin ein spezieller "Point Buffer". Dies ist ein FIFO-Speicher, in den Punkte (im Sinne von Koordinaten) hineingeworfen werden können. Dieser Speicher ist insbesondere dann von Vorteil, wenn mehrere Punkte nacheinander bearbeitet werden sollen.

Die AML kann dazu verwendet werden, um eine grafische Oberfläche zu erzeugen. Dazu muss ein Menü ("Menu") erstellt werden, das eine selbstdefinierte Dialogbox repräsentiert (vgl. [Zeiler94]). Um ein solches Menü zu erzeugen, muss ein "menu file" erstellt werden. Dies ist eine spezielle AML-Datei mit der Endung .menu, die Anweisungen über die Platzierungen von "Widgets", den Menüelementen, enthält. Um die Menüs komfortabler aufbauen zu können, wurde ArcInfo ein grafischer Dialogboxeditor namens "FORMEDIT" beigelegt. Nachdem das Menü erstellt wurde, kann es von einem ArcInfo Prompt und damit selbstverständlich auch aus einem AML-Programm, mit dem Befehl `&menu <menudatei>` gestartet werden. Eine Ereignisbehandlung übernimmt das Menü später selbst, sofern bei der Erstellung den Elementen ("widgets") das jeweilige, aufzurufende AML-Skript zugeordnet wurde. Für einige widgets, wie z.B. einer Dropdown-Liste, ist hier die Angabe einer Variable als Datenquelle zwingend erforderlich.

3. ArcView GIS

3.1. Was ist ArcView GIS?

Im Gegensatz zu ArcInfo ist ArcView GIS, das ebenfalls von ESRI stammt, ein Desktop-GIS. Ursprünglich wurde ArcView GIS entwickelt, um Daten aus ArcInfo zu visualisieren. Mit der Version 2 wurde ArcView durch die Integration von Analysefunktionen, den Änderungsmöglichkeiten der Attributdaten, sowie erweiterten Präsentationsmöglichkeiten zu einem vollständigen GIS. Mit der aktuellen Version 3.2a wurde schließlich auch die Manipulation von Geometriedaten ermöglicht.
ArcView GIS verknüpft traditionelle Datenanalyse-Werkzeuge, wie Tabellen und Wirtschaftsgrafiken mit Landkarten zu einem kompletten, integrierten Analysesystem. Darüber hinaus lassen sich Daten aus ArcInfo, einschließlich der PC-Version und der SDE ("spatial database engine") visualisieren und bearbeiten (vgl. [SUN00]).
ArcView GIS ist für verschiedenen Plattformen (Windows, UNIX) erhältlich und aufgrund seiner objekt-orientiert ausgerichteten Software-Architektur leicht erweiterbar.

3.2. Konzepte und Arbeitsweisen

Wie alle GIS unterscheidet auch ArcView GIS zwischen Raum- und Sachdaten. Die Benutzeroberfläche von ArcView GIS arbeitet dabei mit folgenden Objekten (vgl. [Greve99]):
ArcView GIS fasst den aktuellen Status einer Arbeitssitzung in einem Projekt zusammen. Ein Projekt enthält selbst keine Raumdaten, sondern immer nur die Verknüpfungen zu diesen.
Ein "View" (dt. Ansicht) dient zur Anzeige, Auswertung und Analyse von Raumdaten. Im View-Modul, also der Teil in ArcView, der für die Verwaltung der Views in einem Projekt zuständig ist, lassen sich die Raumdaten in Form von interaktiven Karten bearbeiten. Views selbst bestehen aus Themen. Mit den Werkzeugen der Views können diese Themen angezeigt oder ausgeblendet, sowie Ausschnitt, Maßstab und Projektion der Karte festgelegt werden.
Diagramme dienen zur Visualisierung von Werten aus einer Tabelle oder eines Ausschnittes.
Ein Layout führt alle Elemente (Views, Tabellen und Diagramme) in einer Karte zusammen, die anschließend ausgedruckt werden kann. Im Gegensatz zu einem View, das Analysefunktionen bereit stellt, soll ein Layout primär dem Erstellen einer Druckvorlage dienen.
Ein Skript ist ein Makro in der eingebauten, objekt-orientierten Programmiersprache "Avenue".

Innerhalb dieser Oberflächenobjekte arbeitet ArcView GIS mit folgenden Elementen:
Objektklassen, auch als Objektgruppen bezeichnet, können zum einen Vektordaten, wie Punkte (z.B. Orte), Linien (z.B. Flüsse) oder Polygone bzw. Flächen (letztere sind geschlossene Polygone) sein, zum anderen aber auch Bild- bzw. Rasterdaten. Jedem einzelnen dieser Objekte können Sachdaten zugewiesen werden.
Ein "Thema" (auch "Layer") ist das ArcView GIS Gegenstück zu einem ArcInfo Coverage (vgl. [Haller99]). In ArcView GIS sind die darzustellenden Daten, z.B. Flüsse, Seen, Städte, etc. die durch die oben genannten Objektklassen grafisch kodiert werden, thematisch gruppiert. Jede dieser Gruppen, wie z.B. alle Flüsse einer Karte, bilden ein Thema. Im Gegensatz zu einem ArcInfo Coverage, darf unter ArcView GIS jedes Thema immer nur aus Daten *einer* Objektklasse, also z.B. nur aus Punkten, bestehen. Alle Themen übereinander bilden dann eine komplette Karte (vgl. [Fuest99]). Ein und derselbe Layer kann dabei mehrmals in ein Projekt übernommen werden (vgl.

[Haller99]). Zu jedem Thema gehört auch eine Tabelle, die die Koordinaten der, auf dem Thema abgebildeten Objekte (z.B. Punkte) speichert. Somit kann ein Thema auch als die räumliche bzw. grafische Präsentation einer Tabelle angesehen werden.
ArcView GIS besitzt für Themen ein eigenes Datenformat ("Shapes"). "Ein solches 'Shapethema' kann im Gegensatz zu Themen aus anderen Quellen, direkt in ArcView GIS bearbeitet werden." [Liebig99] Es besteht aber grundsätzlich die Möglichkeit, Themen aus anderen Datenquellen in ein solches Shapethema umzuwandeln.
Tabellen dienen in ArcView GIS primär der Verarbeitung von Sachdaten (vgl. [Liebig99]). Während zu jedem Thema immer eine Tabelle mit gleichem Namen gehört, die "Attributentabelle", muss nicht zwangsläufig zu jeder Tabelle in ArcView GIS ein Thema existieren. Liegen diese Daten bereits in einer Datenbank oder einer Textdatei vor, so erlaubt ArcView das entsprechende Feld in der Attributentabelle mit den Inhalten der Datenbank oder einer anderen Tabelle zu verknüpfen. Der Inhalt einer Tabelle kann graphisch als Thema, Diagramm oder alphanumerisch, d.h. wie gewohnt als Tabelle dargestellt werden.

3.3. Interna

ArcView GIS kennt neben dem eigenen Shapeformat auch folgende, weitere Vektorformate: ARC/INFO-Kartenebenen (Coverage, Layer, Map-Libraries, ArcStorm-Datenbank), Auto-CAD Zeichnungen, sowie Daten im MapInfo-Interchange-Format (MIF) und Daten aus SDE-Datenbanken. "Mit einem externen Programm können auch *.e00 Dateien [ArcInfo "Export-Dateien"] eingelesen werden".[Liebig99]
Das Shapeformat ist ein einfaches, von ESRI freigegebenes Dateiformat, das allerdings keine Topologieinformationen aufnehmen kann. Eine Shapedatei besitzt genau eine Objektklasse und entspricht somit einem Thema. Im Gegensatz dazu können ArcInfo Coverages aus mehreren Objektklassen bestehen, wodurch sie in ArcView nur mit mehreren Themen vollständig erfasst werden können (vgl. [Liebig99]).
Die unter ArcView GIS bekannten Rasterformate sind: ArcInfo Grids, Sun-Raster-Files, Erdas-Rasterdaten, -LAN, -GIS, -IMAGINE, TIFF, TIFF/LZW, RCL, BMP, -JPEG, BIL, BIP, BSQ-Daten. Da ArcView GIS selbst keine Möglichkeit der Georeferenzierung bietet, beschränkt sich die Rasterfunktionalität im wesentlichen auf die Darstellung der Daten. Bevor eine Grafik importiert werden kann, muss aus diesem Grund bei Rasterformaten, in denen keine Projektionsdaten gespeichert werden, ein "World File" (Endung .wld) erstellt werden (vgl. [Fuest99]).
Tabellen werden unter ArcView GIS in folgenden Formaten unterstützt: INFO, DBASE III und IV, diverse ASCII-Texte und SQL-Datenbankabfragen.

3.4. Bedienung und Oberfläche

Die grafische Oberfläche von ArcView GIS ist dynamisch aufgebaut, d.h. je nach Situation werden unterschiedliche Funktionen angeboten.

Abbildung 5: Benutzeroberfläche aus ArcView GIS [Liebig99]

Für jedes der in 3.2. angesprochenen Elemente, existiert ein entsprechendes Modul in ArcView GIS, das die zugehörigen Bearbeitungsfunktionen bereitstellt. Alle Module sind dabei untereinander dynamisch verknüpft. So wirkt sich z.B. die Änderung einer Legende im View-Modul auch auf die Darstellung im Layoutmodul aus.
Der Ablauf einer Kartenerstellung verläuft meistens nach folgendem Schema (Gliederung angelehnt an [Fuest99]):
1. Zunächst müssen entweder die Raumdaten erstellt oder vorhandene Daten, wie z.B. ein ArcInfo Coverage, geöffnet werden (vgl. auch Kapitel 3.2).
2. Je nach Datenquelle müssen die Projektionsdaten ArcView GIS, bekannt gemacht werden. Eine Projektion bezieht sich in ArcView GIS immer auf ein View und nicht auf ein bestimmtes Thema. (vgl. [Liebig99]).
3. Im View kann das Thema ähnlich wie in einem Vektormalprogramm bearbeitet werden.
4. Nun können raumbezogene Abfragen und Analysen vorgenommen werden. "Diese Themenanalyse bedeutet in ArcView GIS im wesentlichen Objekte eines Themas (Zielthema) anhand von selektierten Objekten eines anderen Themas auszuwählen." [Liebig99]
5. Sachdaten können in ArcView direkt in die Attributentabelle eines Themas eingegeben werden. Für eine umfassendere Bearbeitung, insbesondere der Verknüpfung eines Datenfeldes der Attributentabelle mit einer Datenbank, muss zwingend das Tabellenmodul herangezogen werden.
6. In ein View können zur Illustration zusätzliche Texte und Grafiken eingefügt werden. Diese Grafiken sind keine Raumdaten und können daher auch nicht mit Sachdaten verbunden werden.
7. Um das fertige Layout zu erzeugen, kann eine Layoutschablone (Vorlage) verwendet werden. Das Ergebnis ist dann eine druckbare Karte (gemäß der Ansicht des Views).

3.5. Programmierung: Avenue

Avenue ist die objekt-orientierte Skriptsprache in ArcView GIS. Genau betrachtet besteht ArcView GIS fast nur aus Avenue-Programmen (vgl. [Liebig99]). Jedem Steuerelement (Menü, Schaltfläche, Werkzeug) kann ein solches Skript zugeordnet werden, wodurch sich die gesamte Benutzeroberfläche von ArcView GIS den eigenen Bedürfnissen anpassen lässt.
Avenue verfolgt die allgemeinen Konzepte der objekt-orientierten Programmierung, allerdings werden dort Methoden als "Requests" bezeichnet. Im Folgenden werden Grundkenntnisse mit objekt-orientierten Programmiersprachen und den damit verbundenen Konzepten vorausgesetzt (vgl. [DoDi96]).

Ein Avenue-Befehl hat folgende Struktur: `theList=theView.GetActiveThemes`, wobei "`theView`" ein Objekt und "`GetActiveTheme`" eine Methode darstellt. Der Typ der Variable `theList` wird nicht angegeben - ArcView GIS nimmt diese Zuordnung selbst vor. Um das Speichermanagement braucht sich der Programmierer ebenfalls nicht zu kümmern, da dies von der, in Avenue eingebauten "garbage collection" übernommen wird.
Im Gegensatz zu anderen Sprachen, werden Klammern bei Methoden (in diesem Fall "`GetActiveThemes`") nur angegeben, wenn auch tatsächlich Parameter übergeben werden.

Die Klassen besitzen im ArcView GIS-Objektmodell eine hierarchische Ordnung (Vererbungshierarchie). Die Ausgangsklasse bildet dabei die Klasse "Obj"- von ihr sind alle weiteren Klassen abgeleitet. Da es sich bei Avenue um eine Skriptsprache handelt, können keine eigenen Klassen definiert werden.
Methoden können sowohl auf Objekte, als auch auf Klassen angewendet werden, wobei sich die Anwendung auf Klassen hauptsächlich auf die Erzeugung neuer Objekte (Instanzen) oder zur Informationsgewinnung beschränkt.

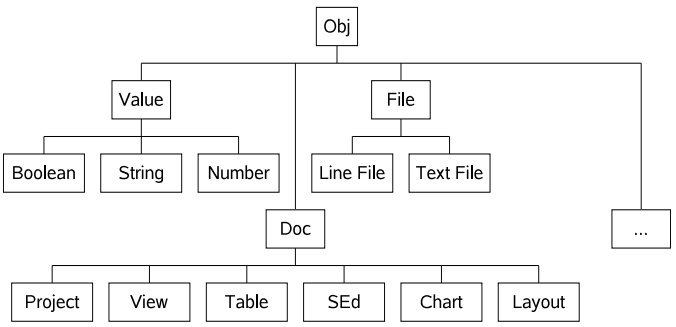

Abbildung 6: Ausschnitt aus der Avenue Objekthierarchie

Avenue-Programme werden im eingebauten Skript-Editor oder alternativ in einem Texteditor eingegeben. Vor einer Ausführung muss das erzeugte Skript im Skripteditor compiliert werden. Das fertige Programm kann anschließend einem Menü, einem Menüeintrag, einer Schaltfläche oder einem Werkzeug zugeordnet werden. Da die Skripte vom Menü oder den Symbolleisten aus gestartet werden, ist ein "Eventhandling" in Avenueprogrammen nicht vorhanden.
Der Skripteditor erlaubt mittels entsprechenden Managementfunktionen, dass mehrere Skripts zu

einem Projekt zusammengefasst werden können. Diese Projektverwaltung sollte insbesondere dann eingesetzt werden, wenn ein Skript mehrere andere Skripte aufruft. Darüber hinaus bietet der Editor Debugger-Funktionen an, mit denen z.B. die Variableninhalte näher untersucht werden können.
Der Aufruf eines anderen Skripts erfolgt mit der Funktion `av.Run(scriptname, selfobject)` (vgl. [Razavi97]). Dabei ist `av` ein reserviertes Wort und `selfobject` der Platzhalter für ein, an das Unterprogramm übergebenes Objekt, das für einen Datenaustausch verwendet werden soll. Im Unterprogramm erhält dieses Objekt automatisch den Namen "`self`". So ruft z.B. `thisView=Self.Get(Subject)` im übergebenen Objekt die Methode "`Get`" auf. Die Rückgabe von Werten an das aufrufende Programm geschieht über die Return-Anweisung. Gibt die Return-Anweisung nichts zurück, so wird immer das zuletzt erzeugte Objekt in dieser Unterroutine zurückgereicht.
Die Interaktion mit dem Benutzer erfolgt über die Klasse `MsgBox`, die für unterschiedlich aufgebaute Dialogboxen verschiedene Methoden bereit stellt.
Im Gegensatz zur AML arbeitet Avenue standardmäßig mit Threads (vgl. [Razavi97]]). Soll z.B. ein Avenue Skript die Kartenabmessungen fünfmal hintereinander anpassen, wartet Avenue nicht auf die Beendigung des ersten Zeichenvorgangs im zugehörenden Fenster, sondern führt bereits die nächste Anpassung im Hintergrund aus.
Im Unterschied zu anderen Programmiersprachen kann unter Avenue ein Objekt "gefunden" werden. Dies kann z.B. nützlich sein, um an aggregierte Objekte über ihren Namen zu gelangen. So findet z.B. der Befehl `myView=myProject.FindDoc(USA)` das ArcView Dokument mit dem Namen "USA", das anschließend über `myView` angesprochen werden kann.

ArcView GIS bietet neben Avenue noch weitere Möglichkeiten zur Integration von anderen Anwendungen mittels IAC ("Inter-Application-Communication") (vgl. [Liebig99]). So kann ArcView GIS z.B. Systembefehle an das Betriebssystem weitergeben. Unter Windows ist ein Datenaustausch zum einen mittels DDE ("Dynamic-Data-Exchange")-fähigen Anwendungen, zum anderen auch über die Zwischenablage möglich. Auf UNIX-Ebene können Funktionen eines anderen Programms per RPC ("Remote Procedure Call") aufgerufen werden. Für jede der oben genannten Methoden existiert in Avenue eine entsprechende Klasse, die einen komfortablen Zugriff auf die einzelnen Funktionen ermöglicht.

4. GRASS

4.1. Was ist GRASS

Das "Geographic Resources Analysis Support System", kurz GRASS, ist ein geographisches Informationssystem (GIS), das für Datenmanagement, Bildbearbeitung, Grafikproduktion, geographische Modellierungen und zur Visualisierung vieler Arten von Daten verwendet werden kann. Es stellt Module zur Bearbeitung von Raster-, Vektor-, Punkt- und Bilddaten zur Verfügung (vgl. [GRASS00]).
Ursprünglich wurde GRASS vom U.S. Army Construction Engineering Research Laboratories (USA-CERL), Teil des US Army Corp of Engineers, als ein Werkzeug zur Landverwaltung und Umgebungsplanung für das Militär entwickelt. Heute ist GRASS ein umfangreiches Werkzeug mit einer großen Bandbreite an Anwendungsmöglichkeiten in den verschiedensten, wissenschaftlichen Bereichen.
Die erste GRASS-Version wurde erstmals 1984 für die SUN-1 vorgestellt und ging aus dem 1982 entwickelten “Ford Hood Informationsystem” und dem 1983 entwickelten “Installation Geographic Information System (IGIS)” hervor. Die erste öffentliche Version wurde 1989 freigegeben.
Die ursprüngliche Anlaufstelle für den GRASS Support, sowie die Forschung und Weiterentwicklung war und ist die “GRASS Research Group” an der Baylor Universität in den USA. Durch den anhaltenden und schnellen Wachstum von GRASS, entstand eine große multi-nationale Entwicklergemeinde, so dass von der Universität Hannover eine zweite, europäische Anlaufstelle für GRASS geschaffen wurde [GRASS00].
GRASS ist unter der "GNU Public License" (GPL) primär für UNIX-Betriebssysteme wie Linux oder SunOS, sowohl im Quellcode, als auch als fertig compiliertes Programm erhältlich. Eine Windowsversion existiert bislang nur als BETA-Version.
GRASS "enthält mehr als 300 Programme und Hilfsmittel um Raster-, Vektor- und Punktdaten zu bearbeiten, Karten am Bildschirm und auf Papier zu erzeugen, Mulitspektralbilddaten zu prozessieren, sowie räumliche Daten zu erstellen, zu handhaben und zu speichern." [Neteler00] GRASS Module können Vektordaten und Rasterdaten wechselseitig im Format umwandeln. Weiterhin ist eine kostenlose Internetschnittstelle vorhanden, die es erlaubt, mittels GRASS ein vollständiges Online-GIS im World-Wide-Web (WWW) aufzubauen.
Die in der Visualisierung erzeugten Ansichten können als Standbilder oder als MPEG-Film für ein späteres Abspielen und Analysieren gesichert werden. Begleitend zu Landschaftsplanung und Ingenieuranwendungen enthält GRASS eine Sammlung an Simulationsmodellen für den Bereich der hydrologischen Modellierung und Analyse.

4.2. Konzepte und Arbeitsweisen

Ähnlich wie ArcInfo mit "workspaces", arbeitet GRASS mit “Locations” und “Mapsets”. “Location” ist der Name für die Projektumgebung (auch Projektgebiet), also eine vorhandene, geographische Umgebung, in der alle Projektdaten zu finden sind. "Diese Umgebung wird über ihre geographischen Ränder mit Koordinatenangaben und zusätzliche Projektionsangaben definiert." [Neteler00] Die Location ist somit ein übergeordneter Name für die Arbeiten, die der Benutzer in einem bestimmten geographischen Bereich vornimmt. Meistens ist dies identisch mit dem Ziel des Projektes, z.B. “Mars”, “Belgien”, “Dortmund”, etc. (vgl. [GRMAN]). Bei der Definition der Abmessungen einer Location sollte bedacht werden, dass sich diese unter GRASS

nachträglich nicht erweitern lassen.
Innerhalb einer Location können Arbeitsgebiete, sogenannte “Mapsets” festgelegt werden. Häufig wird aber nur ein Mapset erzeugt, das so groß wie die “Location” selbst ist. Der Einsatz mehrerer Mapsets bietet sich z.B. auch bei Teamarbeit an (vgl. [Neteler00]). Das Mapset ist der aktive, verwendete Teil, der kleiner oder genau so groß sein darf, wie die Location selbst (vgl. [Neteler98]). Z.B. könnte in der Location “Dortmund” ein Mapset “INFRA” gespeichert sein, das alle Karten mit den Infrastrukturinhalten aus Dortmund aufnimmt (vgl. [GRMAN]). Jede GRASS Sitzung läuft unter einen Namen eines Mapsets. Mit jedem Mapset verknüpft ist ein rechteckiges Koordinatensystem ("coordinate region") und eine Liste aller neu erstellten Karten.

GRASS arbeitet ausschließlich mit dem Layer-Konzept, wobei jede Karte in einem Mapset einen eigenen Layer bildet. Jeder dieser Layer wird in mehreren Dateien gespeichert. Diese Dateien besitzen zwar alle denselben Namen, liegen aber je nach Inhalt in einem anderen Unterverzeichnis. So werden z.B. die ASCII-Daten eines Layers in einem anderen Unterverzeichnis des Mapsets, als die Attributtabelle gespeichert.
"GRASS definiert immer ein Mapset 'PERMANENT', das wie ein Archiv verwendet werden kann; der Benutzer kann in einem selbstdefinierten Mapset arbeiten und hat gleichzeitig den Zugriff auf Basiskarten, die in PERMANENT gespeichert werden." [Neteler00] In der Regel enthält dieses Mapset die originalen, unveränderten Raster- und Vektordateien. Immer nachdem ein Mapset umbenannt wurde, muss GRASS verlassen und mit dem umbenannten Mapset neu gestartet werden (vgl. Kapitel 4.4.).

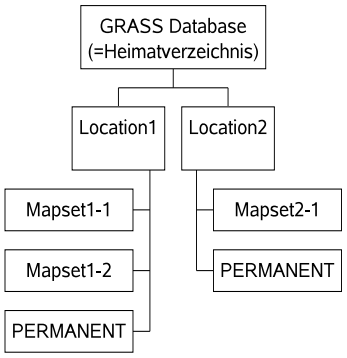

Abbildung 7: Verzeichnishierarchie in GRASS

GRASS hat gegenüber anderer GIS-Software den Vorteil, immer nur in einem aktiven Ausschnitt der gesamten Karte, der “Region” arbeiten zu können (vgl. [Neteler00] und [ClaBya97]). Eine gewählte Region kann selbst noch einmal maskiert werden. Dieser Ausschnitt wird analog als “Maske” bezeichnet. Ein GRASS Programm (vgl. Kapitel 4.5.) erhält immer nur die Daten zur Weiterverarbeitung, die aus diesem Bereich stammen. Auf diese Weise können beispielsweise Kartenausschnitte mit gewünschter Auflösung exportiert werden.

"Vom Konzept her betrachtet ist GRASS ein Modular aufgebautes GIS. Das bedeutet, dass jede Datenbearbeitung mit einem eigenen Teilmodul durchgeführt wird." [Neteler00] Erweiterungsmodule, die direkt auf der Homepage von GRASS bereit gestellt werden, sind u.a. eine Integrationsmöglichkeit von MPEG-Animationen oder mit “GRID3D” ein “raster volume Interface”, das

dreidimensionale Rasterdaten verwalten kann (z.B. für Wettersimulationen).
Aufgrund fehlender Module für Netzanalysen ist die Verwaltung einer Netzwerktopologie, wie dies z.B. ArcInfo durch sein Routensystem bietet, unter der Standardversion von GRASS derzeit nur eingeschränkt möglich.
Simulationsmodelle können in GRASS durch externe Programme eingebunden werden, einige wenige Modelle sind bereits in der Standardversion integriert. Zu beachten ist aber, dass eine Verarbeitung und Analyse von Zeitdaten, wie sie in vielen Simulationsmodellen benötigt werden, erst mit Version 5 zur Verfügung stehen wird.

4.3. Interna

Importieren lassen sich unter GRASS folgende Formate: Rasterformate: ASCII, ArcInfo, ascii grid, ERDAS LAN, HDF, GIF, TIFF, Sun-Raster, PPM, TGA, NHAP, BIL/BSQ, LANDSAT TM,MSS, SPOT, Vektorformate: ASCII, ArcInfo ungenerate, ArcInfo E00, ArcView Shape, DLG (U.S.), DXF, DXF3D, GPS-ASCII, USGS-DEM, IDRISI, MOSS, TIGER, VRML. Exportieren lassen sich die Formate: Rasterdaten: ASCII, BIL, ArcInfo, ascii grid, ARCTIFF, ERDAS LAN, HDF, MPEG, Povray, PPM, TIFF, TGA, Vektorformate: ASCII, ArcInfo ungenerate, ATLAS, DLG, DXF, IDRISI, MAPINFO, MOSS, SDTS, XFIG. Darüber hinaus existieren Befehle für die Verwendung des PostScript-Formates.
Rasterdaten werden in einer Matrix gespeichert. "Unter GRASS bekommt dabei jede Zelle ein Attribut (Eigenschaft, Sachdatum) zugewiesen, das das zu speichernde Phänomen repräsentiert (z.B. einen Temparaturwert)." [Neteler00] Zu beachten ist, dass unter GRASS die Daten in Rasterzellen auf ganzzahlige Werte limitiert sind. Zwar ist die Verarbeitung von Fließkommazahlen unter GRASS 4.3 ansatzweise durch ein Modul gegeben, eine vollständige Unterstützung wird aber erst GRASS 5.0 bieten. Diese Limitierung bezieht sich ausschließlich auf Rasterdaten (vgl. [Neteler98]).
GRASS Vektordaten werden durch “arc-node” Beziehungen gespeichert. Eine Arc ist ein nicht unterbrochener Linienzug, der als eine Serie von x-y-Koordinaten gespeichert wird (vgl. [GRMAN]). Eine Linie verbindet jeweils zwei Endpunkte, die wiederum Koordinaten besitzen und ein oder mehrere Attribute haben. In GRASS können auch Punkte existieren, die nicht Teil einer Linie sind. In einem solchen Fall werden sie als "sites" bezeichnet und besitzen ein eigenes Speicherformat (vgl. [Neteler98]). Unter GRASS können die einzelnen Datenstrukturen ineinander konvertiert werden, indem z.B. aus Rasterdaten Vektorlinien generiert werden. GRASS unterscheidet zwischen Geometrie (Position eines Objektes im Raum) und Topologie (Lage von Teilobjekten zueinander), wobei Letztere intern in GRASS verwaltet wird. Die Berücksichtigung der Topologie bedeutet einen wesentlichen Vorteil gegenüber ArcView, das über diese Methode der Datenverwaltung nicht verfügt (vgl. [Neteler00]).
Die Sachdaten können in Datenbanken oder in GRASS selbst gespeichert werden. Für die Version 5 sind u.a. Schnittstellen zu folgenden Datenbanken geplant: Oracle, Informix und PostgreSQL. Für die Version 4.x existiert bislang nur eine PostgreSQL-Erweiterung.

GRASS speichert seine Daten (u.a. die Locations und Mapsets) in einem Unterverzeichnis, das “GRASS database” genannt wird (vgl. [Neteler98]). Meistens ist dies identisch mit dem Heimatverzeichnis des Benutzers. Vektordaten werden in einem speziellen Binärformat gespeichert, Punktdaten (“sites”) in einer ASCII-Liste und Rasterdaten in einem speziellen, komprimierten Binärformat. Die Datenorganisation darf immer nur mit GRASS eigenen Befehlen durchgeführt werden, da ansonsten interne Verknüpfungen zerstört werden könnten.

4.4. Bedienung und Oberfläche

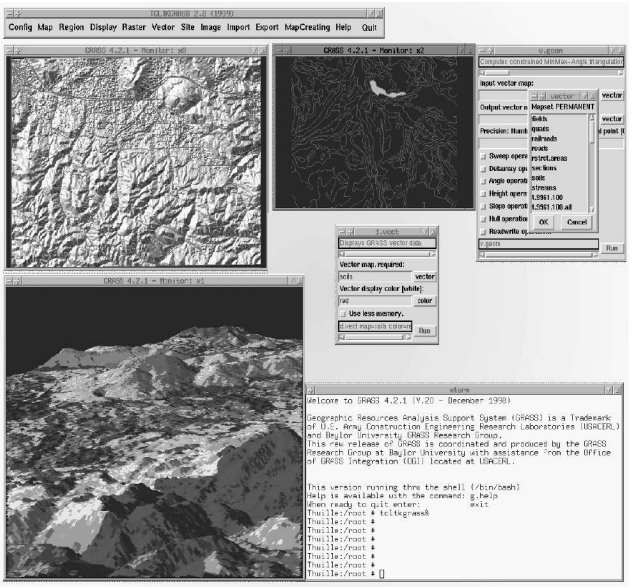

Abbildung 8: Benutzeroberfläche von GRASS [GRASS00]

GRASS wird, wie auch ArcInfo, über standardisierte Text-Kommandos gesteuert. Zusätzlich steht ein, auf Tcl/Tk basierendes, grafisches Frontend ("TclTkGRASS") zur Verfügung.
Nach dem Start muss vom Benutzer stets die "Location", das "Mapset" und der Pfad zur "GRASS-Database" angegeben werden. GRASS zeigt als Hilfe hier und in vielen anderen Situationen eine, aus ASCII-Zeichen bestehende Eingabemaske an.
Um einen grafischen Output einer vorliegenden Karte zu erhalten, muss ein spezielles Fenster geöffnet werden, das unter GRASS als "Monitor" bezeichnet wird (vgl. [Neteler98]). GRASS kann bis zu 8 dieser Monitore (x0 bis x7) gleichzeitig verwalten. Da GRASS-Monitore keine normalen Fenster sind, dürfen sie nur mit den GRASS eigenen Befehlen verändert und geschlossen werden.
"Die GRASS-Kommandos (typischerweise als GRASS-Module bezeichnet) sind den übrigen UNIX-Kommandos nach dem Start von GRASS gleichgestellt." [Neteler00] Dies bedeutet insbesondere, dass innerhalb der GRASS-Umgebung (realisiert innerhalb eines Terminalfensters), auch alle UNIX-Befehle zur Verfügung stehen. Anschaulich kann dies als eine Erweiterung der Shell um zusätzliche Kommandos angesehen werden. Die Syntax der GRASS-Befehle spiegelt die Gliederung der Befehle und ihre Modulzugehörigkeit wieder. Dabei gibt der erste Buchstabe die Funktionsklasse an, anschließend folgt ein Punkt und dann ein oder zwei Bezeichner, letztere wieder durch einen Punkt getrennt. `v.in.shape` ist z.B. ein Vektorbefehl (`v.`) zum Import einer ArcView Shape-Datei.

4.5. Programmierung

Bis auf wenige Ausnahmen in Fortran 77, ist GRASS vollständig in C geschrieben. Dem entsprechend können eigene Erweiterungen über das C-Interface in Form der "GRASS Programming Library" erstellt werden. Optional existiert auch ein, allerdings noch etwas rudimentäres JAVA Interface. Darüber hinaus können in GRASS, wie auch in ArcInfo, Skripts eingesetzt werden, um einfache Anwendungen zu erzeugen oder um Arbeitsabläufe zu automatisieren.

GRASS weist vier Schichten auf, die vom Programmierer jeweils modifiziert werden können (vgl.[ClaBya97]). Die oberste Schicht bilden die "specialized Interfaces". Dies sind grafische Benutzeroberflächen, wie z.B. TclTkGRASS. Darunter liegt der "Command Level". Dies ist die schon unter 4.4. erwähnte Kommandozeilenoberfläche, die durch Skripte gesteuert und erweitert werden kann. Als nächstes in der Hierarchie folgt der "Programming Level", der alle Programme ("Module", vgl. Kapitel 4.4) aus GRASS beinhaltet. Diese können durch eigene C Programme, die die "GRASS Programming Library" verwenden, ergänzt werden. Die unterste Ebene bildet der "Library Level", der die eigentlichen Bibliotheken der "GRASS Programming Library" beinhaltet. Diese Bibliotheken sollten nur in Zusammenarbeit mit dem "GRASS Hauptquartier" (vgl. [GRMAN]) verändert werden.

Skripte
Da die GRASS-Kommandos den übrigen UNIX-Kommandos nach dem Start von GRASS gleichgestellt sind, können alle UNIX-Shellskripte auch unter GRASS ausgeführt werden. In einem solchen Fall ist es weiterhin möglich, ein herkömmliches UNIX-Shellskript mit GRASS-Kommandos zu mischen.
Ein GRASS-Shellscript sollte direkt nach dem Start verifizieren, ob der Benutzer GRASS zuvor ordnungsgemäß gestartet hat und wenn nötig, wichtige GRASS-Variablen in die entsprechende UNIX-Form übersetzten.

Programmierung
Ein GRASS Programm ist ein C-Programm, dass einigen Restriktionen unterliegt. So muss das Programm die Header-Datei "gis.h" einbinden und die Funktionen der oben angesprochenen "GRASS Programming Library" nutzen. Die "GRASS Programming Library" besteht selbst aus mehreren einzelnen Bibliotheken. Funktionen aus diesen Bibliotheken beginnen immer mit einem Großbuchstaben, der meist vom Anfangsbuchstaben der entsprechenden Bibliothek abgeleitet wird, gefolgt von einem Unterstich und dem eigentlichen Funktionsnamen. Ein Beispiel ist die Funktion, die in jedem Programm enthalten sein muss, um die "GIS Library" zu initiallisieren: `G_gisinit(program_name)`, wobei `program_name` vom Typ `char*` ist.
Für Kommandozeilenprogramme, die kein Eingabeinterface zur Verfügung stellen, bietet die GRASS-API Mechanismen an, die zum einen den Programmierer bei der Auswertung der Kommandozeilenargumente unterstützen, und zum anderen die Verarbeitung in allen GRASS-Programmen standardisieren sollen.
Wichtig ist ebenfalls, dass GRASS-Programme einige C-Funktionen aus den Standardbibliotheken nicht verwenden dürfen. Statt dessen müssen sie auf die entsprechenden Funktionen aus der GRASS-Bibliothek zurückreifen. Dies betrifft insbesondere die Funktionen, die Strings manipulieren, wie z.B. `strcpy()`.

Weiterhin dürfen GRASS Programme keine "hartkodierten" Datenbanknamen ansprechen, da

diese normalerweise vom Benutzer zur Laufzeit gewählt werden. Diese Datenbanknamen werden in Variablen gespeichert, die wiederum in einer versteckten Datei im Verzeichnis des Benutzers abgelegt werden. Diese Informationen dürfen anschließend nur über Funktionen aus der "GIS Programming Library" abgefragt werden. Der Vorteil dieses Konzeptes gegenüber den Variablen in einer UNIX-Umgebung ist der, dass diese Variablen von einem GRASS-Programm gesetzt und zu einem späteren Zeitpunkt von einem anderen Programm weiterverwendet werden können.

Compiliert und installiert werden GRASS Programme mit dem Werkzeug gmake4.2. Dieses Programm stellt ein hardwareunabhängiges Frontend zum UNIX-Kommando "make" dar. gmake4.2 liest eine Datei namens Gmakefile ein, um daraus die Datei make.rules zu erzeugen. Anschließend wird automatisch das UNIX-Kommando make gestartet. Das Gmakefile wird dabei durch den Programmierer wie ein herkömmliches "makefile" erzeugt. Die dort verwendeten, vordefinierten Variablen sind in [ClaBya97] beschrieben.
Das fertige Programm muss anschließend in eines von insgesamt sechs Verzeichnissen kopiert werden, je nachdem ob es sich um ein interaktives oder Kommandozeilen-orientiertes Programm handelt.

5. Fazit

Wie [Lehmann] zeigt, eignet sich ArcInfo zum Aufbau und zur Analyse eines Nahverkehrsnetzes. Es werden dort für die einzelnen Linien Vektorlinien und Polygone verwendet, dessen Attributtabellen um die entsprechenden Daten (z.B. Fahrtzeiten) ergänzt wurden. Darüber hinaus besitzt ArcInfo als einziges der drei Systeme die Möglichkeit, ein Routensystem aufzubauen und dieses anschließend mittels der eingebauten Kommandos zu analysieren.
Aufgrund seines Leistungsumfanges kann das ähnlich aufgebaute GRASS ebenfalls für diese Planungsaufgaben eingesetzt werden.
ArcView GIS ist durch seine Ausrichtung als Desktop-GIS mehr für eine Visualisierung der Ergebnisdaten brauchbar. Zudem fehlen ihm einige nützliche, aber nicht zwingend notwendige Analysefunktionen.
Die beiden, für die Projektgruppe in Frage kommenden Systeme "ArcInfo" und "GRASS", unterscheiden sich im wesentlichen durch die unterschiedlichen Begriffs- bzw. Kommandobezeichnungen, sowie die verwendeten Programmiersprachen zur Entwicklung von Erweiterungen.
Ein Nachteil von GRASS ist die zur Zeit nur mäßig gut durchführbare Vekrotkartengestaltung. Zudem ist die Anbindung von GRASS an eine Datenbank zwar prinzipiell möglich, stabil und umfassend arbeiten soll dies aber erst in der geplanten Version 5.
Dem gegenüber ist GRASS kostenlos, der Quellcode frei einsehbar und sowohl die Skriptprogrammierung, als auch die Programmierung in C wesentlich flexibler und mächtiger als die der anderen, hier vorgestellten Systeme.

Abschließend lässt sich sagen, dass sich prinzipiell alle drei Geo-Informationssysteme für einen Einsatz nach dem in [Lehmann] beschriebenen Schema eigenen. Die wesentlichen Unterschiede bestehen in den zusätzlich bereitstehenden Analysefunktionen und den verschiedenen Skript- und Programmiermöglichkeiten.

Literaturverzeichnis

[ClaBya97] Clamons, Steven F.; Byars, Bruce W.: GRASS 4.2 Programmer's Manual, Baylor Univertity GRASS Research Group 1997, http://www.baylor.edu/~grass

[DoDi96] Doberkat, Ernst Erich; Dißmann, Stefan: Einführung in die objektorientierte Programmierung mit BETA, Addison Wesley, Bonn 1996

[ESRI00] ESRI Homepage, http://www.esri-germany.de

[ESRI93] Arc Macro Language, Environmental Systems Research Insitute, Redlands Californien 1993

[Fuest99] Fuest, Reiner: GIS Einführung ArcView, Universität Freiburg, http.//www.kulturgeo.uni-freiburg.de/mitarb/fuest/giskurs/gis.html

[GRASS00] GRASS, europäische Homepage: http://www.geog.uni-hannover.de/grass/

[Greve99] Greve, Klaus: Einführung in die Geographische Informationstechnologie - Vorlesung für fortgeschrittene Studierende, 1999, http://www.giub.uni-bonn.de/Greve/Lehre/SoSe99/VLGIT/2/UeAV.htm

[GRMAN] GRASS Beginner's Manual: http://www.geog.uni-hannover.de/grass/

[Haller99] Haller, Ruedi: Einführung in ArcView 3, http://www.nationalpark.ch/gis/dienstl/schule/einf1.html

[Helm00] Helm, Henry: Einführung in die AML Makrosprache, http://home.t-online.de/home/Henry.Helm/aml_arc.htm

[Hutchin95] Hutchinson, Scott; Daniel, Larry: Inside ArcView, OnWord Press, Santa Fe (USA) 1995

[Lehmann] Lehmann, Ulrich: Herleitung eines nutzerorientierten ÖPNV-Netzes mit Hilfe von ARC/INFO, Dipl. Dortmund

[Liebig99] Liebig, Wolfgang: Desktop-GIS mit ArcView GIS: Leitfaden für Anwender, Wichmann, Heidelberg 1999

[Neteler00] Neteler, Markus: GRASS-Handbuch, http://www.geog.uni-hannover.de/grass

[Neteler98] Neteler, Markus: Introduction to GRASS GIS Software, Hannover 1998, http://www.geog.uni-hannover.de/grass

[Razavi97] Razavi, Amir H.: ArcViewGIS/Avenue Developer's Guide, OnWord Press, Santa Fe (USA) 1997

[SUN00] Sun Solution Catalog: ArcView GIS, http://solutions.sun.com/catalogs/all/Geographic_Information_Systems/GIS_LIS/26553.htm

[Zeiler94] Zeiler, Michael: Inside ARC/INFO, OnWord Press, Santa Fe (USA) 1994

[Zipf96] Zipf, Alexander: Heidelberger Geographische Bausteine - Heft 13: Einführung in GIS und ARC/INFO, Geographisches Institut Heidelberg 1996