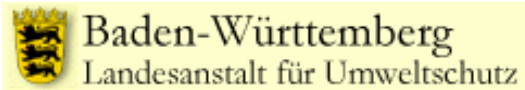


Fachhochschule
Neubrandenburg



FACHBEREICH BAUINGENIEUR- UND VERMESSUNGSWESEN



Studiengang Vermessungswesen

Diplomarbeit

**Evaluierung und prototypische Anpassung
einer Oracle Spatial Datenbank ans
Smallworld GIS**

vorgelegt von

Bastian Ellmenreich

Karlsruhe, 1.1.2002

1.Prüfer: Prof. Dr.-Ing. E. Heil

2.Prüfer: Prof. Dr.-Ing. A. Zimmermann

Diplomaufgabe

Thema: Evaluierung und prototypische Anpassung einer Oracle Spatial Datenbank an das Smallworld GIS

Die ständig wachsende Nachfrage nach raumbezogenen Daten lässt das Informationsvolumen stetig ansteigen. Eine Speicherung in den Dateiformaten der einzelnen CAD und GIS Hersteller reicht für eine effiziente Verwaltung von Geodaten oft nicht mehr aus. Dies und der Wunsch nach einem Standard für die Speicherung derartiger Daten machen die durch Oracle entwickelte „Oracle Spatial Datenbank“ für viele Anwender interessant. Die Umsetzung der Simple Feature Spezifikation ermöglicht die einheitliche Datenspeicherung und ist von großem Vorteil für die Nutzer dieses Standards.

Für die Landesanstalt für Umweltschutz Baden- Württemberg soll geprüft werden, inwieweit eine Oracle Spatial Datenbank die Daten des bestehenden Smallworld GIS übernehmen kann und sich somit als primäres Datenhaltungssystem eignet. Eine Beschreibung des Schemas ist die Basis für die Konzeption einzelner Stufen für die Anpassung an das Smallworld GIS. Innerhalb dieser Arbeitsschritte gilt es unter anderem zu klären, wie eine Datenmigration von Smallworld nach Oracle erfolgen kann. Die Realisierbarkeit des Datentransfers soll anhand eines zu entwickelnden Datenbankadapters nachgewiesen werden.

Fachliche Unterstützung in Form von Dokumentationen und Entwicklungswerkzeugen wird durch die Landesanstalt für Umweltschutz gewährleistet.

Prüfer: Prof. Dr.-Ing. E. Heil
Zweitprüfer: Prof. Dr.-Ing. A. Zimmermann

Termin der Abgabe: 7. Januar 2002

Diplomarbeit

zum

Erwerb des Grades

Diplom-Ingenieur (FH)

an der

Fachhochschule Neubrandenburg

im Fachbereich Bauingenieur- und Vermessungswesen

Diplomand: Bastian Ellmenreich

Organisation: Landesanstalt für Umweltschutz Baden - Württemberg
Griesbachstraße 1
76185 Karlsruhe

Inhaltsverzeichnis

Diplomaufgabe	2
1. Einleitung	6
2. Geodatenhaltung und -nutzung bei der LfU	8
2.1 Geodaten innerhalb des RIPS	13
2.2 Speicherformen	16
2.2.1 Arc Shape	16
2.2.2 Oracle Datenbank	19
2.2.3 Smallworld	23
2.2.3.1 Magik – Die Smallworld Entwicklungsumgebung	26
3. Einführung in Oracle Spatial	29
3.1 Allgemein	29
3.2 Das objekt- relationale Modell	35
3.3 Das relationale Modell	41
3.4 Vergleich beider Modelle	45
4. Konzeption der Anpassung einer Oracle Spatial Datenbank	47
4.1 Phasen der Anpassung	48
4.1.1 Beschaffung und Installation einer Oracle Spatial DB	49
4.1.2 Zugriff des Smallworld GIS auf Oracle Spatial Daten	50
4.1.2.1 Der Oracle Spatial SOM	52
4.1.3 Testen des Systems anhand von Beispieldaten	56
4.1.4 Entwurf eines Datenbankadapters zum Zweck der Datenmigration	57
4.1.5 Einfügen von realen Daten und Testen des Systems	60

5.	Entwicklung eines Datenbankadapters	62
5.1	Der Datenbankadapter	62
5.2	Der SHP2SDO – Editor	78
5.3	Der direkte Weg der Datenmigration	81
6.	Schlussbetrachtung und Ausblick	85
7.	Anhang	87
7.1	Ausschnitt des Oracle Datenbankschemas der LfU	87
7.2	OSPO- SOM Konfigurationsskript	88
7.3	Die ospo_ds.ini Datei	89
7.4	Die SQL Datei	90
7.5	Die LOADER Datei	91
7.6	Quellcode des Datenbankadapters	92
7.7	Quellcode des SHP2SDO Editors	119
7.8	Quellcode der Prozeduren des direkten Datentransfers	125
7.8.1	Die Prozedur make_sql_file_create_layer	125
7.8.2	Die Prozedur make_text_fields	127
7.8.3	Die Prozedur count_poly_cords	128
7.8.4	Die Prozedur make_geom_file_area	129
7.8.5	Die Prozedur write_geometries	132
7.9	Smallworld disjunkte Flächen	135
	Verzeichnis der Abbildungen	136
	Verzeichnis der Tabellen	137
	Literaturverzeichnis	138
	Verzeichnis der Abkürzungen	139
	Danksagung	140
	Erklärung	141

1. Einleitung

Ziel dieser Diplomarbeit ist es, das Schema von Oracle Spatial zu beschreiben und einen möglichen Weg für die Anpassung des Smallworld GIS an eine Oracle Spatial Datenbank aufzuzeigen.

Zu diesem Zweck wird im ersten Teil das bestehende Datenhaltungssystem analysiert. Ausgehend von den Stärken und Schwächen des „alten“ Systems soll der Aufbau und die Funktionsweise des „neuen“ Verwaltungssystems, Oracle Spatial, dokumentiert werden.

Die Anpassung des Smallworld GIS¹ an Oracle Spatial ist ein komplexer Vorgang, der in mehreren Phasen abläuft. Im Rahmen der Diplomarbeit wurde hierfür eine Konzeption entwickelt, die folgende Schwerpunkte untersucht.

1. Wie kann eine vollständige Datenübernahme nach Oracle Spatial aussehen?
2. Wie kann das Smallworld GIS auf diese Daten zugreifen?
3. Welche Probleme treten beim Testen dieser Systemarchitektur auf (Erfahrungen beim Arbeiten mit den Testdaten)?

Für die Beantwortung der ersten Frage wurden im praktischen Teil der Arbeit zwei verschiedene Ansätze entwickelt. Diese beschreiben zum einen ein Transferieren der Daten über das Dateiformat ESRI Shape und zum anderen einen direkten Weg von Smallworld nach Oracle Spatial.

Das bis dahin eingesetzte Smallworld GIS soll hinsichtlich seiner Funktionalität, bis auf die Massendatenverwaltung, vollständig erhalten bleiben. Somit ist im Punkt zwei sicherzustellen, dass nach einer Migration der Daten nach Oracle Spatial das Smallworld GIS auf diese Geodaten weiterhin zugreifen und damit arbeiten kann. Die hierfür vorgestellte Lösung, der OSPO-SOM², beschreibt die von Smallworld entwickelte Schnittstelle, die durch Verwendung von Testdaten auf ihre Funktionsfähigkeit hin überprüft wurde. Im Rahmen der Analyse für die

¹ GIS = Geoinformationssystem (nachfolgend erläutert)

² OSPO-SOM = Oracle Spatial Objects-Spatial Objects Manager (vgl. 4.1.2.1)

Datenumsetzung wurden auftretende Probleme festgehalten und in der Schlussbetrachtung zusammengefasst.

Im Ergebnis soll eine Entscheidungsgrundlage für die Landesanstalt für Umweltschutz entstehen, mit deren Hilfe die Frage nach der Einsetzbarkeit von Oracle Spatial als primärer Datenspeicher beantwortet werden kann.

2. Geodatenhaltung und -nutzung in der LfU

Als Geodaten bezeichnet man Datenobjekte, die durch eine Position im Raum direkt oder indirekt referenzierbar sind. Die Datenerfassung, -haltung und die Datennutzung umweltbezogener Geodaten in Baden- Württemberg werden im Rahmen eines landesweiten **Umweltinformationssystem** dem **UIS** Baden-Württemberg koordiniert.

Allgemein sind Umweltinformationssysteme keine geschlossenen Systeme. Vielmehr bestehen sie aus einer Vielzahl von Teilkomponenten, welche verschiedenen Klassen zuzuordnen sind. Man unterscheidet hierbei in Basissysteme, Fachsysteme, übergeordnete UIS - Komponenten und dem UIS als Konzept und Rahmen. Basissysteme umfassen Normen, Netzinfrastruktur, Rechnerinfrastruktur und Hintergrunddatenbanken. Fachsysteme verarbeiten umweltrelevante Informationen in spezifischen Fachzusammenhängen und bilden die datentechnische Basis des UIS.

Als übergeordnete UIS- Komponenten bezeichnet man Systeme, welche Daten aus den jeweiligen Fachsystemen miteinander auswerten und so themenübergreifende Informationen bereitstellen. UIS als Konzept stellt einen organisierten Zusammenhang zwischen den Einzelkomponenten her.

Innerhalb dieser Komponenten gibt es Anwendungen, die in Form von selbstständigen Informationssystemen Umweltdaten nach dem EVAP – Prinzip³ verwalten. Dies gilt insbesondere für die Fachanwendungen.⁴

Die für diese Programme zugrundeliegenden Daten haben meistens einen räumlichen Bezug, weshalb die eingesetzten Informationssysteme zur Kategorie der **Geoinformationssysteme**⁵ (**GIS**) gehören. Die Aussage von Bill/Fritsch, dass man Umweltinformationssysteme als erweitertes Geoinformationssystem definiert, findet sich somit auch in der Konzeption des UIS - BW wieder.

³ Eingabe, Verarbeitung, Auswertung und Präsentation (Bill/Fritsch 1994)

⁴ vgl.: Klaus Greve - „Ein Referenzmodell zur Beschreibung der Konzeption von UIS“

⁵ Ein Geo-Informationssystem ist ein rechnergestütztes System, das aus Hardware, Software, Daten und den Anwendungen besteht. Mit ihm können raumbezogene Daten digital erfasst und redigiert, gespeichert und reorganisiert, modelliert und analysiert sowie alphanumerisch und graphisch präsentiert werden (Bill / Fritsch, 1991).

Einsatz, Nutzung und Austausch der Geodaten innerhalb des UIS werden in dem ressortübergreifenden Vorhaben „RIPS“⁶ koordiniert. Der dem RIPS zugrundeliegende Datenbestand fasst man in einem Datenpool zusammen, auf den die verschiedenen Fachanwendungen Zugriff haben.⁷

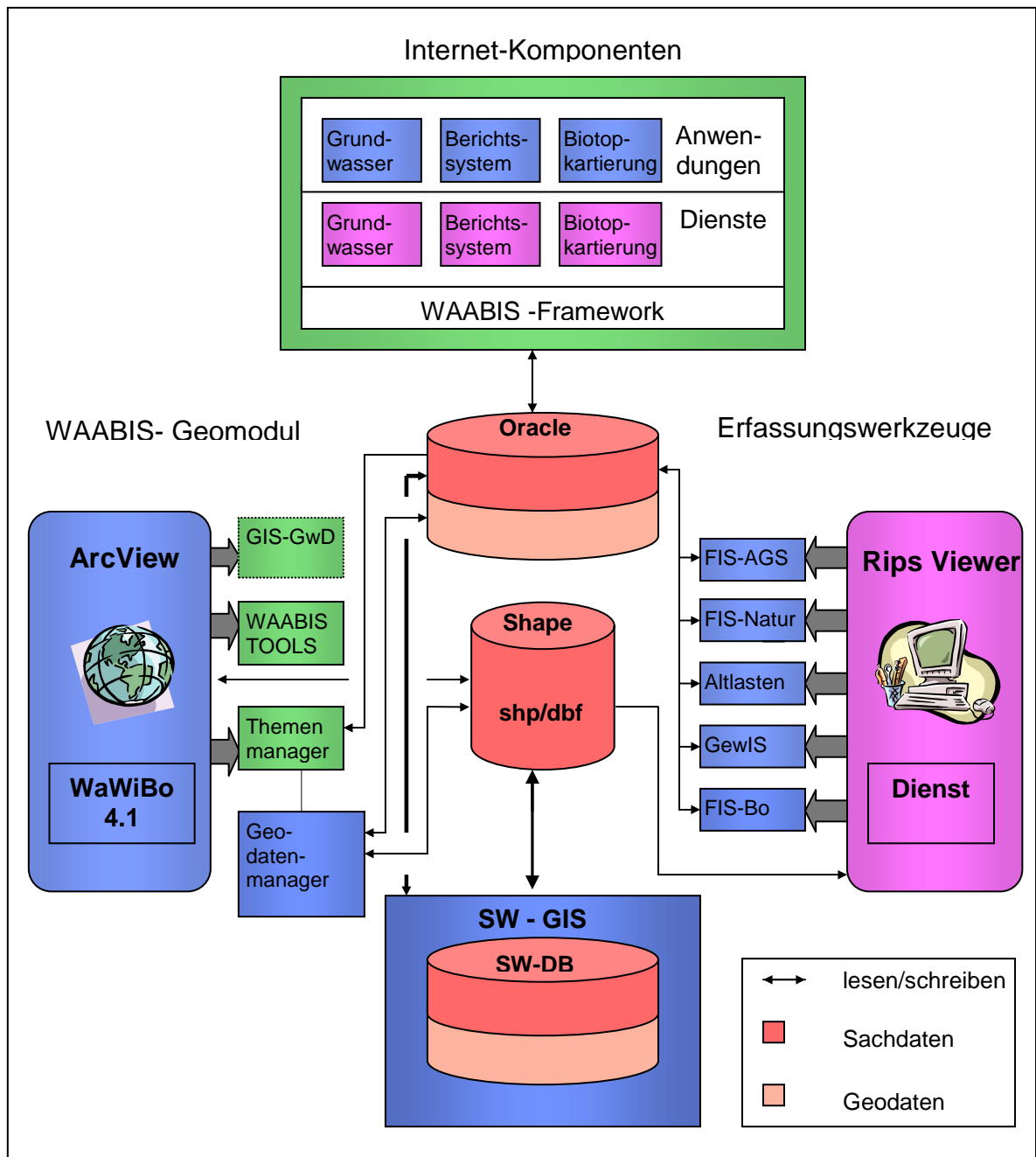


Abbildung 1 : Geodatenorganisation

⁶ Räumliches Informations- und Planungssystem

⁷ vgl.: Manfred Müller- „Anforderungen an Interoperable GIS- Werkzeuge für Umweltinformationssysteme“

Das Smallworld GIS stellt hohe Anforderungen an Rechner und Bearbeiter und dient somit hauptsächlich der Verwaltung der Smallworld Datenbank. Spezielle Themen, wie zum Beispiel Fließgewässer, die eine sehr komplexe Datenstruktur haben, werden außerdem mit dem GIS erfasst.

Weitere Erfassungswerkzeuge für Sach- und Geometriedaten sind Anwendungen aus der ESRI Produktfamilie. Die eigens dafür entwickelte Arc View Fachschale **WaWiBo (Wasser, Wirtschaft, Boden)**, wird von den Fachanwendern sowohl zur Erfassung als auch zur Präsentation umweltbezogener Themen verwendet und gilt innerhalb des RIPS als zentraler kartographischer Arbeitsplatz.

In der LfU unterliegen die Sachdaten im Gegensatz zu den Geometriedaten regelmäßigen Veränderungen von Seiten der Fachanwender. Dazu nutzen Sie spezielle Fachinformationssysteme⁸ (FIS-Natur, GewIS u.a.), die einen fachlich eingeschränkten Zugriff auf die Oracle Datenbank haben. Die Fachinformationssysteme dienen damit der Erfassung aber auch der Auswertung von gespeicherten alphanumerischen Daten.

Nach einer Reform im Jahr 1995 wurden viele, vorher auf Landesebene bearbeitete Umweltaufgaben auf die einzelnen Städte und Landkreise übertragen. Seit diesem Zeitpunkt gehören auch Bürger, Verbände und Planungsbüros zum Nutzerkreis. Um diesen Anwendern die Daten ohne den Einsatz „teurer“ GIS zugänglich zu machen, entstanden verschiedene internetfähige Komponenten. Sie haben bislang eine reine Auskunft- und Präsentationsfunktion, sollen aber demnächst themenspezifisch schreibfähig werden.

Da zur Erfassung, Auswertung und Präsentation der Geodaten in den einzelnen Sachgebieten jeweils verschiedene Anwendungen zum Einsatz kommen, gibt es bislang mehrere Speicheroptionen. Das Zentrale Datenhaltungssystem oder der Master für die Geodaten ist eine **Smallworld Datenbank (SW-DB)**. Zur Präsentation oder für kartographische Zwecke werden jeweils einzelne Themen⁹ der SW-DB ins ESRI – Shape Format transformiert. Ausgehend von diesem Dateiformat können die Daten, wie auch in Abbildung 1 zu sehen, über eine

⁸ Informationssystem, das sich auf ein spezielles Fachgebiet bezieht (Geologie, Bodenkunde u.a.)

⁹ Ein Thema beinhaltet räumliche Objekte, die jeweils die gleichen Eigenschaften aufweisen.

Schnittstelle, den Geodatenmanager, in eine Oracle Datenbank migriert werden und so als Informationsquelle für die meisten Anwendungen fungieren.

Dieser Datenkreislauf ist ein Kompromiss zwischen dem Wunsch der Fachanwender, traditionelle Anwendungen weiterführen zu können, und gleichzeitig ein gewisses Maß an Datenintegrität zu gewährleisten. Des Weiteren spielen aber auch softwarespezifische Einschränkungen und wirtschaftliche Aspekte eine große Rolle. Der Nachteil dieser Datenkonvertierung ist die redundante Datenhaltung, die einhergeht mit dem Problem der „up to date“ Frage. Insbesondere bei der Erstellung von thematischen Karten ist es erforderlich, über aktuelle Daten zu verfügen.

Das Vorhandensein dieser getrennten Datenhaltung ist weniger ein Fehler in der Konzeption des RIPS, als vielmehr fehlende Interoperabilität¹⁰ zwischen den einzelnen Anwendungen. GIS Entwickler haben bislang aus strategischen Gründen zu stark auf eigene Formate und ein starres Datenmodell innerhalb ihrer Anwendungen gesetzt. Mit zunehmender Bedeutung der Geodaten und einer immer größer werdenden Anwenderzahl, erkannten führende Softwareentwickler, dass es Standards für die verschiedenen Datenmodelle und für die softwareseitige Implementierung dieser Modelle geben muss. Das 1994 gegründete **Open GIS Consortium (OGC)** definiert in Zusammenarbeit mit den Herstellern verschiedene Spezifikationen. Eine der ersten Spezifikationen ist die „SIMPLE-FEATURE SPEZIFIKATION“. Diese Richtlinie definiert für die plattformunabhängigen Umgebungen CORBA (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture), COM (**C**omponent **O**bject **M**odel) und SQL (**S**tructured **Q**uery **L**anguage) jeweils eigene Implementierungsspezifikationen, welche auch auf Geometrietypen eingehen. Ein Feature ist die digitale Speicherung eines Objektes oder eines Phänomens der realen Welt, bestehend aus Geometrie, semantischen Eigenschaften und Metadaten.¹¹

Simple bedeutet in diesem Zusammenhang, dass es Einschränkungen bezüglich Geometrietypen gibt, und keine Metadaten gespeichert werden.

¹⁰ Interoperabilität bedeutet, dass Anwendungen Daten, egal in welchem Format sie vorliegen, mit eigenen Funktionen sofort bearbeiten können. Die Daten werden nicht umgewandelt und bleiben physisch am selben Ort.

¹¹ vgl.: The OpenGIS® Guide – Introduction to Interoperable Geoprocessing and the OpenGIS Specification (1998)

Die Umsetzung der einmal beschlossenen Spezifikationen geht sehr schleppend voran. So wurden für die Simple - Feature Spezifikationen erst zwei Produkte ESRI SDE in der Version 3.0.2 für Informix, DB2 und Oracle sowie Oracle8 Spatial Cartridge in der Version 8.0.5, dem sogenannten Konformitätstest erfolgreich unterzogen. Der Prozess der Standardisierung hat für die Anwender wesentliche Bedeutung. Eine Einigung gerade bei den Geometrietypen gestattet eine Trennung zwischen der Datenebene und den Anwendungen. Diese Trennung ermöglicht den Einsatz von sehr spezifischen Anwendungen, was aus wirtschaftlicher Sicht viele Vorteile mit sich bringt.

2.1 Geodaten innerhalb des RIPS

Die im RIPS – Pool enthaltenen Daten werden sowohl in Sach- und Geodaten als auch in Fach- und Basisdaten unterteilt.

Als Basisdaten gelten alle durch die Vermessungsverwaltung des Landes (Landesvermessungsamt BW) geführten Datenbestände. Tabelle 1 zeigt die im RIPS- Pool vorhandenen Basisdaten.

Thema	Maßstab	Kartierzeit	Herkunft	NR	Bemerkungen
ALK/BGRUND	-	Ab 2001	LV	N	geplant
Deutsche Grundkarte DGK5	1:5000	laufend	LV	N	gescannte Einzelblätter
TK 25	1:25000	1994	LV	T	311 Blätter
TK 50	1:50000	1994	LV	T	92Blätter
TÜK 200	1:200000	1995	BKG (IFAG)	N	6 Blätter
ÜK 500	1:500000	1994	BKG (IFAG)	N	1 Blatt
ATKIS DLM 25/BW	1:10000	Ab 1991	LV	T	DLM25/1 komplett
ATKIS DLM	1:10000	Seit 1991	LV/ITZ,LfU	T	1. Version komplett
Digitales Höhenmodell	50 x 50 m	1993	LV	T	Oberfläche der Realnutzung
Verwaltungseinheiten	1:25000/ 1:200000	1999	LV/LfU	N	Gemeinden, Gemarkungen etc.

Tabelle 1 : Basisdaten

Zu den Fachdaten gehören Digitalisierungen genauso wie auch Daten, die von den einzelnen Fachreferaten erfasst wurden. Diese Informationen stammen zum einen aus dem Bereich Naturschutz bzw. Landschaftsökologie und zum anderen aus einem Bereich, der Technosphäre, Wasser, Boden und Luft umfasst.

Naturschutz, Landschaftsökologie					
Thema	Maßstab	Kartierzeit	Herkunft/ Urheber	NR	Bemerkungen
Biotopkartierung nach §24a	1:5 000 / 1:25 000	ab 1992	LfU, Stadt- und Landkreise	U	ca. 60.000 Objekte, lfd. Fortführung
NATURA 2000-Gebiete (vorläufig)	1:25 000	2000	LfU, BNL	U	Gebiete der Flora-Fauna-Habitat-RL
Arterhebungen	Fundorte/ Punkte	laufend	BNL, LfU, Verbände	N	Punkt- und Rasterkartierungen ca. 1 Mio Obj.
Ökologische Standorteignung	1:200 000	1993	FH Nürtingen	U	ca. 30 Faktorkarten
Bann- und Schonwälder	1:50 000	1994	FVA	N	ca. 70 Objekte
Natur- und Landschaftsschutzgebiete	1:25 000/ 1:200 000	1998	LfU, Abt. II	U	ca. 2.500 Objekte
Landschaftspflegepläne	1:1 500	laufend	BNL	U	Projektweise Erhebung
Landschaftsökologie	1:200 000	1999	MLR/Uni.Stgt.	U	ca. 100 Fachthemen
Naturraumeinheiten	1:200 000	1998	LfU/II	U	Überarbeitet nach Meyen/Schmithüsen
Realnutzung aus Landsat TM	30x30 m	1993	UNI KA, IPF	U	Satelliten-Bilddaten, 19 Klassen

Tabelle 2 : Naturschutz, Landschaftsökologie

Technosphäre, Wasser, Boden und Luft					
Thema	Maßstab	Kartierzeit	Herkunft/ Urheber	NR	Bemerkungen
Messnetze Wasser, Boden, Luft	Punkte	laufend	LfU	U	Ca. 120.000 Objekte
Wasser- und Abfallwirtschaftlicher Atlas	1:25 000	laufend	LfU, ITZ	U	Kartierung aller wasserwirtschaftlich rel. Objekte
Wasser- und Bodenatlas BW	1:200 000	laufend	LfU/ Uni.Freiburg	U	ca. 30 Themen
Gewässerläufe	1:10 000/ 1:50 000/ 1:200 000	2000	LfU, Abt. IV/ITZ	U	ca. 30.000 Objekte
Grundwasserlandschaften	1:200 000	1998	LfU, Abt. IV	U	Grobgliederung
Gewässereinzugsgebiete	1:50 000	2000	LfU, Abt. IV	U	440 Objekte
Arbeitsstätten und Anlagen	Standorte	1996	LfU, Abt. II	N	Über 1.000.000 Obj.; Georeferenz tw. nur über Adressen
Verkehr (Straßen- und Schienennetz)	1:150 000	1989	LfS	U	Knoten aus der Straßendatenbank
Schallimmissionspläne	1:1.500	laufend	LfU, Abt. III	U	Gemeindeweise Modell-Rechnungen nach RLS

Tabelle 3 : Technosphäre, Wasser, Boden und Luft

Innerhalb dieser Klassifizierung verwendet man sowohl Raster als auch Vektordaten. Beide Datentypen sind die Basis von hybriden Geoinformationssystemen. Sie werden benötigt um Reale – Welt - Objekte, die neben verschiedenen Sachinformationen eben auch eine räumliche Komponente haben, innerhalb eines Datenmodells darzustellen.¹²

¹² vgl.: http://www.lfu.bwl.de/local/abt5/itz/rips/rips_documente.htm (local Server)

2.2 Speicherformen

In der LfU werden die genannten Geodaten durch drei komplett verschiedene Systeme verwaltet. Das erste ist ein dateibasiertes Format, das Shape. Im wesentlichen dient es dem schnellen Transfer der Daten zwischen den einzelnen Fachreferaten. Die Fachdaten werden zumeist mit Arc View erfasst und somit im systemeigenen Dateiformat, dem Shape, gespeichert.

Das zweite System ist eine Oracle Datenbank der Version 8.0.5. Die Datenbank dient vor allem der Speicherung von Sachdaten, auf welche die meisten Anwender Zugriff haben. Über eine Schnittstelle, den Geodatenmanager, können auch Geometrien abgelegt werden. Dazu werden die Shapes in einen geschlossenen binären Stream gewandelt und als Datentyp Long Raw in die Datenbank integriert. Der Hauptdatenspeicher ist jedoch eine **Smallworld Datenbank (SW-DB)**. Die Geodaten liegen auf einem Grafikserver, welcher unter Linux arbeitet. Bis zu 20 Anwender haben die Möglichkeit, über jeweils eigene Alternativen¹³ auf diesen Datenbestand zuzugreifen. Über eine Shape Schnittstelle ist es möglich, neue Themen in die Datenbank aufzunehmen oder für Präsentationszwecke Themen zu entladen.

2.2.1 Arc Shape

Arc Shape ist ein Fileformat, welches innerhalb der verschiedenen ESRI Produkte (ARC/INFO, Spatial Database Engine (**SDE**), ARC VIEW u.a.) zur Speicherung von nichttopologischen Geometrien und Sachdaten dient. Durch die bloße Verwaltung der Geometrie ohne die topologischen Strukturen kann die Datei einfacher editiert und auch schneller graphisch dargestellt werden. Diese Eigenschaften, aber auch die Offenlegung der Struktur des Shapefiles durch ESRI, haben es zu einem der verbreitetsten Formate in der GIS - Welt gemacht. Als geometrische Grundformen werden Punkte, Linien und Flächen unterstützt. Ein Shape setzt sich zusammen aus dem „Main file“ *.shp dem „Index file“ *.shx und einer „dBASE table“ *.dbf.

¹³ vgl.: Kapitel 2.2.3

Das Main - File speichert alle Informationen über den räumlichen Teil eines Features. Seine Struktur ist binär, jedoch ist die Zuordnung der einzelnen Bytes bekannt.

Es beginnt mit einem größtmäßig immer konstanten Header¹⁴, gefolgt von den einzelnen Geometriedaten. Im Header werden Informationen wie Dateigröße, Version, Geometriotyp und Bounding Box¹⁵ der nachfolgenden Geometrien angegeben. Die einzelnen Geometrien besitzen wiederum jeweils einen eigenen Header (Record Header), gefolgt von der eigentlichen Geometrie (Record Contents). Damit ergibt sich der in Tabelle 4 skizzierte Aufbau.

File Header	
Record Header	Record Contents
Record Header	Record Contents
Record Header	Record Contents
Record Header	Record Contents

Tabelle 4 : Shape Aufbau (Main – File)

Im Record Header steht sowohl die Nummer des Records als auch die Länge des folgenden Datenteils. Der Record Content gibt Auskunft über den Geometriotyp und dessen Koordinaten. Folgende Geometriotypen sind zulässig:

2D (X,Y)

Dient dem Darstellen von ebenen Objekten im zweidimensionalen Raum.

- Point und Multipoint
- Polyline bestehend aus einem Teilstück oder mehreren Teilstücken
- Polygon bestehend aus einer Fläche oder mehreren Flächen, welche sich nicht selbst überschneiden dürfen, aber Löcher/ Inseln beinhalten können.

¹⁴ Teil einer Datei, der Metainformationen über die ihm folgenden Daten enthält.

¹⁵ Ein Rechteck, das um eine Geometrie gelegt werden kann und dabei minimale Seitenlängen hat.

2D + Messwert (X,Y,M)

Dient dem Darstellen von ebenen Objekten im zweidimensionalen Raum mit der Möglichkeit einen Messwert zu hinterlegen.

- PointM und MultipointM
- PolylineM
- PolygonM

3D + Messwert(X,Y,Z,M)

Geometrietypen verfügen sowohl über eine Lage- als auch über eine Höheninformation. Zudem ist es auch hier möglich einen Messwert zu speichern.

- PointZ und MultipointZ
- PolylineZ
- PolygonZ

Der Index - File speichert Metadaten zu den im Main - File hinterlegten Geometrien. Er besitzt ebenfalls einen Header, welcher mit dem des Main - Files übereinstimmt. Im Anschluss an diese Dateiinformationen, werden der Offset¹⁶ und die Länge des Main - File Records in der gleichen Reihenfolge wie im Main - File gespeichert.

Der Index - File dient somit der Datenverwaltung und gibt Auskunft über den Standort einer Geometrie innerhalb des Main - Files. Dies ist besonders wichtig bei der Darstellung eines bestimmten Features.

Die dritte Komponente der Shape Datei ist das „dBase file“. Es speichert alle zu den Geometrien gehörenden Sachdaten und enthält eine Tabelle, in der pro Objekt jeweils ein Sachdatensatz vorhanden sein muss. Die Verknüpfung zu den Geometrien im Main file erfolgt über die „record number“, die in der dBASE- Datei und im „Main file“ enthalten ist. Zwischen Geometrien und Sachdaten besteht somit eine 1:1 Beziehung.

Aufgrund dieser offenen Struktur hat das Shape Format verschiedene Aufgaben innerhalb des UIS-BW. Vorrangig dient es dem Datenaustausch zwischen den Fachanwendern und dem primären Datenhaltungssystem, der Smallworld

¹⁶ Bezeichnet die Anfangsposition bestimmter Daten innerhalb einer Datei.

Datenbank. Hierzu werden die meist mit einem ESRI Produkt erfassten oder veränderten Geodaten regelmäßig über eine Smallworld/Shape Schnittstelle mit dem Master abgeglichen. Umgekehrt werden die Daten aus der SW - DB wieder in Shape - Files gewandelt und als „IST Zustand“ den Anwendern zur Verfügung gestellt.

Auf diese Weise ist es den Fachreferaten möglich Ihre Aufgaben wie zum Beispiel das Anfertigen von thematischen Karten auf der Grundlage von aktuellen Informationen zu bewältigen.¹⁷

2.2.2 Oracle Datenbank

Der Titel „Oracle Datenbank“ dieses Kapitels besteht aus den Wörtern Oracle und Datenbank, wobei der Begriff „Oracle“ der Name einer Firma ist, die neben verschiedenen anderen Produkten, vor allem Datenverwaltungssysteme also Datenbanken entwickelt. Bei einer Datenbank handelt es sich nach Scott Martin, einen der wichtigsten Entwickler bei Oracle, um „... ein Bündel von Programmen, die Daten manipulieren“. Zu den Daten einer Oracle Datenbank gehören neben den Benutzerdaten die Systemdaten. Beide Datentypen werden physisch in sogenannten Datendateien abgelegt, welche jeweils einem Tablespace zugeordnet sind. Der Tablespace dient der logischen Gruppierung der Daten und muss vor dem Anlegen der Datenobjekte, die im Fall einer Oracle Datenbank meistens Tabellen sind, definiert werden. Veranschaulicht kann man die Datenbank mit einem Aktenschrank vergleichen. In dessen Schubladen (Tablespace) es Akten (Datendateien) gibt und in diesen Akten existieren einzelne Seiten (Tabellen), welche die eigentlichen Informationen beinhalten. Die Verwaltung dieser Daten übernimmt ein **Datenbank Management System (DBMS)**. Im Falle von Oracle handelt es sich hierbei traditionell um ein **Relationales Datenbank Management System (RDBMS)**. In einem solchen System isoliert man die zu erfassenden Informationstypen einer Entität¹⁸, also die Attribute, und identifiziert anschließend die Beziehungen zwischen den

¹⁷ vgl.: ESRI Shapefile Technical Description - ESRI White Paper—July 1998

¹⁸ Entität beschreibt ein wohl unterscheidbares Objekt der realen Welt.

Informationstypen. Das logische Abbild dieser Struktur ist eine Tabelle, die Daten eines bestimmten Objekttyps aufnehmen kann.

Spalten dieser Tabelle sind die Attribute und besitzen jeweils einen bestimmten Wertebereich. Die in den einzelnen Zeilen gespeicherten Objekte sind innerhalb eines Datenmodells über ihre Informationstypen mit anderen Entitäten verbunden. Ein relationales System ist somit datengesteuert, das heißt, dass sich ein Ändern der Daten nicht auf die ursprüngliche Struktur des Datenmodells auswirkt. Das DBMS benutzt zur Manipulation der Daten die in der Aussage von Scott Martin angesprochenen Programme. Auf Oracle bezogen stellen diese Programme einzelne Prozesse dar, welche beim Starten eines Programms zum Zweck der Kommunikation mit dem DBMS ausgelöst werden. Auch hier gibt es wieder eine Unterscheidung in Benutzerprozesse, also Prozesse, die der Anwender auslöst, um Zugriff auf die Datenbank zu erlangen, und den Serverprozessen, welche die Anforderungen der Benutzerprozesse entgegennehmen und zur Erfüllung dieser Anforderungen direkt mit der Datenbank kommunizieren. Ein Beispiel für einen Benutzer- oder auch Clientprozess, ist das Programm SQL*PLUS. Über diese Anwendung ist der Benutzer in der Lage, alle der Datenbank zur Verfügung stehenden Funktionen auszuführen. Dazu gehören Aufgaben wie das Anlegen oder Löschen von Datenbankobjekten (Tabellen, View's, Indizes u.a.) also Struktur beeinflussende Aktionen (vgl. **DDL - Data Definition Language**) und Operationen, welche die Daten an sich betreffen, wie das Abfragen, Einfügen und das Löschen von Informationen (vgl. **DML - Data Manipulation Language**).

Serverprozesse sind Programme wie der **Database Writer (DBWR)**, der die geänderten Daten in die Datendateien zurückschreibt oder der **Log Writer (LGWR)**, der die Daten einer Transaktion als Kopie in sogenannte Redo - Log - Dateien speichert. Redo Log's werden benötigt, wenn beispielsweise die Datenbank nicht richtig beendet wurde und es beim Neustart zu Problemen kommt. Um den letzten Stand der Daten wiederzuerlangen, werden die Redo Log Dateien ausgelesen und die letzten Transaktionen, so sie nicht korrekt ausgeführt wurden, wiederholt.

Mit Hilfe dieser und anderer Programme ist Oracle in der Lage Informationen in großen Mengen effizient zu verwalten und als Datenbasis vielen äußeren

Anwendungen zur Verfügung zu stellen. Weiterhin stellt das DBMS sicher, dass die Daten immer einen eindeutigen Zustand haben. Das ist eine der wichtigsten Eigenschaften des Systems, erlaubt es doch den multiplen Datenzugriff vieler Anwender. Oracle nutzt dafür das Prinzip der kurzen Transaktion. Bearbeitet ein Benutzer ein bestimmtes Objekt, erlaubt das System zwar einen zweiten lesenden Zugriff, verhindert jedoch eine Manipulation der Daten durch Dritte. Eine Transaktion beinhaltet also immer den Datenzugriff, das Bearbeiten der Daten und das Zurückschreiben der Daten in die Datenbank („commit“).

Ein RDBMS wie es von Oracle entwickelt wird, eignet sich für Organisationen mit vielen Anwendern, großen Datenmengen und vielen zeitlich begrenzten Einzeltransaktionen.

Um dieses Spektrum an Datenverwaltungsmechanismen zu erweitern, nimmt Oracle mit der Datenbankversion 8i erstmals objektorientierte Erweiterungen auf. Damit können Ecksteine der objektorientierten Technologie wie Objekte, Klassen, Einkapselungen, Vererbungen u.a. zumindest in Ansätzen, erstmals genutzt werden. Oracle als Datenspeicher gibt Anwendungen, die auf die Daten zugreifen und meistens in einer objektorientierten Sprache entworfen worden sind, die Möglichkeit, in einer einheitlichen objektbasierten Welt zu arbeiten. Die 8i Version wird deshalb als objektrelationale Datenbank bezeichnet. Sie behält das relationale Modell bei und erweitert dieses um Objekttechnologie.¹⁹

Die in der LfU zum Einsatz kommende Oracle Datenbank (8.0.5) liegt verteilt auf zwei Datenservern (Debora1/2, Betriebssystem Linux). Ihre Hauptaufgabe besteht in der Verwaltung, der durch die einzelnen Fachreferate erfassten Sachdaten. Dabei werden die Daten, themenspezifisch, in jeweils eigene Schemata integriert. Neben den Sachdaten befinden sich auch Geometriedaten in der Datenbank. Sie werden ausgehend vom Shape Format in binäre Informationen zerlegt und über ein selbst entwickeltes Modul, den Geodatenmanager, in die Datenbank eingefügt. Bei den Geometrien handelt es sich vor allem um übergeordnete Themen, welche oft Grundlage von umweltbezogenen Auswertungen sind. Das Datenmodell, in welchem die Geometrien verwaltet werden, trägt den Namen „Geodienst“ mit dem

¹⁹ vgl.: Loney/Theriault – „Oracle 8i – Einsteiger Handbuch“

Alias - Namen „Geo“ und ist im Anhang unter Punkt 7.1 ausschnittsweise abgebildet. Es enthält Geometrien und Sachdaten, die entweder schon bei der Geometrieerfassung erhoben wurden oder aus den oben erwähnten Fachschemata stammen. Letztere werden, um eine redundante Datenhaltung zu vermeiden, über View's ins Datenmodell eingebunden. Eine Objektklasse oder ein Thema wird in diesem Modell durch einen **Fachführungscode (FFC)**, eine dreistellige Zahl, welche die Erfassungsstelle identifiziert, und einen **Objektartencode (OAC)**, eine achtstellige Zahl, definiert. Jedes Objekt als Teilmenge dieser Klasse wird durch eine Objekt_id eindeutig. Mit FFC und OAC erhält man, über eine entsprechende Tabelle „UIS_OBJEKTART“ den Namen der Sachdatentabelle und mit der Objekt_id den gesuchten Record. Die Geometrien befinden sich in eigenen Tabellen, in Spalten vom Datentyp „long raw“. Dieser Datentyp kann die durch den Geodatenmanager erzeugten Binärdaten mit einer maximalen Größe von bis zu 2GB fassen. Der Name der Tabelle, in welcher die Geometrien gespeichert sind, setzt sich zusammen aus dem Alias Namen des Schemas („geo“), dem Wort „geom“, dem FFC und dem OAC (Bsp. GEO_GEOM_0100000001). Die Verbindung zwischen Sachdaten und Geometrien ist somit lose, also disjoint. Die einmal so hinterlegten Daten werden im Einjahreszyklus aktualisiert und im Rahmen des UIS-BW an verschiedene öffentliche Verwaltungen in Form von Datendumps weitergegeben.

Oracle ist ein sehr ausgereiftes Werkzeug zur Speicherung von alphanumerischen Massendaten. Über den Datentyp Long Raw ist zwar eine Speicherung von räumlichen Daten möglich, aber eine effiziente Verwaltung ist im Standard - Oracleschema nicht möglich. So müssen die Daten sowohl beim Einspielen als auch beim Lesen jeweils umgewandelt werden. Eine gute Performance oder gar ein Onlinezugriff sind dadurch nicht realisierbar.

2.2.3 Smallworld

Smallworld ist eines der weitverbreitetsten Geoinformationssysteme und wird innerhalb der LfU vorwiegend als Datenhaltungssystem verwendet. Abbildung 2 zeigt die mehrschichtige Architektur, mit welcher Smallworld in der Lage ist, Objekte der realen Welt zu modellieren und nach dem EVAP Prinzip zu verarbeiten. Die einmal gespeicherten Daten werden dem Anwender interaktiv und blattschnittfrei in einem grafischen Fenster dargestellt.

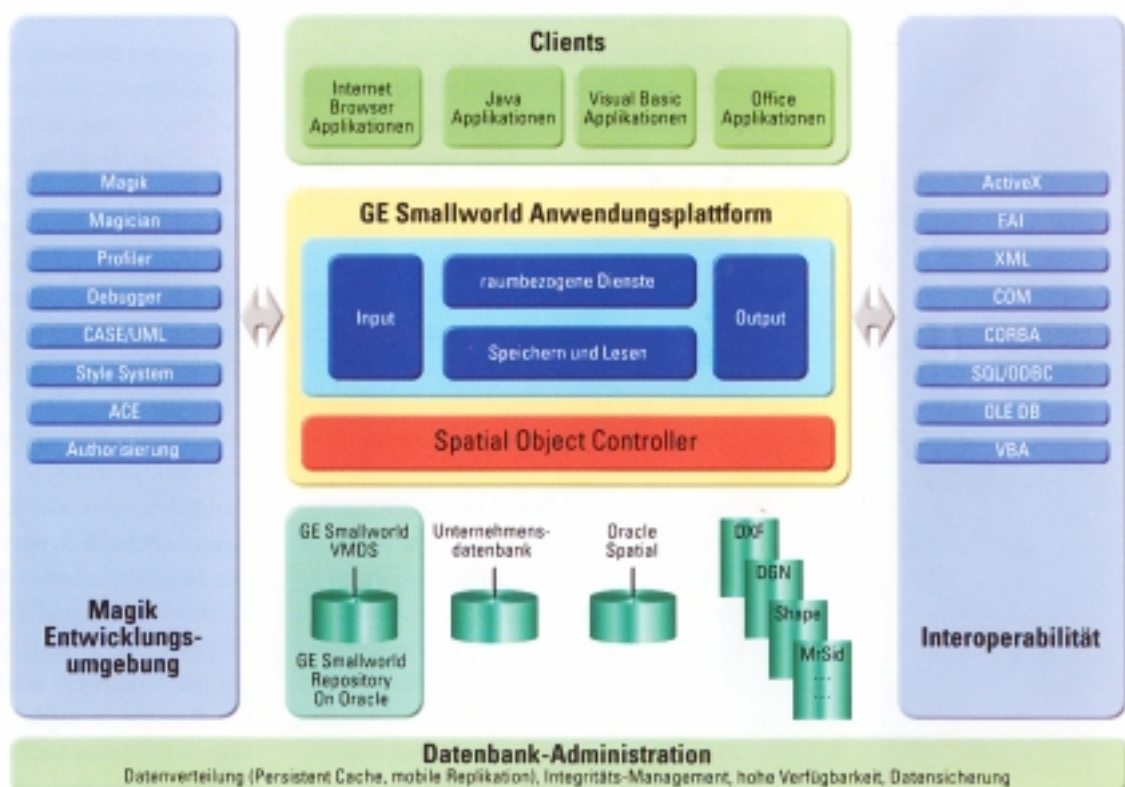


Abbildung 2 : Smallworld - Systemarchitektur

Innerhalb des mehrschichtigen Aufbaus unterscheidet man eine Datenhaltungs-, Applikations- und Präsentationsschicht. Die Datenhaltungsschicht des Smallworld GIS wird durch eine eigene Datenbank realisiert. Sie nutzt intern das relationale Prinzip und ist mit Funktionen zur Versionsverwaltung ausgestattet. Mit einem **V**ersion **M**anaged **D**ata **S**tore (**VMDS**) erhält jeder Anwender eine eigene aktuelle Version der Datenbank. Ein Ändern dieser Version erzeugt eine Alternative der

Datenbank die durch Senden an die Topaltnative allen weiteren Benutzern zugänglich ist.

So ist es dadurch möglich, Objekte zu bearbeiten, ohne das diese für andere Anwender gesperrt werden müssen. Die Daten werden zweckgebunden in eigenständige und lose miteinander verbundene Partitionen und Datenbestände abgelegt. Die wichtigsten Systempartitionen heißen Style, **ACE (Application Configuration Environment)**, Message und **Auth (Authorisation)**. Wobei die Style-Partition Informationen über das Erscheinungsbild von Geometrien enthält (z.B. Linienbreite, Farbe u.s.w.). Die ACE-Partition speichert Daten über Änderungen an Menüs, geladene Komponenten und sichtbare Felder. Die Message – Partition, dient dem Speichern aller Meldungen, Beschriftungen und auch die Lizenzierung ist hier dokumentiert. In der Auth - Partition wird für alle Anwender der Zugriff auf das GIS geregelt. Die Datenbestände liegen je nach Datentyp in einzelnen Datastore Dateien. So enthält die „rwo.ds“ sämtliche RWO`s und Sachdaten, Rasterdaten werden in der „raster.ds“ gespeichert. Die Vektordaten und Metainformationen über dieselben befinden sich in der „gdb.ds“ bzw. in der „dd.ds“.

Die Daten in den einzelnen Tabellen werden beim gis_reinit() mit Magikobjekten verknüpft. Diese Objekte (**Real World Objects (RWO)**) suchen alle relevanten Daten einer Entität aus den einzelnen Datastore Dateien zusammen. Die gespeicherten Daten werden immer nur über Ihren Repräsentanten angesprochen. Auf diese Weise ist es nicht nur möglich neue Verhaltensweisen zu definieren, weiterhin können so neue plattformunabhängige Applikationen (Applikationsschicht) entwickelt werden. Die Applikationen laufen dann auf einer plattformabhängigen virtuellen Maschine, in C geschrieben, und können für spezielle Aufgaben (z.B. Flächenverschnidungen) externe Programme (ACP/ Alien Co Processor) anstoßen.

Dies ist erforderlich, um rechenintensive Prozeduren (z.B. Topologien) zu beschleunigen oder Windows Systemaufrufe in plattformabhängige Anweisungen zu wandeln.

Zur Modellierung von eigenen Datenmodellen hat der Anwender die Möglichkeit, das **CASE** - Tool (**C**omputer **A**ided **S**oftware **E**ngineering) zu benutzen. Dabei handelt es sich eine grafische Oberfläche, die es erlaubt, neue Tabellen mit entsprechenden Beziehungen ohne Kenntnis der Programmiersprache Magik festzulegen.

Nach außen hin können die Smallworlddaten über verschiedene Schnittstellen (COM, CORBA, VBA u.a.) einer großen Bandbreite von verschiedenen, ggf. auch nichtgrafischen Anwendungen wie Microsoft Excel zugänglich gemacht werden.²⁰

Das an der Landesanstalt für Umweltschutz eingesetzte Smallworld GIS entspricht der Version 3.1 und beinhaltet das Service Pack 3. Über einen Applikationsserver sind bis zu 20 Anwender - PCs (Windows NT 4.0 SP5 oder Windows 2000 SP1) mit einem „HP Datenserver“ unter Linux verbunden. Die genannten Basis- und Fachdaten werden vollständig von Smallworld verwaltet. Das Smallworld GIS wird hauptsächlich von Versorgern eingesetzt, welche oft auf der Grundlage von linienhaften Objekten arbeiten. Bei den von der LfU verwalteten Daten handelt es sich zumeist um Flächen. Speziell hierfür gibt es verschiedene Eigenentwicklungen, die den Bedürfnissen der Fachanwender entsprechen.

Aufgrund der relativ hohen Kosten eines Smallworld Arbeitsplatzes werden nur qualitativ hochwertige Themen, z.B. Fließgewässer Baden - Württemberg

1: 50000, mit dem System direkt erfasst. Eines der wesentlichen Vorteile des GIS ist die Fähigkeit zur Integration von Rasterdaten. Entsprechende, oben genannte Daten liegen so Geo- Referenziert und nahtlos in der Smallworld - Datenbank.

Innerhalb des RIPS- Pools benötigen die meisten Fachanwender nur Zugriff auf die Sachdaten. Diese verändern sich durch die Arbeit der einzelnen Referate kontinuierlich.

Ein Zugriff auf die Smallworld eigenen Sachdaten ausgehend von jeder Fachanwendung ist zwar generell möglich, jedoch ist der Entwicklungsaufwand hierfür erheblich.

²⁰ vgl.: Smallworld 3 Dokumentation Version 3.1 (0) SP1 – Smallworld 3 Overview

Für den Datenzugriff auf Oracle gibt es jedoch bestehende frei zugängliche Mechanismen (z.B. **ActiveX Data Objects (ADO)**²¹). Die betroffenen Sachdaten befinden sich deshalb in einer Oracle Datenbank.

Smallworld ist in der Lage über die bestehende Klasse „extdb_rwo“ ein Objekt zu bilden, welches Sachdaten aus einer Oracle DB und Geometrien aus einer Smallworld DB beinhaltet. Bei der Auswahl eines Objektes in Smallworld werden die aktuellen Sachdaten aus der Oracle Datenbank gelesen.

Themen, welche häufig durch die Fachanwender bearbeitet werden, sind also in einer Oracle Datenbank abgelegt und mit den entsprechenden Smallworld Geometrien verknüpft.

Für den Datenaustausch zwischen den verschiedenen Systemen existiert eine Shape - Schnittstelle. Daten die in der Smallworld Datenbank liegen, können Themenweise selektiert und mit oder ohne Sachdaten entladen werden. Die einzelnen Objektattribute können im Gegensatz zu Topologien oder Darstellungen mit exportiert werden. Die Schnittstelle ermöglicht einen Datenaustausch in beide Richtungen, so dass die durch die Fachreferate erfassten Daten nach Smallworld importiert werden können.

2.2.3.1 Magik – Die Smallworld Entwicklungsumgebung

Smallworld Magik ist eine Programmiersprache, mit der sich große Systeme interaktiv beeinflussen lassen. Sie ist ein Hybrid bestehend aus prozeduralen und objektorientierten Elementen. Die objektorientierte Struktur ermöglicht es dem Anwender, eigene Exemplare zu definieren, und sich Funktionen, wie Polymorphismus, mehrfach Vererbung, Abhängigkeiten und Anderen, zu bedienen. Magik enthält weiterhin Standardbibliotheken von Objektklassen, die das Arbeiten mit externen Datenbanken und den versionsverwalteten Smallworld eigenen Datastore genauso ermöglichen, wie auch die direkte Steuerung der graphischen Oberfläche. Die meisten interaktiven Systeme sind mit nicht interaktiver Software z.B. in C geschrieben. Der Anwender hat hier keine

²¹ ADO stellt Objekte und Methoden zur Verfügung um auf fast alle geläufigen Datenbanken zugreifen zu können.

Möglichkeit, das System, während es arbeitet, zu modifizieren. Diese Systeme verfügen im Rahmen einer Anwendung über ein vorher festgelegtes Budget an Funktionen, welches nur durch Updates oder ähnliche Mechanismen verändert werden kann. Mit Magik kann der Anwender sofort auf das System, die Daten und die graphische Oberfläche zugreifen, ferner ist es möglich, eigene Modifikationen oder auch Applikationen zu definieren.

Zum Starten von Magik benötigt man ein Magik- „Executable“ und ein Image. Magik läuft auf einer virtuellen Maschine und das Executable ist ein Programm, das mit dieser Maschine verbunden ist. Das Image enthält sämtliche Objekte und bildet damit die Magikumgebung. Innerhalb dieser Umgebung gibt es einen Compiler, der den Magikcode in Befehle für die virtuelle Maschine übersetzt. Damit ist Magik plattformunabhängig und kann sowohl unter Windows als auch unter Unix eingesetzt werden.

In der objektorientierten Welt von Magik gibt es Objekte die auf verschiedene Nachrichten reagieren. Jede dieser Nachricht enthält den Verweis auf eine objektspezifische Methode, die in der sogenannten „method_table“ gespeichert ist. Objekte, mit gleichen Eigenschaften und Methoden, werden in einer Klasse zusammengefasst. Im Unterschied zu anderen objektorientierten Sprachen sind Magikklassen selber keine Objekte, sie dienen lediglich als Gruppendifinition. Spezielle Objekte einer Klasse bezeichnet man als Instanzen. So kann z.B. eine bestimmte Person „Schmitt“ eine Instanz der Klasse „Person“ sein. Jede Klasse hat eine Oberinstanz, von der sich alle anderen Objekte ableiten. Diese schablonenähnliche Instanz wird als Exemplar bezeichnet. Alle Magikobjekte lassen sich in drei Kategorien einteilen:

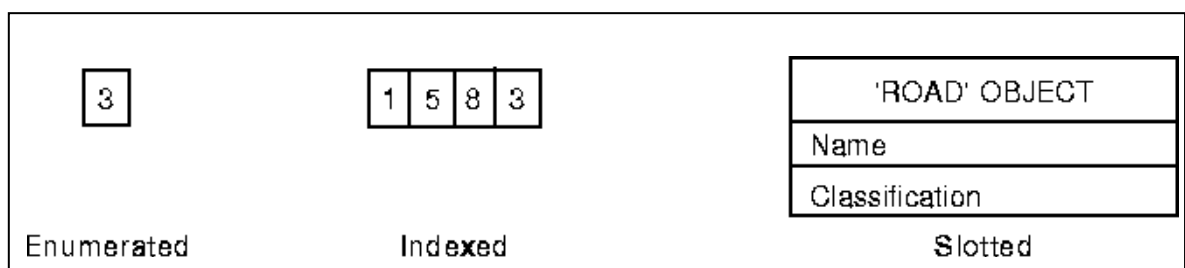


Abbildung 3 : Magikobjekte

Objekte vom Typ „Enumerated“ sind eindeutig, das heißt, sie besitzen nur einen Zustand. Hierzu zählen Datentypen wie Integer, Character oder Boolean. „Indexed“- Objekte setzen sich aus einer Menge von bestimmten Teilobjekten zusammen. Der klassische Vertreter ist z.B. der Datentyp String. Ein Wort besteht aus mehreren Teilelementen (Buchstaben) des Typs Character. Der dritte Typ enthält verschiedene Variablen (Instanz Variablen), in denen Daten gespeichert werden. Die Attribute werden auf jedes Objekt dieser Klasse weitergegeben, nicht aber die Daten. Dieses Verhalten bezeichnet man auch als Einkapselung.

Magik bietet natürlich auch die Standardfunktionalitäten einer Programmiersprache. Dazu zählt das Anlegen von Variablen, das Vergleichen mit Hilfe von verschiedenen Operatoren, Schleifen und Anderes.²²

²² vgl.: Smallworld 3 Dokumentation Version 3.1 (0) SP1 – Magik

3. Einführung in Oracle Spatial

Oracle Spatial ist eine Erweiterung zur Oracle8i Datenbank (ab Release 8.0.5) und bietet das erweiterte Management von räumlichen Daten. Oracle Spatial nutzt dafür ein integriertes Set von Funktionen und Prozeduren für das Sichern, Verwalten und Anbieten von räumlichen Daten, die sich über Analysen auswerten lassen.

Durch folgende Komponenten werden diese Leistungsmerkmale verwirklicht.

- Das Schema (MDSYS) beschreibt das Speichern, die Syntax und die Semantik der unterstützten geometrischen Datentypen.
- Räumlicher Index, der den Zugriff auf die räumlichen Daten beschleunigt.
- Verschiedene Funktionen und Operatoren
- Administrative Werkzeuge

3.1 Allgemein

Das Oracle Spatial Schema unterstützt zwei Modelle für die Repräsentation der Geometrien. Zum einen den objekt-relationalen Ansatz, welcher eine Tabelle definiert, in der es eine Spalte vom Typ „MDSYS.SDO_GEOMETRY“ gibt und jede Zeile eine Geometrie verkörpert, und zum anderen den relationalen Ansatz, der die hierarchische Struktur der Geometrien in jeweils vier zusammenhängenden Tabellen abbildet.

Beide Modelle greifen die durch das OpenGIS- Konsortium definierte ODBC/SQL Spezifikation auf, in der sowohl „SQL mit Geometrietypen“ (objekt- relational) als auch „numerisches SQL“(relational) zum Abbilden der räumlichen Daten in einer Datenbank zulässig sind.

Das Datenmodell ist hierarchisch aufgebaut und gliedert raumbezogene Objekte in Elemente, Geometrien und Layer. Geometrien bestehen aus einem oder mehreren Elementen, die durch geometrische Primitiven definiert sind, also Punkt, Linie oder Fläche. Sie repräsentieren das räumliche Merkmal eines Objektes, das jeweils durch eine eindeutige Geometrie_id gekennzeichnet ist. Ein Layer fasst Geometrien zusammen, die dieselben Attribute besitzen.

Das heißt, dass Entitäten mit den gleichen alphanumerischen Eigenschaften ihre Geometrien in einem gemeinsamen Layer ablegen. Das Schema umfasst weiterhin die Möglichkeit, räumliche Indizes für die gespeicherten Geodaten anzulegen.

Ein räumlicher Index ist, wie jeder andere Index, ein Mechanismus mit dem sich das Suchen nach Daten aufgrund eines verringerten und strukturierten Datenaufkommens innerhalb eines Indexes verkürzt. Der Unterschied besteht aber darin, dass die Grundlage dieses Indexes räumliche Kriterien sind. Diese werden gebraucht, um sowohl Gebiete, die sich mit einem gegebenen Punkt oder einer Fläche („area-of-interest“) überlappen, zu finden (window query), als auch Beziehungen zwischen zwei, sich unterscheidenden Geometrietypen, aufzuzeigen (spatial join). Oracle Spatial unterstützt zwei Indexarten Quad-Tree und R-Tree für Geodaten. Der R-Tree Index umschließt jede Geometrie durch ein Rechteck mit minimaler Größe (**minimum bounding rectangle (MBR)**). In einem Layer, welcher mehrere Geometrien besitzt, werden die durch die Indexbildung entstandenen MBR`s hierarchisch angeordnet. Gespeichert wird dieser Index in räumlichen Indextabellen (SDO_INDEX_TABLE). Um das gleichzeitige „updaten“ des Index durch verschiedene Anwender sicherzustellen, existiert ein Sequenzgenerator (SDO_RTREE_SEQ_NAME), welcher für jeden R-Tree Index einen eindeutigen Schlüssel generiert. Abbildung 4 zeigt die Zerlegung der Objekte in Rechtecke und stellt den so angelegten Index als Baumdiagramm dar.

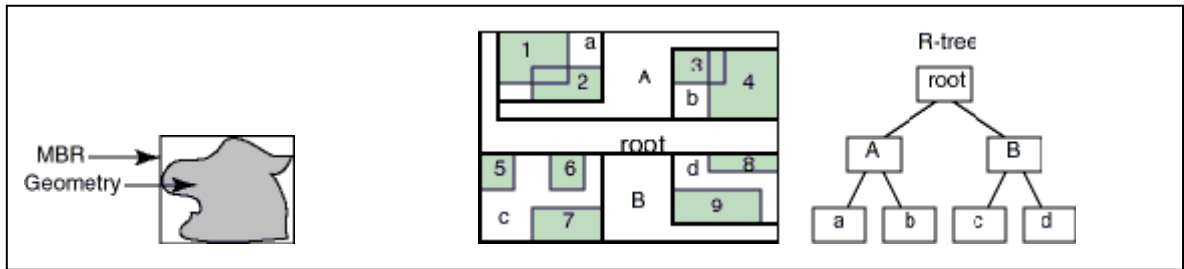


Abbildung 4 : R- Tree Index

Der Quad - Tree Index ist ein linearer Index, bei dem das Koordinatensystem (für einen Layer der alle Geometrien umfasst) durch einen Algorithmus zerlegt wird (tessellation). Die Unterteilung findet in mehreren Schritten, je nach Einstellungen, statt. Dabei werden die einzelnen Quadranten des Koordinatensystems durch verschieben der Achsen in jeweils vier rechteckige, gleichmäßige Stücke zerlegt. Für die Zerlegung gibt es zwei Vorgehensweisen. Zum einen wird über die zu indizierenden Objekte ein Raster aus Rechtecken gleicher Größe gelegt (fixed size).

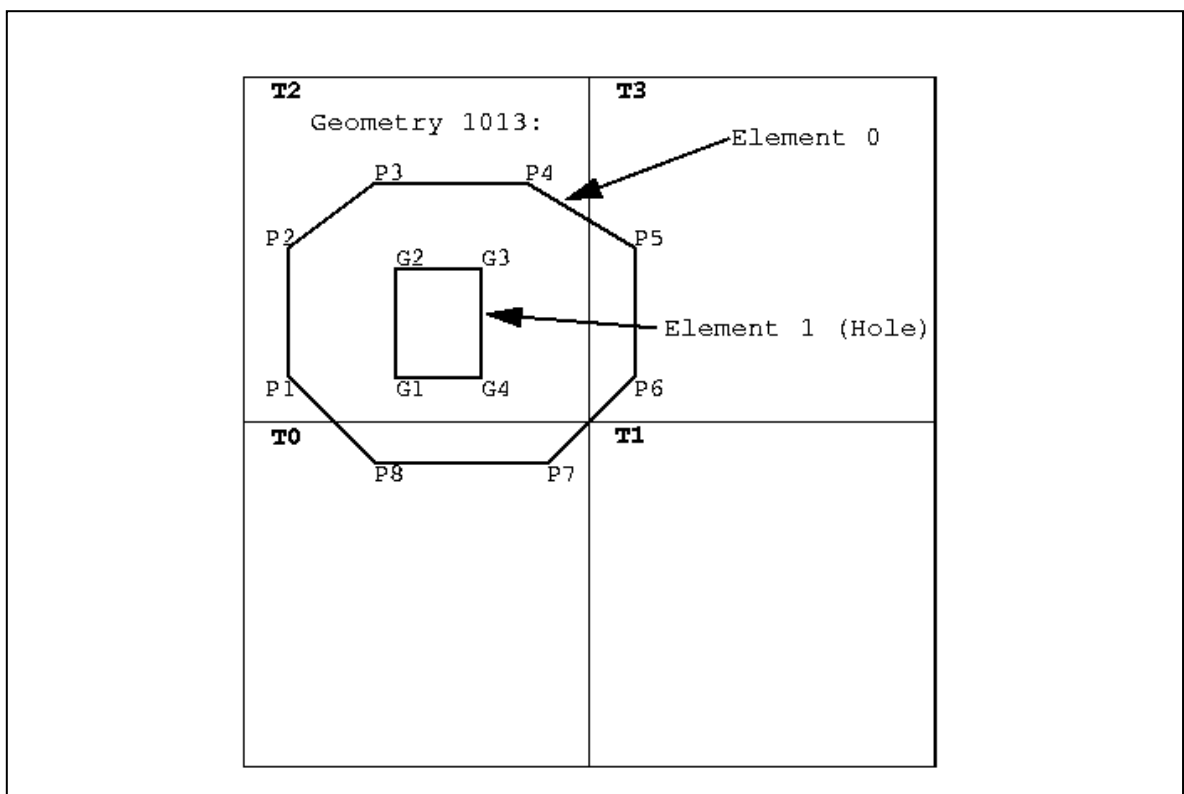


Abbildung 5 : Quad- Tree (fixed size)

Eine entsprechende Index – Tabelle für das in Tabelle 6 dargestellte Objekt hätte folgendes Aussehen.

SDO_GID <number>	SDO_CODE <raw>
1013	T0
1013	T2
1013	T3

Tabelle 5 : SDO_Index Tabelle

Bei der zweiten Variante besteht die Möglichkeit ein Raster, bestehend aus gleichmäßigen Rechtecken, zu erzeugen und innerhalb dieser Rechtecke ein engmaschigeres Raster zu entwickeln (hybrid size).

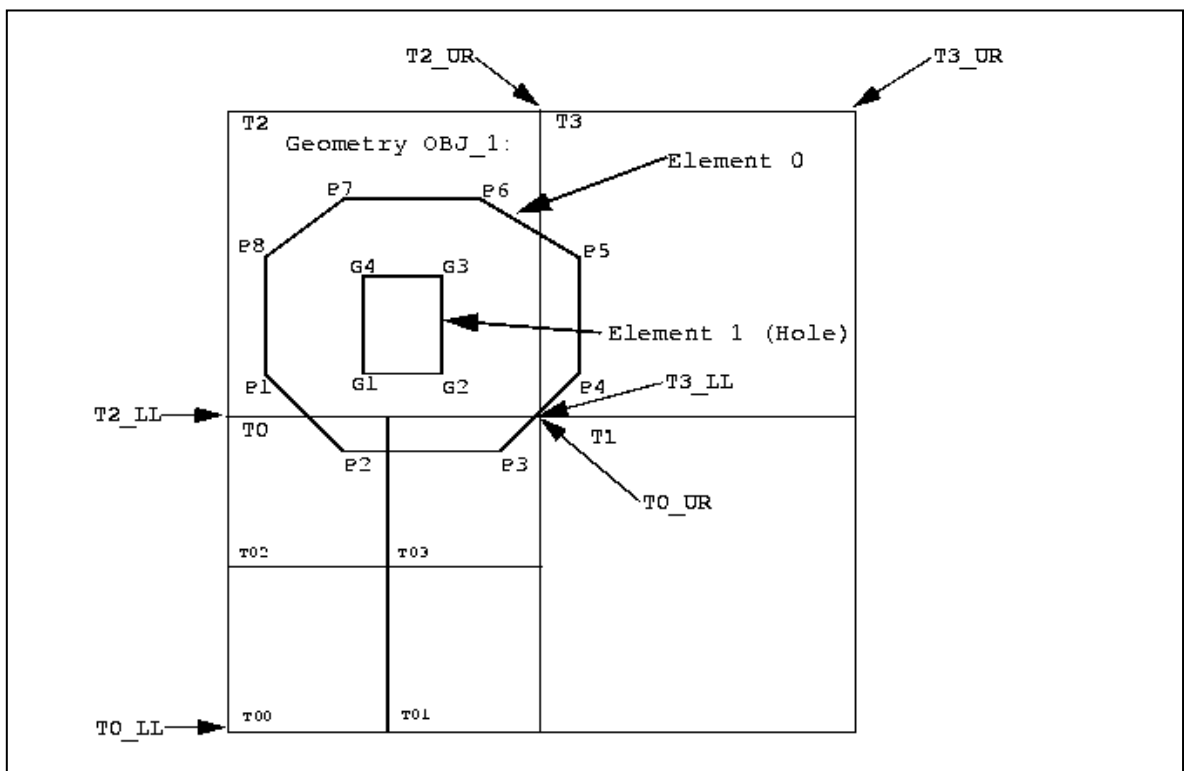


Abbildung 6 : Quad- Tree (hybrid size)

Die Werte der Index- Tabelle würden sich folgendermaßen zusammensetzen.

SDO_ROWID <RAW>	SDO_CODE <RAW>	SDO_MAXCODE <RAW>	SDO_GROUPCODE <RAW>	SDO_META <RAW>
GID_OBJ_1	T02	<binary data>	T0	<binary data>
GID_OBJ_1	T03	<binary data>	T0	<binary data>
GID_OBJ_1	T2	<binary data>	T2	<binary data>
GID_OBJ_1	T3	<binary data>	T3	<binary data>

Tabelle 6 : SDO_INDEX Tabelle

Der Quadrant T0 ist im Beispiel ein zweites Mal unterteilt worden. Für die Indextabelle bedeutet dies, dass sowohl die Rechtecke T02 und T03 als auch die Rechtecke der übergeordneten Struktur T0, T2 und T3 gespeichert werden müssen.

Beide Indextypen können getrennt oder gemeinsam eingesetzt werden, je nach Daten und Anwendung. Die Bildung eines R- Tree Index ist zwar einfacher, jedoch hat man auch keinen Einfluss auf die Gestaltung. Im Gegensatz zum Quad- Tree, wo die Indizierung in Stufen abläuft, welche man selbst definieren kann. Dieses Abstufen benötigt wiederum mehr Speicher und auch Pflege als für den R- Tree Index, allerdings sorgt diese Struktur auch dafür, das viele gleichzeitige Updates der Geometrietabellen möglich sind.

Oracle verwendet in beiden Modellen, relational und objekt-relational, ein „two-tier“ (- zwei Schichten) Modell um sowohl räumliche, als auch verknüpfende Abfragen durchzuführen. Das Ergebnis beider Schritte ist ein eindeutiges und exaktes Resultat. Die einzelnen Schichten werden jeweils durch einen Filter implementiert, welcher die räumlichen Beziehungen zwischen den Entitäten einer Oracle Spatial Datenbank nutzt. Diese räumlichen Beziehungen basieren auf den Lageinformationen der einzelnen Geometrien und nutzen ihre Eigenschaften hinsichtlich Topologie und Entfernungen untereinander.

Ein Filter ist somit der Oberbegriff für alle Operatoren und Funktionen, welche die geometrischen Daten zum Zwecke der räumlichen Analyse strukturieren.

- Der erste Filter ist der „primary filter“. Er sucht in der gesamten Datenbank nach Kandidaten, welche ein flaches Abfragekriterium erfüllen, und liefert so einen Set von Daten, in denen sich das exakte Ergebnis befindet.
- Der zweite Filter sucht aus dem Datensatz vom ersten Filter die Kandidaten heraus, welche die Abfrage vollständig und exakt erfüllen.

Das Schema zeigt den Ablauf einer Suche

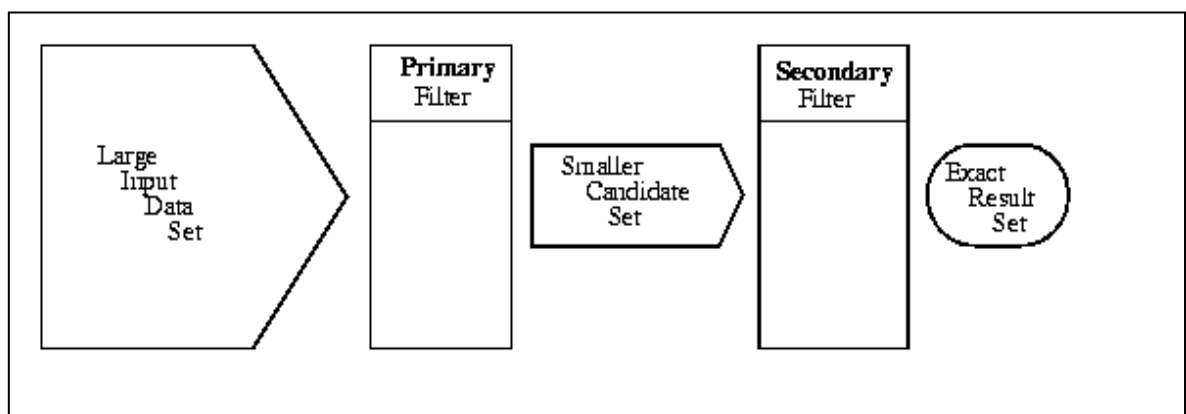


Abbildung 7 : Datenfilterung

Bei „spatial query’s“ ist immer ein Teil der Abfrage fest definiert, z. B.: Zeige alle Straßen die innerhalb einer Gemeinde X liegen. Bei „spatial join“ - Abfragen werden alle Geometrien eines Layer’s mit denen eines anderen Layer’s verglichen. Beide Teile der Anfrage beziehen sich nicht auf ein bestimmtes Objekt. z.B.: Zeige alle Strassen, die Naturschutzgebiete durchqueren. Beide Abfrageschemen nutzen primäre und sekundäre Filter, die jedoch für den einen oder anderen Abfragemodus festgelegt werden müssen. Wichtig für das Verstehen der Arbeitsweise solcher Filter ist das mathematische Konzept, welches allen zugrunde liegt.

Ausgehend davon das jedes geometrische Objekt einen bestimmten Bereich umschließt, also eine Fläche belegt (interior), einen Umring (boundary) hat und es

ein Umfeld um das Objekt gibt (exterior), lassen sich verschiedene topologische Regeln definieren. „Disjoint“ beschreibt zum Beispiel zwei Objekte, welche sich weder überlappen noch in ihren Grenzen berühren. Das bedeutet, dass sich das jeweilige andere Objekt gänzlich im „äußeren Raum“ des Ersten befindet. „Touch“ würde bedeuten, dass sich die Objekte zwar nicht überlappen, jedoch die Grenze des einen Objektes partiell auch die Grenze des anderen Objektes ist.

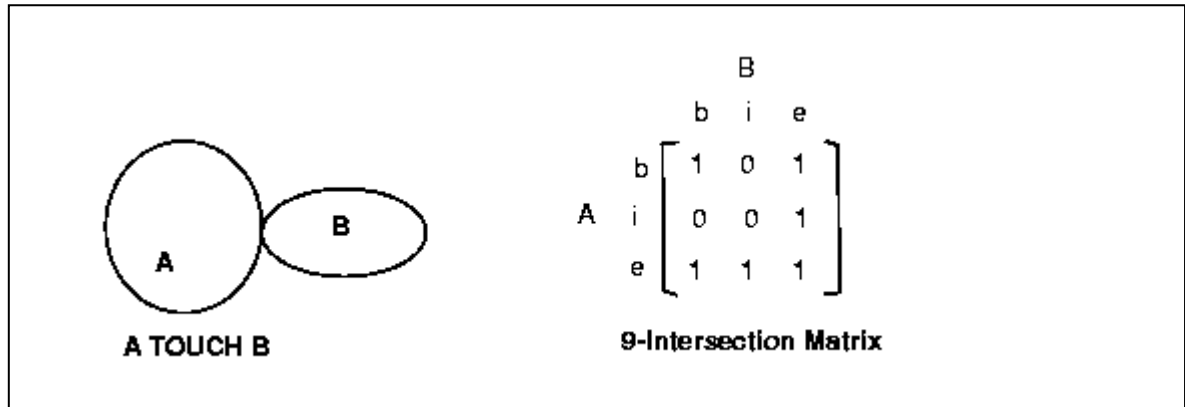


Abbildung 8 : Räumliche Beziehung zwischen geometrischen Objekten

Alle diese Regeln lassen sich für zwei Objekte in Form von Matrizen ausdrücken.²³ Die oben dargestellte Grafik beschreibt genau die „touch“ – Regel: so steht die 1 im Feld $M_{1,1}$ für ein interagieren der Grenze (boundary b) von Objekt A zu der von Objekt B usw. .

3.2 Das objekt - relationale Modell

Das objekt- relationale Modell umfasst mehrere Objekttypen, einen Index und Operatoren auf diese Typen. Geometrien werden als Objekte in einer Spalte des Typs SDO_GEOMETRY gespeichert. Der räumliche Index kann durch den Gebrauch von Data Definition Language -DDL (z.B. CREATE, ALTER, DROP) und Data Manipulation Language -DML (z.B. INSERT, UPDATE, DELETE) angelegt und bearbeitet werden. Das Vorgehen soll anhand eines Beispiels, indem das Anlegen einer Tabelle, das Einfügen von Daten und die Abfragesyntax durchgeführt wird, näher erläutert werden.

²³ vgl.: Professor Max Egenhofer , University of Maine, Orono

```
CREATE TABLE regbezirk (  
  nummer_id NUMBER PRIMARY KEY,  
  name VARCHAR2(32), shape MDSYS.SDO_GEOMETRY);
```

Quellcode 1 : Anlegen einer Tabelle auf Oracle mittels SQL

Zunächst wird eine Tabelle „regbezirk“ angelegt, in der es drei Spalten gibt. Die erste ist der primäre Schlüssel (nummer_id), die zweite Spalte enthält den Namen und in der Letzten (shape) stehen direkt die Geometriedaten zu diesem Objekt. Das Füllen der Tabelle folgt in einem zweiten Schritt.

```
INSERT INTO regbezirk VALUES(  
  1,'stuttgart', MDSYS.SDO_GEOMETRY(2003, NULL, NULL,  
  MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1, 11,2003,1),  
  MDSYS.SDO_ORDINATE_ARRAY(  
  30000,120000,60000,120000,60000,150000,30000,150000,30000,120000,  
  35000,125000,35000,130000,36000,130000,36000,125000,35000,125000)));
```

Quellcode 2 : Einfügen von Daten mittels SQL

Damit wird die erste Zeile dieser Tabelle mit der nummer_id =1, dem Namen = stuttgart und der Geometrie gefüllt. Diese muss vom Datentyp SDO_GEOMETRY sein, der sich wie folgt aufbaut:

Das erste Argument (2003) gehört zum Parameter SDO_GTYPE und definiert den Geometriotyp. Diese Typen entsprechen im Namen und der Semantik denen durch das OGIS – Konsortium spezifizierten. Angewendet auf das Beispiel 2003 ist der Geometriotyp Polygon, wobei die 2 zu Beginn für die Dimension steht (2D).

Das zweite Argument steht für SDO_SRID. Es kann verwendet werden, um diese Geometrie auf ein bestimmtes Koordinatensystem abzubilden. Alle gültigen Systeme werden in der MDSYS-CS_SRS Tabelle beschrieben und müssen bei Verwendung in die USER_SDO_GEOM_METADATA eingefügt werden. Ist dieser Wert „null“, wird kein spezielles Koordinatensystem für die Abbildung verwendet.

Alle Geometrien in einer Spalte müssen auf dem selben Koordinatensystem beruhen. Der nächste Parameter ist SDO_POINT, er dient dem Speichern von Punktelementen. Ist er definiert und die nachfolgenden Elemente sind null, so speichert er die X,Y und Z Koordinate als Punktobjekt in der DB. SDO_ELEM_INFO_ARRAY(1,1003,3,11,2003,1) dient der Interpretation der in SDO_ORDINATES gespeicherten Ordinaten. Zu einem Element gehören immer drei Werte.

Die Zahlen „1“ und „11“ geben den Offset der Koordinaten an, also welche Ordinate ist die erste für dieses Objekt (SDO_STARTING_OFFSET). SDO_ETYPE definiert den Elementtyp. So steht 1003 für einen äußeren (1 von 1003) Polygonring (3 von 1003). Das zweite Element definiert sich als innerer (2 von 2003) Polygonring (3 von 2003) und ist ein Loch innerhalb des ersten Elementes.

Die jeweils letzte Zahl „3“ ist ein Argument des SDO_INTERPRETATION Parameters. Er legt die Ausprägung des in SDO_ETYPE definierten Elementes fest. SDO_ORDINATES enthält alle Koordinaten dieser Geometrie in der Form X,Y,X,Y... .

Im nächsten Schritt muss die USER_SDO_GEOM_METADATA Tabelle aktualisiert werden, um im Anschluss einen Index definieren zu können. Die Tabelle speichert dabei den Tabellennamen, den Namen der Geometriespalte und definiert ein Rechteck, in dem sich alle Geometrien des Layer's befinden müssen.

```
INSERT INTO USER_SDO_GEOM_METADATA
VALUES (
'regbezirk',
'SHAPE',
MDSYS.SDO_DIM_ARRAY(
MDSYS.SDO_DIM_ELEMENT('X', 0, 900000000, 0.005),
MDSYS.SDO_DIM_ELEMENT('Y', 0, 900000000, 0.005)
),
NULL
);
```

Quellcode 3 : Einfügen von Metainformationen

Als letzter Schritt wird der Index, jeweils nur einmal pro Tabelle, angelegt.

```
CREATE INDEX reg_idx
ON regbezirk(shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('SDO_LEVEL = 8');
```

Quellcode 4 : Anlegen eines Index mittels SQL

Hier handelt es sich um den Quad- Tree Index mit einem SDO_LEVEL von 8 . Ohne den Parameter SDO_LEVEL würde automatisch ein R- Tree erstellt werden. Zusätzlich zu den im relationalen Modell vorgestellten Geometrietypen können beim objekt- relationalen Ansatz folgende Typen verwendet werden, die nicht Bestandteil der „Simple Feature“ Spezifikation sind:

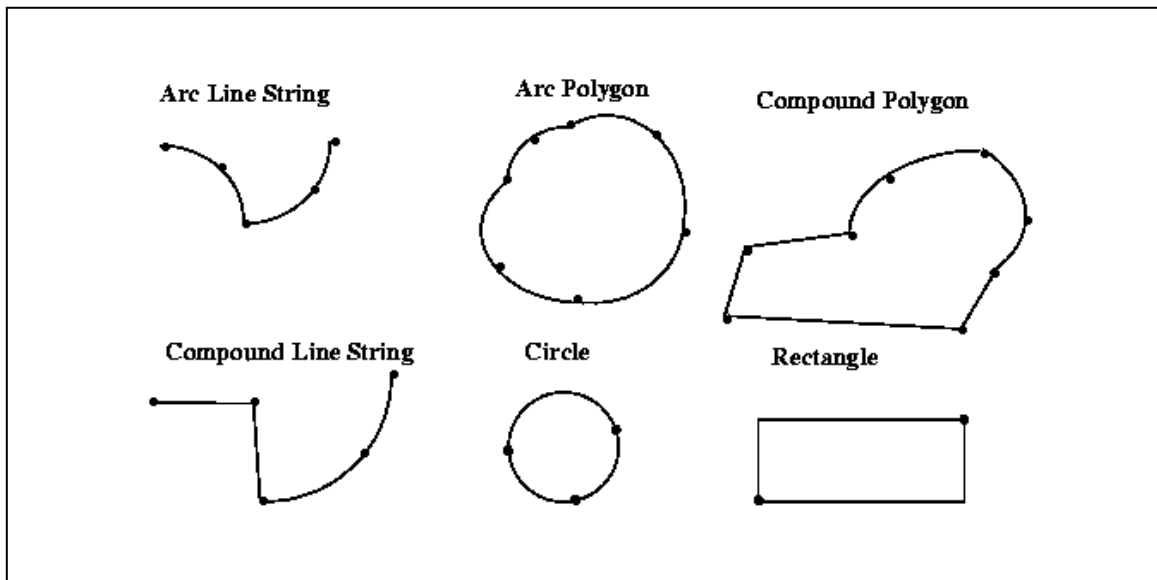


Abbildung 9 : Zusätzliche Geometrietypen des objekt- relationalen Schemas

Für das objekt-relationale Schema können verschiedene Operatoren genutzt werden.

Ein Beispiel für einen Operator ist der „SDO_WITHIN_DISTANCE“ - Filter. Mit diesem Filter können Objekte gesucht werden, die sich in einer bestimmten euklidischen Entfernung zu einem gegebenen Objekt befinden. Der Abfragetyp spielt dabei keine Rolle.

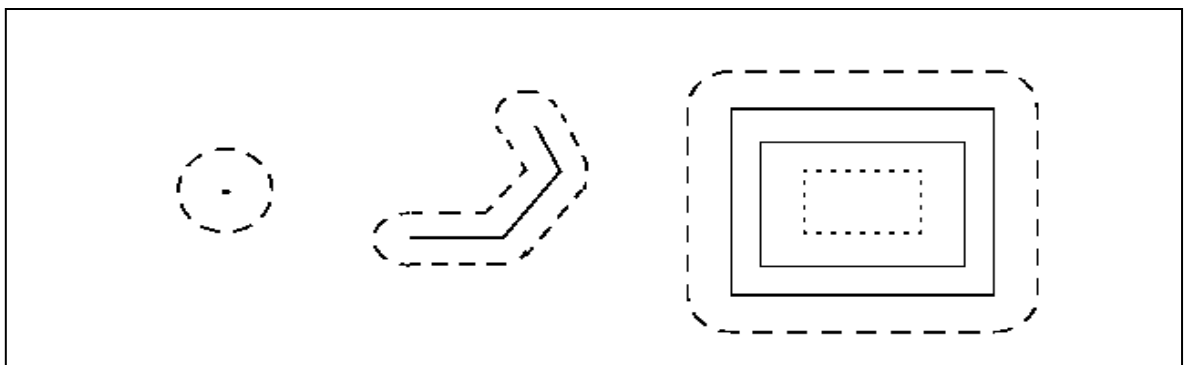


Abbildung 10 : Bufferbildung bei geometrischen Objekten

Beispiel mit einer Entfernung von 10 Längeneinheiten wobei „:aGeom“ die Variable für eine entsprechende Referenzgeometrie ist.

```
SELECT A.GID
FROM POLYGONS A
WHERE
SDO_WITHIN_DISTANCE(A.Geometry, :aGeom, 'distance = 10') ='TRUE';
```

Quellcode 5 : Bufferbildung mittels SQL

Neben den räumlichen Operatoren gibt es noch verschiedene Funktionen, welche man auf das Geometrieobjekt in der Form SDO_GEOMETRY.<FUNKTION(...)> anwenden kann.

Diese Funktionen können verschiedenen Kategorien zugewiesen werden.

- Testen auf verschiedene Beziehungen zwischen zwei Objekten und ausgeben von true oder false : RELATE, WITHIN_DISTANCE
- Überprüfung auf Gültigkeit: VALIDATE_GEOMETRIE, VALIDATE_LAYER
- Funktionen, die nur auf ein Objekt anwendbar sind: SDO_AREA, SDO_BUFFER, SDO_CENTROID, SDO_CONVEXHULL, SDO_LENGTH, SDO_POINTONSURFACE
- Funktionen, die auf zwei Objekte anwendbar sind: SDO_DISTANCE, SDO_DIFFERENCE, SDO_INTERSECTION, SDO_UNION, SDO_XOR

3.3 Das relationale Modell

Im relationalen Ansatz entwirft der Anwender eine Tabellenstruktur, in welcher die Sachdaten und ein Verweis auf die zugehörigen Geometrien (GID) gespeichert werden. Weiterhin bedarf es eines primären Schlüssels, der durch ein oder mehrere Sachdatenfelder definiert werden kann. Die GID ist dabei der Zugang zu den Geometrietabellen. Im relationalen Modell werden die Geometrien in Layer zusammengefasst, die durch vier Tabellen repräsentiert werden. Anhand eines Beispiels werden Struktur und Argumente der einzelnen Tabellen erklärt.

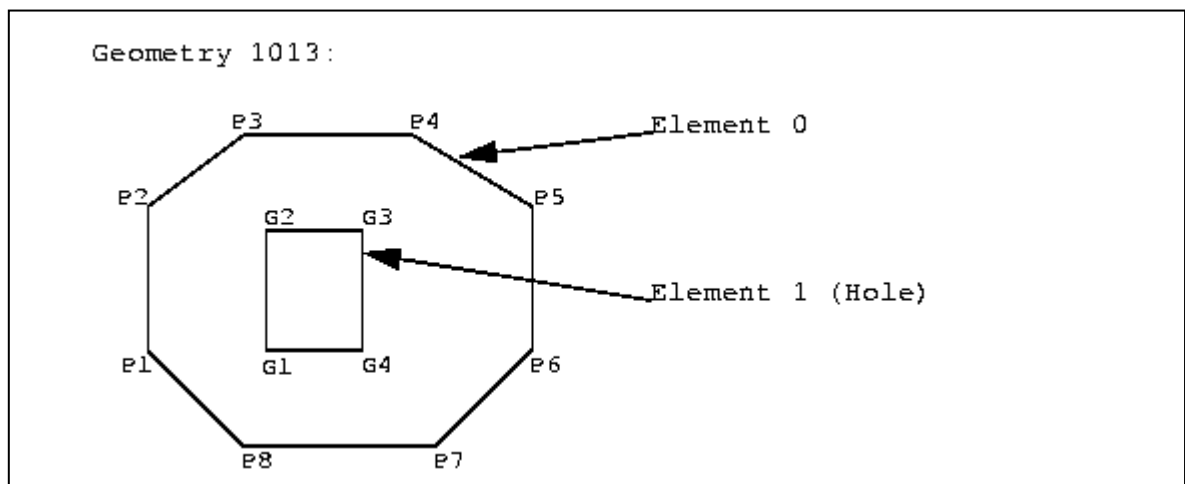


Abbildung 11 : Polygon mit Loch

Die oben dargestellte Geometrie entspricht dem Geometriotyp „complex polygon“. Dieser Typ bildet durch sein äußeres Polygon eine Fläche, welche durch das Rechteck ausgehöhlt wird. Nachfolgend werden die einzelnen Tabellen, die diese Geometrie beschreiben, näher erläutert.

a) <layername>_SDOLAYER

SDO_ORDCNT (number)
4

Tabelle 7 : SDO_LAYER

Die SDOLAYER Tabelle definiert die Form der SDOGEOM Tabelle, in welcher die Geometrien gespeichert sind. Es handelt sich also um Metadaten, welche die eigentliche Datenstruktur beschreiben. Die Spalte SDO_ORDCNT gibt Auskunft über die Anzahl der in einer Zeile gespeicherten Daten (ordinates).

Weitere Parameter gehen auf die Mosaikbildung und auf das zugrunde liegende Koordinatensystem ein.

b) <layername>_SDODIM

SDO_DIMNUM (number)	SDO_LB (number)	SDO_UB (number)	SDO_TOLERANCE (number)	SDO_DIMNAME (varchar)
1	0	100	.05	X axis
2	0	100	.05	Y axis

Tabelle 8 : SDODIM

Diese Tabelle definiert die Dimension der Geometrien. So stehen in der ersten Spalte die jeweiligen Dimensionen beginnend mit eins und einem Inkrement von ebenfalls eins. SDO_LB und SDO_UB geben jeweils die obere und die untere Grenze der Dimensionen an. In diesem Fall also 0 als untere und 100 als obere Grenze. Das Argument von SDO_TOLERANCE ist ein Wert für die Entfernung, die ein Element von einem anderen mindestens haben muss, um als eigenes Element abgelegt zu werden. SDO_DIMNAME ist die jeweilige Beschreibung einer Dimension.

c) <layername>_SDOGEOM

SDO_ GID (number)	SDO_ ESEQ (number)	SDO_ ETYPE (number)	SDO_ SEQ (number)	SDO_ X1 (number)	SDO_ Y1 (number)	SDO_ X2 (number)	SDO_ Y2 (number)
1013	0	3	0	P1(X)	P1(Y)	P2(X)	P2(Y)
1013	0	3	1	P2(X)	P2(Y)	P3(X)	P3(Y)
1013	0	3	2	P3(X)	P3(Y)	P4(X)	P4(Y)
1013	0	3	3	P4(X)	P4(Y)	P5(X)	P5(Y)
1013	0	3	4	P5(X)	P5(Y)	P6(X)	P6(Y)
1013	0	3	5	P6(X)	P6(Y)	P7(X)	P7(Y)
1013	0	3	6	P7(X)	P7(Y)	P8(X)	P8(Y)
1013	0	3	7	P8(X)	P8(Y)	P1(X)	P1(Y)
1013	1	3	0	G1(X)	G1(Y)	G2(X)	G2(Y)
1013	1	3	1	G2(X)	G2(Y)	G3(X)	G3(Y)
1013	1	3	2	G3(X)	G3(Y)	G4(X)	G4(Y)
1013	1	3	3	G4(X)	G4(Y)	G1(X)	G1(Y)

Tabelle 9 : SDOGEOM

Hier befinden sich die eigentlichen Geometriedaten. Jede Geometrie in einem Layer ist durch eine eigene Geometrie_id GID gekennzeichnet (1013). Weiterhin wird jedes Element einer Geometrie durch eine aufsteigende Zahlenreihe in der zweiten Spalte voneinander getrennt. SDO_ETYPE definiert den Geometriertyp, so steht 1 für SDO_GEOM.POINT_TYPE, 2 für SDO_GEOM.LINESTRING_TYPE und 3 entspricht SDO_GEOM.POLYGON_TYPE. Die vierte Spalte stellt eine Sequenz dar, die jedem Element einer Geometrie eine Zahl zuweist. Dieser Spalte folgen nun die Koordinaten der einzelnen Elemente, hier sind es jeweils die eines Startpunktes und seines Nachfolgers.

d) <layername>_SDOINDEX Table

SDO_GID (number)	SDO_CODE (raw)
1013	T1
1013	T2
1013	T3
1013	T4

Tabelle 10 : SDOINDEX

Die letzte Tabelle bildet einen Index, enthalten sind die GID und die Teile eines Mosaiks in der sich Elemente dieser Geometrie befinden.

Das Laden der Geometrien in dieses Schema vollzieht sich in drei Schritten. Zuerst werden die einzelnen Tabellen Layer für Layer erstellt. Im Anschluss daran werden die in einem ASCII Format gespeicherten Daten mittels einzelnen Transaktionen (insert) oder bei großen Datenmengen durch das Anlegen von PL*SQL- Skripten über den SQL*LOADER in die Tabellen eingefügt. Zum Schluss wird ein Index auf die Geometrien angelegt.

Beide Modelle sind in der Lage folgende Typen sowie Gruppen aus diesen Typen als ein geometrisches Objekt zu bilden.

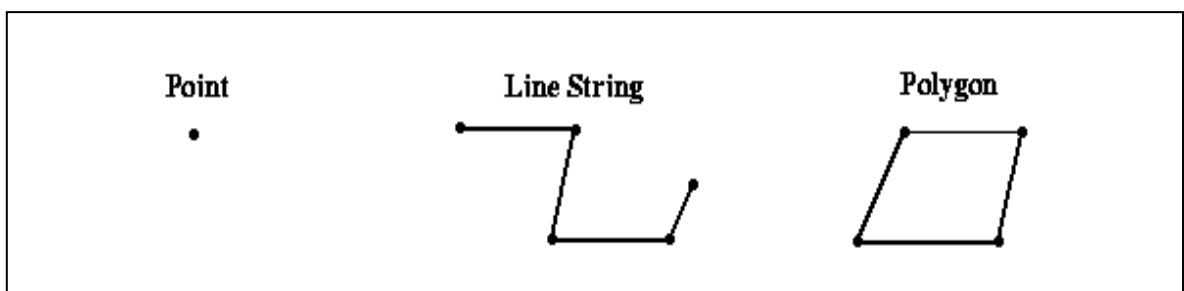


Abbildung 12 : Geometrische Grundtypen

Sich selbst überschneidende Polygone werden im Gegenteil zu selbst überschneidenden Linien nicht unterstützt.

Für das relationale Modell steht nur der Quad- Tree Index (fixed size) zur Verfügung.

Auch bei den Funktionen gibt es Einschränkungen. Ähnlich wie beim objekt-relationalen Schema haben diese Funktionen ebenfalls die Aufgabe Informationen über Geometrien zu erzeugen. So kann man sowohl die Art der Interaktion zweier Geometrien testen, als auch die Gültigkeit eines Layer's oder einer Geometrie feststellen. Weiterhin stehen dem Anwender Tuning Funktionen zur Verfügung, mit denen sich Metadaten zu den Geometrien sammeln lassen (z.B.: kleinstes, die Geometrie umfassendes Rechteck berechnen; beste Stufe für die „fixed size“ Teilung berechnen). Daraus ergeben sich Möglichkeiten der Performanceverbesserungen. Der Gebrauch von Operatoren ist in diesem relationalen Datenmodell nicht vorgesehen.

3.4 Vergleich beider Modelle

Vorteile des relationalen Modells :

- abgleichen von Datenbanken (replication) wird unterstützt
- verteilte Datenbanken werden unterstützt
- Partition von Tabellen wird unterstützt

Vorteile des objekt-relationalen Modells :

- Weitere Geometrietypen wie Arc (Bögen), Circle (Kreise), compound Polygons (zusammengesetzte Flächen), compound line Strings (zusammengesetzte Linien) und vereinfachte Rechteckdarstellungen (zwei Punkte) werden unterstützt.
- hybrid Quad- Tree Index
- Index wird durch den Oracle8i Datenbankserver unterstützt.
- Geometrie wird in einer Spalte und einer Zeile gehalten

- sehr gute Performance
- Funktionen

Der einzige nennenswerte Nachteil des objekt-relationalen Modells liegt in der nicht Replikationsfähigkeit der Daten. Dieses Manko soll im kommenden Release abgestellt werden. Beide Modelle sollen zwar weitergeführt werden, jedoch wird sich Oracle immer weiter auf den objekt- relationalen Ansatz festlegen.

Da das objekt- relationale Modell fast die gesamte Funktionalität besitzt und noch zusätzliche Prozeduren liefert, ist der Einsatz dieses Schemas empfehlenswert. So ist der Verwaltungsaufwand, der durch den neuen Datentyp SDO_GEOMETRIE entsteht, wesentlich geringer als die Administration der jeweils vier Tabellen des relationalen Schemas. Für die LFU bedeutet der Einsatz des objekt- relationalen Schemas den minimalsten Arbeitsaufwand, da alle Felder der Geometrietabellen (geom) vollständig übernommen werden können und dort, wo das Long Raw Feld steht, der neue Datentyp eingefügt werden kann. Das Suchen über vordefinierte Bounding Box Werte entfällt, da der R- Tree Index dies automatisch übernimmt.²⁴

²⁴ vgl.: http://otn.oracle.com/docs/products/oracle8i/doc_library/817_doc/-inter.817/a85337/sdo_intr.htm#871809

4. Konzeption der Anpassung einer Oracle Spatial Datenbank

Die bisher eingesetzten Datenhaltungssysteme haben aufgrund software-spezifischer Rahmenbedingungen aber besonders auch aus der Sicht der Anwender ihren Platz innerhalb des RIPS. Die Verwendung von drei, vom Aufbau her komplett unterschiedlichen, Verwaltungssystemen bringt jedoch auch verschiedene Nachteile mit sich.

- verschiedene Datenformate
- mehrere Schnittstellen
- mehr an Arbeitsschritten
- redundante Datenhaltung
- zeitlich inkonsistente Datensätze

Eine Lösung hierfür wäre der Einsatz einer Oracle Spatial Datenbank. Diese Datenbank bietet eine Struktur, in der nach wie vor Sachdaten und als Neuerung auch Geometrien, genauer Vektorgeometrien, gespeichert werden können. Bei einer Einführung der Oracle Spatial Datenbank, kann man bestehende Operationen nutzen und sich der verschiedenen Zugriffsoptionen, die Oracle bereitstellt, bedienen. Vor der Umstellung soll zunächst geklärt werden, wie man ein solches System in den RIPS Verbund aufnehmen kann. Dabei müssen die bestehenden Anwendungen und die sich im Einsatz befindenden Datenhaltungssysteme, hinsichtlich der Zusammenarbeit mit einer Oracle Spatial Datenbank geprüft werden.

Eine Vereinbarkeit zwischen Smallworld und Oracle steht hier im Vordergrund einer solchen Betrachtung, da die Smallworld Datenbank der bisherige primäre Datenspeicher ist, und fast alle Basis- und Fachdaten, egal ob Raster oder Vektor enthält.

Eine prototypische Anpassung der Oracle Spatial Datenbank an das Smallworld GIS erfolgt mit dem Ziel, bisherige Smallworld-Daten, zumindest teilweise, in die

neue Datenbank zu transferieren und für den Zugriff durch das Smallworld GIS vorzukonfigurieren.

4.1 Phasen der Anpassung

Die Einführung eines neuen Systems wie die Oracle Spatial Datenbank ist ein sehr komplexer Vorgang, der große Auswirkungen auf Anwender und Programme hat. Eine Planung ist für die erfolgreiche Installation und die Abschätzung etwaiger Probleme unabdingbar.

Die Aufstellung einzelner Arbeitsschritte und deren geschätzter Zeitaufwand ist in Abbildung 13 dargestellt. Die meiste Zeit und damit auch der größte Aufwand liegt demnach im Transferieren der Daten aus der Smallworld Datenbank in das Oracle Spatial System. Weiterhin muss ein beträchtlicher Teil an Zeit für die Analyse der Datenzugriffsmöglichkeit des Smallworld GIS auf die nun extern gespeicherten Daten aufgebracht werden.

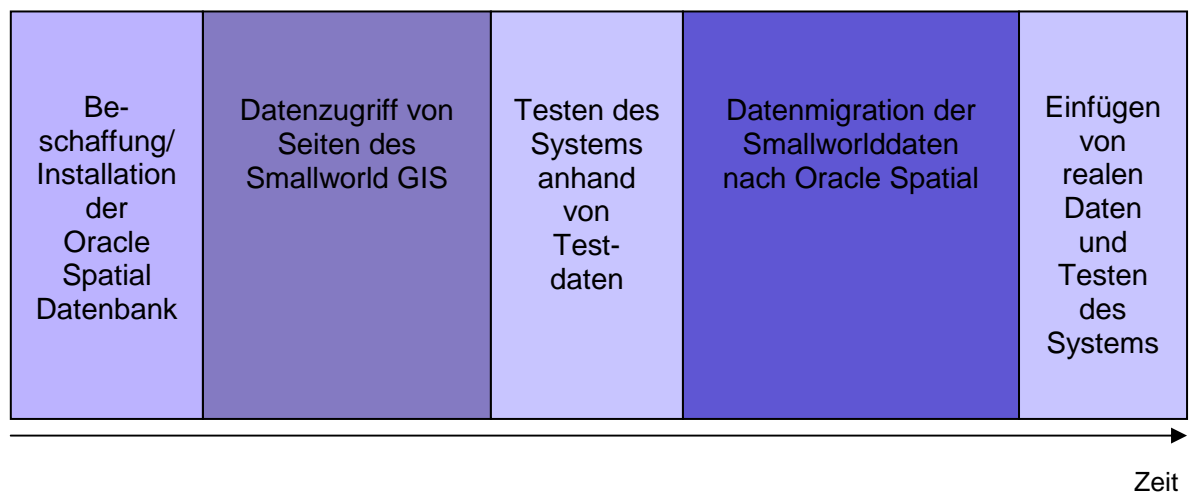


Abbildung 13 : Phasen der Anpassung

4.1.1 Beschaffung und Installation einer Oracle Spatial DB

Die bestehende Oracle Datenbank ist lizenziert durch die Oracle Vertretung Stuttgart. Ausgehend von den hardwareseitigen Rahmenbedingungen (Intel Pentium PC, 2x 333 MHz, Windows 2000) konnte in Zusammenarbeit mit Oracle eine Testlizenz für eine Oracle 8.1.7 beschafft werden.

Die Grundinstallation mit datenbankadministrativen Werkzeugen benötigt ca. 1 GB Festplattenspeicher. Der eingesetzte PC sollte hier als „stand alone“ Plattform fungieren dessen Festplatte auf die zu erwartende Datenmenge vorbereitet wurde. Wichtig ist, dass sämtliche bestehenden Installationen von Oracle Datenbanken oder Oracle Clients deinstalliert werden. Hierfür sollte der Oracle Uninstaller verwendet werden. Es empfiehlt sich dennoch die Registrierungsdatenbank von Windows auf eventuell noch vorhandene Einträge zu untersuchen. Gegebenenfalls sollten auch Systempfade editiert werden.

Anschließend kann auf dem konfigurierten und bereinigten System mit der Neuinstallation begonnen werden. Oracle bietet hierfür zwei Wege an. Zum einen kann der „normale“ Anwender eine Standardinstallation starten, in der bereits eine Datenbank definiert wird. Fortgeschrittene Benutzer können die zu installierenden Komponenten eigenständig auswählen und automatisch installieren lassen.

Nach der Installation besteht die Möglichkeit, existierende Datenbanken in einem eventuell vorhandenen Netzwerk, entweder durch eine Benutzeroberfläche oder durch direkte Manipulation der Dateien „tnsnames.ora und sqlnet.ora“ anzubinden. Damit sind alle wesentlichen Schritte zur Vorbereitung der Datenbank abgeschlossen.

4.1.2 Zugriff des Smallworld GIS auf Oracle Spatial Daten

Smallworld ist in der Lage während einer Anwendung auf mehrere verschiedene Datensätze zurückzugreifen. Ein Datensatz ist eine abgeschlossene Datenmenge, welche aus räumlichen und nicht räumlichen Daten besteht. Die Datensätze setzen sich dabei entweder aus smallworld-eigenen oder aus externen Daten zusammen.

Dieses Datenmanagement ist aufgrund eines mehrschichtigen Aufbaus, also das Hintereinanderschalten verschiedener Teilkomponenten, möglich. In Abbildung 14 ist die Systemarchitektur einer solchen verteilten Datenhaltung dargestellt.

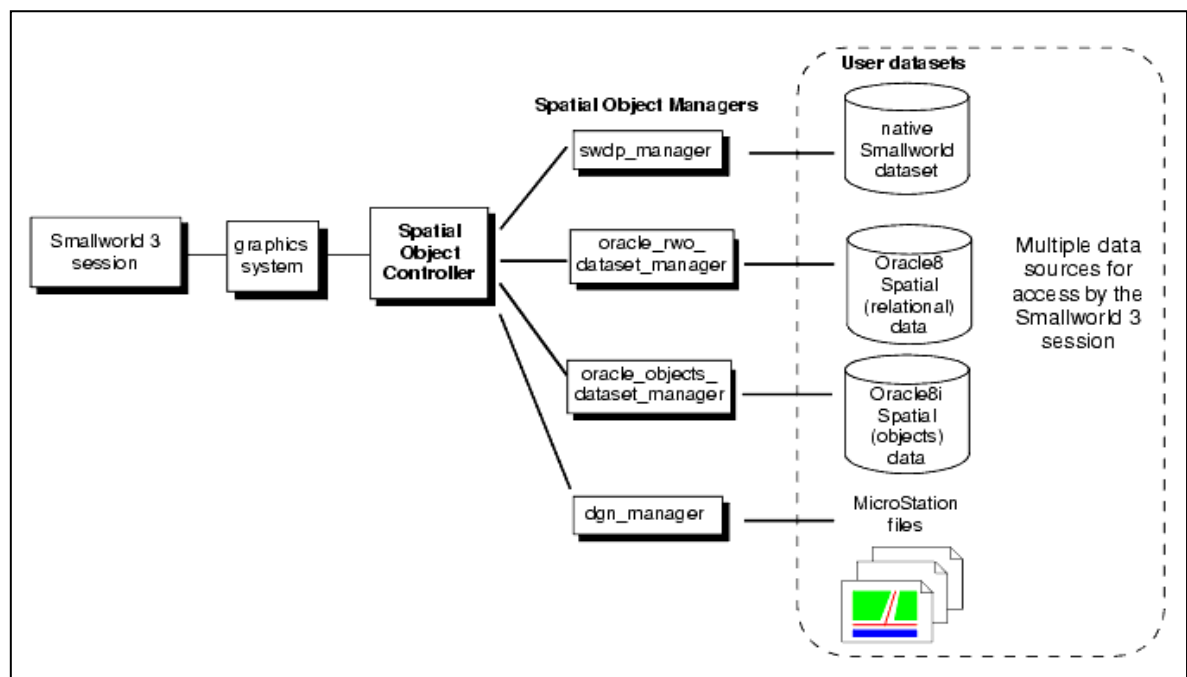


Abbildung 14 : Aufbau einer Smallworldsitzung

Mit dem Starten einer Smallworld 3 Sitzung wird mindestens ein Grafiksystem, welches die Kontrolle über die Benutzerdaten steuert, gestartet. Jedes Grafiksystem verfügt über ein Grafikenster, das die Grafik aller relevanten Daten enthält. Ein Grafiksystem ist immer mit einem **Spatial Objekt Controller (SOC)** verbunden, der den Zugriff auf Benutzerdatenbestände erlaubt. Alle angeschlossenen Datenbestände müssen innerhalb des SOC definiert sein. Ein einzelner Datenbestand wird durch einen Namen und durch den jeweiligen **Spatial**

Object Manager (SOM) eindeutig beschrieben. Der SOM stellt eine Art Plug-In dar, welches den Zugriff auf eine spezielle Datenquelle steuert. Dabei erhält der Anwender Zugriff auf Fremddaten ohne das Kopier- oder Ladevorgänge vonstatten gehen. Ein SOM benötigt zum einen Informationen für das Aufbauen einer Verbindung zum jeweiligen Datenbestand und zum anderen eine Beschreibung wie letztendlich der Datenzugriff umgesetzt wird. Grundlegende Aufgaben eines SOM's sind

- öffnen/schließen des Datensatzes
- der Zugriff auf Sach- und Geometriedaten
- das Zeichnen
- das Hervorheben von selektierten Benutzerobjekten.

Bei smallworld-eigenen Datenbeständen (Datastore Files) erfolgt eine Anmeldung direkt beim Smallworld Datastore Server (**swmfs**). Für den Zugriff auf externe Daten wird der jeweilige Datenserver über einen „Alien **Co-Processor**“ (**ACP**) angesprochen. Das ACP übersetzt die Smallworld Befehle in eine dem Fremdserver verständliche Sprache.

Somit erfolgt der Datenzugriff auf einen Datensatz in mehreren Schritten. Das Grafik System löst im ersten Schritt eine Prozedur auf der Grundlage einer bestimmten Datenmenge aus. Der Spatial Object Controller bestimmt aus der Datenmenge den Namen des Datenbestandes und spricht den jeweiligen Spatial Object Manager an. Dieser gibt die durch die Prozedur ausgelösten Befehle, bei smallworld-eigenen Datenbeständen direkt an den Smallworld Daten Server weiter, oder ruft ein zuständiges ACP auf. Innerhalb des ACP's wird der Befehl in eine für den jeweiligen Datenserver gültige Anweisung übersetzt und an den Fremdserver weitergeleitet.

Eine Verteilung des Gesamtproblems auf verschiedene Komponenten ermöglicht das schnelle und zeitsparende Anbinden vieler verschiedenartiger Datenquellen. So muss für jeden Datenbestand jeweils nur die letzte Ebene der SOM, also der direkte Datenzugriff implementiert werden. Übergeordnete Ebenen können jedoch

gemeinsam benutzt werden und müssen nicht neu entwickelt werden. Dieses Prinzip gilt auch für den Datenzugriff auf Oracle Spatial.

4.1.2.1 Der Oracle Spatial SOM

Smallworld ist, wie viele GIS Entwickler, strategischer Partner von Oracle und hat erkannt, dass dem von Oracle entwickelten und von einer Vielzahl von Anwendern eingesetzten Relationalen Datenbank Management System eine besondere Bedeutung zukommt.

Dem Rechnung tragend, ist es bislang möglich, die auf Seiten Oracle's gespeicherten Sachdaten in Verbindung mit den unter Smallworld abgelegten Geometrien gemeinsam zu nutzen. Mit der Version 8i führte Oracle erstmals eine Möglichkeit zur gemeinsamen Speicherung von Sach- und Geometriedaten ein. Aufgrund des Engagements Oracles in verschiedenen Organisationen zum Beispiel im Open Gis Konsortium und der offenen Struktur der Datenbank, wird Oracle Spatial auch von Smallworld als primäres Datenhaltungssystem anerkannt. Für den Zugriff auf Oracle und den Gebrauch der Datenbankseitig implementierten Funktionen (stored procedures), entwickelte Smallworld den Oracle Spatial SOM. Er unterstützt sowohl das relationale, als auch das objekt- relationale Model von Oracle. Der objekt- relationale Ansatz entspricht der neuen Entwicklungsphilosophie Oracles, in der man von der reinen relationalen Datenbank abgeht und eine hybride Datenbank, bestehend aus einem relationalen und einem objektbasierten Teil, favorisiert. Der Datenzugriff durch das Smallworld GIS sollte diesen Fakt berücksichtigen und sich auf die Struktur des **OSPO (Oracle Spatial Objects) SOM's** konzentrieren.

Wie alle SOM's stellt der OSPO SOM ein Bindeglied zwischen zwei verschiedenen Systemen dar. Bei der Entwicklung einer solchen Schnittstelle will man möglichst viele bestehende Funktionen auf dem anzubinden Datenbestand anwenden. Gleichzeitig ist es jedoch durch die Verschiedenartigkeit der zugrundeliegenden Datenmodelle unabdingbar, die Funktionalitäten spezifisch auf die jeweilige Datenquelle abzustimmen. Smallworld besitzt eine objektorientierte

Umgebung, das heißt es gibt Objekte und es gibt definierte Verhaltensweisen für diese Objekte. Das GIS arbeitet auf der Basis von Benutzerobjekten den **RWO's** (**Real World Object**), die mit den eigentlichen Datentabellen verbunden sind. Beim Bearbeiten der Daten werden immer nur die Objekte manipuliert, nicht direkt die Tabellen. Diese logische Trennung ist für Smallworld die Basis für das Arbeiten mit verschiedenartigen Datenbeständen. So ist es möglich globale Funktionalitäten zu entwickeln und diese dann unabhängig von der physischen Datenstruktur auf verschiedene Objekte zu übertragen. Unter dem Aspekt der Programmierung wird dies durch den Vorgang der Vererbung umgesetzt.

Bezogen auf das Arbeiten mit externen Datenbeständen bedeutet das, dass sowohl das Verhalten der Dataset Manager, das der Datensätze und auch das Verhalten der einzelnen Elemente der Datensätze von den bestehenden Objektklassen abgeleitet werden kann und so die Funktionen nach außen offen sind. Dies sind zum Beispiel Systemfunktionen, die das ACE oder das Style System betreffen, aber auch solche Prozesse wie das Erzeugen von Objekteditoren oder das Initialisieren des Themenmanagers, die auf diese Weise automatisch umgesetzt werden. Weiterhin müssen jedoch Funktionen und Methoden definiert werden, die das „Fremddatenmodell“ als Rahmenbedingungen vorgibt (Zugriffsformen- rechte, Hinzufügen/Löschen von Daten u.s.w.).

Besonderes wichtig ist dabei das Problem des Transaktionsmanagements. Smallworld unterstützt die „Lange Transaktion“. Dies ist aufgrund des vorhandenen Version Managed Data Store (VMDS) möglich. Oracle hingegen arbeitet mit einem Relationalen Datenbank Management System (RDBMS), das kurze Transaktionen implementiert. Der OSPO SOM vermittelt zwischen beiden Datenbankwelten und nutzt zur Kommunikation mit dem Oracle Server ein ACP dem „Oracle ACP“. Es ist via SQL*Net mit dem Oracle Server verbunden und übersetzt die Smallworld Anweisungen für den Server. Der Aufruf des ACP's durch den SOM ist der letzte Schritt innerhalb der Datenzugriffskette. Um den OSPO-SOM nutzen zu können, muss er zunächst geladen, konfiguriert und auf das jeweilige Anwenderdatenmodell angepasst werden. Der Oracle Spatial SOM ist Bestandteil von Smallworld 3 und befindet sich als Code im Stammverzeichnis. Mit dem Start einer Sitzung wird zuerst ein Image geladen, in dem der gesamte

relevante Quellcode enthalten ist. Der SOM - Quellcode muss, beispielsweise durch den „gbm_software_component_manager“ ins aktuelle Image geladen werden. Ist dies erfolgt, sollte das Image abgespeichert („gis_save_image(Pfad)“) werden, um den Code dauerhaft zu speichern. Unter der Voraussetzung, dass in Oracle alle Daten abgelegt sind, ein Index erzeugt wurde und der eingesetzte PC mittels SQL*Net mit einer Oracle Datenbank verbunden werden kann, besteht der nächste Schritt in der Erzeugung eines Konfigurationsskriptes. Ein solches Skript enthält alle wichtigen Informationen über einen neu zu öffnenden Datensatz. Im wesentlichen hat es die Aufgabe, die auf Oracle liegenden Entitäten auf Smallworld Benutzerobjekte zu wandeln.

```
oracle_objects_dataset.define_shared_constant(  
    :dataset_instance_metadata,  
    property_list.new_with(  
        :objects_demo, property_list.new_with(  
            :collections, property_list.new_with(  
                :roads_oo, property_list.new_with(  
                    :geom_fields,property_list.new_with( :centreline, :chain),  
                    :exemplar, road_obj),  
                :area_oo, property_list.new_with(  
                    :geom_fields,property_list.new_with(:coverage, :area,  
                                                         :annotation, :point,  
                                                         :centroid, :point),  
                    :exemplar, area_obj ) ),  
                :short_trans, _true ) ),  
        :private )
```

Quellcode 6 : Konfigurationsskript

Der Quellcode 6 zeigt den Aufbau eines solchen Konfigurationsskriptes. Es definiert für einen neuen Oracle Datensatz („**objects_demo**“) alle anzubindenden Tabellen („**roads_oo, area_oo**“) inklusive deren Geometriespalten („:**centreline, :coverage u.a**“). Nachdem die Geometriespalten deklariert worden sind, muss eine Zuordnung zur jeweiligen Smallworld Geometrie Objektklasse erfolgen.

Unterstützt werden hierbei Punkte, Linien und Flächen, wobei es Unterschiede in den Auslegungen der Geometrietypen zwischen Smallworld und Oracle gibt.

Die Oracle Geometrietypen sind Simple Feature konform, das heißt, sie beinhalten Punkte, Linien, Flächen und jeweils Gruppen dieser Grundtypen. Damit hält sich Oracle an die 1998 verabschiedete Simple Feature Spezifikationen des OpenGIS-Konsortiums (OpenGIS Consortium 1998:3-11) und ist neben ESRI SDE von genannter Organisation zertifiziert. Neben elementaren Geometrietypen unterstützt Oracle ebenfalls das Abbilden von Rechtecken, Kreisen und weitere Formen (keine Simple Feature Elemente). Das Geometriemodell des Smallworld GIS weicht vor allem in der Definition der flächenhaften Objekte von Oracle ab. Ist es bei Oracle möglich disjoint(e), also unabhängige, Flächen in einem Objekt zusammen zu fassen, funktioniert dieses in der installierten Smallworld Version nicht.²⁵

Nachdem die Tabellenstruktur aufgeschlüsselt wurde, muss ein Objekt, bei Smallworld spricht man von Exemplaren, definiert werden („**exemplar,road_obj, exemplar, area_obj**“). Auf diese Weise ist es möglich, neue Methoden speziell für dieses Objekt zu entwickeln.

Wie bereits beschrieben, ist eines der größten Probleme, zwischen Smallworld und Oracle die Inhomogenität des Transaktions-managements. Der OSPO SOM kann so konfiguriert werden, dass jede Änderung, die aus einem insert, delete oder update stammt, sofort zurückgeschrieben wird und so eine kurze Transaktion darstellt. Ohne diese Anpassung muss jede Änderung explizit committed bzw. durch ein roll back rückgängig gemacht werden. Im Skript wird für den ersten Fall die Variable „short_transaktion“ auf den Wert „_true“ gesetzt („**short_trans, _true**“). Das so erstellte Skript kann direkt in die Magikumgebung eingefügt und anschließend durch das Speichern des Images auf Dauer verfügbar gemacht werden.

Damit das aktuelle Grafiksystem auf den neuen Datenbestand zugreifen kann, muss dieser beim Spatial Object Controller registriert werden. Mit „manage_soc()“ kann bei geöffneter Datenbank ein neuer Datensatz definiert werden. Wichtig ist hierbei, dass immer ein nativer Datenbestand, allgemein der GIS Datenbestand,

²⁵ vgl.: Anhang 7.9 – Bestätigung des auftretenden Fehlers durch Smallworld

definiert sein muss. Nach dem Eintragen des Datensatznamens müssen die für Oracle benötigten Verbindungsinformationen wie Benutzer, Passwort und Datenbank eingegeben werden. An dieser Stelle entscheidet sich auch, unabhängig von der Smallworldanmeldung, ob der Benutzer das Recht hat, Daten zurückzuschreiben. Nach dem Öffnen des Datenbestandes sind die üblichen Objektdefinitionen im ACE (Sichtbarkeit, Selektierbarkeit u.a.) und die graphische Darstellung im Style festzulegen.

Ein auf diese Weise definierter Oracledatensatz verfügt über die gesamte Funktionalität des Smallworld 3 GIS.²⁶

4.1.3 Testen des Systems anhand von Beispieldaten

Zum Testen der Funktionsfähigkeit beider Systeme liefert Smallworld im Rahmen seiner Onlinedokumentation einen Testdatensatz. Es handelt sich dabei um zwei Datendumps einer Oracle 8.0.5 Datenbank aus dem Smallworld Cambridge Datenmodell. Das Importieren der Daten erfolgt über das Oracle Standardwerkzeug „imp80“. Es ist jedoch darauf zu achten, dass die Daten aus einer 8.0.5 Version stammen und somit vor dem Einfügen in eine höhere Version gewandelt werden müssen.

Nachdem die Daten bereitstehen und Oracle und das Smallworld GIS konfiguriert worden ist, sollten die Daten im Smallworld darstellbar sein.

Durch die Verwendung des Testdatensatzes kann sichergestellt werden, dass die Verbindung zwischen beiden Systemen funktioniert und die Konfiguration erfolgreich war. Sollten Fehler auftreten, können diese entweder mit Hilfe der Onlinedokumentation oder, da das Beispiel von Smallworld unterstützt wird, direkt mit der Smallworldhilfe nachvollzogen werden.

²⁶ vgl.: Smallworld 3 Dokumentation Version 3.1 (0) SP1 – Open Systems: Smallworld on Oracle Spatial reference

4.1.4 Entwurf eines Datenbankadapters zum Zweck der Datenmigration

Beim Einsatz der Oracle Spatial Datenbank als primärer Datenspeicher, stellt sich die Frage nach der Datenkonvertierung. Hierfür gibt es verschiedene Ansätze, die jeweils Vor- und Nachteile mit sich bringen. Für einen großen Datenbestand gilt es, die einzelnen Schritte der Datenübernahme zu identifizieren, und anschließend ein Modul zu entwickeln, das diese Aufgabe übernimmt. Die Anforderungen an einen solchen Datenbankadapter werden zum einen durch die beiden Systeme, Smallworld und Oracle, und zum anderen durch den Benutzerkreis definiert. Für die praktische Umsetzung können sich hieraus verschiedene Vereinbarungen ableiten lassen.

Die LfU als zukünftiger Benutzer gibt als äußeren Rahmen folgende Bedingungen vor.

- automatisierte Datenübernahme aller Smallworlddaten
- eventuell Einbeziehen von bestehenden Entwicklungen aus Zeit- und Kostengründen
- Steuerung über eigene Oberfläche

Eine automatisierte Datenübernahme hat den Vorteil, dass alle Daten auf dem gleichen Weg konvertiert werden. Bei eventuell auftretenden Fehlern handelt es sich dann meistens um systematische. Sollte es hierfür schon bestehende Software geben, so ist zu überprüfen, ob diese Entwicklungen eingebunden werden können. Der Punkt 3 gewährleistet, dass auch Anwender, welche nicht mit dem Datenmodell von Oracle Spatial vertraut sind, den vollständigen Transfer steuern können. Die Vorgaben der beiden Systeme sind durch die unterschiedlichen Datenmodelle begründet. Die Prozedur des Datentransfers muss diese Vorgaben berücksichtigen und durch einzelne Schritte nachvollziehen. Die Aufgabe eines Datenbankadapters besteht im wesentlichen darin, die komplexen Abläufe der Datenübernahme zu vereinfachen und dem Anwender das detaillierte Einarbeiten

in die Materie zu ersparen. Der Programmablauf eines typischen Datentransfers gestaltet sich dabei folgendermaßen.

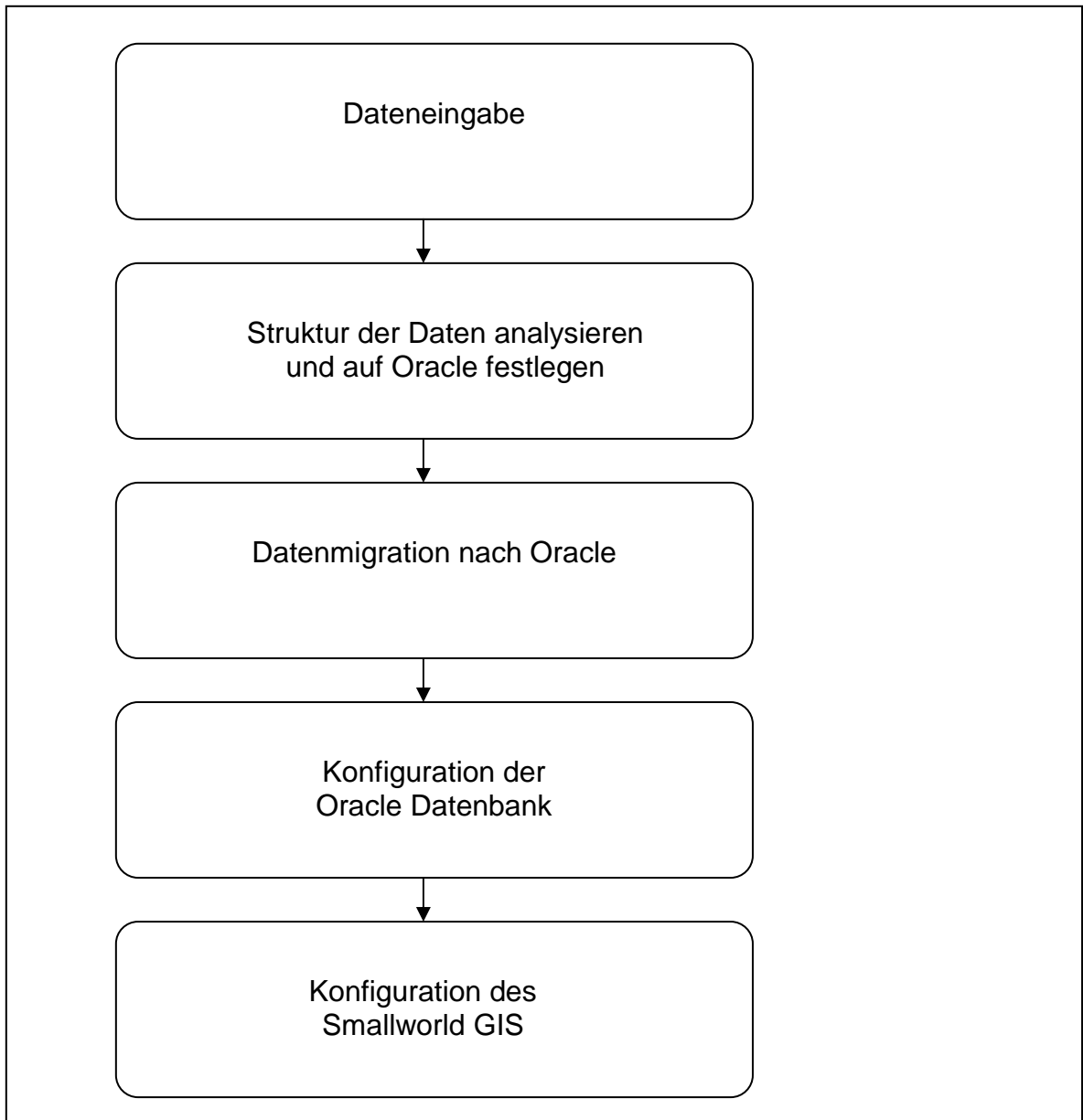


Abbildung 15 : Allgemeiner Programmablauf

Demnach ist zunächst zu klären, wie das Ausgangsformat der Daten aussieht und ob ein direktes Lesen der Smallworldbank möglich ist. Ist die Datenquelle festgelegt, muss jeder Datensatz, der zumeist Sach- und Geometriedaten beinhaltet, hinsichtlich seiner Struktur untersucht werden. Dabei gilt es festzulegen, welche Attribute nach Oracle migriert werden können. Eine

Unterteilung in Sach- bzw. Geometriedaten ist zwingend, da es für beide Datenformen unterschiedliche Vorgaben gibt. Bei den Sachdaten empfiehlt es sich, alle physikalischen Felder für die Datenkonvertierung zu identifizieren. Diese enthalten alle durch den Benutzer direkt erzeugten Daten eines Objektes, dazu gehören Zahlen und Wörter. Für diese reinen Sachdaten sind in einer zweiten Stufe die Feldtypen zu bestimmen und auf die Oracle Datentypen umzuändern.

Von den Geometriedaten können nur Vektordaten konvertiert werden, da das Oracle Datenbankschema keine Verwaltung von Rasterdaten vorsieht. Deren Übernahme bleibt vorerst noch offen.

Bei einer Festlegung der Grundstruktur reicht es bei den Geometriedaten vorerst aus, den Spaltennamen der Spalte, in der die Vektordaten gespeichert werden sollen, festzulegen. Die so erhaltene Datengrundstruktur kann nun für das Anlegen eines Layer's auf Oracle genutzt werden.

In einem dritten Schritt können nun die eigentlichen Daten migriert werden. Die Geodaten der LfU beziehen sich auf die gesamte Landesfläche von Baden-Württemberg und haben somit, was die Geometrien angeht, ein erhebliches Datenvolumen. Für die Darstellung des Rheins werden z.B. an die 3000 Stützpunkte benötigt. Zur Übernahme solcher Datenmengen ist der normale „Insert“ Befehl nicht ausreichend. Das ist auf die Länge des vom Oracle Server zu interpretierenden Kommandos zurückzuführen. Eine bessere Variante stellt der Oracle SQL*LOADER dar. Hierbei handelt es sich um ein Dienstprogramm, welches Daten aus externen Dateien in Oracle-Tabellen laden kann. Oracle ist damit in der Lage auch große Datenmengen effizient und in angemessener Zeit zu verarbeiten.

Im nächsten Arbeitsgang muss ein Index erstellt werden. Oracle bietet im objekt-relationalen Modell zwei Typen an. Der zu entwickelnde Datenbankadapter kann hier jedoch keine Entscheidung treffen und muss es dem Anwender überlassen, für seine geometrischen Daten den richtigen Typ zu wählen. Allerdings kann eine entsprechende Hilfe die Entscheidung beeinflussen. Auf die gleiche Weise soll es möglich sein, etwaige „Constraints“, wie das Anlegen von primären Schlüsseln zu gewähren. Zudem müssen die Daten publiziert werden. Oracle nutzt dabei eine

Tabelle „*USER_SDO_GEOM_METADATA*“, in der für jeden Layer der Tabellename, die Bezeichnung der Geometriespalte und eine Bounding Box für alle enthaltenen Geometrien definiert wird. Das Eintragen von Layer- und Spaltennamen erfordert programmiertechnisch wenig Aufwand, jedoch bedarf das Festlegen der Bounding Box genauere Kenntnis über die einzufügenden Objekte. Die Daten der LfU sind auf das Gebiet von Baden-Württemberg beschränkt und einer allgemeinen Festlegung der Umrisse stehen im wesentlichen Performancenachteile im Wege. Vordergründiges Ziel ist jedoch die Frage nach der Einsetzbarkeit der Oracle Spatial Datenbank als Hauptdatenspeicher. Die Datenübernahme ist dabei „nur“ ein wichtiger Teilschritt. Performancesteigerungen oder Funktionserweiterungen des Adapters sollten erst nach einer Entscheidung für den Einsatz der Datenbank erörtert werden.

Die genannten Punkte betreffen alle Oracle und spielen sich ausschließlich in der Datenbankumgebung ab. Der zu entwickelnde Datenbankadapter muss in der Lage sein, diese Funktionen nach expliziten Aufruf etwaiger Anwendungen wie *SQL*LOADER* oder *SQL*PLUS* durchzuführen.

Nach dem Abschluss dieser Phase liegt ein homogener Datensatz vor, auf den verschiedene Geoinformationssysteme aufgesetzt werden können. Für eine Anpassung der Daten an das Smallworld GIS ist eine entsprechende Konfiguration nötig, welche die bestehende Schnittstelle, der OSPO SOM, vorgibt.

4.1.5 Einfügen von realen Daten und Testen des Systems

Innerhalb dieses Arbeitsschritts wurden reale Daten mit dem entwickelten Datenbankadapter in die Oracle Spatial Datenbank transferiert. Die Auswahl der Daten war für die von der LfU gespeicherten Geodaten eingeschränkt repräsentativ.

Vor dem Einfügen realer Daten wurden jedoch zunächst jeweils kleinere Objekte manuell eingefügt. Dabei handelte es sich um punkt-, linien- und flächenhafte Objekte, die sich zum Teil aus einzelnen Elementen zusammensetzten.

Auf diese Weise sollte die Funktionsweise des OSPO SOM untersucht werden.

So konnte festgestellt werden, dass Smallworld Probleme mit der Darstellung von topologisch getrennten Objekten hat. Geometrien, die sich aus mehreren Elementen zusammensetzen, können nicht im GIS bearbeitet werden. Dies wurde, dann in einer Anfrage an die Entwicklungsabteilung von Smallworld bestätigt.²⁷

Das Einfügen von Originaldaten des RIPS Pools hatte dagegen eher die Aufgabe den Datenbankadapter hinsichtlich möglicher Fehler zu analysieren und Probleme im Umgang mit Massendaten festzustellen. Hierfür wurden zum einen die landesweit erfassten Fließgewässer, ca. 15000, im Maßstab 1: 50000 und verschiedene Flächen wie Kreise, Regionen u.a. mittels Datenbankadapter nach Oracle transferiert.

Die bei den Testobjekten auftretenden Fehler stellten sich auch bei der Verwendung echter Daten ein. Zudem konnte festgestellt werden, dass die Datenmigration und das Anlegen der Indexes, je nach Datenvolumen, in einem angemessenen Zeitraum vor sich ging.

Die Erzeugung des Konfigurationsskriptes verlief bei jedem Datensatz erfolgreich. Hinsichtlich der Verwendung smallworld-eigener Funktionen traten Probleme bei der Erzeugung des Objekteditors und im Bereich der Abfrageprozesse auf. Beide konnten mittels Patch zwar kurzfristig gelöst werden, sind aber Ausdruck bestehender Inkonsistenzen zwischen smallworld-eigenen Funktionen und LfU interner Softtentwicklungen, und können nicht verallgemeinert werden. Die Anwendung jeder Nicht-Standard-Smallworldfunktion ist demnach erwartungsgemäß problembehaftet und im Einzelfall zu überprüfen.

²⁷ vgl.: Anlage 7.9

5. Entwicklung eines Datenbankadapters

Die Umsetzung der in der Abbildung 15 dargestellten Einzelprozesse erfolgt mittels geeigneter Entwicklungsumgebung. Es gibt verschiedene Programmiersprachen die hier in Frage kommen. Entscheidend ist der Fakt, dass Smallworld nur eingeschränkten Zugriff auf seine Daten gewährt und sich damit wesentlich von der Oraclephilosophie unterscheidet. Der Weg zu den Smallworlddaten ist am besten mit der Sprache gangbar, welche auch für die Definition der Datenstruktur benutzt wurde.

5.1 Der Datenbankadapter

Der entwickelte Datenbankadapter greift die Vorteile von Magik auf und setzt die gestellten Bedingungen in Form eines Prototyps um. Dabei wird die Möglichkeit der Verwendung von freien Programmen genutzt und um diese eine anwenderoptimierte Schale konstruiert. Die Smallworlddaten werden nicht direkt von einem System ins andere überspielt, sondern vorher durch die Verwendung der bestehenden Shape - Schnittstelle ins Shape Format gewandelt. Diese Lösung vernachlässigt damit zwar die Möglichkeit eines direkten Datenzugriffs auf die Smallworld Datenbank, kompensiert dieses jedoch durch die Berücksichtigung bereits entwickelter Anwendungen.

Aus der Entwicklersicht stellt der Datenbankadapter eine eigene Magikklasse mit verschiedenen Methoden und Prozeduren dar. Der Klassenname des Datenbankadapters ist „SHP2SDO“ und steht im Zusammenhang mit den Ausgangs- und Zieldatenformaten, SHAPE und SDO. Im Rahmen der Klassendefinition ist zu klären, ob es bestehende Klassen gibt, von denen man Attribute und Eigenschaften erben kann. Für die Gestaltung von Oberflächen bietet Magik die Objektklasse „Model“ an. In ihr sind bereits alle Grundfunktionalitäten wie Fenster, Button, Menüleisten und andere enthalten. Bei der Entwicklung einer neuen Klasse kann man auf diese Eigenschaften

zurückgreifen und sich auf die Kernfunktionalität, hier dem Datentransfer, konzentrieren.

Die Abbildung 16 zeigt die entsprechende Klassenhierarchie, in welche sich die neue Klasse einreicht.

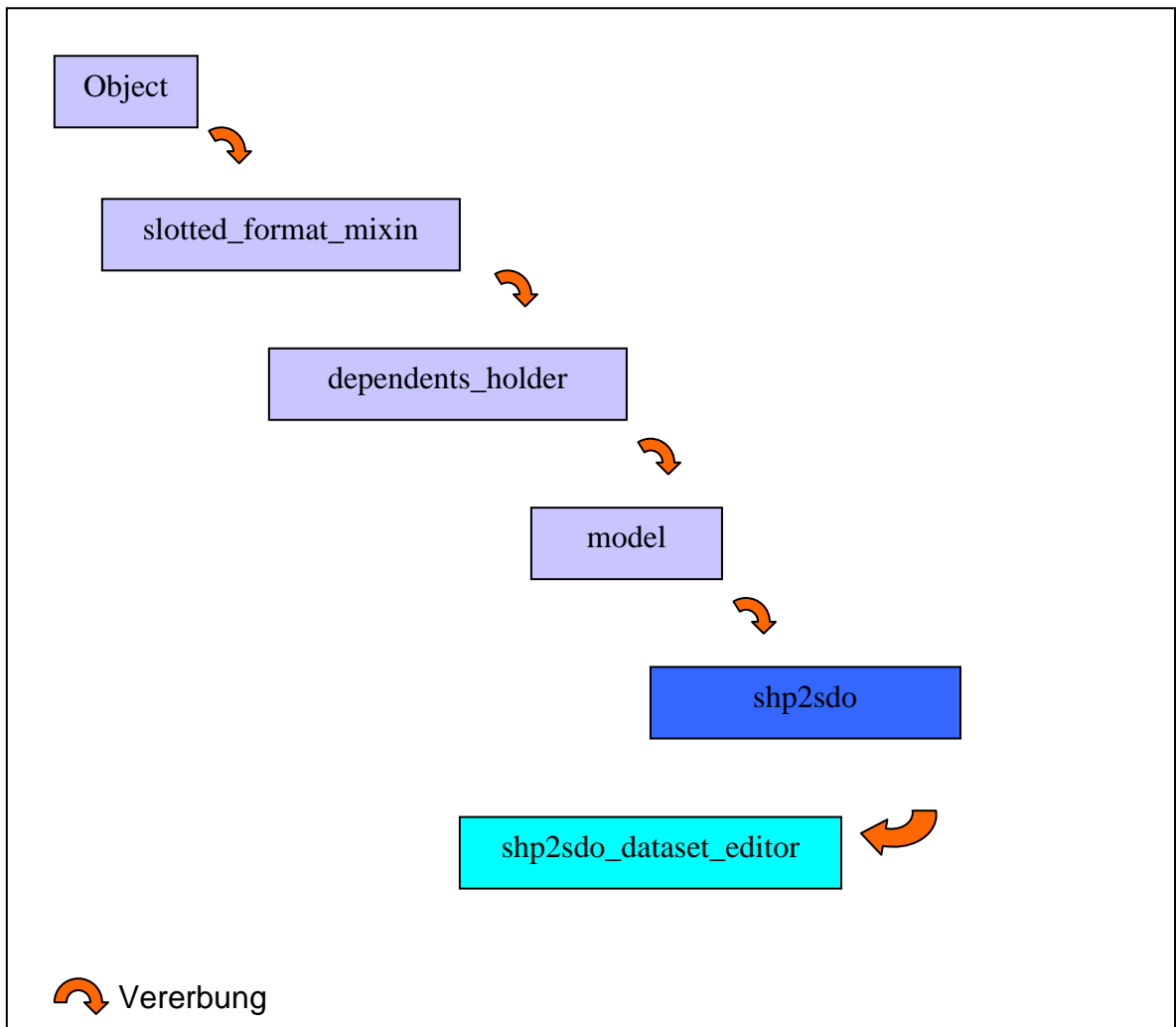


Abbildung 16 : Klassenhierarchie

Wie alle Klassen leitet sich auch SHP2SDO durch Vererbung über mehrere Superklassen von der obersten Klasse Object ab. Die unterste Klasse „shp2sdo_dataset_editor“ leitet sich direkt von SHP2SDO ab und dient der Erstellung und Verwaltung der Konfigurationsskripte .

Mit der Definition eines neuen Exemplars müssen gleichzeitig die Instanzvariablen deklariert werden. Diese können sich in der Entwicklungsphase ändern und sind

nicht von Anfang an bekannt. Aus diesem Grund sollte das Abspeichern des Codes im Image vermieden werden.

Die Abbildung 17 stellt skizzenhaft den allgemeinen Aufbau der Klasse SHP2SDO dar. Im ersten Drittel steht zunächst der Name der Klasse. Ihm folgen Instanzvariablen auch Slot's genannt. Sie sind in jeder Methode ansprechbar und tragen die eigentlichen Daten. Der letzte Teil soll symbolisch für die Methoden stehen, auf die das Objekt später reagieren soll.

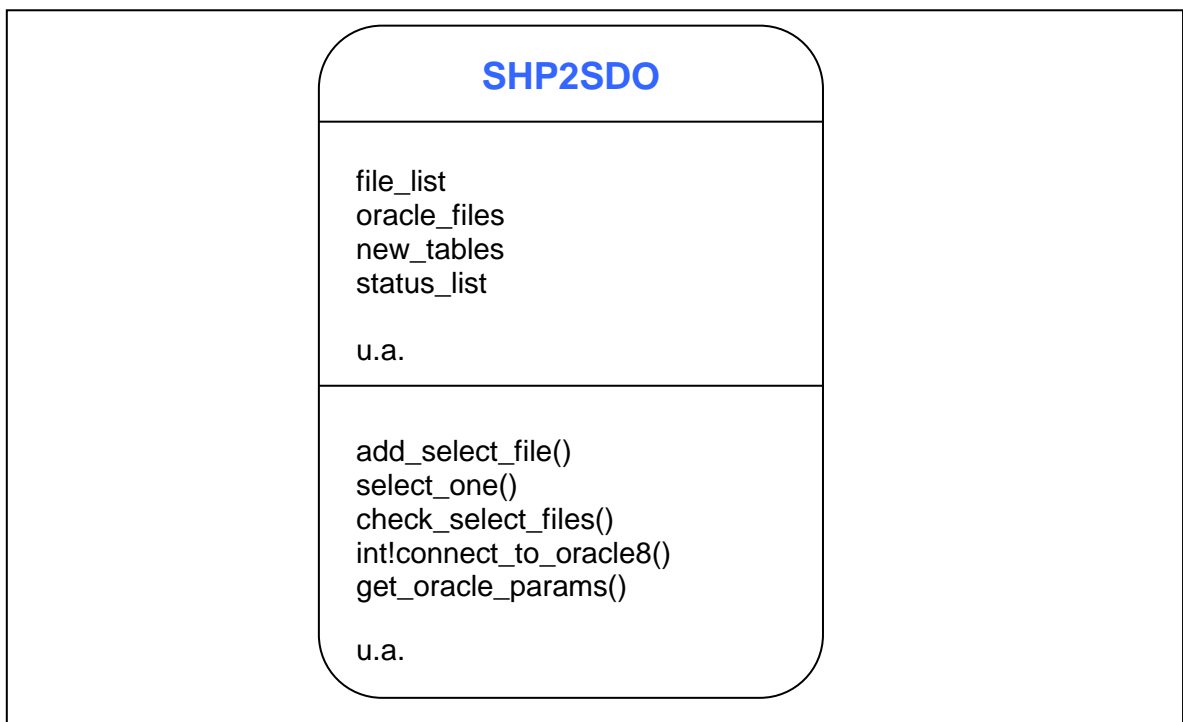


Abbildung 17 : SHP2SDO Datenbankadapter

Der Prozess der Objektdefinition ist damit abgeschlossen und die Entwicklung der Oberfläche und deren Funktionen kann beginnen. Der Datentransfer läuft in mehreren Phasen und ist kein geschlossener Prozess. Der Anwender steuert die einzelnen Phasen über eine graphische Oberfläche. Diese Oberfläche führt den Benutzer durch den Datentransfer, kontrolliert seine Eingaben und kommuniziert mit verschiedenen Unterprogrammen. Die Unterprogramme fungieren als selbstständige Prozesse und behandeln jeweils ein Teilproblem des Transfers. Kernstück der Konvertierung ist das gleichnamige von Oracle publizierte Programm <shp2sdo>. Es wandelt die Informationen aus einem Shapefiles in den

Oracledatentyp SDO_GEOMETRY. Dabei wird zuerst die Struktur der gespeicherten Daten analysiert und in einer anzugebenen *.SQL- Datei gespeichert. In einer zweiten Phase werden sowohl die Sachdaten, als auch die Geometrien in eine SQL*LOADER- fähige *.CTL- Datei exportiert. Das Ergebnis sind zwei Dateien, die mit dem Benutzerprogramm SQL*PLUS bzw. mit dem SQL*LOADER in die Datenbank importiert werden können. Diese Vorgänge werden durch den Datenbankadapter automatisiert.

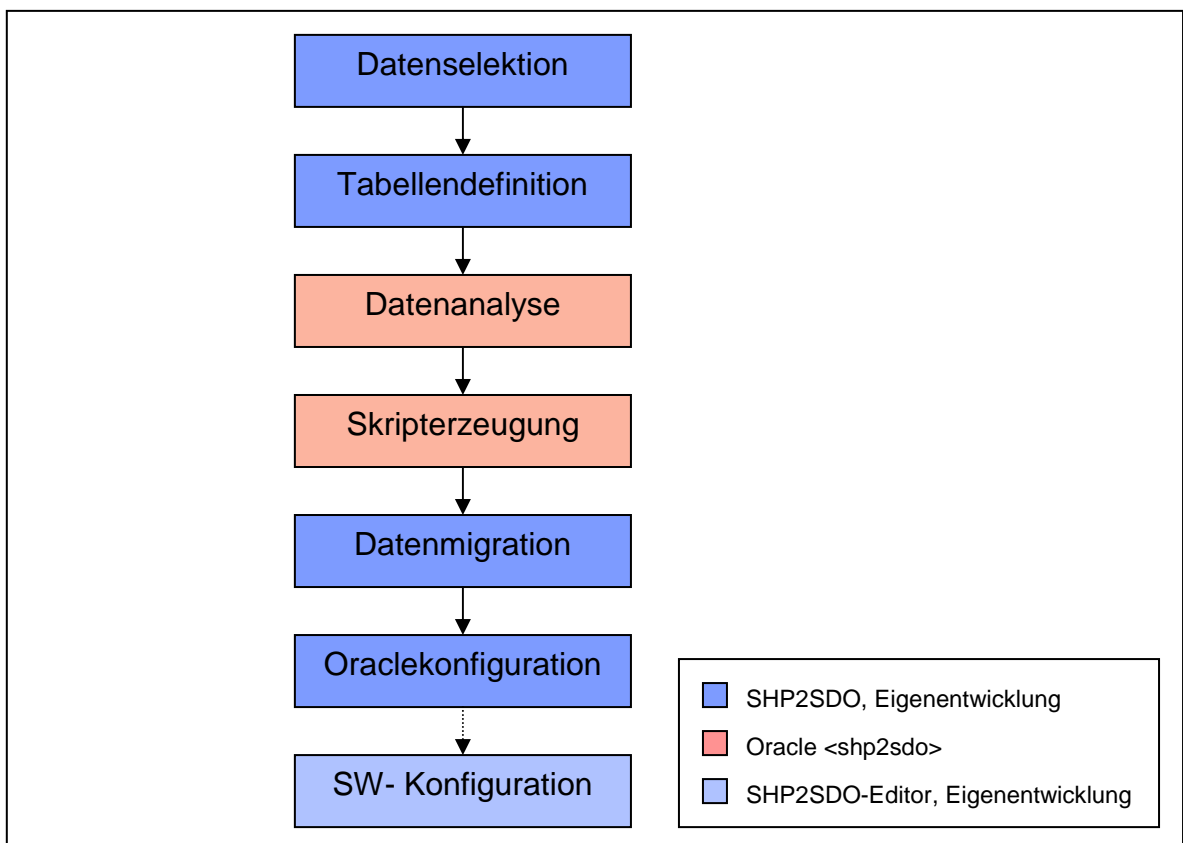


Abbildung 18 : Datenbankadapter Programmablauf

Beim Starten des Transfers muss ein neues Objekt initialisiert werden. Im Magikprompt wird dies durch die Eingabe folgendes Befehls realisiert.

```
Magik2> q<<shp2sdo.new(g)
```

Quellcode 7 : Objektinitialisierung

Die Variable „q“ ist nun ein Objekt aus der Klasse SHP2SDO. Das „g“ steht hier für das aktuelle Grafiksystem, welches mit den Tasten F11+g (nur im Emacs) vorher in der Magikeingabe definiert werden kann. Im Anschluss kann das erzeugte Objekt aktiviert werden.

```
Magik2> q.activate()
```

Quellcode 8 : Objektaktivierung

Dem Anwender zeigt sich nun eine erste Eingabeoberfläche.

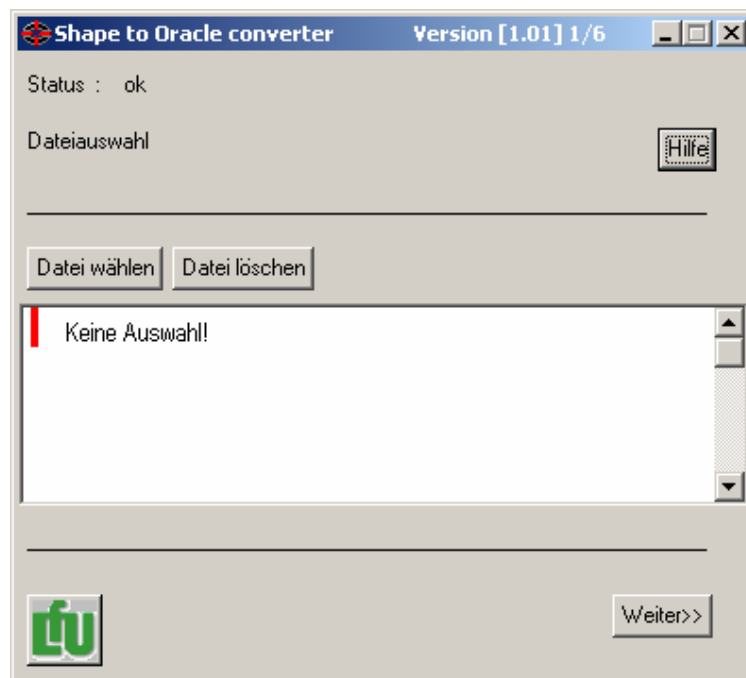


Abbildung 19 : Menü Dateiauswahl

Im oberen Teil des Menüs befinden sich allgemeine Informationen. So steht in der Headerzeile der Name und die Version der Anwendung. Des weiteren sieht der Anwender welche Stufe des Transfers er gerade durchläuft („1/6“). Im darunter liegenden Abschnitt wird der Zustand des Systems, der Name der Oberfläche und eine Hilfeoption dargestellt.

Die bis zu dieser Stelle definierten Elemente sind in allen weiteren Oberflächen gleich und erleichtern so die Orientierung im Menü. Im unteren Teil des Menüs

kann der Anwender über das bekannte Windows Dateiauswahlmenü, ein oder mehrere Shape - Files auswählen. Es können bis zu fünf Dateien gleichzeitig bearbeitet werden. Alle, sich in der Auswahl befindlichen, Dateien werden nach dem Betätigen der „Weiter- Taste“ innerhalb des Objektes mit vollständigem Pfad gespeichert. Dabei wird auch sichergestellt, dass Dateien nicht doppelt ausgewählt wurden und sich mindestens eine Datei in der Auswahl befindet.

Im nächsten Schritt muss der Anwender den Namen der Tabelle und den der Geometriespalte des auf Oracle zu definierenden Layer's angeben. Zusammen mit den Attributen eines Objektes, dienen diese Angaben dem Festlegen der Oracle Tabellenstruktur.

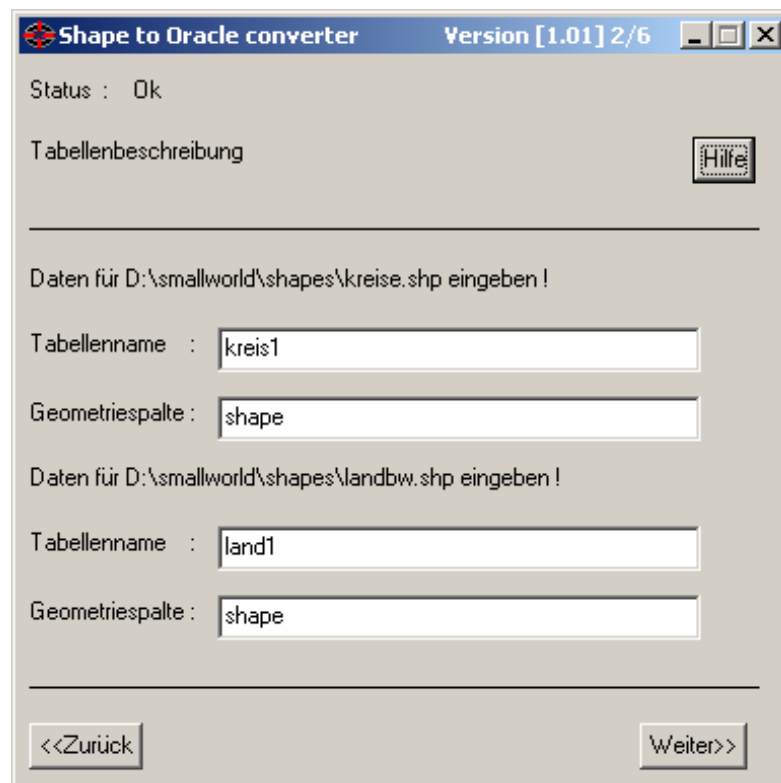


Abbildung 20 : Menü Tabellenbeschreibung

Nach dem Eingeben der Daten muss eine Verbindung mit Oracle hergestellt werden, um zu überprüfen, ob es diese Tabellen schon gibt. Eine Bejahung würde ein Vergleich der bestehenden Spalten mit den neu einzufügen Daten nach sich ziehen. Stimmen beide Strukturen nicht überein, so kann der Transfer nicht

weiterlaufen. Zum Anpassen der Strukturen muss das Shape editiert oder die Tabellendefinition mit Hilfe der entsprechen Oracle Programme (SQL*WORKSHEET/PLUS) neu definiert werden. Zum Login greift die Anwendung auf bestehende Strukturen zurück. So nutzt es ein bestehendes Oracle Login Panel , also ein Fenster, welches die Verbindungsparameter zur Datenbank entgegennimmt. Zum Übermitteln der Daten und der damit verbundenen Anmeldung am Oracle Server verwendet Smallworld das Oracle ACP und SQL*NET. Das nachfolgende Codefragment zeigt den Aufbau einer solchen Verbindung.

```

_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.int!connect_to_oracle8(user,passwd)
_handling warning _with _proc(cond) cond.report_contents_on(!terminal!)
_endproc
_self.oracle_login<<rope.new()
_self.oracle_login[1]<<user
_self.oracle_login[2]<<passwd
_local g<<!grs!
_local v<<g.current_dsview
_local spa_spec << extdb_user.connect_template(
    :dbtype, :oracle8,
    :connect_name, "External Database",
    :command, rope.new_with("sworacle8"),
    :dbusername, user,
    :dbpassword, passwd,
    :no_mandatory_field_validation, _true,
    :command, rope.new_with("sworacle8"),
    :configuration,
(property_list.new_with(:max_table_fetch,10)))

_global spatial_connection
spatial_connection << extdb_user.new(spa_spec )
v.add_external_database(spatial_connection)
g.ace_control.reset()
_self.status<<"Ok"
#spatial_connection.discard(v)
_self.get_oracle_params()
_endmethod

```

Quellcode 9 : Verbindungsaufbau mittels Magik

Den wichtigsten Teil dieser Methode, hier blau markiert, bezeichnet man als Verbindungsspezifikation. Es handelt sich dabei um eine Gruppe von Verbindungsparametern, die eine direkte Verbindung zu einer externen Datenbank beschreiben. Zum Einrichten der Verbindung muss ein neues Objekt der Subklasse `extdb_user` mit der Methode `extdb_user.new()` erstellt werden (rot gekennzeichnet). Innerhalb dieser Methode wird das ACP mit den vorher definierten Parametern gestartet. In der vorliegenden Verbindungsspezifikation werden keine Oracle Tabellen beschrieben. Dies liegt daran, dass der User frei wählbar sein soll und alle in seinem Schema enthaltenen Tabellen später angebunden werden können. Die grün markierte Zeile ermöglicht das Hinzufügen der Datenbank und damit eventuell definierter Tabellen zur aktuellen Datenbankansicht. Ein Arbeiten von Seiten Smallworld wird hierdurch erst möglich. Ist eine Verbindung zustande gekommen, so bleibt sie für die Dauer der Smallworld Sitzung aktiv und muss nicht bei jedem Transfer neu hergestellt werden. Das folgende Fenster gibt Auskunft, inwieweit einzelne Tabellen in Oracle bereits bestehen oder neu angelegt werden müssen.

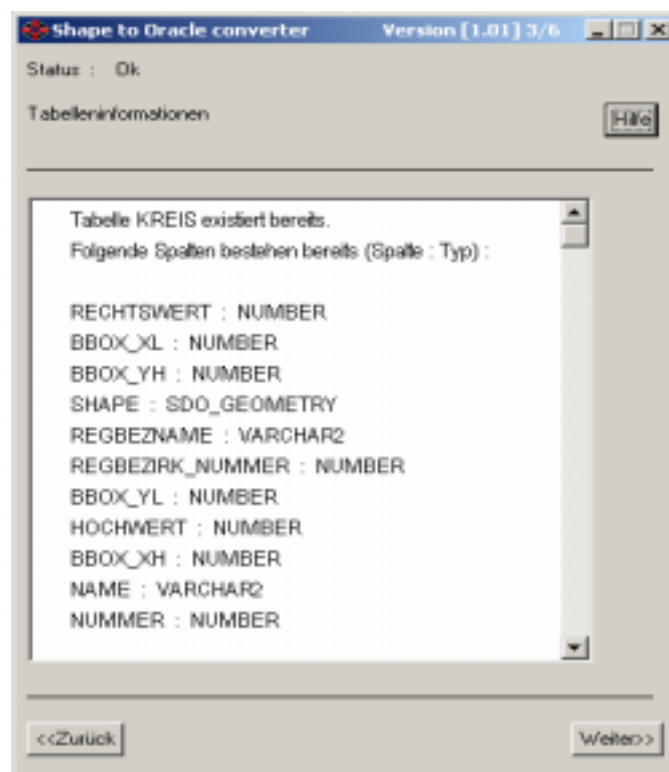


Abbildung 21 : Menü Tabelleninformationen

Mit einem Klick auf „Weiter“ werden pro Shapefile durch das externe Oracle Programm <shp2sdo> jeweils zwei Dateien erzeugt. Zuerst wird eine *.SQL- Datei angelegt, die Anweisungen für den Oracle Datenserver enthält.

```
DROP TABLE FLIES50;

CREATE TABLE FLIES50 (
  gew_id      VARCHAR2(38),
  gew_kennza  VARCHAR2(38),
  gew_name    VARCHAR2(80),
  vor_gew_ke  VARCHAR2(38),
  laenge     NUMBER,
  user_name   VARCHAR2(32),
  SHAPE      MDSYS.SDO_GEOMETRY);

DELETE FROM USER_SDO_GEOM_METADATA
WHERE TABLE_NAME = 'FLIES50' AND COLUMN_NAME = 'SHAPE' ;

INSERT INTO USER_SDO_GEOM_METADATA(TABLE_NAME,COLUMN_NAME,DIMINFO)
VALUES ('FLIES50', 'SHAPE',
  MDSYS.SDO_DIM_ARRAY
    (MDSYS.SDO_DIM_ELEMENT('X',337456153.000000000,364231221.000000000,
0.000000050),
    MDSYS.SDO_DIM_ELEMENT('Y',522898599.000000000,555568636.000000000,
0.000000050)
  )
);
COMMIT;
```

Dateiauszug 1 : SQL - Datei

Beim Laden der Datei mittels SQL*PLUS wird zunächst eine eventuell bestehende Tabelle mit dem gleichen Namen gelöscht. Das führt unweigerlich zu Problemen, wenn neue Daten in eine bestehende Tabelle eingefügt werden sollen. Der Datenbankadapter hat jedoch zu diesem Zeitpunkt bereits festgestellt ob eine entsprechende Tabelle bereits besteht. Die Datei wird zwar angelegt jedoch nur geladen, wenn die Tabelle neu definiert werden soll.

Die zweite Datei enthält die eigentlichen Daten und besteht aus einem Header und einem Datenteil.

```

LOAD DATA
  INFILE *
  TRUNCATE
  CONTINUEIF NEXT(1:1) = '#'
  INTO TABLE FLIES50
  FIELDS TERMINATED BY '|'
  TRAILING NULLCOLS (
    gew_id      NULLIF gew_id = BLANKS,
    gew_kennza  NULLIF gew_kennza = BLANKS,
    gew_name    NULLIF gew_name = BLANKS,
    vor_gew_ke  NULLIF vor_gew_ke = BLANKS,
    laenge,
    user_name  NULLIF user_name = BLANKS,
  SHAPE COLUMN OBJECT
  (
    SDO_GTYPE    INTEGER EXTERNAL,
    SDO_ELEM_INFO VARRAY TERMINATED BY '/'
    (X          FLOAT EXTERNAL),
    SDO_ORDINATES VARRAY TERMINATED BY '/'
    (X          FLOAT EXTERNAL)
  )
  )
  )
  BEGINDATA

```

Dateiauszug 2 : Header der SQL*LOADER- Datei

Der Header beschreibt die Datenstruktur der nachfolgenden Daten mittels PL*SQL und bestimmt deren Zielort innerhalb der Datenbank.

Der Datenteil enthält unter Einhaltung, der durch den Header vorgegebenen Struktur, die eigentlichen Daten.

```

2|238626980000|NN|238626000000|0.56300000|53_FETT|
#2|1|2|1|/
#356696727.000000|541349194.000000|356700983.000000|541348454.000000|35670
4733.000000|541347890.000000|356708080.000000|541347285.000000|356711951.0
00000|541346075.000000|356716710.000000|541346236.000000|356722678.000000|5
41347365.000000|356730260.000000|541348333.000000|356738970.000000|5413483
33.000000|356746068.000000|541348656.000000|356752198.000000|541346559.000
000|/
3|2386252400000|Buttenbach|2386252000000|4.349000000|53_FETT|
#2|1|2|1|/

```

Dateiauszug 3 : Datenteil der SQL*LOADER- Datei

Der Aufruf des Programms erfolgt aus der Magikumgebung heraus. Innerhalb einer Methode wird hierfür eine Prozedur gestartet, welche das Programm durch den im Quellcode 10 beschriebenen Befehl aufruft.

```
system.do_command("shp2sdo -o "+file_name+" "+table_name+" -g  
"+geom_spalte+" -d",output_dir)
```

Quellcode 10 : Aufruf des Unterprogramms <shp2sdo> mittels Magik

Bedingung hierfür ist die Aufnahme des Programmpfads in den Windows Path. Im Programmaufruf ist zunächst der Dateiname des zu wandelnden Shape - Files zu deklarieren. Tabellename und die Bezeichnung der Geometriespalte werden in der zweiten Stufe des Transfers definiert. Das SQL- Skript wird durch den Aufruf von SQL*PLUS ausgelesen und die Tabellendefinitionen interpretiert. Der Aufruf erfolgt innerhalb derselben Prozedur .

```
system.do_command("sqlplus"+a_shp2sdo.oracle_login[1]+"/"+  
a_shp2sdo.oracle_login[2]+" @" +output_dir+sql_file_name)
```

Quellcode 11 : Aufruf von SQL*PLUS mittels Magik

Die Loginparameter werden der bereits definierten Verbindungsspezifikation entnommen und müssen nicht neu eingegeben werden.

Im Anschluss werden die Daten durch das Aufrufen des SQL*LOADERS eingelesen.

```
system.do_command("sqlldr"+a_shp2sdo.oracle_login[1]+"/"+  
a_shp2sdo.oracle_login[2]+" "+output_dir+ctl_file_name+"  
log="+output_dir+table_name+".log")
```

Quellcode 12 : Aufruf des SQL*LOADER mittels Magik

Der Benutzer wird in der vierten Oberfläche über das Fortschreiten des Ladevorgangs unterrichtet.

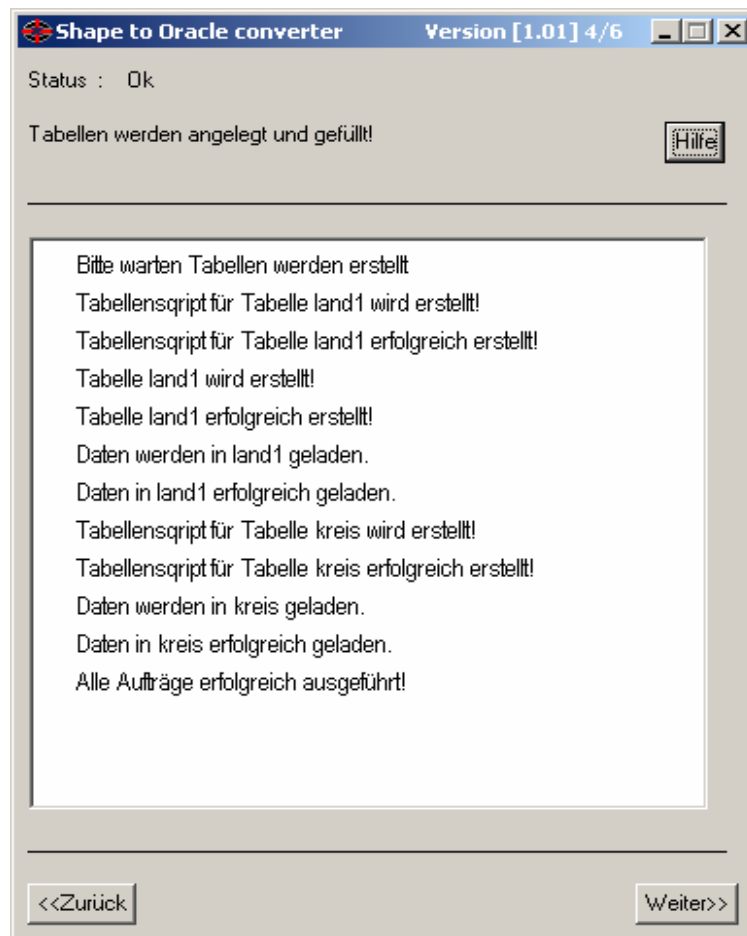


Abbildung 22 : Informationsmenü

Das Ziel, die Daten zu wandeln und in Oracle Spatial zu migrieren, ist damit erreicht. Die Spezifikation des Datenbankadapters sieht aber vor, dass eine automatische Konfiguration vorgenommen werden soll. So besteht der nächste Schritt im Anlegen eines Indexes. Dieser kann in der folgenden Oberfläche zusammen mit dem einem neu anzulegenden primären Schlüssel gewählt werden. Das Programm erzeugt hierfür automatisch pro neu angelegter Tabelle die entsprechenden Eingabefelder.

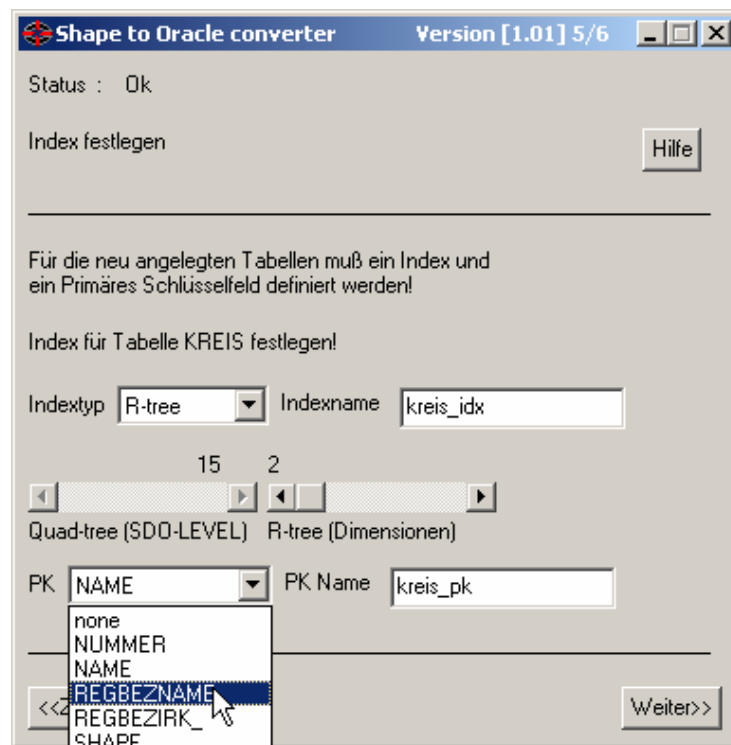


Abbildung 23 : Indexgenerierung

Für die Definition des Index stehen die schon beschriebenen Indextypen R- Tree und Quad - Tree zur Verfügung. Je nach Auswahl werden die Scrollleisten des einen oder anderen Typs aktiv geschaltet. Es besteht die Möglichkeit, keinen Index festzulegen („none“), und diesen nachträglich selbst zu definieren. Ein Arbeiten mit Oracle Daten von Seiten Smallworld's ist jedoch nur mit bestehenden Index möglich. Für den Primärschlüssel werden beim Menüaufbau die neu entstandenen Spalten auf Oracle ausgelesen. Auf diese kann dann ein entsprechender Constraint²⁸ gelegt werden. Für den Fall, dass der Schlüssel sich über mehrere Spalten erstreckt, muss hier das Feld „none“ selektiert werden und der Schlüssel mit Hilfe der Oracle Standardwerkzeuge manuell festgelegt werden. Die Bereitstellung derartiger Funktionen ist bewusst nur eingeschränkt möglich. Die bestehenden Programme von Oracle sollen und können innerhalb einer solchen Anwendung nicht ersetzt werden.

²⁸ Bezeichnet eine Bedingung die auf eine bestimmte Spalte einer Tabelle bezogen ist und für Datenintegrität sorgt.

Die Auswertung der angegebenen Werte erfolgt beim Betätigen der „Weiter-Taste“. Index und Primärschlüssel können über die noch bestehende Verbindung zu Oracle angelegt werden. Hierfür wird auf die Instanz der Klasse `extdb_user`, welche die Datenbankverbindung enthält, die Methode `„exec_sql(Kommando)“` angewendet. Diese Methode übermittelt den in den Klammern stehenden Kommandostring an das aktive Oracle ACP. Der „Create- Befehl“ für den Index sieht folgendermaßen aus.

```
a_sql_string<<"CREATE INDEX "+a_hash[2].value.uppercase+" ON  
"+a_table_name.uppercase+"("+a_geom.uppercase+") INDEXTYPE IS  
MDSYS.SPATIAL_INDEX PARAMETERS  
(SDO_LEVEL="+write_string(a_value)+" SDO_COMMIT_INTERVAL=-1)"  
  
spatial_connection.exec_sql(a_sql_string)
```

Quellcode 13 : Anlegen eines Index mittels Magik

Im blauen Teil wird zunächst der Kommandostring definiert und auf die Variable `„a_sql_string“` gelegt. Der String besteht zum einen aus den obligatorischen SQL Befehlen und den vom Benutzer eingegebenen Daten. Die Ausführung des Befehls übernimmt dann die Methode `„exec_sql“`, die auf die Verbindung angewendet wird (rot markiert).

Das Anlegen des primären Schlüssels erfolgt auf die gleiche Weise.

```
a_pkstring<<"ALTER TABLE "+a_table_name.uppercase+"  
ADD(CONSTRAINT "+a_pkname+" primary key ("+a_pkrow+"))"  
  
spatial_connection.exec_sql(a_pkstring)
```

Quellcode 14 : Anlegen des primären Schlüssels mittels Magik

Damit sind alle Konfigurationen, die auf Oracle vorgenommen müssen, durchgeführt. Abschließend ist ein Konfigurationsskript zur Anpassung der Daten an das Smallworld GIS anzulegen. Die Funktion übernimmt ein weiteres Programm, der „SHP2SDO- Editor“, der jedoch auf die Daten, die während des Transfers anfallen, angewiesen ist. Mit dem Beenden des Programms SHP2SDO über das letzte Menü wird automatisch eine Datei erzeugt, die Grundlage für die Erstellung eines Konfigurationsskriptes zur Anpassung Smallworld's ist.

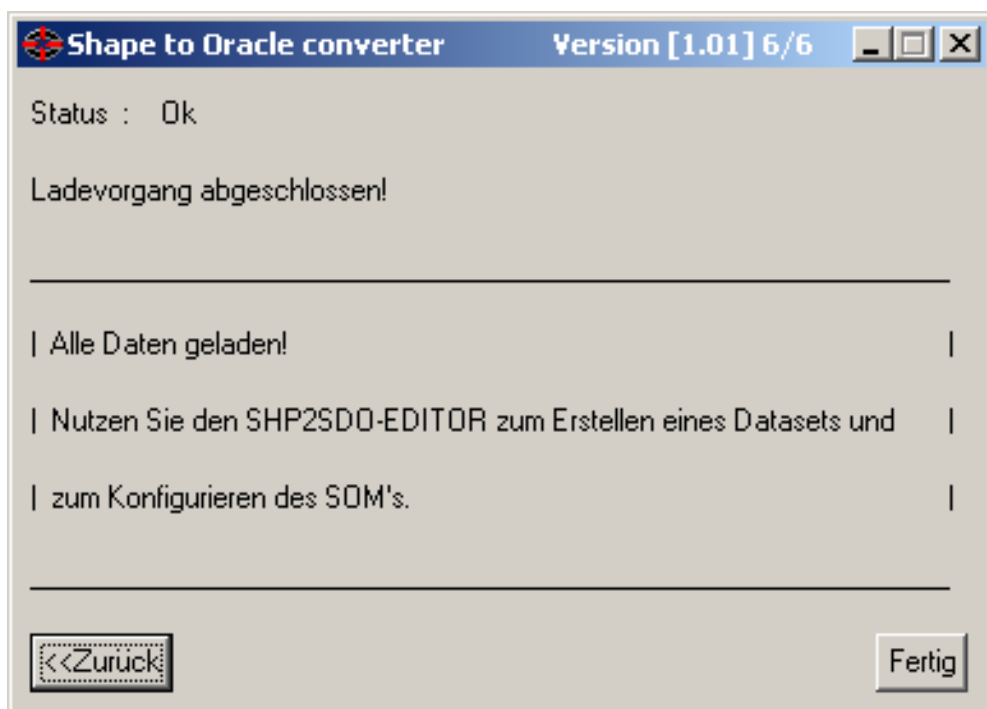


Abbildung 24 : Beenden des Datenbankadapters

Genauer gesagt wird zuerst geprüft, ob die Datei gegebenenfalls bereits besteht. In einem solchen Fall wird die Datei zuerst eingelesen, um die neuen Daten ergänzt und zum Schluss neu geschrieben.

Die Datei hat folgenden Aufbau, und ist im Stammverzeichnis des Programmes SHP2SDO unter dem Namen „ospo_ds.ini“ abgelegt.

```
kreise  
shape  
area  
fliess50  
shape  
chain
```

Dateiauszug 4 : Aufbau der ospo.ini Datei

Ein Block besteht aus dem Namen der Oracle Tabelle, dem Namen der Geometriespalte und dem Geometrietyp. Alle mit dem Datenbankadapter transferierten Themen werden auf diese Weise festgehalten.

5.2 Der SHP2SDO – Editor

Der letzte Schritt, in dem in Abbildung 15 dargestellten Programmablauf, ist die Konfiguration des Smallworld GIS, genauer des OSPO SOM's. Das Anfertigen einer Konfigurationsdatei ist durch die Verwendung der bestehenden Klasse „external_text_output_stream“ unproblematisch. Zur Erzeugung des Skriptes sind sowohl der Name des Datensatzes als auch Informationen über die anzubindenden Tabellen zu definieren. Letztere sind in der Datei ospo_ds.ini enthalten.

Gesteuert wird dieser Schritt durch ein eigenes Programm, das auf dem SHP2SDO Datenbankadapter aufsetzt. Die Abtrennung dieser Funktion vom Datenbankadapter hat den Vorteil, dass Datensätze, auch wenn diese nicht mit dem Datenbankadapter in die Oracle Datenbank transferiert wurden, dennoch ans Smallworld GIS angebunden werden können. Hierfür stellt der SHP2SDO – Editor die Möglichkeit zur Verfügung, Tabellen nachträglich zu publizieren.

Mit dem Starten des Programms durch wird zunächst die eventuell vorhandene ospo_ds.ini Datei ausgelesen, ehe die Oberfläche generiert wird.

```
Magik>w<<shp2sdo_editor.new(g)
```

Quellcode 15 : Initialisierung des Editors

Über das in Abbildung 25 dargestellte Menü kann der Anwender dann verschiedenen Datensätze für Smallworld definieren. So können Themen wie Strassen, Bahnlinien, Schifffahrtswege und andere in einem Datensatz namens Verkehrswege zusammen gefasst werden. Der Dataset – Name, der in der ersten Eingabezeile definiert werden muss, wird später im Smallworld Themenbrowser der Oberbegriff aller ausgewählten Themen sein. Themen, die in der ospo_ds.ini enthalten sind, werden in einer Auswahl automatisch angezeigt.

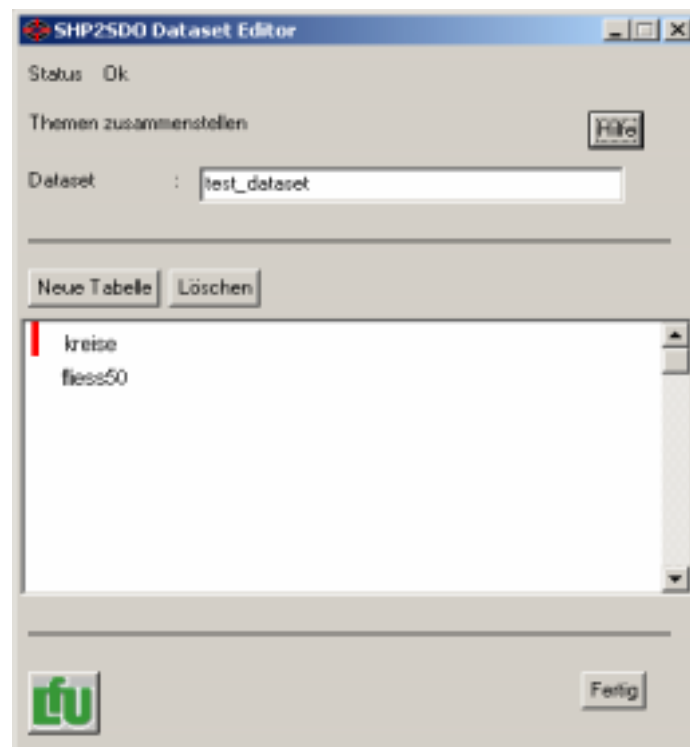


Abbildung 25 : Themenauswahl im Editor

Aus dieser Auswahl können unerwünschte Themen durch Drücken der Löschen-Taste entfernt oder über den Knopf „Neue Tabellen“ hinzugefügt werden.

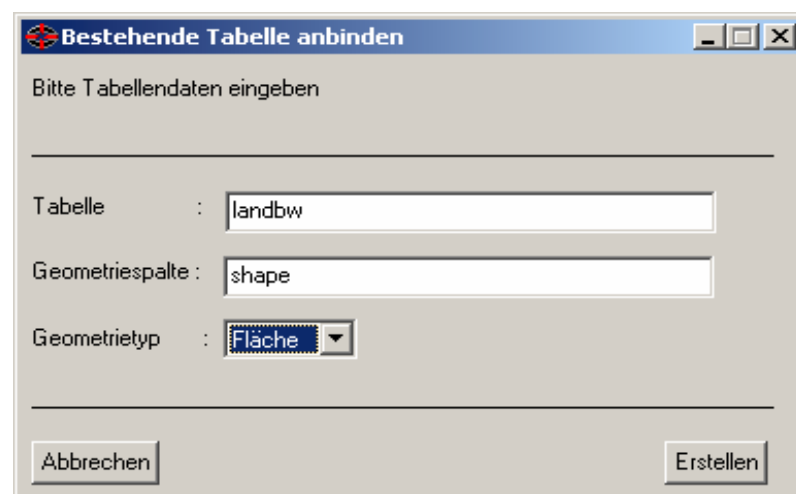


Abbildung 26 : Tabellendefinition

Ist ein Datensatz zusammengestellt, werden die Angaben auf Vollständigkeit überprüft. Programmtechnisch wird der Schreibvorgang durch eine Prozedur implementiert.

```
_global write_ospo_config<<_proc(a_filename,a_ospo_hash,a_dataset)
  ##
  ##Prozedur schreibt ein Konfigurationsskript für den Ospo SOM.
  ##Achtung auf die Struktur der ospohash achten. (siehe shp2sdo
  ##Exemplar definition)
  ##

  _local a_size<<a_ospo_hash.size
  _local runner<<0
  a_stream<<external_text_output_stream.new(a_filename)
  a_stream.write("def_mixin(:ssd_annotation_mixin, {})"
  a_stream.newline()
  a_stream.write("$")
  a_stream.newline()
  _for a_table _over a_ospo_hash.keys()
  _loop
    a_stream.write("def_slotted_exemplar(:"+a_table+"_object,{},
    {:oracle_objects_record, :ssd_annotation_mixin})"
    a_stream.newline()
    a_stream.write("$")
    a_stream.newline()
  _endloop

  ...
```

Quellcode 16 : Anlegen eines Konfigurationsskriptes mittels Magik

Nach der Erzeugung des Skriptes wird es automatisch ins Image geladen und kann hier, durch das Abspeichern desselben, dauerhaft gesichert werden. Der so definierte Datensatz muss nun bei dem Spatial Object Manager angemeldet werden. Mit dessen Funktionalität kann der Datensatz im Anschluss geöffnet werden.

Damit sind alle Phasen der Datenmigration, beginnend mit dem Lesen der Daten auf Smallworld, dem Konvertieren und Laden derselben nach Oracle bis hin zum Konfigurieren der Oracle Datenbank und dem Anlegen eines neuen Datensatzes im Smallworld GIS, abgeschlossen.

5.3 Der direkte Weg der Datenmigration

Bei der Verwendung von externen Programmen gibt es jedoch einige Nachteile. Zum einen besteht eine gewisse Abhängigkeit (z.B. Version des Programms, Version des Datenschemas u.a.), die dann zum Tragen kommt, wenn sich am Beispiel von <shp2sdo> die Form des Typs SDO_GEOMETRY auf Oracle ändert. Einschränkungen treten hinsichtlich Funktionalität oder Performance auf. Der direkte Weg, also von einer Datenbank zu einer anderen Datenbank, ist programmtechnisch der bessere. Mit der Umsetzung einer eigenen Schnittstelle kann man besser auf Veränderungen reagieren. Der Ablauf würde dann innerhalb eines Prozesses erfolgen und mögliche Fehler, die während des Transfers auftreten, könnten mit den Standardroutinen behandelt werden. Aus diesem Grund wurde ein weiterer Prototyp entwickelt, der nur eine Möglichkeit für einen direkten Datentransfer vorsieht. Es handelt sich bei dem Prototypen um einen Verbund von Prozeduren, also keine eigene Klasse, kein Menü, die als Eingabewerte eine Tabelle aus einer Magikcollection haben. Beim Starten einer Smallworldsitzung kann eine bestimmte Datenbankview aufgerufen werden. Ein View besteht aus einer Sammlung (collection) von Tabellen die gemeinsam verwaltet werden. Je nachdem woher die Tabellen stammen oder woraus sie sich zusammensetzen, unterteilt man eine Collection in

- ds_collection - enthält Smallworld Datastore Tabellen
- select_collection - Ansammlung von Tabellen unterschiedlichen Ursprungs
- view_collection - setzt sich aus anderen Collection's zusammen
- extddb_collection - Tabellen stammen aus fremden Datenbanken.

Tabellen die in einer View definiert sind verfügen über verschiedene Feldtypen.

- Physical_fields - reale Felder (Texte, Zahl)
- Logical_fields - durch Methoden, Join's entstanden
- Geometry_fields -enthalten geometrische Daten

Ausgehend von dieser hierarchischen Struktur interessieren für den reinen Datentransfer zunächst nur die sachlichen und die geometrischen Felder. Für den Prototyp ergibt sich damit folgendes allgemeines Ablaufszenarium.

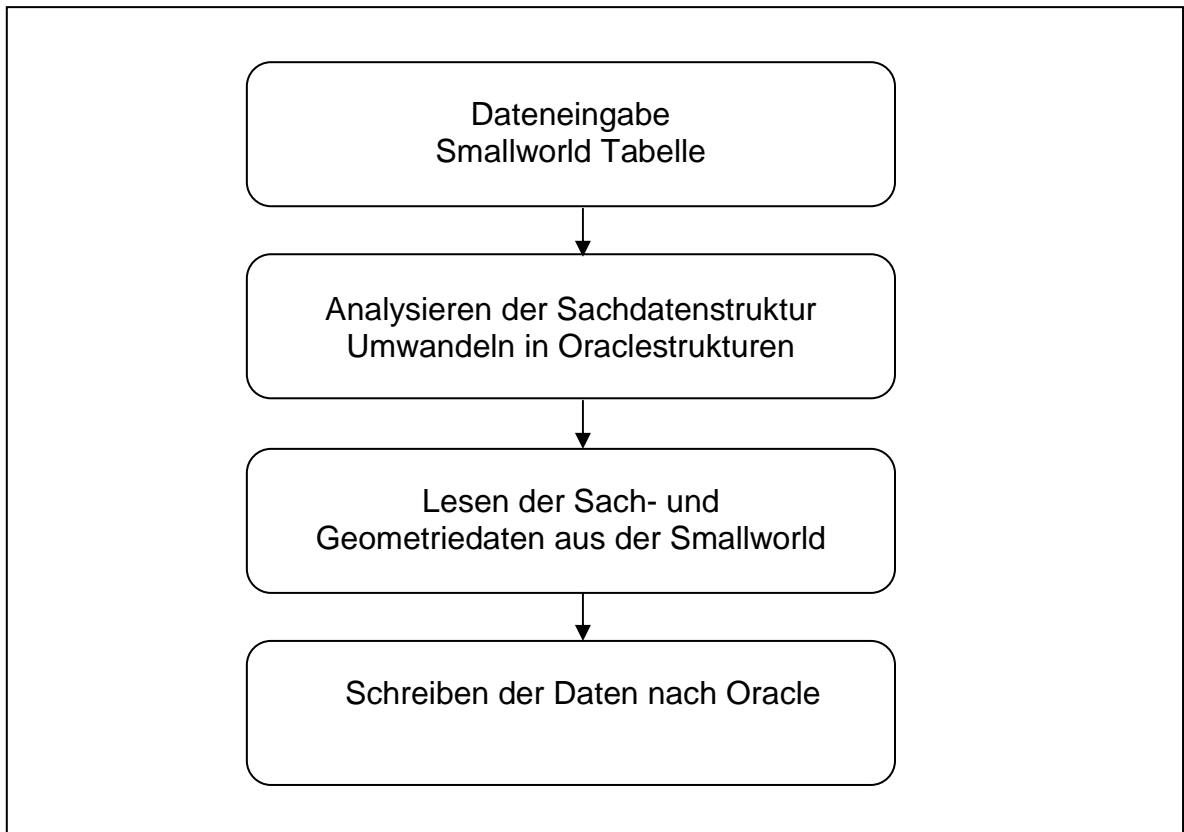


Abbildung 27 : Prozedurenablauf

Der erste Schritt, die Auswahl einer Tabelle, muss vom Anwender übernommen werden.

```
Magik>a_collection<<v.collections[:kreise]
```

Quellcode 17 : Zugriff auf Tabellen mittels Magik

Das Beispiel identifiziert aus der bestehenden Collection die Tabelle Kreise. Mit diesem Eingangswert kann dann die erste Prozedur, „make_sql_file_create_layer“ aufgerufen werden. Innerhalb der Prozedur wird ein Element der Tabelle einer Untersuchung hinsichtlich seiner Struktur unterzogen.

```
a_record<<a_collection.an_element()
key_field<<a_record.key_field_names[1].as_charvec()
table_name<<a_collection.name.as_charvec()
valu_vec<<a_record.values_as_vector()
phys_field<<a_record.physical_field_names
text_fields<<make_text_fields(phys_field,valu_vec)
a_stream<<external_text_output_stream.new(a_filename)
a_stream.write("DROP TABLE "+table_name+";")
a_stream.newline()
a_stream.write("CREATE TABLE "+table_name+" (")
a_stream.newline()
_for a_field,a_hash _over text_fields.keys_and_elements()
_loop
_for a_val,a_type _over a_hash.keys_and_elements()
_loop
a_stream.write(a_field.as_charvec().uppercase+" "+a_type+",")
a_stream.newline()
_endloop
_endloop
a_stream.write("SHAPE MDSYS.SDO_GEOMETRY);")
a_stream.newline()
a_stream.newline()
a_stream.write("ALTER TABLE "+table_name+" ADD CONSTRAINT
"+key_field+table_name.slice(1,3)+"_pk primary key ("+key_field+");")
a_stream.newline()
a_stream.write("commit;")
a_stream.close()
```




Quellcode 18 : Generierung einer SQL- Datei

Das Ergebnis ist eine *.SQL- Datei , welche die gleiche Struktur wie die *.SQL- Datei des Datenbankadapters hat. In der nächsten Phase wird eine *.SQL – LOADER Datei erzeugt, ebenfalls aufgebaut wie die des Datenbankadapters und der Header mit der bereits bekannten Struktur geschrieben. Die Daten werden dann für jedes Objekt der Tabelle einzeln in der Reihenfolge Sachdaten dann Geometriedaten in die LOADER- Datei eingefügt. Das Schreiben der Sachdaten ist dabei problemlos und kann für jeden Datentyp gleichermaßen erfolgen. Schwieriger ist es bei den Geometrien. Hier muss zunächst der Geometriotyp analysiert werden. Die entwickelte Prozedur „make_geom_file_area“ ist zunächst nur für flächenhafte Objekte geeignet. Sie sind jedoch die anspruchsvollsten Geometrien und haben unter Umständen einen sehr komplexen Aufbau. Die Strukturanalyse muss, da die Daten nicht zwischengespeichert, sondern zur

Laufzeit der Prozedur geschrieben werden, die einzelnen Definitionen des Oracle SDO_GEOMETRY Feldtyps hintereinander definieren. So stellt sich zuerst die Frage ob das Objekt ein einzelnes Polygon ist oder sich aus mehreren Teilen zusammensetzt. Für das Schreiben der Koordinaten gilt folgende Regel. An erster Stelle kommt der äußere Polygonring und sollten innerhalb desselben Löcher existieren, folgen sie im Anschluss. Erschwerend ist, dass vor dem Schreiben der Koordinaten der Offset, also die Startordinate eines jeden Elementes der Geometrie vordefiniert werden muss .

```
MDSYS.SDO_GEOMETRY(2007, NULL, NULL,  
MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1,11,1003,1),  
MDSYS.SDO_ORDINATE_ARRAY(  
20000,120000,25000,120000,25000,150000,20000,150000,20000,120000,  
20000,100000,25000,100000,25000,110000,20000,110000,20000,100000)));
```

Quellcode 19 : Aufbau des Objekttyps SDO_GEOMETRY

-  Definition der Geometrieform, Koordinatensystem u.a.
-  Definition des Offset , Geometriotyp u.a.
-  Ordinaten (X,Y,X...)

Hierfür analysiert der Prototyp die Länge der Smallworld Koordinaten Sektoren und berechnet den Startpunkt jedes Elementes der Geometrie. Im Anschluss werden dann die Koordinaten hintereinander aus Smallworld ausgelesen und in die SQL*LOADER – Datei geschrieben.

Beide Dateien werden in einem vorher angegebenen Pfad gespeichert und können mit den Oracle Standardwerkzeugen verarbeitet werden.

Die programmierten Prozeduren könnten in einem weiteren Schritt die Funktionen des Oracle Programms <shp2sdo> ersetzen und an dessen Stelle innerhalb des Datenbankadapters treten.

6. Schlussbetrachtung und Ausblick

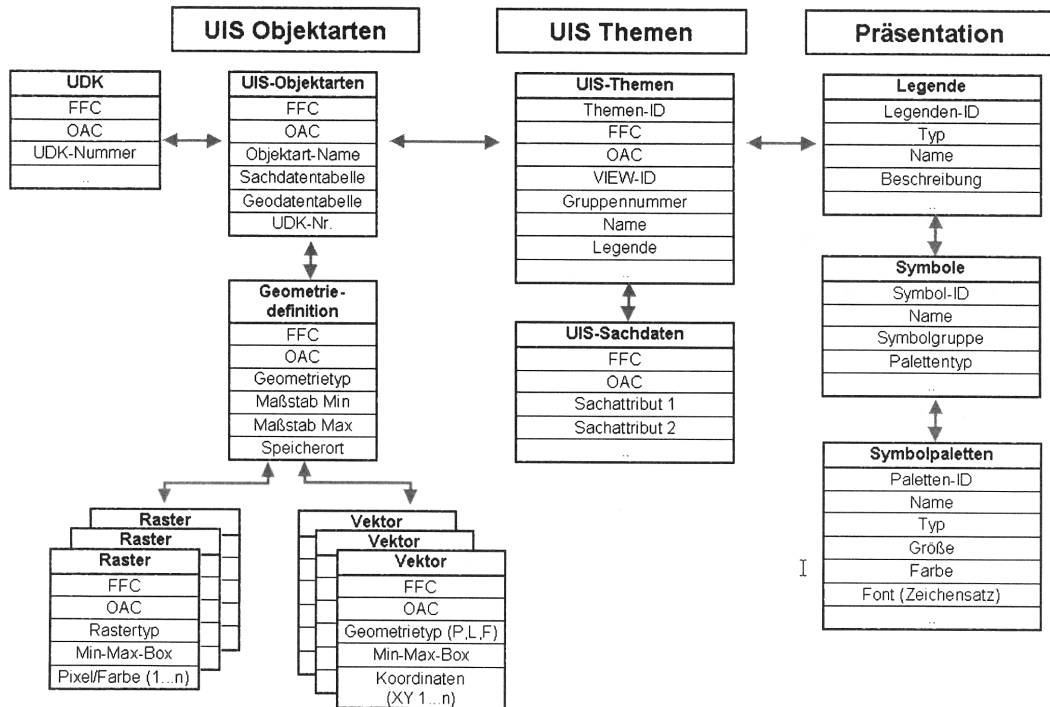
Mit der nahezu vollständigen Beschreibung des Datenverwaltungssystem Oracle Spatial und den geschaffenen Werkzeugen zur Anpassung der Datenbank an das Smallworld GIS als Kernelement des RIPS, konnte das Ziel, eine Entscheidungsgrundlage zu schaffen, erreicht werden. Die Dokumentation des Oracle Spatial Schemas macht deutlich, dass eine vollständige Datenübernahme in dieses System wesentliche Vorteile im Vergleich zum bisherigen System hat. Besonders markant ist hier die strikte Umsetzung der Simple Feature Spezifikation. Viele GIS Hersteller schließen sich diesen Vereinbarungen an und bieten entsprechende Schnittstellen. Die Datenverwaltung wird damit transparent und systemunabhängig. Das Datenbankmanagementsystem bringt zudem eine Vielzahl von Funktionen und Prozeduren mit sich, die nicht durch einzelne Anwendungen neu entwickelt werden müssen. Ein wesentlicher Nachteil ist jedoch, dass Oracle Spatial keine Möglichkeit für die Speicherung von Rasterdaten vorsieht. Für die LfU bedeutet dies, dass Smallworld weiterhin das effektivste Werkzeug für die Verwaltung von Rasterdaten bleibt.

Das entwickelte Konzept geht über das Ziel einer bloßen Anpassung der Datenbank an das Smallworld GIS hinaus. Hierfür würde der programmierte SHP2SDO-Editor ausreichen. Aufgrund des vorhandenen OSPO-SOM ist es nicht notwendig und auch nicht sinnvoll, eine eigene Schnittstelle zwischen Smallworld und Oracle Spatial zu entwickeln. Die Konzeption zeigt zudem, durch die entwickelten Programme, zwei mögliche Wege für eine Datenübernahme nach Oracle Spatial. Mit dem Programm SHP2SDO ist es möglich, Smallworlddaten über die bestehende Shapeschnittstelle nach Oracle zu migrieren und die ebenfalls im RIPS enthaltenen ESRI Shape Dateien direkt umzuwandeln. Die Nutzung der unter Punkt 5.3 vorgestellten Prozeduren ermöglicht einen direkten Datentransfer. Der Anwender ist unabhängig von anderen Programmen und kann den für Polygone entwickelten Quellcode leicht als Vorlage für andere Geometrietypen nutzen.

Auch nach einer Entscheidung für Oracle Spatial als primäres Datenhaltungssystem kann das Smallworld GIS weiterhin genutzt werden. Der dokumentierte Fehler bezüglich der Verwendung von Multipolygonen stellt den Anwender - speziell die LfU - vor große Probleme. Smallworld ist jedoch dabei, die Funktionalität des OSPO-SOM weiterzuentwickeln, und als Partner von Oracle derartige Fragen nachhaltig zu beantworten.

7. Anhang

7.1 Ausschnitt des Oracle Datenbankschemas der LfU



7.2 OSPO- SOM Konfigurationskript

Zur Anpassung des Oracle Spatial SOM verwendet Smallworld ein Konfigurationskript. (vgl. 4.1.2.1 Der Oracle Spatial SOM)

```
def_mixin(:ssd_annotation_mixin, {})  
$  
def_slotted_exemplar(:strassen_object, {}, { :oracle_objects_record,  
:ssd_annotation_mixin})  
$  
def_slotted_exemplar(:spa99_object, {}, { :oracle_objects_record,  
:ssd_annotation_mixin})  
$  
def_slotted_exemplar(:fliess_object, {}, { :oracle_objects_record,  
:ssd_annotation_mixin})  
$  
_pragma(classify_level=advanced, topic={oracle_som}, usage={redefinable})  
oracle_objects_dataset.define_shared_constant(  
:dataset_instance_metadata,  
property_list.new_with(  
:spa99, property_list.new_with(  
:collections, property_list.new_with(  
:spa, property_list.new_with(  
:geom_fields, property_list.new_with( :shape, :area),  
:exemplar, spa99_object)),  
:short_trans, _true ),  
:stra99, property_list.new_with(  
:collections, property_list.new_with(  
:strassen, property_list.new_with(  
:geom_fields, property_list.new_with( :shape, :chain),  
:exemplar, strassen_object)),  
:short_trans, _true )),  
:private )  
$
```


7.3 Die ospo_ds.ini Datei

Die ospo_ds.ini Datei speichert Angaben zu allen mit dem Datenbankadapter bearbeiteten Themen. Sie ist Grundlage für die Erstellung eines neuen Smallworld Dataset und enthält alle Informationen, für das Anlegen des Konfigurationskriptes. (vgl. 5.2 Der Datenbankadapter)

```
kreise  
shape  
area  
test2  
shape  
point  
fliess50  
shape  
chain
```

7.4 Die SQL Datei

Die Datei dient dem Anlegen neuer Tabellenstrukturen auf Oracle. Sie definiert für jedes neue Smallworld Thema, das in die Oracledatenbank überführt werden soll, die Tabelle mit Spalten und Wertebereich. (vgl. 5.2 Der Datenbankadapter)

```
CREATE TABLE regbezirk (  
  BBOX_YH NUMBER,  
  BBOX_XH NUMBER,  
  BBOX_YL NUMBER,  
  RECHTSWERT NUMBER,  
  NUMMER NUMBER,  
  HOCHWERT NUMBER,  
  BBOX_XL NUMBER,  
  NAME VARCHAR2(32),  
  SHAPE MDSYS.SDO_GEOMETRY);
```

7.5 Die LOADER Datei

Die LOADER Datei enthält alle Daten eines Objektes, welche in die vorher definierte Struktur eingelesen werden sollen. Dabei werden die einzelnen Datensätze durch Steuerzeichen voneinander getrennt hintereinander in eine ASCII- Datei vom Typ „*.ctl“ abgelegt. Mittels Oracle SQL*LOADER können die Daten dann in einem Zug in die im Header angegebene Tabelle migriert werden. (vgl. 5.2 Der Datenbankadapter)

```

LOAD DATA
INFILE *
TRUNCATE
CONTINUEIF NEXT(1:1) = '#'
INTO TABLE FLIES50
FIELDS TERMINATED BY '|'
TRAILING NULLCOLS (
  gew_id      NULLIF gew_id = BLANKS,
  gew_kennza  NULLIF gew_kennza = BLANKS,
  gew_name    NULLIF gew_name = BLANKS,
  vor_gew_ke  NULLIF vor_gew_ke = BLANKS,
  laenge,
  user_name   NULLIF user_name = BLANKS,
  SHAPE COLUMN OBJECT (
    SDO_GTYPE   INTEGER EXTERNAL,
    SDO_ELEM_INFO VARRAY TERMINATED BY '/'
    (X          FLOAT EXTERNAL),
    SDO_ORDINATES VARRAY TERMINATED BY '/'
    (X          FLOAT EXTERNAL)
  ))
BEGINDATA
2|2386269800000|NN|2386260000000|0.563000000|53_FETT|
#2|1|2|1|/
#356696727.000000|541349194.000000|356700983.000000|541348454.000000|35670
4733.000000|541347890.000000|356708080.000000|541347285.000000|356711951.0
00000|541346075.000000|356716710.000000|541346236.000000|356722678.000000|5
41347365.000000|356730260.000000|541348333.000000|356738970.000000|5413483
33.000000|356746068.000000|541348656.000000|356752198.000000|541346559.000
000|/
4|2386252420000|NN|2386252400000|0.475000000|53_FETT|
#2|1|2|1|/
#356478257.000000|541702523.000000|356498459.000000|541681789.000000|35651
0844.000000|541667947.000000|/....

```

7.6 Quellcode des Datenbankadapters

```
##% text_encoding = iso8859_1
#-----
#
#> Name:    shp2sdo.magik
#
#> Description: Convert shape to oracle sdo and configured ospo som
#
#> Author:   Bastian Ellmenreich
#
#> Date:    30. Aug 2001
#
# Copyright (C) 2001 by Landesanstalt für Umweltschutz, Karlsruhe
#
#-----
#> Exemplar
#-----
# Entfernen des Exemplars vor Neudefinition. Nach der
# Entwicklungsphase zu entfernen!
#_block
#   _if !current_package![:shp2sdo] _isnt _unset
#   _then
#       remove_exemplar(:shp2sdo)
#   _endif
#_endblock
$
_pragma(classify_level=basic, topic={shp2sdo})
def_slotted_exemplar(:shp2sdo,
  ##
  ## Definiert neues Exemplar von shp2sdo.
  ##
  {
    {:file_list,_unset},

    #Hält alle zu ladenden Files aus dem file-select Fenster

    {:label_hold,_unset},

    #Hashtable hält alle wichtigen Eingabeinformationen.

    {:oracle_files,_unset},

    #Rope zum halten von Informationen über die Oracle_DB.

    {:window,_unset},

    #Rope zum halten aller Fenster. Ermöglicht "Vor / Zurück" - Funktion

    {:buffer,_unset},

    #Hält Daten als Zwischenspeicher (Ablage)
```

```
        {:oracle_login,_unset},
        #Rope zum halten von Datenbank Login Informationen.

        {:new_tables,_unset},
        #Hashtable zum halten von Informationen über neu anzulegende Tabellen.

        {:exist_tables,_unset},
        #Hashtable hält alle Informationen über bestehende Tabellen.

        {:ospo_ds,_unset},
        #Hashtable hält alle Informationen über bereits an SW angebundene Spatial
        Tabellen.
        {:ospo_tables,_unset},
        #Rope hält temporär die Tabellennamen aller angebundenen Tabellen.

        {:status_list,_unset}

        #Rope zum halten aller den Statusberichte über das erfolgreiche Anlegen von
        Oracletabellen.

    },
    {:model})
$
#-----
#> Slot Zugriff
#-----

    _pragma(classify_level=basic, topic={shp2sdo})
    shp2sdo.define_slot_access(
        :file_list,
        ##
        ## Setzt diesen Slot auf writeable (beschreibbar)
        ##
        ##
        :writable)
$
    _pragma(classify_level=basic, topic={shp2sdo})
    shp2sdo.define_slot_access(
        :ospo_tables,
        ##
        ## Setzt diesen Slot auf writeable (beschreibbar)
        ##
        ##
        :writable)
$
```

```
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :window,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :exist_tables,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :ospo_ds,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :oracle_login,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :new_tables,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :oracle_files,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
```

```
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :label_hold,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :buffer,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :oracle_login,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
_pragma(classify_level=basic, topic={shp2sdo})
shp2sdo.define_slot_access(
    :status_list,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
#-----
#> Objektklasse initialisieren
#-----
##
## Mit shp2sdo.open() wird auf model.open() zurückgegriffen und
## eine neue Instanz der Klasse shp2sdo erzeugt.
##

_pragma(classify_level=basic, topic={sas}, usage = {external})
_method shp2sdo.new ( a_grs )
##
## Initialisiert neues shp2sdo.
##

    >> _clone.init ( a_grs )

_endmethod
$
```

```

_pragma(classify_level=basic, topic={sas}, usage = {internal})
_method shp2sdo.init ( a_grs )
  ##
  ## Initialises the Configuration Interface Agent.
  ##

  #graphic system wird festgelegt
  _global !grs!
  !grs!                << a_grs

  #legt message accessor fest
  .message_accessor   << message_handler.new(:shp2sdo)

  >> _super.init ()

_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.activate_in(f)
  ##
  ## Fenster konfiguration
  ##

  #setzt den Status auf ok
  _self.status<<"Ok"

  .window<<rope.new()

  #setzt den titel des frames
  _self.title<<_self.message(:title1)

  #legt die position fest
  f.position<<pixel_coordinate(0,0)
  .window[1]<<f

  p << panel.new(f)
  label_item.new(p,_self.message(:status))
  label_item.new(p,_self.status_string,:model,_self, :aspect,:status_string)
  p.start_row()
  label_item.new(p,_self.message(:file_selection))
  label_item.new(p," "*41)
  button_item.new(p,_self.message(:help_label),_self,:helptext1|(),:help_id,:help_help)
  p.start_row()
  label_item.new(p,"_*30)
  p.start_row()
  button_item.new(p,_self.message(:select_file),_self,:file_select_mask|(),:help_id,:file_selection_help)
  button_item.new(p,_self.message(:deselect_file),_self,:deselect|(),:help_id,:deselection_help)

```



```

#Legt den Startwert für file_list fest
.file_list<<rope.new_with(_self.message(:no_choice))
list_view.new(_self, f, :file_list,:select_one|())
s<<panel.new(f)
s.start_row()
s.start_row()
label_item.new(s,"_"*30)
s.start_row()
t << panel.new(f)
t.start_column()
last<<image_button_item.new_safe(t,
"D:\smallworld\sonstiges\eigene\shp2sdo\bitmaps\lfugross.bmp",_self, :info|(),:help_id,
:info_help)
t.start_column(last,270)
button_item.new(t,_self.message(:forward_label),_self,:step2|(),:help_id,:forward_help)

>> _self
_endmethod

$
_pragma(classify_level=basic, topic={shape_import})
_method shp2sdo.help_text1()
##
##Methode des Help-button zum öffnen der Hilfe
##
>>_self.help_wanted(:step1_help)
_endmethod

$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.file_list(new_val)
##
## Change Methode für die List View. Werden neue Werte in die List View aufgenommen,
## findet eine Aktualisierung statt.
##
.file_list << new_val
_self.changed(:file_list)
_endmethod

$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.info()
##
##Definiert das info fenster
##
>>_self.show_alert(_self.message ( :info))

_endmethod

$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.file_select_mask()
##
##Ermöglicht die Fileauswahl über Windows-Fenster. Achtung geht nur mit sas_filler!
##

mask << "".concatenation ( "Shape-Datei" ,
" (*.shp)|*.shp|",
"Datei wählen",
" (*.*)|*.*||" )

```

```

_local a_filename << _self.get_filename_shape ( _self.message(:mask_label),
                                             system.getenv("SW_SHAPE_PATH"), _true, mask )

_if a_filename _is _unset _orif          # Check if we have a filename
  a_filename.size _is 0
_then
  _return                                # When nothing do nothing
_endif

_local a_file_rope<<a_filename
# ruft Methode zum überprüfen der ausgewählten file_name`s auf.
_self.check_select_files(a_file_rope)
# setzt erstes Element von .file_list(enthält alle File einträge) auf q
_local q<<.file_list[1]
# spaltet Pfad und Dateinamen
(fn, pt) << system.pathname_components ( q )
#setzt den Umgebungspfad auf pt, damit wird bei erneuter Fileauswahl vom aktuellen
#Verzeichnis gestartet.
system.putenv("SW_SHAPE_PATH",pt)
_endmethod
$
#-----
#   Die folgenden Methoden anwenden wenn sas_filler aus shape_som nicht vorhanden!   |
#-----

#_pragma(classify_level=basic, topic={shp2sdo})
# _method shp2sdo.file_select_mask ()
#   _local
a_file_selection<<file_selection.new(_self,_self.message(:mask_label),:get_filename_shp|(),_un
set,:directory,"c:\",:filter,"*.shp")
#   >>a_file_selection.activate()
#_endmethod
#$

#_pragma(classify_level=basic, topic={shp2sdo})
# _method shp2sdo.get_filename_shp(a_filename)
#   write(a_filename)
#   _if a_filename _is _unset _orif          # Check if we have a filename
#     a_filename.size _is 0
#   _then
#     _return                                # When nothing do nothing
#   _endif
#
#
#   _local a_file_rope<<rope.new()
#     a_file_rope[1]<<a_filename
#
#
#   # ruft Methode zum überprüfen der ausgewählten file_name`s auf.
#
#   _self.check_select_files(a_file_rope)
#
#   # setzt erstes Element von .file_list(enthält alle File einträge) auf q
#
#   _local q<<.file_list[1]
#
#

```

```
# # spaltet Pfad und Dateinamen
#
# (fn, pt) << system.pathname_components ( q )
#
# #setzt den Umgebungspfad auf pt
#
# system.putenv("SW_SHAPE_PATH",pt)
#
# _endmethod
#
#
#
#
#$
  _pragma(classify_level=basic, topic={shp2sdo})
  _method shp2sdo.get_filename_shape ( title, directory, multiple?, mask )
    ##
    ## Startet das Interface für die Fileselection
    ##

    _local a_file_menu << sas_filer.new ( title, directory, mask, multiple?, .top_frame )
    _try _with cond
      #
      # Make the startup of the dialog safe, because sas_drop may be
      # missing in the <product>/etc/x86 directory.
      #
      a_result << a_file_menu.do_dialog ()

    _when error
      #
      # Notofy user that the acp for the sas_filer is missing. This
      # acp is actually the sas_drop executable which serves for
      # opening files as well.
      #

      condition.raise ( :user_error,
                        :string, _self.message ( :no_sas_drop_found ) )
    _endtry

    _if multiple? _is _false
    _then
      _if a_result _is _unset _orif
      a_result.size _isnt 1
      _then
        _return _unset
      _endif

      _return a_result [ 1 ]
    _endif

    _return a_result
  _endmethod
$
  _pragma(classify_level=basic, topic={shp2sdo})
  _method shp2sdo.new_select_file(a_file_rop)
    ##
    ##Setzt ersten Fileeintrag in .file_list. (.file_list enthält alle ausgewählten Files!)
    ##
```

```

_local w<<.file_list.size
  _local x<<a_file_rope.size
  _local z<<1

  # Prüft ob das erste Element in .file_list = none ist.

  _if .file_list[1]=_self.message(:no_choice)
  _then
  .file_list<<a_file_rope
  _self.file_list(a_file_rope)
  _else
  >>_self.add_select_file(a_file_rope)
  _endif
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.add_select_file(a_file_rope)
  ##
  ##Fügt in .file_list Elemente neue Einträge hinzu.
  ##
  _local w<<.file_list.size
  _local x<<a_file_rope.size
  _local z<<1
  _loop
  _if z<=x
  _then
  _local q<<w+z
  .file_list[q]<<a_file_rope[z]
  z<<z+1
  _else
  _self.file_list(.file_list)
  _return
  _endif
  _endloop
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.select_one(ind)
  ##
  ##Setzt angeklickte Fileeinträge in den Slot .buffer.
  ##
  .buffer<<rope.new()
  .buffer<<.file_list[ind]
  _endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.deselect()
  ##
  ##Methode zum deselektieren der Einträge in .file_list. Greift dabei auf .buffer zu.
  ##Nach dem Löschen werden die Elemente in .file_list neu angeordnet.
  ##
  _local i<<1
  _local t<<rope.new()

  _for f _over .file_list.elements()
  _loop

```

```
                _if f=.buffer
                _then
                _continue
                _else
                t[i]<<f
                i<<i+1
                _endif
            _endloop
        _if t.size=0
        _then
        t[1]<<_self.message(:no_choice)
        _endif
        .file_list<<t
        _self.file_list(.file_list)
        .buffer<<_unset
    _endmethod
$
    _pragma(classify_level=basic, topic={shp2sdo})
    _method shp2sdo.check_select_files(a_file_rope)
        ##
        ##Überprüft, ob Dateien mehrmals in .file_list aufgeführt sind.
        ##
        _local n<<rope.new()
        _local c<<_false

        _for t _over a_file_rope.elements()
            _loop
                _for m _over .file_list.elements()
                    _loop

                        #Wenn File schon in der Auswahl wird show_alert ausgelöst

                        _if t=m
                        _then
                        _self.show_alert
                        (_self.message(:alert1)+m+_self.message(:alert2))
                        c<<_true
                        _endif
                    _endloop
                _endloop
            _if c=_false
            _then
                w<<a_file_rope.size
                e<<.file_list.size
                _if .file_list[1]=_self.message(:no_choice)
                _then
                e<<0
                _endif
                r<<w+e
                _if r>5
                _then
                c<<_true
                _self.show_alert(_self.message(:too_much_files))
                _endif
            _endif
        _endmethod
endmethod
```

```

_endif
  _if c=_false
  _then
    _self.new_select_file(a_file_rope)
  _endif
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.step2()
  ##
  ##Startet zweites Fenster zur Dateneingabe. Für jeden ausgewählten File werden eigene
  Label und Choice - Items erzeugt.
  ##

  #Überprüfung ob überhaupt Files ausgewählt wurden.
  _local q
  _if .file_list[1] = _self.message(:no_choice)
  _then
    _self.show_alert(_self.message(:no_choice))

  #Wenn Files ausgewählt wird das neue Fenster erstellt.
  _else
    _self.close()
    .label_hold<<hash_table.new()
    .window[2]<<frame.new(_self.message(:title2))
    p<<panel.new(.window[2])
    _self.status<<"Ok"
    label_item.new(p,_self.message(:status))
    label_item.new(p,_self.status_string,:model,_self,:aspect,:status_string)
    p.start_row()
    label_item.new(p,_self.message(:table_desc))
    label_item.new(p,"  "*33)

    button_item.new(p,_self.message(:help_label),_self,:helptext2|(),:help_id,:help_help)
    p.start_row()
    label_item.new(p,"  "*30)
    p.start_row()

    _for i_over _self.file_list.elements()
    _loop
      w<<1
      p.start_row()
      label_item.new(p,_self.message(:sub_string1)+" "+i+"
"+_self.message(:sub_string2))
      p.start_row()
      q<<.label_hold[i] <<hash_table.new()
      q[w]<<text_item.new(p,_self.message(:table_name))
      w<<w+1
      p.start_row()
      q[w]<< text_item.new(p,_self.message(:geom_row))

    _endloop
    p.start_row()
    label_item.new(p,"  "*30)
    p.start_row()
    .buffer<<rope.new()

```

```

.buffer[1]<<button_item.new(p,_self.message(:backward_label),_self,:move_back|(),:help_id,:b
ackward_help)
      label_item.new(p,"  **38)

      .buffer[2]<<button_item.new(p,_self.message(:forward_label),_self,:step2_control_input|()
|,:help_id,:forward_help)
      .window[2].activate()
    _endif
  _endmethod
$
_pragma(classify_level=basic, topic={shape_import})
_method shp2sdo.help_text2()
  ##
  ##Methode des Help-button zum öffnen der Hilfe
  ##
  >>_self.help_wanted(:step2_help)
_endmethod
$

_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.step2_control_input()
  ##
  ##Kontrolliert ob alle Felder im Menü gefüllt wurden und startet wenn alles ok ist das
Oracle Login Panel.
  ##
  _local loginvar
  _local c<<_true
  _local w<<1
  _for k,e _over .label_hold.keys_and_elements()
    _loop
      _for i _over e.elements()
        _loop
          _if i.value=""
            _then
              _self.show_alert(_self.message(:sub_string3)+
"+w.write_string+" "+_self.message(:sub_string4))
              c<<_false
              w<<w+1
            _else
              w<<w+1
            _endif
          _endloop
        _endloop

      _if spatial_connection _isnt _unset
        _then
          loginvar<<spatial_connection.connected?()
        _else
          loginvar<<_false
        _endif
        #Wenn alle Felder ausgefüllt sind und eine Connection besteht wird erneuter Login
übersprungen.

        _if c=_true _andif loginvar _is _true
          _then

```

```

    _self.oracle_login<<rope.new()
    _self.oracle_login[1]<<spatial_connection.connect_spec[:dbusername]
    _self.oracle_login[2]<<spatial_connection.connect_spec[:dbpassword]
    _self.get_oracle_params()
  _endif

  #Wenn alle Felder ausgefüllt sind und noch kein Login erfolgte, wird das Oracle Login
  Panel gestartet.
  _if c=_true _andif loginvar _is _false
  _then
    _self.status<<_self.message(:connect_db)
    _self.connect_oracle()
  _endif
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.connect_oracle()
  ##
  ##Prüft ob log_in auf DB ok ist. Dabei wird eine Verbindung aufgebaut.
  ##
  _local o<<_self
  _dynamic !grs!
  _local g << !grs!
  _local sw_view<<g.current_dsview
  _local login << g.activate_sub_menu(:oracle_login_panel)
  login.action <<
    _proc(p)
      _import o
      _import sw_view

      user << p.user_name
      passwd << p.password.value
      host << p.host_name

      o.int!connect_to_oracle8(user, passwd + "@" + host)

    _endproc

  login.clear()
  login.activate()
  login.title << o.message(:oracle_login)
  login.top_frame.raise()

_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.int!connect_to_oracle8(user,passwd)
  ##
  ##Connectscript zum login auf Oracle. Es werden keine tabellen angebunden da User
  nicht festgelegt.
  ##Die login Daten werden nach oracle_login geschrieben.
  ##
  _handling warning _with _proc(cond) cond.report_contents_on(!terminal!) _endproc
  _self.oracle_login<<rope.new()

```



```

    _self.oracle_login[1]<<user
    _self.oracle_login[2]<<passwd
    _local g<<!grs!
    _local v<<g.current_dsview
    _local spa_spec << extdb_user.connect_template(
        :dbtype, :oracle8,
        :connect_name, "External Database",
        :command, rope.new_with("sworacle8"),
        :dbusername, user,
        :dbpassword, passwd,
        :no_mandatory_field_validation, _true,
        :command, rope.new_with("sworacle8"),
        :configuration, (property_list.new_with(:max_table_fetch,10)))

    _global spatial_connection
    spatial_connection << extdb_user.new(spa_spec )
    v.add_external_database(spatial_connection)
    g.ace_control.reset()
    _self.status<<"Ok"
    #spatial_connection.discard(v)

    _self.get_oracle_params()
    _endmethod
$
    _pragma(classify_level=basic, topic={shp2sdo})
    _method shp2sdo.get_oracle_params()
        ##
        ##Methode zum abfragen von Tabelleninformationen aus Oracle. Tabellen und zugehörige
        Spalten werden abgefragt.
        ##
        _local c<<_false
        _local exist_table_hash
            exist_table_hash<<hash_table.new()
        _local new_table_hash
            new_table_hash<<hash_table.new()
        _local a_select
        _local a_select2

        _for key,element _over .label_hold.keys_and_elements()
            _loop

                #Prüft ob eingebene Tabelle schon vorhanden
                a_select<<spatial_connection.sql_select("select table_name from
                cat where table_name='"+element[1].value.uppercase+"'")
                a_select<<a_select.get()
                _if a_select_isnt_unset
                _then
                    a_select_table<<a_select.table_name.lowercase
                    c<<_true
                    a_select_table_sym<<a_select_table.as_symbol()
                    _local
                a_hash<<exist_table_hash[a_select_table_sym]<<hash_table.new()
                    a_view << gis_program_manager.databases[:gis]

                #wenn vorhanden werden Spalten ausgelesen

```

```

_local tab<<spatial_connection.int!dd_fetch_table_columns_sdo(_self.oracle_login[1].
uppercase,a_select.table_name,a_view)
                                _for i _over tab.elements()
                                _loop
                                a_hash[i.column_name]<<i.data_type
                                _endloop

                                _elif a_select _is _unset
                                _then

new_table_hash[element[1].value]<<hash_table.new()

                                _endif

                                _endloop

_self.step3(c,exist_table_hash,new_table_hash)
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.step3(c,exist_table_hash,new_table_hash)
##
##Öffnet neues Fenster, welches Informationen über die anzubindenden Tabellen gibt.
##

.new_tables<<new_table_hash
.exist_tables<<exist_table_hash
.window[2].deactivate()
.window[3]<<frame.new(_self.message(:title3))
p<<panel.new(.window[3])
_self.status<<"Ok"
label_item.new(p,_self.message(:status))
label_item.new(p,_self.status_string,:model,_self, :aspect,:status_string)
p.start_row()
label_item.new(p,_self.message(:table_info))
label_item.new(p," "*34)
button_item.new(p,_self.message(:help_label),_self,:helptext3|(),:help_id,:help_help)
.status_list<<rope.new()
p.start_row()
label_item.new(p,"_*30)
p.start_row()
list_view.new(_self,p,:status_list,_unset,_unset,15,60,:none)
p.start_row()
label_item.new(p,"_*30)
p.start_row()
.buffer<<rope.new()
.buffer[1]<<button_item.new(p,_self.message(:backward_label),_self,:move_back|(),:help_
id,:backward_help)
label_item.new(p,"_*40)
.buffer[2]<<button_item.new(p,_self.message(:forward_label),_self,:step4|(),:help_id,:forwa
rd_help)
.buffer[2].visibility<<_false
.window[3].activate()
e<<0
_local a_buffer_rope<<rope.new()

```

#wenn Tabellen schon auf Oracle existieren, werden die dort vorhandenen Spalten und deren Wertebereiche angezeigt.

```

_if exist_table_hash _isnt _unset
_then
  _for a_table,a_row_col _over exist_table_hash.keys_and_elements()
  _loop
    e<<a_buffer_rope.size+1
    _if e _isnt 1
    _then
      a_buffer_rope[e]<<write_string("")
      e<<e+1
    _endif
    a<<a_table.write_string.uppercase
    a_buffer_rope[e]<<write_string(_self.message(:sub_string5)," ",a,"
",_self.message(:sub_string6))
    e<<e+1
    a_buffer_rope[e]<<write_string(_self.message(:sub_string7))
    e<<e+1
    a_buffer_rope[e]<<write_string("")
    _for a_row,a_data_type _over a_row_col.keys_and_elements()
    _loop
      _if a_row _isnt _unset
      _then
        e<<a_buffer_rope.size+1
        a_buffer_rope[e]<<write_string(a_row," : ",a_data_type)
        _if a_data_type = "SDO_GEOMETRY"
        _then
          _for a_hash _over _self.label_hold.elements()
          _loop

              #stimmt der eingegebene
Geometriespaltenname nicht mit dem in Oracle überein wird der eingegebene Name
überschrieben.
          _if a_hash[1].value.uppercase =
write_string(a_table).uppercase _andif a_hash[2].value.uppercase ~= a_row.uppercase
          _then
            _self.list3(a_buffer_rope)

            _self.show_alert(_self.message(:sub_string8)+" "+a_hash[2].value.uppercase+"
"+_self.message(:sub_string9)+a_row.uppercase+" "+_self.message(:sub_string10))

            a_hash[2].value<<a_row.uppercase
            _endif
          _endloop
        _endif
      _endif
    _endloop
  _endloop
_endif
_if new_table_hash.size _isnt 0
_then
  _for i _over new_table_hash.keys()
  _loop
    e<<a_buffer_rope.size+1
    a_buffer_rope[e]<<" "

```

```

                e<<e+1
                a<<i.write_string.uppercase
                a_buffer_rope[e]<<write_string(_self.message(:sub_string5)," ",a,"
",_self.message(:sub_string11))
            _endloop
        _endif

        _self.list3(a_buffer_rope)
        .buffer[2].visibility<<_true
        _endmethod
$
_pragma(classify_level=basic, topic={shape_import})
_method shp2sdo.help_text3()
    ##
    ##Methode des Help-button zum öffnen der Hilfe
    ##
    >> _self.help_wanted(:step3_help)
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.list3(new_val)
    ##
    ## Change Methode für die List View. Werden neue Werte in die List View aufgenommen,
    ## findet eine Aktualisierung statt.
    ##
    .status_list << new_val
    _self.changed(:status_list)
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.move_back()
    ##
    ##Methode die jeweils feststellt welches Fenster gerade aktive ist, dieses dann schließt
    und dessen vorgänger reaktiviert.
    ##Ermöglicht die "Zurück" - Funktion
    ##
    _local a_number
    _for k,e _over .window.keys_and_elements()
    _loop
        _if e~=_unset
        _then
            _if e.active? _is _true
            _then
                a_number<<k
            _endif
        _endif
    _endloop
    .window[a_number].deactivate()
    a_number<<a_number-1
    _if .window[a_number]=_unset
    _then
        a_number<<a_number-1
    _endif
    .window[a_number].activate()

_endmethod

```

```

$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.step4()
  ##
  ##Öffnet 4-tes Fenster. Weiterhin werden die Skripte für das Importieren der Daten nach
  Oracle erzeugt. Alle Arbeitsschritte werden aufgelistet.
  ##

  .oracle_files<<rope.new()
  .window[3].deactivate()
  .window[4]<<frame.new(_self.message(:title4))
  p<<panel.new(.window[4])
  _self.status<<"Ok"
  label_item.new(p,_self.message(:status))
  label_item.new(p,_self.status_string,:model,_self,:aspect,:status_string)
  p.start_row()
  label_item.new(p,_self.message(:sub_string12))
  label_item.new(p," "*21)
  button_item.new(p,_self.message(:help_label),_self,:helptext4|(),:help_id,:help_help)
  p.start_row()
  label_item.new(p," "*30)
  p.start_row()
  .oracle_files[1]<<_self.message(:sub_string13)
  list_view.new(_self,p,:oracle_files,_unset,_unset,15,65,:none)
  p.start_row()
  label_item.new(p," "*30)
  p.start_row()
  .buffer<<rope.new()
  .buffer[1]<<button_item.new(p,_self.message(:backward_label),_self,:move_back|(),:help_
id,:backward_help)
  label_item.new(p," "*40)
  .buffer[2]<<button_item.new(p,_self.message(:forward_label),_self,:step5|(),:help_id,:forwa
rd_help)
  _for i _over .buffer.elements()
    _loop
    i.visibility<<_false
  _endloop

  .window[4].activate()
  _self.status<<_self.message(:sub_string14)
  _local a_number<<1
  _for k,e _over .label_hold.keys_and_elements()
    _loop
      shape_file_name<<k
      output_dir<<system.getenv("DATA_PATH")
      sql_file_name<<e[1].value+".sql"
      ctl_file_name<<e[1].value+".ctl"
      geom_spalte<<e[2].value
      table_name<<e[1].value

      skript_erzeuger(_self,shape_file_name,output_dir,sql_file_name,ctl_file_name,geom_spalt
e,a_number,table_name)
      a_number<<a_number+1
    _endloop
  _self.status<<"Ok"

```

```
_for i _over .buffer.elements()
  _loop
  i.visibility<<_true
  _endloop

  _self.list4(_self.message(:sub_string15))
  _endmethod
$
_pragma(classify_level=basic, topic={shape_import})
_method shp2sdo.help_text4()
  ##
  ##Methode des Help-button zum öffnen der Hilfe
  ##
  >>_self.help_wanted(:step4_help)
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.list4(new_val)
  ##
  ## Change Methode für die List View. Werden neue Werte in die List View aufgenommen,
  ## findet eine Aktualisierung statt.
  ##
  q<<.oracle_files.size+1
  .oracle_files[q] << new_val
  _self.changed(:oracle_files)
_endmethod
$

_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.step5()
  ##
  ##Methode entscheidet, ob nur bestehende Tabellen vorhanden sind. Für neue Tabellen
  muß jedoch ein Index auf die Geometrien erstellt werden.
  ##
  _if .new_tables.size > 0
  _then
  _self.select_idx()
  _else
  _self.show_alert(_self.message(:sub_string16))
  .window[5]<<_unset
  .window[4].deactivate()
  _self.step6()
  _endif
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.select_idx()
  ##
  ##Neues Fenster zum eingeben von Index und Primary Key Daten wird über alle neuen
  Tabellen generiert.
  ##
  _local a_choice
  _local q
  _local w<<1
  .window[4].deactivate()
```

```

.window[5]<<frame.new(_self.message(:title5))
p<<panel.new(.window[5])
_self.status<<"Ok"
label_item.new(p,_self.message(:status))
label_item.new(p,_self.status_string,:model,_self,:aspect,:status_string)
p.start_row()
label_item.new(p,_self.message(:sub_string35))
label_item.new(p,"_*38)
button_item.new(p,_self.message(:help_label),_self,:helptext5|(),:help_id,:help_help)
p.start_row()
label_item.new(p,"_*30)
p.start_row()
label_item.new(p,_self.message(:sub_string17))
_for a_table _over .new_tables.keys()
  _loop
  _local a_runner<<2
  _local new_files_temp<<hash_table.new()
  p.start_row()
  label_item.new(p,_self.message(:sub_string18)+" "+a_table.uppercase+"
"+_self.message(:sub_string19))
  p.start_row()
  a_choice<< choice_item.new(p,"Indextyp",{ "none","Quad-tree","R-
tree"},{"none","q"+a_table,"r"+a_table},:model,_self,:change_selector,
:|switch_visi(),:display_all?,_false)
  _local hash_temp<<new_files_temp[1]<<hash_table.new()
  _local a_rope<<hash_temp[a_choice]<<rope.new()
  _local a_text_item<<text_item.new(p,_self.message(:index_name))
  a_text_item.display_length<<15
  p.start_row()
  a_rope[1]<<slider_item.new(p,"Quad-tree (SDO-LEVEL)",1,15)
  a_rope[2]<<slider_item.new(p,"R-tree (Dimensionen)",2,2)
  a_rope[2].visibility<<_false
  p.start_row()
  _local a_row_rope<<rope.new()
  a_row_rope[1]<<"none"
  _local a_view << gis_program_manager.databases[:gis]
  _local
tab<<spatial_connection.int!dd_fetch_table_columns_sdo(_self.oracle_login[1].uppercase,a_tabl
e.uppercase,a_view)
  _for i _over tab.elements()
    _loop
    a_row_rope[a_runner]<<i.column_name
    a_runner<<a_runner+1
  _endloop

  a_rope[3]<<choice_item.new(p,"PK",a_row_rope,a_row_rope,:model,
_self,:change_selector,_unset,:display_all?,_false)
  a_rope[4]<<text_item.new(p,_self.message(:sub_string36))
  a_rope[4].display_length<<15
  p.start_row()
  new_files_temp[2]<<a_text_item
  .new_tables[a_table]<<new_files_temp
  _endloop
  p.start_row()
  label_item.new(p,"_*30)
  p.start_row()

```

```

        .buffer<<rope.new()

        .buffer[1]<<button_item.new(p,_self.message(:backward_label),_self,:move_back|(),:help_
id,:backward_help)
            label_item.new(p," "*40)

        .buffer[2]<<button_item.new(p,_self.message(:forward_label),_self,:controlstep5|(),:help_id
,:forward_help)
            .buffer[3]<<a_choice
            .window[5].activate()
    _endmethod
$
    _pragma(classify_level=basic, topic={shape_import})
    _method shp2sdo.helpertext5()
        ##
        ##Methode des Help-button zum öffnen der Hilfe
        ##
        >> _self.help_wanted(:step5_help)
    _endmethod
$
    _pragma(classify_level=basic, topic={shp2sdo})
    _method shp2sdo.switch_visi(value)
        ##
        ##Je nach gewähltem Indextyp im oben erzeugten Choice_item werden der die jeweils
anderen Felder inaktiv geschaltet.
        ##

        a_size<<value.size
        indx<<value.slice(1,1)
        table<<value.slice(2,a_size)
        _for a_table,a_hash _over .new_tables.keys_and_elements()
        _loop
        _if a_table = table
        _then
            _for i _over a_hash[1].elements()
            _loop
            _if indx="r"
            _then
                i[1].visibility<<_false
                i[2].visibility<<_true
            _else
                i[1].visibility<<_true
                i[2].visibility<<_false
            _endif
            _endloop
        _endif
        _endloop
    _endmethod
$
    _pragma(classify_level=basic, topic={shp2sdo})
    _method shp2sdo.controlstep5()
        ##
        ##Wertet die im 5-ten Fenster gemachtenangaben aus und erzeugt den fetsgelegten
Index.
        ##

```



```

_local checker<<_true
_local a_sql_string
_local table_hash<<hash_table.new()
  _for a_text_item _over _self.label_hold.elements()
    _loop
      a_table<<a_text_item[1].value.as_symbol()
      table_hash[a_table]<<a_text_item[2].value.lowercase
    _endloop

_for a_table_name,a_hash _over .new_tables.keys_and_elements()
  _loop
    _local a_idx_name<<a_hash[2].value
    _for a_slide _over a_hash[1].elements()
      _loop
        _if a_slide[1].visibility=_true
          _then
            _if a_hash[2].value="" _andif _self.buffer[3].value~="none"
              _then
                _self.show_alert(_self.message(:sub_string20)+
"+write_string(a_table_name).uppercase+" "+_self.message(:sub_string21))
                checker<<_false
              _else

                #_for i _over .buffer.elements()
                #_loop
                #i.visibility<<_false
                #_endloop
                _local a_pkrow<<a_slide[3].value
                _local a_pkname<<a_slide[4].value
                _local a_pk_checker<<_true
                _if a_pkname="" _andif a_pkrow~="none"
                  _then
                    a_pk_checker<<_false
                    checker<<_false
                    _self.show_alert(_self.message(:sub_string37))
                  _endif
                    _if a_pkrow~="none" _andif a_pk_checker=_true
                      _then
                        _local a_pkstring<<"ALTER TABLE
"+a_table_name.uppercase+" ADD(CONSTRAINT "+a_pkname+" primary key
  ("+a_pkrow+"))"
                        spatial_connection.exec_sql(a_pkstring)
                      _endif
                      _local a_value<<a_slide[1].value
                      a_symbol<<a_table_name.as_symbol()
                      a_geom<<table_hash[a_symbol]
                      _self.status<<_self.message(:sub_string22)
                      _if _self.buffer[3].value~="none"
                        _then
                          a_sql_string<<"CREATE INDEX
"+a_hash[2].value.uppercase+" ON "+a_table_name.uppercase+"("+a_geom.uppercase+")
INDEXTYPE IS MDSYS.SPATIAL_INDEX PARAMETERS
('SDO_LEVEL="+write_string(a_value)+" SDO_COMMIT_INTERVAL=-1)"
                          spatial_connection.exec_sql(a_sql_string)

```

```

        _endif
        _self.status<<"OK"
        _for i _over .buffer.elements()
        _loop
        i.visibility<<_true
        _endloop
    _endif
_endif

_if a_slide[2].visibility=_true
_then
    _if a_hash[2].value="" _andif _self.buffer[3].value~="none"
    _then
        _self.show_alert(_self.message(:sub_string20)+
"+write_string(a_table_name).uppercase+" "+_self.message(:sub_string21))
    _else
        #_for i _over .buffer.elements()
        #_loop
        #i.visibility<<_false
        #_endloop
        _local a_pkrow<<a_slide[3].value
        _local a_pkname<<a_slide[4].value
        _local a_pk_checker<<_true
        _if a_pkname="" _andif a_pkrow~="none"
        _then
            a_pk_checker<<_false
            checker<<_false
            _self.show_alert(_self.message(:sub_string37))
        _endif
        _if a_pkrow~="none" _andif a_pk_checker=_true
        _then
            _local a_pkstring<<"ALTER TABLE
"+a_table_name.uppercase+" ADD(CONSTRAINT "+a_pkname+" primary key
("+a_pkrow+"))"
            spatial_connection.exec_sql(a_pkstring)
        _endif
        _local a_value<<a_slide[2].value
        a_symbol<<a_table_name.as_symbol()
        a_geom<<table_hash[a_symbol]
        _if _self.buffer[3].value~="none"
        _then
            _self.status<<"Index wird angelegt"
            a_sql_string<<"CREATE INDEX
"+a_hash[2].value.uppercase+" ON "+a_table_name.uppercase+"("+a_geom.uppercase+)
INDEXTYPE IS MDSYS.SPATIAL_INDEX PARAMETERS
('SDO_INDX_DIMS="+write_string(a_value)+")"
            spatial_connection.exec_sql(a_sql_string)
        _endif
        _self.status<<"OK"
        _for i _over .buffer.elements()
        _loop
        i.visibility<<_true
        _endloop
    
```

```

        _endif
        _endif
        _endloop
    _endloop
    _if checker=true
    _then
    .window[5].deactivate()
    _self.step6()
    _endif
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.step6()
    ##
    ##Letztes Fenster wird geöffnet.
    ##

    _self.get_and_create_ospo_ds ()
    .window[6]<<frame.new(_self.message(:title6))
    p<<panel.new(.window[6])
    _self.status<<"Ok"
    label_item.new(p,_self.message(:status))
    label_item.new(p,_self.status_string,:model,_self, :aspect,:status_string)
    p.start_row()
    label_item.new(p,_self.message(:sub_string23))
    p.start_row()
    label_item.new(p,"__"*30)
    p.start_row()
    label_item.new(p,_self.message(:sub_string25))
    p.start_row()
    label_item.new(p,_self.message(:sub_string26))
    p.start_row()
    label_item.new(p,_self.message(:sub_string27))
    p.start_row()
    label_item.new(p,"__"*30)
    p.start_row()
    button_item.new(p,_self.message(:backward_label),_self,:move_back|(),:help_id,:backward_help)
    label_item.new(p,"__"*43)
    button_item.new(p,_self.message(:ready_label),_self,:ready_for_take_off|(),:help_id,:ready_help)
    .window[6].activate()
_endmethod
$
_pragma(classify_level=basic, topic={sas}, usage = {internal})
_method shp2sdo.get_and_create_ospo_ds ()
    ##
    ##Methode zum lesen und hinzufügen von Informationen über alle anzubindenden Tabellen.
    ##
    _local a_data_type_rope<<rope.new_with("point","chain","area")
    _local a_type
    _self.ospo_tables<<rope.new()
    _self.ospo_ds<<hash_table.new()

    #enthält daten über bereits angebundene Tabellen

```

```

a_filename<<system.getenv("OSPO_PATH")+"ospo_ds.ini"

#datei für die konfigurationsdaten des ospo soms
a_filename_config<<system.getenv("OSPO_PATH")+"ospo_ds.config"
_if system.file_exists?(a_filename)
  _then
    _if _true _then

      #Wenn es bereits einen ospo_ds.ini File gibt werden seine Informationen
      #ausgelesen und diese File anschließend gelöscht.
      read_ospo_ds(a_filename,_self)
      system.unlink(a_filename)
    _endif

  _endif

#Überprüft, ob Tabellen bereits angebunden sind, wenn nicht werden sie in die Liste der
anzubindenden Files aufgenommen.
_for i _over _self.label_hold.elements()
  _loop
    _local checker<<_true
    _for a_table _over _self.ospo_ds.keys()
      _loop
        _if a_table=i[1].value
          _then
            _local a_size<<.ospo_tables.size
            _self.ospo_tables[a_size+1]<<a_table
            checker<<_false
          _endif
        _endif
      _endloop
    _endif

    a_sql_string<<"select b."+i[2].value.uppercase+".SDO_GTYPE from
"+i[1].value.uppercase+" b where rownum=1"
    a_select<<spatial_connection.sql_select(a_sql_string)
    a_select<<a_select.get()
    _if a_select._isnt_unset
      _then
        _local a_vec<<a_select.sys!slot(:slot_values)
        _if a_vec[1]=5
          _then
            a_vec[1]<<1
          _elif a_vec[1]=6
            _then
              a_vec[1]<<2
            _elif a_vec[1]=7
              _then
                a_vec[1]<<3
              _endif
            _endif

            a_type<<a_data_type_rope[a_vec[1]]

          _endif

        _if checker=_true

```

```

    _then
        a_size<<.ospo_tables.size
        _self.ospo_tables[a_size+1]<<i[1].value.lowercase
        a_hash<<.ospo_ds[i[1].value.lowercase]<<hash_table.new()
        a_row<<a_hash[i[2].value.lowercase]<<rope.new()
        a_row[1]<<a_type
    _endif
_endloop

#eine neue ospo_ds.ini wird erstellt
write_ospo_ds(a_filename,_self)

#ein neues konfigurationsskript für den ospo som wird erstellt.
#write_ospo_config(a_filename_config,_self.ospo_ds,"Oracle_spatial")

#konfigurationskript wird geladen
#load_file(a_filename_config)
_endmethod
$
_pragma(classify_level=basic, topic={sas}, usage = {internal})
_method shp2sdo.ready_for_take_off()
##
##Schließt das letzte Fenster und #####
##

.window[6].deactivate()
#spatial_connection.discard(v)
_endmethod
$

_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.select_a_ospo(ind)
##
##Setzt angeklickte Fileeinträge in den Slot .buffer.
##
.buffer<<rope.new()
.buffer<<.ospo_tables[ind]
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo})
_method shp2sdo.ospo_tables(new_val)
##
## Change Methode für die List View. Werden neue Werte in die List View aufgenommen,
## findet eine Aktualisierung statt.
##
.ospo_tables << new_val
_self.changed(:ospo_tables)
_endmethod
$
_pragma(classify_level=restricted, topic={oracle_som})
_method oracle_objects_dataset_manager.default_connect_specification
##
## Returns a default connect specification for this dataset manager
##
## This can be subclassed to return a suitable default or
## template connect spec.

```

```
##  
  
  >> property_list.new_with( :dbtype, :oracle8,  
                             :connect_name, "deep1",  
                             :dbusername, "sw_user",  
                             :dbpassword, "manager@bas8_db",  
                             :command, rope.new_with("sworacle8"))  
  
_endmethod
```

7.7 Quellcode des SHP2SDO Editors

```
## text_encoding = iso8859_1
#-----
#
#> Name:    shp2sdo_dataset_editor.magik
#
#> Description: Convert shape to oracle sdo and configured ospo som
#
#> Author:   Bastian Ellmenreich
#
#> Date:    30. Aug 2001
#
# Copyright (C) 2001 by Landesanstalt für Umweltschutz, Karlsruhe
#
#-----
#> Exemplar
#-----
# Entfernen des Exemplars vor Neudefinition. Nach der
# Entwicklungsphase zu entfernen!
#_block
#   _if !current_package![:shp2sdo_dataset_editor] _isnt _unset
#   _then
#       remove_exemplar(:shp2sdo_dataset_editor)
#   _endif
#_endblock
$
  _pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
  def_slotted_exemplar(:shp2sdo_dataset_editor,
    ##
    ## Definiert neues Exemplar von shp2sdo_dataset_editor.
    ##
    {
      },
    {:shp2sdo})
$
#-----
#> Slot Zugriff
#-----

  _pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
  shp2sdo_dataset_editor.define_slot_access(
    :file_list,
    ##
    ## Setzt diesen Slot auf writeable (beschreibbar)
    ##
    ##
    :writable)
$
```

```

_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
shp2sdo_dataset_editor.define_slot_access(
  :ospo_ds,
  ##
  ## Setzt diesen Slot auf writeable (beschreibbar)
  ##
  ##
  :writable)
$
_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
shp2sdo_dataset_editor.define_slot_access(
  :buffer,
  ##
  ## Setzt diesen Slot auf writeable (beschreibbar)
  ##
  ##
  :writable)
$
#-----
#> Objektklasse initialisieren
#-----
##
## Mit shp2sdo_dataset_editor.open() wird auf model.open() zurückgegriffen und
## eine neue Instanz der Klasse shp2sdo_dataset_editor erzeugt.
##

_pragma(classify_level=basic, topic={sas}, usage = {external})
_method shp2sdo_dataset_editor.new ( a_grs )
##
## Initialisiert neues shp2sdo_dataset_editor.
##
  >> _clone.init ( a_grs )

_endmethod
$

_pragma(classify_level=basic, topic={sas}, usage = {internal})
_method shp2sdo_dataset_editor.init ( a_grs )
##
## Initialises the Configuration Interface Agent.
##

#graphic system wird festgelegt
_global !grs!
!grs!          << a_grs

#legt message accessor fest
#message_accessor    << message_handler.new(:shp2sdo_dataset_editor)

>> _super.init (a_grs)

_endmethod
$

```



```

_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.activate_in(f)
  ##
  ## Fenster konfiguration
  ##
  .label_hold<<rope.new()
  .ospo_tables<<hash_table.new()
  _local a_filename
  .file_list<<rope.new()
  .ospo_ds<<hash_table.new()
  #setzt den Status auf ok
  _self.status<<"Ok"

  #setzt den titel des frames
  _self.title<<"SHP2SDO Dataset Editor"

  #legt die position fest
  f.position<<pixel_coordinate(0,0)
  p << panel.new(f)
  label_item.new(p,"Status")
  label_item.new(p,_self.status_string,:model,_self, :aspect,:status_string)
  p.start_row()
  label_item.new(p,"Themen zusammenstellen")
  label_item.new(p," **28)
  button_item.new(p,"Hilfe",_self,:helptext5|(),:help_id,:help_help)
  p.start_row()
  .label_hold[1]<<text_item.new(p,"Dataset          :")
  p.start_row()
  label_item.new(p,"__"*30)
  p.start_row()
  button_item.new(p,"Neue
Tabelle",_self,:new_table_definition|(),:help_id,:file_selection_help)
  button_item.new(p,"Löschen",_self,:deselect|(),:help_id,:deselection_help)
  #Legt den Startwert für file_list fest
  a_filename<<system.getenv("OSPO_PATH")+ospo_ds.ini"
  read_ospo2(a_filename,_self)
  list_view.new(_self, f, :file_list,:select_one|(),_unset,8,60)

  s<<panel.new(f)
  s.start_row()
  s.start_row()
  label_item.new(s,"__"*30)
  s.start_row()
  t << panel.new(f)
  t.start_column()
  last<<image_button_item.new_safe(t,
"D:\smallworld\sonstiges\eigene\shp2sdo\bitmaps\lfugross.bmp",_self, :info|(),:help_id,
:info_help)
  t.start_column(last,270)
  button_item.new(t,"Fertig",_self,:ready|(),:help_id,:ready_help)

  >> _self
_endmethod
$

```

```

_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.help_text5()
  ##
  ##Methode des Help-button zum öffnen der Hilfe
  ##
  >> _self.help_wanted(:edi_help)
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.file_list(new_val)
  ##
  ## Change Methode für die List View. Werden neue Werte in die List View aufgenommen,
  ## findet eine Aktualisierung statt.
  ##
  .file_list << new_val
  _self.changed(:file_list)
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.info()
  ##
  ##Definiert das info fenster
  ##
  >> _self.show_alert(_self.message ( :info))

_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.new_table_definition()

.window<<rope.new()
frame2<<frame.new("Bestehende Tabelle anbinden")
p<<panel.new(frame2)
label_item.new(p,"Bitte Tabellendaten eingeben")
p.start_row()
label_item.new(p,"__"*30)
p.start_row()
.label_hold[2]<<text_item.new(p,"Tabelle      :")
p.start_row()
.label_hold[3]<<text_item.new(p,"Geometriespalte :")
p.start_row()
.label_hold[4]<<choice_item.new(p,"Geometrietyp
:",{ "Punkt", "Linie", "Fläche"}, {"point", "chain", "area"}, :model, _self, :display_all?, _false)
p.start_row()
label_item.new(p,"__"*30)
p.start_row()
button_item.new(p,"Abbrechen", _self, :break|(), :help_id, :break_help)
label_item.new(p,"__"*38)
button_item.new(p,"Erstellen", _self, :built|(), :help_id, :built_help)
frame2.activate()
.window[1]<<frame2
_endmethod
$

```

```
_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.break()
.window[1].deactivate()
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.built()
_local a_dump_rope<<rope.new()
_local a_tab
_if .label_hold[2].value.lowercase=""
_then
.window[1].deactivate()
_else
a_dump_rope[1]<<.label_hold[2].value.lowercase
a_tab<<.ospo_tables[.label_hold[2].value.lowercase.as_symbol()]<<rope.new()
a_tab[1]<<.label_hold[2].value.lowercase
a_tab[2]<<.label_hold[3].value.lowercase
a_tab[3]<<.label_hold[4].value.lowercase
_self.new_select_file(a_dump_rope)
.window[1].deactivate()
_endif
_endmethod
$
_pragma(classify_level=basic, topic={shp2sdo_dataset_editor})
_method shp2sdo_dataset_editor.ready()
.ospo_ds<<hash_table.new()
_if .file_list[1]=_self.message(:no_choice)
_then
_self.quit()
_else
_if .label_hold[1].value=""
_then
_self.show_alert("Dataset festlegen")
_else
_for i _over .file_list.elements()
_loop
a_hash<<.ospo_ds[i]<<hash_table.new()
w<<i.as_symbol()
a_rope<<.ospo_tables[w]
a_row<<a_hash[a_rope[2]]<<rope.new()
a_row[1]<<a_rope[3]
_endloop
a_filename<<system.getenv("OSPO_PATH")+ospo_ds_new.ini"
write_ospo_ds(a_filename,_self)
a_filename<<system.getenv("OSPO_PATH")+ospo_ds.config"
write_ospo_config(a_filename,.ospo_ds,.label_hold[1].value)
load_file(a_filename)
_self.quit()
_endif
_endif
_endmethod
$
```

```
_global read_ospo2<<_proc(a_filename,a_model)
  ##
  ##Prozedur zum lesen von externen textfiles

  ##
  _local a_tab
  stream<<external_text_input_stream.new(a_filename) #create
stream
  i<<0
  q<<1

  _loop

  i<<i+1
  line<<stream.get_line() #get a line of source data from the stream
  _if line _is _unset _then
  _leave
  _endif
  _if i=1
  _then

  a_tab<<a_model.ospo_tables[line.lowercase.as_symbol()]<<rope.new()
  a_tab[i]<<line
  a_model.file_list[q]<<line
  _endif
  _if i=2
  _then
  a_tab[i]<<line
  _endif
  _if i=3
  _then
  a_tab[i]<<line
  i<<0
  q<<q+1
  _endif

  _endloop
  stream.close()
  _if a_model.file_list.size=0
  _then
  a_model.file_list[1]<<a_model.message(:no_choice)
  _endif
_endproc
⌘
```

7.8 Quellcode der Prozeduren des direkten Datentransfers

Die Prozeduren für den direkten Datentransfer sind hinsichtlich ihrer Funktion so eng wie möglich gekapselt. Dies hat den Vorteil, dass die Struktur übersichtlicher, sicherer im Ablauf und, im Falle einer Erweiterung der Aufgaben, leicht in andere Strukturen eingebunden werden kann.

7.8.1 Die Prozedur `make_sql_file_create_layer`

Zur Erstellung eines SQL Skripts, mit dem die Tabellenstruktur auf Oracle neu definiert wird, kann die folgende Prozedur genutzt werden. Aus der gegebenen Tabelle wird ein Datensatz identifiziert und hinsichtlich seiner Struktur analysiert. Das Ergebnis dieser Prozedur ist eine Datei, in der die Oracle- Tabellendefinition für die gegebene Smallworldtabelle definiert wird.

```
_global make_sql_file_create_layer<<_proc(a_collection,a_filename)
_local a_record
_local key_field
_local text_fields<<hash_table.new()
_local tabel_name
_local valu_vec
_local phys_field

a_record<<a_collection.an_element()
key_field<<a_record.key_field_names[1].as_charvec()
table_name<<a_collection.name.as_charvec()

valu_vec<<a_record.values_as_vector()
phys_field<<a_record.physical_field_names

text_fields<<make_text_fields(phys_field,valu_vec)

a_stream<<external_text_output_stream.new(a_filename)
a_stream.write("DROP TABLE "+table_name+";")
a_stream.newline()
a_stream.write("CREATE TABLE "+table_name+" (")
a_stream.newline()
_for a_field,a_hash _over text_fields.keys_and_elements()
_loop
_for a_val,a_type _over a_hash.keys_and_elements()
_loop
a_stream.write(a_field.as_charvec().uppercase+" "+a_type+",")
a_stream.newline()
_endloop
```

```
_endloop
a_stream.write("SHAPE MDSYS.SDO_GEOMETRY;")
a_stream.newline()
a_stream.newline()
a_stream.write("ALTER TABLE "+table_name+" ADD CONSTRAINT
"+key_field+table_name.slice(1,3)+"_pk primary key ("+key_field+");")
a_stream.newline()
a_stream.write("commit;")
a_stream.close()
_endproc
$
```

7.8.2 Die Prozedur make_text_fields

Die Prozedur make_text_fields ist eine Hilfsprozedur die beim Schreiben der SQL-Datei aktiviert wird. Sie hat die Aufgabe die Smallworld Datentypen in Oracle Datentypen umzuwandeln.

```
_global make_text_fields<<_proc(phys_field,valu_vec)

_local text_fields<<hash_table.new()
_for a_ind,a_elem _over phys_field.keys_and_elements()
_loop
_if a_elem=:rwo_id _orif a_elem=:ds!version
_then
_continue
_else

a_buffer<<text_fields[a_elem]<<hash_table.new()
_if valu_vec[a_ind].class_name=:integer _orif valu_vec[a_ind].class_name=:float
_then
_a_buffer[valu_vec[a_ind]]<<"NUMBER(15,4)"
_elif valu_vec[a_ind].class_name=:char16_vector _orif
valu_vec[a_ind].class_name=:ds_char16_vector
_then
_a_buffer[valu_vec[a_ind]]<<"VARCHAR2(32)"
_else
_write("Unbekannter Typ : "+valu_vec[a_ind].class_name)
_endif
_endif
_endloop
>>text_fields
_endproc
$
```

7.8.3 Die Prozedur count_poly_cords

Für den Objekt Typ SDO_GEOMETRY werden Metadaten, welche die Struktur der folgenden Informationen definieren, benötigt. Die Prozedur bestimmt die Anzahl der einzelnen Stützpunkte eines geometrischen Objektes. Auf diese Weise kann beim Schreiben des LOADER Skriptes der Offset einer jeden Geometrie bestimmt werden.

```
_global count_poly_coords<<_proc(outer_poly)
poly_coord<<0
_for a_key,a_poly_sector _over outer_poly.keys_and_elements()
_loop
  poly_coord<<poly_coord+a_poly_sector.size*2
  _if a_key~=1
  _then
    poly_coord<<poly_coord-2
  _endif
_endloop
>>poly_coord
_endproc
$
```


7.8.4 Die Prozedur make_geom_file_area

Innerhalb dieser Prozedur werden die Geometriedaten in eine LOADER- Datei geschrieben. Hierfür wird zunächst ein einheitlicher Header aufgestellt. Ihm folgen die Definitionen zu den einzelnen Spalten in den die Daten geschrieben werden sollen. Ist die Struktur der Daten festgelegt, können abschließend die Koordinaten, mittels einer weiteren Prozedur, die innerhalb dieser Routine aufgerufen wird, geschrieben werden.

```
_global make_geom_file_area<<_proc(a_collection,a_filename)
  _local valu_vec
  _local phys_field
  _local text_fields
  _local table_name
  _local a_runner<<1
  !print_float_precision!<<12
  a_test_record<<a_collection.an_element()
  valu_vec<<a_test_record.values_as_vector()
  phys_field<<a_test_record.physical_field_names
  text_fields<<make_text_fields(phys_field,valu_vec)
  table_name<<a_collection.name.as_charvec()
  a_stream<<external_text_output_stream.new(a_filename)
  a_stream.write("LOAD DATA")
  a_stream.newline()
  a_stream.write("INFILE *")
  a_stream.newline()
  a_stream.write("TRUNCATE")
  a_stream.newline()
  a_stream.write("CONTINUEIF NEXT(1:1) = '#")
  a_stream.newline()
  a_stream.write("INTO TABLE "+table_name)
  a_stream.newline()
  a_stream.write("FIELDS TERMINATED BY '|")
  a_stream.newline()
  a_stream.write("TRAILING NULLCOLS (")
  a_stream.newline()
  _for a_attribute,a_hash _over text_fields.keys_and_elements()
  _loop
    _if a_attribute~="SHAPE"
    _then
      a_stream.write(a_attribute+" NULLIF "+a_attribute+" = BLANKS,")
      a_stream.newline()
    _endif
  _endloop
  a_stream.write("SHAPE COLUMN OBJECT")
  a_stream.newline()
  a_stream.write("(")
  a_stream.newline()
  a_stream.write("SDO_GTYPE      INTEGER EXTERNAL,")
```

```

a_stream.newline()
a_stream.write("SDO_ELEM_INFO  VARRAY TERMINATED BY '|/'")
a_stream.newline()
a_stream.write("X          FLOAT EXTERNAL),")
a_stream.newline()
a_stream.write("SDO_ORDINATES  VARRAY TERMINATED BY '|/'")
a_stream.newline()
a_stream.write("X          FLOAT EXTERNAL)")
a_stream.newline()
a_stream.write("")
a_stream.newline()
a_stream.write("")
a_stream.newline()
a_stream.write("BEGINDATA")
a_stream.newline()
_for a_record _over a_collection.elements()
_loop
    valu_vec<<a_record.values_as_vector()
    phys_field<<a_record.physical_field_names
    write(write_string(a_runner)+" von "+write_string(a_collection.size)+" wird bearbeitet!")
    a_runner<<a_runner+1
    text_fields<<make_text_fields(phys_field,valu_vec)
    all_geometries<<a_record.geometries()
    _if all_geometries.size>1
    _then
        write("Mehrere Geometrien vorhanden!")
    _endif
    (out,hol)<<all_geometries[1].int!outers_and_holes()
    _if out.size=1
    _then
        a_geom_type<<3
    _else
        a_geom_type<<7
    _endif
    _for a_attri,a_hash _over text_fields.keys_and_elements()
    _loop
        _if a_attri~="SHAPE"
        _then
            _for a_value,a_type _over a_hash.keys_and_elements()
            _loop
                a_stream.write(write_string(a_value)+"|")
            _endloop
        _endif
    _endloop
    a_stream.newline()
    a_stream.write("#")
    a_stream.write("200"+write_string(a_geom_type)+"|")
    a_stream.write("1|1003|1|")
    poly_coord<<0
    a_poly_runner<<0
    _for a_outer_poly _over out.elements()
    _loop
        outer_poly_sector<<a_outer_poly.sectors
        poly_coord<<poly_coord+count_poly_coords(outer_poly_sector)
        a_poly_runner<<a_poly_runner+1
        _if a_outer_poly.holes.size>0

```

```
        _then
            _for a_hole _over a_outer_poly.holes.elements()
            _loop
                a_stream.write(write_string(poly_coord+1)+"|2003|1|")
                poly_coord<<poly_coord+count_poly_coords(a_hole.sectors)
            _endloop
        _endif
        _if a_poly_runner<out.size
        _then
            a_stream.write(write_string(poly_coord+1)+"|1003|1|")
        _endif
    _endloop
    a_stream.write("/")
    a_stream.newline()
    a_stream.write("#+")
    a_stream.newline()
    a_stream.write("#")
#prozedur startet
    a_checker<<0
    _for a_outer_poly _over out.elements()
    _loop
        outer_poly_sector<<a_outer_poly.sectors
        a_checker<<write_geometries(outer_poly_sector,a_stream,a_checker)
        a_stream.write("|")
        _if a_outer_poly.holes.size>0
        _then
            _for a_hole _over a_outer_poly.holes.elements()
            _loop
                a_hole_sector<<a_hole.sectors
                a_checker<<write_geometries(a_hole_sector,a_stream,a_checker)
                a_stream.write("|")
            _endloop
        _endif
    _endloop

    a_stream.write("/")
    a_stream.newline()
_endloop
a_stream.close()
write("Datentransfer abgeschlossen")
_endproc
$
```

7.8.5 Die Prozedur write_geometries

Das Schreiben der Koordinaten übernimmt die folgende Prozedur. Sie ruft dabei die Koordinaten der einzelnen Tabellenobjekte nacheinander ab. Besonderheiten in dieser Prozedur ist das Einfügen von Steuerzeichen wie „#“. Dies ist nötig, da der SQL*LOADER eine Begrenzung für die Länge des zu interpretierenden Befehls hat.

```

_global write_geometries<<_proc(inn_or_out_rope,a_stream,a_checker)
_for a_sector_key,a_out_sector_over inn_or_out_rope.keys_and_elements()
  _loop
    _if a_sector_key=1 _andif a_sector_key=inn_or_out_rope.size
      _then
        _for a_coord_key,a_coodinate_set_over
a_out_sector.keys_and_elements()
          _loop
            _if a_coord_key~=a_out_sector.size
              _then
                _if a_checker=1752
                  _then
                    a_stream.newline()
                    a_stream.write("#")
                    a_checker<<0
                  _endif
                a_stream.write(write_string(a_coodinate_set.x)+"|"+write_string(a_coodinate_set.y)+"|")
                a_checker<<a_checker+2
              _else
                _if a_checker=1752
                  _then
                    a_stream.newline()
                    a_stream.write("#")
                    a_checker<<0
                  _endif
                a_stream.write(write_string(a_coodinate_set.x)+"|"+write_string(a_coodinate_set.y))
                a_checker<<a_checker+2
              _endif
            _endloop
          _endif
        _if a_sector_key=1 _andif a_sector_key~=inn_or_out_rope.size
          _then
            _for a_coord_key,a_coodinate_set_over
a_out_sector.keys_and_elements()
              _loop
                _if a_checker=1752

```

```

        _then
            a_stream.newline()
            a_stream.write("#")
            a_checker<<0
        _endif

    a_stream.write(write_string(a_coodinate_set.x)+"|"+write_string(a_coodinate_set.y)+"|")
        a_checker<<a_checker+2
    _endloop
_endif

    _if a_sector_key~=1 _andif a_sector_key~=inn_or_out_rope.size
    _then
        _for a_coord_key,a_coodinate_set _over
a_out_sector.keys_and_elements()
        _loop
            _if a_coord_key~=1
            _then
                _if a_checker=1752
                _then
                    a_stream.newline()
                    a_stream.write("#")
                    a_checker<<0
                _endif

                a_stream.write(write_string(a_coodinate_set.x)+"|"+write_string(a_coodinate_set.y)+"|")
                    a_checker<<a_checker+2
            _endif
        _endloop
    _endif

    _if a_sector_key~=1 _andif a_sector_key=inn_or_out_rope.size
    _then
        _for a_coord_key,a_coodinate_set _over
a_out_sector.keys_and_elements()
        _loop
            _if a_coord_key~=1 _andif
a_coord_key~=a_out_sector.size
            _then
                _if a_checker=1752
                _then
                    a_stream.newline()
                    a_stream.write("#")
                    a_checker<<0
                _endif

                a_stream.write(write_string(a_coodinate_set.x)+"|"+write_string(a_coodinate_set.y)+"|")
                    a_checker<<a_checker+2
            _endif
            _if a_coord_key=a_out_sector.size
            _then
                _if a_checker=1752
                _then
                    a_stream.newline()
                    a_stream.write("#")

```

```

a_checker<<0
  _endif
a_stream.write(write_string(a_coordinate_set.x)+"|"+write_string(a_coordinate_set.y))
  a_checker<<a_checker+2
  _endif
  _endloop
  _endif
  _endloop
>>a_checker
  _endproc
$
```

7.9 Smallworld – Disjointe Flächen

Call-Nummer: SWS 511000
Thema: Areas in Oracle 8.1.7 Spatial
Kundenreferenz: ---
Ansprechpartner: Herr Pankow
Bearbeiter: Erich Nagy
Status: Antwort zu Ihrer Anfrage

Hallo Herr Pankow,

bei dem gemeldeten Problem handelt es sich um eine Einschränkung der Funktionalität der aktuellen GIS Version 3.1(0) SP2:

> This is confirmed as a limitation.
>
> Here are developer's comments :
>
> "We do not support disjoint area at that release and it is only a side
> effect that it is returned. ..."

Grund sind die Unterschiede zwischen den Geometriemodellen von Smallworld und Oracle.

Wir haben bereits eine Fehlermeldung hierzu gestellt und diese hat folgenden Status:

Ref Number: 39344
Status: Open

Weiteres Vorgehen: Ohne Rückmeldung wird der Call am 24.10.01 geschlossen.

Mit freundlichen Grüßen

Erich Nagy

GE Smallworld (Germany) GmbH
- Support -
E-mail: GIS-Hilfe@smallworld.de
Tel.: +49(2102) 108-137 Europaring 60
Fax: +49(2102) 108-168 D-40878 Ratingen

Verzeichnis der Abbildungen

Abbildung	Bezeichnung	Seite
ABBILDUNG 1	GEODATENORGANISATION	9
ABBILDUNG 2	GEODATENORGANISATION LFU	23
ABBILDUNG 3	MAGIKOBJEKTE	27
ABBILDUNG 4	R- TREE INDEX	31
ABBILDUNG 5	QUAD- TREE (FIXED SIZE)	31
ABBILDUNG 6	QUAD- TREE (HYBRID SIZE)	32
ABBILDUNG 7	DATENFILTERUNG	34
ABBILDUNG 8	RÄUMLICHE BEZIEHUNG ZWISCHEN GEOMETRISCHEN OBJEKTEN	35
ABBILDUNG 9	ZUSÄTZLICHE GEOMETRIETYPEN DES OBJEKT- RELATIONALEN SCHEMAS	39
ABBILDUNG 10	BUFFERBILDUNG BEI GEOMETRISCHEN OBJEKTEN	39
ABBILDUNG 11	POLYGON MIT LOCH	41
ABBILDUNG 12	GEOMETRISCHE GRUNDTYPEN	44
ABBILDUNG 13	PHASEN DER ANPASSUNG	48
ABBILDUNG 14	AUFBAU EINER SMALLWORLDSITZUNG	50
ABBILDUNG 15	ALLGEMEINER PROGRAMMABLAUF	58
ABBILDUNG 16	KLASSENHIERARCHIE	63
ABBILDUNG 17	SHP2SDO DATENBANKADAPTER	64
ABBILDUNG 18	DATENBANKADAPTER PROGRAMMABLAUF	65
ABBILDUNG 19	MENÜ DATEIAUSWAHL	66
ABBILDUNG 20	MENÜ TABELLENBESCHREIBUNG	67
ABBILDUNG 21	MENÜ TABELLENINFORMATIONEN	69
ABBILDUNG 22	INFORMATIONSMENÜ	73
ABBILDUNG 23	INDEXGENERIERUNG	74
ABBILDUNG 24	BEENDEN DES DATENBANKADAPTERS	76
ABBILDUNG 25	THEMENAUSSWAHL IM EDITOR	79
ABBILDUNG 26	TABELLENDEFINITION	79
ABBILDUNG 27	PROZEDURENABLAUF	82

Verzeichnis der Tabellen

Tabelle	Bezeichnung	Seite
TABELLE 1	BASISDATEN	13
TABELLE 2	NATURSCHUTZ, LANDSCHAFTSÖKOLOGIE	14
TABELLE 3	TECHNOSPHERE, WASSER, BODEN UND LUFT	15
TABELLE 4	SHAPE AUFBAU (Main – File)	17
TABELLE 5	SDO_INDEX TABELLE	32
TABELLE 6	SDO_INDEX TABELLE	33
TABELLE 7	SDO_LAYER	41
TABELLE 8	SDODIM	42
TABELLE 9	SDOGEOM	43
TABELLE 10	SDOINDEX	44

Literaturverzeichnis

- Loney/Theriault; Oracle 8i - Für Einsteiger
Carl Hanser Verlag München Wien ,2001
- Loney/Theriault; Oracle 8i – DBA Handbuch
Carl Hanser Verlag München Wien ,2001
- Oracle 7 – Server SQL Language Reference Manual, 1992
- Bock/Greve/Kuhn; Offene Umweltinformationssysteme- Chancen und
Möglichkeiten der OpenGIS- Entwicklungen im Umweltbereich
Verlag Natur & Wissenschaft Solingen ,1999
- ESRI Shapefile Technical Description - ESRI White Paper—July 1998
- Online Handbuch - Smallworld 3 Dokumentation Version 3.1 (0) SP1
- Smallworld Schulungsunterlagen – Magik Programming Course

Firmenproduktbeschreibungen

- GE Smallworld Core Spatial Technology

Internetpublikationen

- <http://www.bgr.de/z6/glossar.htm>
- <http://www.oracle>
- http://www.lfu.bwl.de/local/abt5/itz/rips/rips_documente.htm (local Server)
- http://otn.oracle.com/docs/products/oracle8i/doc_library/817_doc/inter.817/a85337/sdo_intr.htm#871809
- <http://www.smallworld.com>
- <http://www.gis-tutor.de>
- www.opengis.org/techno/guide.htm

Verzeichnis der Abkürzungen

Abkürzung	Bedeutung
UIS	U mwelt i nformationssystem dem
GIS	G eoinformationssysteme
WaWiBo	W asser, W irtschaft und B oden
SW-DB	S mall w orld D aten b ank
OGC	O pen G IS C onsortium
CORBA	C ommon O bject R equest B roker A rchitecture
COM	C omponent O bject M odel
SQL	S tructured Q uery L anguage
DBMS	D atenbank M anagement S ystem
RDBMS	R elationales D atenbank M anagement S ystem
DDL	D ata D efinition L anguage
DML	D ata M anipulation L anguage
DBWR	D atabase W riter
LGWR	L og W riter
FFC	F achführungscode
OAC	O bjektartencode
VMDS	V ersion M anaged D ata S to r e
ACE	A pplication C onfiguration E nvironment
Auth	A uthorisation
RWO	R eal W orld O bjects
CASE	C omputer A ided S oftware E ngineering
ADO.	A ctiveX D ata O bjects
MBR	M inimum B ounding R ectangle
SOC	S patial O bjekt C ontroller
SOM	S patial O bjekt M anager
SWMF	Smallworld Datastore Server
ACP	A lien C o- P rocessor

Danksagung

An dieser Stelle möchte ich mich für die gute Zusammenarbeit mit Herrn Professor Dr. Ing. Ernst Heil bedanken. Trotz der Entfernung zwischen Karlsruhe und Neubrandenburg erfuhr ich stets Unterstützung von seiner Seite.

Für die Bereitschaft meine Arbeit trotz beruflicher Neuorientierung zu betreuen danke ich ebenfalls Herrn Professor Dr. Ing. Albert Zimmermann.

Ein großer Dank gilt Herrn Dipl. Agrar Biologen Manfred Müller und seinem Team. Er und seine Mitarbeiter erleichterten mir den praktischen Einstieg in die Geoinformatik durch Ihren fachlichen Rat und viele wertvolle Anregungen. Durch die vielfältigen Arbeitsbereiche innerhalb dieses Referates konnte ich stets auf kompetente Ansprechpartner zurückgreifen.

Erklärung

Gemäss der Diplomprüfungsordnung für den Studiengang Vermessungswesen an der Fachhochschule Neubrandenburg versichere ich, dass diese Diplomarbeit ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen genutzt wurden.

Karlsruhe, den 1.1.2002

Bastian Ellmenreich, Matr. Nr. 262597 _____

Bemerkungen

Bemerkungen
