

Diplomarbeit

Integration raumbezogener Daten in ein Umweltinformationssystem

Konzeption und Implementierung eines grafischen Assistenten für die
Eingabe, Zusammenstellung und Integration von Geodaten in ein WebGIS.
Erweiterung der Integrationssoftware für den Import von in-situ Sensormessdaten.

eingereicht von

Malte Ahrens

HOCHSCHULE KARLSRUHE –TECHNIK UND WIRTSCHAFT
Fakultät Geomatik – Diplomstudiengang Kartographie und Geomatik



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES



Dezember 2010

**Diplomarbeit zur Erreichung des akademischen Grades
Dipl.-Ing. (FH) für Kartographie und Geomatik**

für Herrn Malte Ahrens
Matrikelnummer: 21779

Thema: Integration raumbezogener Daten in ein Umweltinformationssystem

Konzeption und Implementierung eines grafischen Assistenten für die Eingabe, Zusammenstellung und Integration von Geodaten in ein WebGIS. Erweiterung der Integrationssoftware für den
Import von in-situ Sensormessdaten.

Im Rahmen der Projekte WISDOM und CAWa entwickelt das Deutsche Fernerkundungsdatenzentrum (DFD), als Institut des Deutschen Luft- und Raumfahrtzentrum (DLR), ein internetbasiertes Umweltinformationssystem für das Mekong Delta in Vietnam und fünf Länder Zentralasiens. Die Konzeption des Systems sieht vor, dass Inhalte von den Nutzern bereitgestellt und in den Speichersystemen gespeichert und verwaltet werden. Für die Integration von Geodaten wurde vom DFD ein Datenaustauschformat („WISDOM Geodata Exchange Format“, WGEF) entwickelt, das als „Behälter“ für die verschiedenen Bestandteile eines Geodatensatzes dient. Bestandteile eines Geodatensatzes sind neben den Geometriedaten im Raster- oder Vektordateiformat, eine Dokumentation der Daten und deren Erhebung mittels Metadaten und Regeln für die Symbolisierung.

In dieser Diplomarbeit wird eine grafische Oberfläche für die schrittweise Eingabe der Bestandteile eines Geodatensatzes entwickelt. Diese umfasst Funktionen zum Hochladen von Geodaten und Formulare für die Eingabe von Metadaten und Darstellungsregeln. Die Bestandteile werden durch die entwickelte Software in das WGEF-Dateiformat überführt und die Integration in das Umweltinformationssystem durchgeführt. Des Weiteren wird die Integrationssoftware erweitert, so dass Messdaten von in-situ Sensormessstationen automatisch in das System integriert werden können.

Rahmenbedingungen

Erstprüfer

Ausgabedatum: 15.09.2010

Prof. Dr. Bernhard Bürg (Hochschule Karlsruhe)

Abgabedatum: 13.12.2010

Zweitprüferin

Bearbeitungszeit: 4 Monate

Dipl.-Informatikerin Verena Klinger (DLR)

Vorwort

Mit dieser Diplomarbeit schlieÙe ich mein Studium der Kartographie und Geomatik ab. Während der Studienzeiit konnte ich den faszinierenden Wandel hin zur digitalen Kartographie hautnah miterleben.

Google legte zu Beginn meines Studiums mit *Google Maps*[™] im Jahre 2005, durch die gute Bedienbarkeit und die kostenlose Bereitstellung, den Grundstein für die Entwicklung von Kartenanwendungen im Internet hin zu einer digitalen Kulturtechnik. Google handelt aus finanziellem Interesse. Kartennutzer werden als Konsumenten gesehen und restriktive Lizenzen beschränken künstlich die Kreativität der Menschen.

Die Kreativität ist mittlerweile entfesselt worden. Mit dem freien Projekt *OpenStreetMap* entsteht ein Weltwunder der Neuzeit, das ich, zusammen mit hunderttausenden anderen Menschen mitgestalten kann. Man versteht das Ausmaß erst, wenn man sich veranschaulicht, dass dabei ein Bild der Erde mit unglaublicher Detaildichte gezeichnet wird, das gedruckt eine Breite von 80km hätte. Getrieben von der Faszination, gemeinschaftlich etwas Großes zu schaffen.

*We now use the country itself, as its own map, and I
assure you it does nearly as well. (Lewis Carroll, 1893)*

Besonders danken möchte ich meinen Eltern Uwe und Gabi und meinen beiden Geschwistern Meike und Wibke, die in schwierigen Situationen zu mir gestanden und mir Mut zugesprochen haben.

Für die Betreuung meiner Diplomarbeit und das entgegengebrachte Vertrauen danke ich Verena Klinger und auch Thilo Wehrmann. Binh für den leck'ren vietnamesischen Kaffee, der so gut nach Schokolade riecht und Magith als Vorbild dafür, wie man die Arbeit auch unter Zeitdruck zu Ende bringt.

Danke auch an Herrn Prof. Dr. Bürg, der die kurzfristige Abgabe möglich gemacht hat.

Inhaltsverzeichnis

Vorwort	IIV
Inhaltsverzeichnis.....	IVI
Abbildungsverzeichnis.....	VI
Tabellenverzeichnis	VII
Listings (Programmausdrücke).....	VII
Abkürzungsverzeichnis	IX
Glossar.....	X
1 Einleitung.....	11
1.1 Problembeschreibung.....	12
1.2 Zielsetzung.....	14
1.3 Vorgehensweise.....	15
2 Stand der Technik	16
2.1 Geodaten	16
2.1.1 Datenformate	16
2.1.2 Metadaten	18
2.1.3 Visualisierung.....	19
2.2 Geodatendienste	20
2.2.1 Web Processing Service (WPS)	20
2.2.2 Sensor Observation Service (SOS)	21
2.3 Internetbasierte Geoinformationssysteme.....	22
2.4 Softwarearchitekturmuster	22
2.4.1 Model – View – Controller (MVC).....	22
2.4.2 Representational State Transfer (REST)	24
2.4.3 Muster für grafische Benutzeroberflächen	24
3 „Status quo“ der Datenintegration in ELVIS	25
3.1 ELVIS Dateninhalte.....	25
3.2 Automatische Datenintegration	26
3.2.1 WISDOM Geodata Exchange Format (WGEF).....	26
3.2.2 DatenEingangsPortal (DEP).....	27
4 Entwicklung eines grafischen Assistenten für die Zusammenstellung und Integration von Geodaten in ELVIS	32
4.1 Rahmenbedingungen	32
4.1.1 Google Web Toolkit (GWT).....	32
4.1.2 SmartGWT	33
4.1.3 Grafische Benutzeroberfläche von ELVIS	33

4.2	Anforderungserhebung.....	35
4.2.1	Nichtfunktionale Anforderungen.....	35
4.2.2	Funktionale Anforderungen.....	36
4.3	Entwurfsplanung.....	38
4.3.1	Teilkonzepte.....	38
4.3.2	Gesamtkonzept.....	41
4.4	Implementierung	50
4.4.1	Grafische Oberfläche	50
4.4.2	Funktionen.....	55
4.5	Ergebnisse	63
4.5.1	Diskussion	68
5	Erweiterung DEP für den Import von Sensormessdaten	69
5.1	Rahmenbedingungen	70
5.1.1	Sensoren und Sensormessdaten in ELVIS	70
5.2	Anforderungserhebung.....	72
5.2.1	Nichtfunktionale Anforderungen.....	72
5.2.2	Funktionale Anforderungen.....	72
5.3	Entwurfsplanung.....	74
5.3.1	Teilkonzepte.....	74
5.4	Implementierung	79
5.4.1	Erweiterung.....	79
5.4.2	Verarbeitung.....	80
5.4.3	Einlesen & Validierung.....	81
5.5	Ergebnisse	85
5.5.1	Aufruf.....	85
5.5.2	Softwaretest	85
6	Schlussbetrachtung	87
6.6	Zusammenfassung.....	87
6.7	Fazit.....	88
	Quellenverzeichnis	89
	Anhang	95

Abbildungsverzeichnis

Abbildung 1.01 – Kartenansicht des „Environmental Visualisation and Information System“ (ELVIS)	12
Abbildung 2.01 – Kernprofil des ISO 19115 Metadatenstandards	18
Abbildung 2.02 – Schematische Darstellung der Integration von Sensormessdaten	21
Abbildung 2.03 – UML-Diagramm der MVC-Architektur in ELVIS	23
Abbildung 3.01 – ELVIS Systemarchitektur	27
Abbildung 3.02 – Automatische Datenintegration mittels des DatenEingangsPortal.....	28
Abbildung 3.03 – WPS Kapselung des DEP	30
Abbildung 4.01 – ELVIS Layout Gestaltung.....	34
Abbildung 4.02 – WGEF Zusammenstellung durch einen WPS-Dienst	39
Abbildung 4.03 – Datenfluss	43
Abbildung 4.04 – Bedienkonzept.....	44
Abbildung 4.05 – Layoutkonzept.....	46
Abbildung 4.06 – Startbildschirm	48
Abbildung 4.07 – Initialisierung der Editoren für Datensatzbestandteile.....	49
Abbildung 4.08 – Schaltflächen für die Navigation	49
Abbildung 4.09 –Navigationsschaltflächen zum Abschließen der Aufgabe	50
Abbildung 4.10 – Verknüpfung der JAVA-Klassen mit den Toolboxes in der GUI Konfigurationsdatei	51
Abbildung 4.11 – Instanziierung der Toolboxes.....	51
Abbildung 4.12 – Initialisierung der Bearbeitungstabs am Beispiel eines Geodatensatzes .	55
Abbildung 4.13 – Funktion updateTabs(type)	56
Abbildung 4.14 – Sequenzdiagramm für den Aufruf des nächsten Bearbeitungsschritt.....	57
Abbildung 4.15 – Erstellung WGEF und Integration	61
Abbildung 4.16 – Startbildschirm	63
Abbildung 4.17 – Eingabe von Geodaten.....	64
Abbildung 4.18 – Metadateneingabe.....	65
Abbildung 4.19 – Eingabe Darstellungsvorschriften.....	66
Abbildung 4.20 – Datenintegration	67
Abbildung 5.01 – Datenbankmodellierung für die Verwaltung von Sensoren und Sensormesswerte	70
Abbildung 5.02 – WPS Kapselung des DatenEingangsPortal.....	74
Abbildung 5.03 – JAVA-Klassen	76
Abbildung 5.04 – Verarbeitung der Sensormessdaten.....	80
Abbildung 5.05 – Validierung der Dateneingaben	82

Tabellenverzeichnis

Tabelle 3.01 – ELVIS Dateninhalte	25
Tabelle 4.01 – Nichtfunktionale Anforderungen.....	35
Tabelle 4.02 – Funktionale Anforderungen.....	36
Tabelle 4.03 – Schritte für die Zusammenstellung und Integration eines WGEF Datensatzes in ELVIS	42
Tabelle 4.04 – Overview Toolbox, Auswahl Datensatztyp und Dateneingabe	47
Tabelle 4.05 – Zu speichernde Daten für einen Datensatztyp	52
Tabelle 4.06 – Ereignistypen für die Kommunikation zwischen Toolboxes.....	54
Tabelle 5.01 – Nichtfunktionale Anforderungen.....	72
Tabelle 5.02 – Funktionale Anforderungen.....	72

Listings (Programmausdrücke)

Listing 3.01 – Ordner- und Dateistruktur eines Geodatensatzes nach dem WGEF Standard (Zip-Archiv)	26
Listing 3.02 – Kommandozeilenaufruf des DatenEingangsPartal	28
Listing 3.03 – HTTP Anfrage zur Bestimmung der Kennziffer des Datenanbieters („Provider“)	29
Listing 3.04 – HTTP Antwort zur Bestimmung der Kennziffer des Datenanbieters.....	29
Listing 3.05 – HTTP Anfrage zur Bestimmung der Kennziffer der thematischen Gruppe	29
Listing 3.06 – HTTP Antwort zur Bestimmung der Kennziffer der thematisch Gruppe ..	30
Listing 3.07 – Aufruf des gekapselten DatenEingangsPortal über das Internet	30
Listing 4.01 – Beispielaufruf des WPS Dienstes zum Erstellen eines WGEF Archivs.....	40
Listing 4.02 – Datenmodellierung innerhalb der OverviewTBDataSource.....	52
Listing 4.03 – Umsetzung der Datenmodellierung der OverviewTBDataSource in JAVA	52
Listing 4.04 – „Weiter“-Schaltfläche in der Control Toolbox, NextEvent Ereignis.....	58
Listing 4.05 – Verarbeitung des NextEvent-Ereignisses durch die DataEntry Toolbox.....	58
Listing 4.06 – Veranlassen der Speicherung der Eingaben	59
Listing 4.07 – Validierung und Speicherung innerhalb des MetadataTabController	59
Listing 4.08 – Weiterleitung der Antwort an den DataEntry-Controller	59
Listing 4.09 – Verarbeitung der Antwort durch den DataEntry-Controller.....	60
Listing 4.10 – Generieren eines CompleteEvent: Funktion tabCompleetEvent.....	60
Listing 4:11 – Reaktion auf das CompleteEvent durch DataEntryTB.java	60
Listing 4.12 – Reaktion auf das „CompleteEvent“ durch die OverviewTB.....	60
Listing 4.13 – Setzen des Status des Bearbeitungsschrittes in der OverviewTB.....	61
Listing 5.01 – Kodierung von Sensormessdaten im kommagetrennten CSV-Format.....	71
Listing 5.02 – Bestimmen der UUID anhand des Modul Akronyms über die Tabelle observations.module.....	78

Listing 5.03 – Speicherung der Werte in der Tabelle observations.{uuid} über „Copy“ (PostgreSQL)	78
Listing 5.04 – Speicherung der Werte in der Tabelle observations.{uuid} über eine INSERT-Anweisung	79
Listing 5.05 – Weiche zum Aufruf der Startfunktion	79
Listing 5.06 – Adresse des Sensor WPS Dienstes.....	80
Listing 5.07 – Einlesen der Dateien	81
Listing 5.08 – Klasse Measurements.....	81
Listing 5.09 – Typumwandlung von String zu Date der Werte in der Spalte „time“	82
Listing 5.10 – Überführung der Ausgelesenen Werte in Measurement-Objekte	83
Listing 5.11 – Fehlermeldung bei Verletzung des Fremdschlüssel-Contstraint	84
Listing 5.12 – Methode save() der Klasse Module.java	84
Listing 5.13 – Aufruf der Erweiterung.....	85
Listing 5.14 – Beispieldatensatz Sensormessdaten (measurements.csv)	85

Abkürzungsverzeichnis

CAW ^a	Central Asian Water
DEP	DatenEingangsPortal / DataEntryPortal
DFD	Deutsches Fernerkundungsdatenzentrum
DLR	Deutsches Zentrum für Luft- und Raumfahrt
ELVIS	EnvironmentaL Visualisation and Information System
GDI	Geodateninfrastruktur
GIS	Geoinformationssystem
GML	Geography Markup Language
GUI	Graphical User Interface
GWT	Google Web Toolkit
HTTP	Hyper Text Transfer Protocol
ISO	International Standards Organisation
KML	Keyhole Markup Language
MVC	Model-View-Controller
OGC	Open Geospatial Consortium
REST	Representational State Transfer
SLD	Styled Layer Descriptor
SOS	Sensor Observation Service
SWE	Sensor Web Enablement
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
WGEF	WISDOM Geodata Exchange Format
WISDOM	Water Related Information System for the Sustainable Development of the Mekong, Vietnam
WMS	Web Map Service
WPS	Web Processing Service
XML	Extensible Markup Language

Glossar

Assistent

Ein Assistent (engl. „Assistant“ oder „Wizard“) ist in der Datenverarbeitung eine Softwarekomponente, die den Nutzer bei der Erreichung eines Aufgabenziels unterstützt. Dafür wird der Nutzer, unter Anleitung, durch eine Folge von Teilschritten geführt.

Nach Beendigung eines Teilschrittes kann er mit einer „Weiter“-Schaltfläche zum folgenden Schritt navigieren. Mit einer „Zurück“-Schaltfläche lassen sich zuvor getätigte Eingaben überprüfen und gegebenenfalls korrigieren.

Softwareassistenten finden häufig bei Installationsroutinen von Desktop-Anwendungen oder beispielsweise bei Bestellprozessen in Online-Shops Anwendung.

Datenintegration

Unter Datenintegration wird in dieser Arbeit das Überführen von Daten in Inhalte des Informationssystems verstanden. Dafür werden die Daten von einem Datenlieferanten bereitgestellt und in den Datenstrukturen des Zielsystems gespeichert. Das Ziel des Vorgangs ist es die Daten innerhalb des Informationssystems für Analysen und die Darstellung verfügbar zu machen.

Datensatz

Als Datensatz wird in dieser Arbeit eine Sammlung von Dateien verstanden, die zusammen alle benötigten Informationen für eine erfolgreiche Integration in das Informationssystem enthalten. Die Vollständigkeit legt ein definierter Standard fest. Ein Datensatz vom Typ „Geodaten“ gilt z. B. nach dem „WISDOM Geodata Exchange Format“ als vollständig, wenn Geodaten, Metadaten, und Darstellungsregeln innerhalb einer bestimmten Ordnerstruktur in einem Zip-Archiv vorliegen. Einzelne Dateien eines Datensatzes werden als Datensatzbestandteile oder Datensatzelemente bezeichnet.

Grafische Benutzeroberfläche (GUI)

Eine grafische Benutzeroberfläche (*Graphical User Interface, GUI*) ist ein Hilfsmittel für die Bedienung eines technischen Systems durch Menschen. Sie besteht aus grafischen Komponenten, die in einem Layout angeordnet sind und der Interaktion, Dateneingabe und Datenpräsentation dienen. Die Bedienung erfolgt mit einem Zeigergerät und einer Tastatur.

1 Einleitung

Die Natur beeinflusst das Leben auf der Erde - der Mensch beeinflusst die Natur. Technologie hilft Veränderungen in unserer Umwelt zu erkennen, den Grund dafür zu verstehen und gezielt zu handeln. Daten bilden das Gedächtnis und erlauben über Datenverarbeitung den Blick auf Vergangenes, die Gegenwart und die vermeintliche Zukunft. Die gewonnenen Informationen können die Grundlage für zukünftiges Handeln sein, um das menschliche Zusammenleben auf der Erde positiv zu gestalten.

Durch das Bevölkerungswachstum, Umweltverschmutzung, Klimawandel und den kontinuierlich steigenden Wasserverbrauch wird es zukünftig vermehrt zu Konflikten um die Ressource Süßwasser kommen.

Die beiden Projekte WISDOM¹ und CAWa² adressieren diese Problematik. Mit Hilfe von Fernerkundungsdaten und Sensornessstationen werden großflächig Daten zum Wasserhaushalt in den Regionen des Mekong Deltas und Zentralasiens gewonnen und verarbeitet.

An diesen Projekten sind, neben dem Deutschen Fernerkundungsdatenzentrum (DFD) zahlreiche andere Organisationen an der Bearbeitung verschiedener Fragestellungen beteiligt. Die Ergebnisse sollen einem breiten Benutzerkreis zur Verfügung gestellt werden, der einerseits aus den beteiligten, deutschen Institutionen - andererseits aus Projektpartnern der Länder Zentralasiens und Vietnam besteht. Die gewonnenen Erkenntnisse sollen Entscheidungen im Bereich der Planung zur Nutzung von Wasserressourcen unterstützen. Die technischen Werkzeuge werden von Menschen bedient und sollen einen einfachen Zugang zu den Informationen ermöglichen. Dabei sollten technische Details der Umsetzung für die Nutzer unsichtbar sein, damit diese sich auf ihr Fachwissen konzentrieren können und nicht an der Hürde der Bedienung des Systems scheitern. Daraus ergibt sich die wichtige Rolle der Schnittstellen zwischen Mensch und Technik.

¹ WISODM - **W**ater Related **I**nformation **S**ystem for the Sustainable **D**evelopment of the **M**ekong

Homepage: http://www.wisdom.caf.dlr.de/intro_en.html

² CAWa - **C**entral **A**sian **W**ater

Homepage: <http://www.cawa-project.net/>

1.1 Problembeschreibung

Im Rahmen der Projekte WISDOM und CAWa entwickelt das DFD (Deutsches Fernerkundungsdatenzentrum), ein Institut des DLR (Deutsches Zentrum für Luft- und Raumfahrt), ein internetbasiertes Umweltinformationssystem mit dem Namen *ELVIS* (*Environmental Visualisation and Information System*). Das System stellt den Nutzern Informationen aus den Bereichen der Hydrologie, Soziologie und der Erdbeobachtung für die Regionen des Mekong Delta in Vietnam (Projekt WISDOM) und die fünf Länder Kasachstan, Usbekistan, Kirgisistan, Turkmenistan und Tajikistan in Zentralasien (Projekt CAWa) zur Verfügung. **Abbildung 1.01** zeigt die grafische Oberfläche von ELVIS in der Kartenansicht.

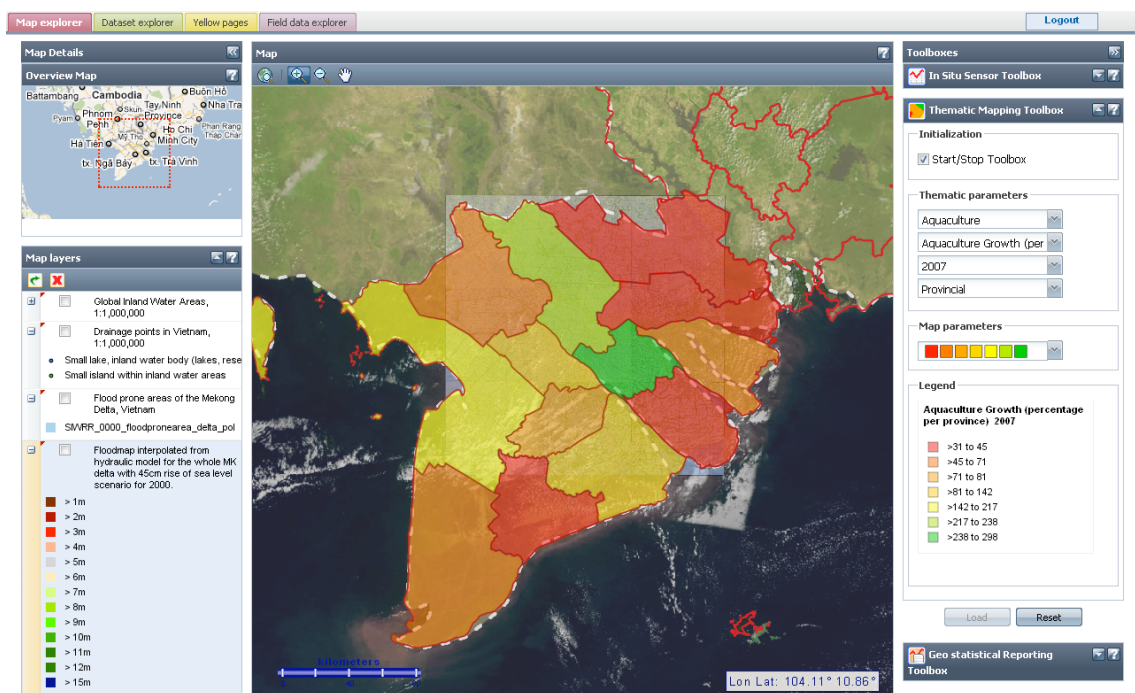


Abbildung 1.01 – Kartenansicht des „Environmental Visualisation and Information System“ (ELVIS),
Quelle: WISDOM Projekt Dokumentation

Nutzer können innerhalb von ELVIS nach Datensätzen suchen. Dabei können zeitliche, räumliche und thematische Suchkriterien angegeben werden. Hat der Nutzer einen oder mehrere Datensätze identifiziert, die für ihn von Interesse sind, lassen sich die Daten als „Ebene“ zu einer Basiskarte hinzufügen. Neben Geodaten im Raster- oder Vektorformat, verwaltet das System unter anderem Sensormessstationen und erlaubt den Abruf von Messdaten dieser Stationen.

Die Konzeption des Systems sieht vor, dass Inhalte von Nutzern oder technischen Systemen bereitgestellt und innerhalb der ELVIS Speichersysteme gespeichert und verwaltet werden.

Für die Integration von Geodaten wurde ein Datenaustauschformat WGEF (*WISDOM Geodata Exchange Format*,) entwickelt, das als „Behälter“ für die Bestandteile eines Geodatenatzes dient. Bestandteile des Geodatenatzes sind neben den Geometriedaten im Raster- oder Vektordateiformat (*GeoTiff / ESRI Shapefile*), eine Dokumentation der Daten und deren Erhebung mittels Metadaten nach der *ISO 19115 Norm – Geographic Information*³ und Regeln für die Symbolisierung nach dem *SLD-Standard*⁴ (*Styled Layer Descriptor*).

Eine vom DFD entwickelte Datenintegrationssoftware nimmt einen WGEF-Datenatz entgegen und verteilt dessen Bestandteile in der Geodateninfrastruktur. Die automatische Integration von Sensormessdaten wird von der Software nicht unterstützt.

Zur Erzeugung eines gültigen Datenatzes nach dem WGEF-Standard ist es für die Eingabe der Datenatzbestandteile zunächst notwendig verschiedene Programme lokal auf dem Rechner zu installieren und deren Bedienung zu erlernen. Die Bestandteile müssen in einem ZIP-Archiv in einer vorgegebenen Datei- und Ordnerstruktur zusammengestellt werden. Erst wenn diese Voraussetzungen erfüllt sind, ist die Datenintegration in das System möglich. Die Vorbereitungen für den Import sind zeitaufwendig und erfordern ein fundiertes technisches Wissen des Datenlieferanten über zugrunde liegende Datenstandards, Datenstrukturen und Vorgänge zum Einbinden der Daten in das System.

³ http://www.gdi-de.org/de_neu/thema/2009/c_thema_iso_uebersetzung.html
(Abgerufen am 7.8.2010)

⁴ <http://www.opengeospatial.org/standards/sld> (Abgerufen am 7.8.2010)

1.2 Zielsetzung

Diese Diplomarbeit verfolgt zwei Ziele. Ein Ziel ist es, den Datenlieferanten ein Werkzeug an die Hand zu geben, um Geodaten für die Integration vorzubereiten und sie durch den Integrationsvorgang zu führen. Das zweite Ziel ist die Erweiterung der bestehenden Integrationssoftware für die Speicherung von Sensormessdaten.

Für die Erreichung des ersten Ziels wird ein grafischer Assistent entworfen und in der Programmiersprache JAVA und mit den Frameworks *GWT (Google Web Toolkit)* und *SmartGWT* realisiert.

Der Assistent unterstützt die Nutzer bei der Eingabe und der Zusammenstellung von Geodatenätzen nach dem WGEF-Standard, indem er Editoren für die einzelnen Datensatzbestandteile bereitstellt. Durch klar definierte Schritte und Handlungsanweisungen wird der Nutzer durch den Eingabeprozess geführt. Ein Studium der WGEF-Spezifikation wird überflüssig. Der Assistent stellt Funktionen und eine grafische Benutzeroberfläche bereit, um Geodaten hochzuladen, mittels Metadaten zu beschreiben, die grafische Darstellung festzulegen und den Integrationsprozess zu starten. Er fügt sich in die bestehende grafische Oberfläche ein, so dass eine Installation von proprietärer Einzelplatzsoftware nicht mehr notwendig ist.

Das zweite Ziel besteht in der Erweiterung der bestehenden Integrationssoftware für den automatischen Import von Messdaten von in-situ Sensormessstationen in ELVIS. Der Software werden Messdaten als Eingabe übergeben. Diese werden ohne weitere Benutzerinteraktion verarbeitet und zu registrierten Sensoren hinzugefügt. Die Daten können nach der Integration durch das System abgerufen werden. Eine Registrierung von neuen Sensoren wird nicht angestrebt.

1.3 Vorgehensweise

In **Kapitel 1** wurde das *Environmental Information and Visualisation System (ELVIS)* kurz vorgestellt, die Problemstellung beschrieben und Ziele für die Lösung formuliert. Dieses Unterkapitel 1.3 beschreibt die Vorgehensweise zur Zielerreichung.

Kapitel 2 gibt einen Einblick in den Stand der Technik. Das Kapitel beschreibt die technischen Grundlagen. Dafür wird auf die Architektur von Geoinformationssystemen eingegangen, verschiedene Aspekte von Geodaten wie die Kodierung in Datenformaten, deren Dokumentation mittels Metadaten und die kartografische Visualisierung mit dem SLD-Standard vorgestellt. Für den Austausch und die Verarbeitung von Geodaten werden die beiden Dienste *Web Processing Service* und *Sensor Observation Service* des *Open Geospatial Consortium* eingeführt.

Kapitel 3 beschreibt den aktuellen Entwicklungsstand der Datenintegration in ELVIS. Es wird auf die Rahmenbedingungen eingegangen, die durch die bisherigen Entwicklungsarbeiten vom DFD geschaffen wurden und die Grundlage für die folgenden Arbeiten bilden. Dazu wird eine Übersicht über die durch ELVIS verwalteten Inhalte und deren Speicherung gegeben. Darüber hinaus stellt das Kapitel die WGEF-Spezifikation und die Software für die automatische Integration von Geodaten in ELVIS vor.

In **Kapitel 4** wird der grafische Assistent für die Zusammenstellung und Integration von Geodaten in ELVIS entwickelt. Die Entwicklung führt über die Anforderungserhebung, der Entwurfsplanung hin zu der Implementierung des Konzeptes. Am Ende des Kapitels werden die Ergebnisse vorgestellt.

In **Kapitel 5** wird die bestehende Integrationssoftware für den Import von Sensormessdaten erweitert. Analog zur Vorgehensweise in Kapitel 4 werden Anforderungen an die Erweiterung erhoben, ein Konzept erstellt und das Konzept implementiert. Am Ende des Kapitels werden die Ergebnisse vorgestellt.

In **Kapitel 6** werden die Arbeiten aus den vorherigen Kapiteln zusammengefasst und ein Fazit gezogen.

2 Stand der Technik

2.1 Geodaten

Der Begriff Geodaten ist die Kurzform für geographische Daten. Dies sind Daten, denen über Koordinaten Orte auf der Erdoberfläche zugeordnet werden. Geodaten können einen direkten räumlichen Bezug haben, wenn sie Koordinaten innerhalb der Daten enthalten, oder einen indirekten Raumbezug haben, wenn die Daten keine Koordinaten enthalten, diese jedoch auf Objekte mit Raumbezug verweisen.

Ein Bezugssystem legt die angenommene, dreidimensionale Gestalt der Erde fest (z. B. *WGS84*⁵, Erde = Ellipsoid mit spezifischen Eigenschaften). Eine Projektion überführt Orte auf der Oberfläche des Bezugssystems auf eine Fläche (z. B. *UTM*, Projektion auf einen gedachten Schnitzzylinder und Abrollen der Zylinderoberfläche). Damit werden den Orten auf der dreidimensionalen Oberfläche Koordinaten in einem zweidimensionalen Koordinatensystem zugewiesen. Dadurch lassen sich Geodaten auf ebenen Karten darstellen. Es entstehen dabei jedoch zwangsläufig Verzerrungen bei Winkeln und/oder Strecken und/oder Flächen.

2.1.1 Datenformate

Für die Speicherung bzw. den Austausch von Geodaten lassen sich zwei verschiedene Formattypen unterscheiden. Vektordaten und Rasterdaten. Vektordaten besitzen als geometrische Primitiven Punkt, Linie und Fläche. Bei Rasterdaten ist die geometrische Primitive ein Pixel. Die geometrischen Primitiven werden in verschiedenen Datenformaten formatspezifisch modelliert. Geodaten können innerhalb eines Dateisystems oder in Datenbanksystemen gespeichert werden.

ESRI Shapefile

Ein ESRI-Shapefile ist ein Datenformat für die Speicherung von Vektordaten. Es wurde von der Firma ESRI (*Environmental Systems Research Institute*) bereits Anfang der 1990er entwickelt⁶ und ist zu einem Quasi-Standard im Umfeld von Geodaten geworden. Durch die weite Verbreitung von ESRI-Software liegt eine große Anzahl an Daten in diesem Da-

⁵ http://de.wikipedia.org/wiki/World_Geodetic_System_1984 (Stand: 17.11.2011, 21:19 Uhr)

⁶ <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> (abgerufen am 08.10.2010)

teiformat vor und wird von vielen (freien) Desktop-Geoinformationssystemen unterstützt⁷. Im Zusammenhang von Geodaten und Internet haben sich die Standards mit dem Namen GML (*Geography Markup Language*) und KML (*Keyhole Markup Language*) etabliert. Diese besitzen jedoch eine geringere Unterstützung durch die verschiedenen Desktop-Informationssysteme⁶.

Die geometrischen Primitiven innerhalb von Shapefiles werden unter anderen als Punkte, Polylinien (Linienzüge: miteinander verbundene Koordinaten) und Polygone (Flächen: geschlossene Polylinien) modelliert. Zu den enthaltenen Objekten (*features*) können Sachdaten (*attributes*) hinzugefügt werden (z. B. Name eines Flusses).

Ein ESRI-Shapefile besteht aus mehreren Dateien. Die Dateien besitzen das gleiche Präfix im Dateinamen, jedoch unterschiedliche Dateiendungen. Es gibt optionale und verpflichtende Dateien. Verpflichtende Dateien sind die Hauptdatei (*.shp), eine Tabelle, die die Sachdaten der einzelnen Objekte (*.dbf) enthält und eine Indexdatei (*.shx) für die Optimierung des Zugriffs auf die Inhalte. Optionale Dateien sind zum Beispiel eine *.prj Datei, die das Koordinatensystem festlegt oder eine *.shp.xml Datei, die Metadaten beinhaltet. Das Format legt nicht fest, wie die Daten zum Beispiel auf einer Karte dargestellt werden sollen.

GeoTiff

Ein GeoTiff ist ein Dateiformat für die Speicherung von Rasterdaten. Es ist eine spezielle Form eines Tiff-Bildes, das innerhalb der Datei Koordinaten für die Georeferenzierung des Bildausschnittes sowie zur verwendeten Kartenprojektion enthält. Ein Tiff-Bild erlaubt die verlustfreie Kodierung von Daten in einer rechteckigen Rastermatrix, bestehend aus Zeilen und Spalten. Die Dateien können mehrere „Bänder“ enthalten. Dies sind mehrere übereinander gelegte Bilder, deren Pixel jeweils eigene Werte beinhalten und z. B. bei Fernerkundungsdaten optische Objekteigenschaften verschiedener Wellenlängen widerspiegeln. GeoTiff-Dateien besitzen die Dateiendungen *.tiff oder *.tif oder *.geotiff.

Das Format beinhaltet keine Regeln für die Visualisierung, wie zum Beispiel die Zuweisung von Pixelwerten zu Klassen und die Darstellung mittels Farbwerten.

⁷ spreadsheets.google.com/ccc?key=0Albk_XRkhVkdGxyYk8tNEZvLUp1UTUzTFN5bjlLX2c&hl=en
(Stand: 2010, abgerufen am 08.10.2010)

2.1.2 Metadaten

„Metadaten sind als strukturierte Dokumentation von Datensätzen zu verstehen, die es Erstellern, Verwaltern und Nutzern ermöglicht, den Inhalt und damit den anwendungsbezogenen Nutzwert der beschriebenen Datensätze zu verstehen.“⁸

Der Inhalt von Metadaten ist abhängig von den Eigenschaften der zu beschreibenden Daten und dem Zweck der Metadatenerfassung. Viele Datenformate sehen die Speicherung von Metadaten innerhalb der jeweiligen Formate vor. Da verschiedene Formate jedoch unterschiedliche Metadateninhalte festlegen, ist es sinnvoll einen formatübergreifenden Standard zu verwenden, der an die jeweiligen Bedürfnisse angepasst werden kann.

Metadaten nach ISO 19115

Für die Beschreibung von Geodaten gibt es verschiedene Normen, die die Metadatenerhebung und Speicherung standardisieren. Unter anderen hat die *International Standards Organisation (ISO)* dafür eine internationale Norm *Geographic Information – Metadata (ISO 19115)* entwickelt.

Sie besteht aus 409 unterschiedlichen Elementen. Die Elemente werden 14 Klassen zugeordnet. Bestimmte, aussagekräftige Elemente sind in der Norm als Hauptelemente deklariert und im

Dataset title (M) (MD_Metadata > MD_DataIdentification.citation > CI_Citation.title)	Spatial representation type (O) (MD_Metadata > MD_DataIdentification.spatialRepresentationType)
Dataset reference date (M) (MD_Metadata > MD_DataIdentification.citation > CI_Citation.date)	Reference system (O) (MD_Metadata > MD_ReferenceSystem)
Dataset responsible party (O) (MD_Metadata > MD_DataIdentification.pointOfContact > CI_ResponsibleParty)	Lineage (O) (MD_Metadata > DQ_DataQuality.lineage > LI_Lineage)
Geographic location of the dataset (by four coordinates or by geographic identifier) (C) (MD_Metadata > MD_DataIdentification.extent > EX_Extent > EX_GeographicExtent > EX_GeographicBoundingBox or EX_GeographicDescription)	On-line resource (O) (MD_Metadata > MD_Distribution > MD_DigitalTransferOption.onLine > CI_OnlineResource)
Dataset language (M) (MD_Metadata > MD_DataIdentification.language)	Metadata file identifier (O) (MD_Metadata.fileIdentifier)
Dataset character set (C) (MD_Metadata > MD_DataIdentification.characterSet)	Metadata standard name (O) (MD_Metadata.metadataStandardName)
Dataset topic category (M) (MD_Metadata > MD_DataIdentification.topicCategory)	Metadata standard version (O) (MD_Metadata.metadataStandardVersion)
Spatial resolution of the dataset (O) (MD_Metadata > MD_DataIdentification.spatialResolution > MD_Resolution.equivalentScale or MD_Resolution.distance)	Metadata language (C) (MD_Metadata.language)
Abstract describing the dataset (M) (MD_Metadata > MD_DataIdentification.abstract)	Metadata character set (C) (MD_Metadata.characterSet)
Distribution format (O) (MD_Metadata > MD_Distribution > MD_Format.name and MD_Format.version)	Metadata point of contact (M) (MD_Metadata.contact > CI_ResponsibleParty)
Additional extent information for the dataset (vertical and temporal) (O) (MD_Metadata > MD_DataIdentification.extent > EX_Extent > EX_TemporalExtent or EX_VerticalExtent)	Metadata date stamp (M) (MD_Metadata.dateStamp)

Abbildung 2.01 – Kernprofil des ISO 19115 Metadatenstandards, Quelle: ISO 19115:2003

so genannten Kernprofil (*core metadata*) zusammengefasst (siehe **Abbildung 2.01**).

Neben sieben verpflichtenden („M“ = *Mandatory*) gibt es optionale („O“ = *Optional*) oder konditionell verpflichtende („C“ = *Conditional*) Elemente. Für die Erfüllung des Standards ist mindestens die Dokumentation der sieben Elemente *Dataset title*, *Dataset reference*, *Dataset language*, *Dataset topic category*, *Dataset Abstract*, *Metadata point of contact* und *Metadata*

⁸ http://ispace.researchstudio.at/downloads/2004/mittlboeck_schreilechner_2004_agit_iso_profil_at.pdf

ta date stamp verpflichtend. Je nach Bedürfnis können weitere Elemente aus dem Standard hinzugenommen oder der Standard nach bestimmten, im Standard festgehaltenen Regeln erweitert werden. Die Zusammenstellung wird als *Profil* bezeichnet.

Die GDI-DE (Geodateninfrastruktur Deutschland) merkt an, dass es „selbst im englischsprachigen Raum [...] schwierig [sein dürfte], alle Felder einheitlich zu interpretieren. Das ergibt sich zum einen aus sprachlichen Mehrdeutigkeiten (je nachdem, welche Fachbereich den Standard betrachtet) als auch aus verschiedenen Möglichkeiten, die Felder und Klassen in Beziehung zu setzen.“⁹

2.1.3 Visualisierung

Styled Layer Descriptor (SLD)

„Styled Layer Descriptor (SLD) ist ein XML Schema, das vom *Open Geospatial Consortium (OGC)* definiert wurde, um das Aussehen von Kartenebenen zu definieren. Mit der SLD-Darstellungsbeschreibungssprache kann das Aussehen von Vektor- und Rasterdaten beschrieben werden“¹⁰. Die Darstellung von Objekten wird über so genannte *Symbolizer*-Elemente festgelegt. Im Anhang auf Seite 97 ist ein Beispiel für den Inhalt einer SLD-Datei abgedruckt, mit der die Darstellung von Rasterdaten über einen *Rastersymbolizer* umgesetzt wurde.

Verschiedene Kartenebenen können über die SLD-Datei miteinander kombiniert werden, so dass daraus eine Karte entsteht. Die Konformität eines SLD kann über den Vergleich mit einem XML-Schemata softwaretechnisch überprüft werden.

Unterstützt wird der SLD-Standard von verschiedenen Map Servern wie beispielsweise dem UMN-MapServer¹¹ oder GeoServer¹² und von JAVA-Bibliotheken wie GeoTools¹³. Die aktuelle Version besitzt die Nummer 1.1.0 (Stand: 29.6.2007). Um eine bessere Wiederverwendbarkeit der Darstellungsregeln zu ermöglichen, wurden im Vergleich zu der Versi-

⁹ http://www.gdi-de.org/de_neu/thema/2009/c_thema_iso_uebersetzung.html (abgerufen am 07.08.2010)

¹⁰ http://de.wikipedia.org/wiki/Styled_Layer_Descriptor (Stand: 03.08.2009, 09:17 Uhr)

¹¹ <http://mapserver.org/> (abgerufen am 29.11.2010)

¹² <http://geoserver.org/display/GEOS/Welcome> (abgerufen am 29.11.2010)

¹³ <http://docs.geotools.org/stable/userguide/faq.html> (abgerufen am 29.11.2010)

on 1.0.0 die Regeln für die Symbolisierung in eine *OpenGIS Symbology Encoding* Datei ausgelagert¹⁴.

2.2 Geodatendienste

Ein Geodatendienst ist eine spezialisierte Form eines Webservice. „Das *World Wide Web Consortium* definiert die Bereitstellung eines Webservices als Unterstützung zur Zusammenarbeit zwischen verschiedenen Anwendungsprogrammen, die auf unterschiedlichen Plattformen und/oder Frameworks betrieben werden“^{15,16}. Sie sind für den Einsatz im Internet, genauer gesagt für das *Hypertext Transfer Protocol (HTTP)*¹⁷ konzipiert.

Darauf aufbauend bieten Geodatendienste Schnittstellen für den Austausch und die Verarbeitung von Daten mit Raumbezug (Geodaten). Im Bereich der Geodaten ist das *OGC (Open Geospatial Consortium)* aktiv an der Geodatendiensten beteiligt. Die Standards beinhalten häufig Leistungsbeschreibungen, jedoch keine fertigen Implementierungen. Verbreitete OGC-Dienste sind der *WMS (Web Mapping Service)* und der *WFS (Web Feature Service)* für den Abruf von Kartendaten. Für diese Arbeit sind vor allem der *WPS (Web Processing Service)* und der *SOS (Sensor Observation Service)* von Bedeutung. Diese werden im Folgenden vorgestellt.

2.2.1 Web Processing Service (WPS)

„Ein Web Processing Service (WPS) ist ein Mechanismus, um über das Internet eine räumliche Analyse von Geodaten durchzuführen. Im Jahr 2007 wurde vom *Open Geospatial Consortium (OGC)* die Version 1.0.0 des WPS-Standards definiert“¹⁸.

Ein WPS kann aber auch dazu verwendet werden, beliebige Prozessoren zu kapseln und über das Internet zur Verfügung zu stellen. Es muss sich nicht zwangsläufig um eine räumliche Analyse handeln.

Der Standard definiert die drei Operationen *GetCapabilities*, *DescribeProcess* und *Execute*. Über *GetCapabilities* wird eine Beschreibung aller, vom WPS-Dienst zur Verfügung gestellten Prozesse abgerufen. *DescribeProcess* erzeugt eine Beschreibung der jeweiligen Prozesse

¹⁴ <http://www.opengeospatial.org/pressroom/pressreleases/761> (abgerufen am 25.11.2010)

¹⁵ <http://de.wikipedia.org/wiki/Webservice> (Stand: 15.10.2010, 20:24 Uhr)

¹⁶ <http://www.w3.org/TR/ws-arch/#introduction> (abgerufen am 29.10.2010)

¹⁷ http://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol (Stand 05.07.2010, 12:15)

¹⁸ http://de.wikipedia.org/wiki/Web_Processing_Service (Stand: 27. Mär. 2010, 17:19)

mit Eingabe und Ausgabeparametern. *Execute* startet einen Prozess. Die Operationen werden über eine Internetadresse aufgerufen, an die eine XML-Datei mit Anweisungen übergeben werden kann.

2.2.2 Sensor Observation Service (SOS)

Der SOS (*Sensor Observation Service*)¹⁹ ist ein Paket verschiedener Standards, das innerhalb der SWE (*Sensor Web Enablement*) Initiative des OGC entstanden ist.

Er „[...] stellt eine einheitliche Webserviceschnittstelle zur Abfrage von Echtzeit-Sensordaten sowie Sensordatenzeitreihen dar. Die angebotenen Sensordaten umfassen einerseits Beschreibungen von Sensoren, die in der *SensorML* (*Sensor Model Language*) kodiert werden, sowie die Messwerte, die in dem *O&M-Format* (*Observations&Measurements*) kodiert werden.“²⁰

Die Funktionalitäten werden in ein verpflichtendes Kernprofil (*Core Profile*) und ein optionales, transaktionales Profil (*Transactional Profile*) unterteilt. Implementierungen eines SOS müssen mindestens die Operationen *GetCapabilities* für die Beschreibung des Dienstes, *DescribeSensor* für den Abruf von Metadaten eines Sensors und *GetObservation* für den Abruf von Messdaten unterstützen. Optional können Implementierungen die beiden Operationen des transaktionalen Profils *RegisterSensor*, für das Hinzufügen neuer Sensoren und *InsertObservation*, für das Hinzufügen von Sensormessdaten zu einem Sensor anbieten.

Die Begrifflichkeiten des SOS werden in **Abbildung 2.02** veranschaulicht.

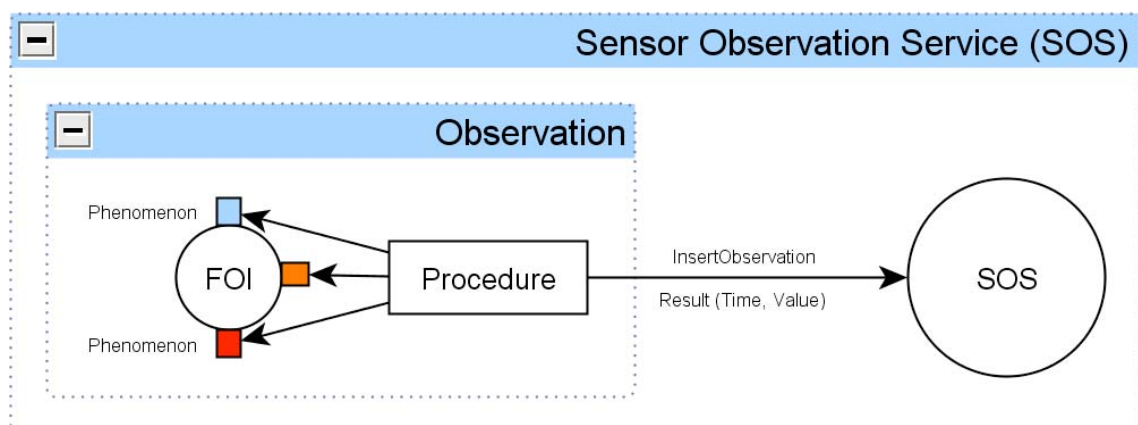


Abbildung 2.02 – Schematische Darstellung der Integration von Sensormessdaten (SOS)

¹⁹ <http://www.opengeospatial.org/standards/sos> (Abgerufen am: 02+.07.2010)

²⁰ http://de.wikipedia.org/wiki/Sensor_Observation_Service (Stand: 06.08.2010, 17:57)

Der Standard bezeichnet eine Messung als *Observation*. Eine Observation liefert einen Messwert (*Result*) für die Eigenschaft (*Phenomenon*) eines observierten Objektes (*FeatureOfInterest, FOI*). Der Vorgang der Messung wird als *Procedure* bezeichnet. Über die Operation *InsertObservation* können Messwerte zu einem registrierten Sensor hinzugefügt werden.

2.3 Internetbasierte Geoinformationssysteme

Zentraler Bestandteil von Geoinformationssystemen (GIS) sind Geodaten, die zu digitalen Karten weiterverarbeitet werden. Ein GIS bietet Funktionalitäten zum Erfassen, Verwalten, der Analyse und Ausgabe von Geodaten²¹. Kann man über das Internet darauf zugreifen, so spricht man von einem internetbasierten Geoinformationssystem. Für eine solche Art eines Informationssystems werden Begriffe wie WebGIS, InternetGIS oder OnlineGIS verwendet. Dient das GIS zur Verarbeitung von Daten über die Umwelt, so spricht man von einem Umweltinformationssystem. Es gibt jedoch keine allgemeingültigen Definitionen, die die Begriffe eindeutig untereinander abgrenzen.

Ein WebGIS besteht aus Client- und Server-Komponenten (*Client-Server-Architektur*). Bausteine sind Geodatendienste wie zum Beispiel die OGC-Dienste WMS (*Web Map Service*), WPS (*Web Processing Service*) und SOS (*Sensor Observation Service*). Clients senden Anfragen an die Server-Dienste, der Server verarbeitet die Anfrage und liefert eine Antwort zurück an den Client. Die Antwort wird daraufhin clientseitig verarbeitet und visualisiert.

2.4 Softwarearchitekturmuster

2.4.1 Model – View – Controller (MVC)

Die MVC-Architektur (*Model-View-Controller*) ist ein weit verbreitetes Softwarearchitekturmuster für die Umsetzung von grafischen Benutzungsoberflächen. Sie wurde Ende der 1970er Jahre für grafische Oberflächen von Desktop-Anwendungen in der Programmiersprache Smalltalk entwickelt²². Inzwischen wird das Prinzip auch für Internet-Anwendungen eingesetzt. Das Ziel ist eine flexible Softwarearchitektur, die eine spätere Anpassung und die Wiederverwendbarkeit von einzelnen Teilen der Anwendung ermöglicht.

²¹ Lexikon der Kartographie und Geomatik, Geoinformationssystem

²² <http://c2.com/cgi/wiki?ModelViewControllerHistory> (abgerufen am 05.09.2010)

Das Modell (*Model*) dient der Datenhaltung. Der *Controller* dient der Steuerung und reagiert auf Eingaben mit entsprechenden Aktionen. Für die Darstellung ist die Präsentationsschicht (*View*) zuständig. Sie erhält entweder über ein Push-Verfahren geänderte Daten oder fragt diese in einem Pull-Verfahren vom Modell ab.

In ELVIS ist die GUI nach dem MVC als Architekturmuster umgesetzt (siehe **Abbildung 2.03**).

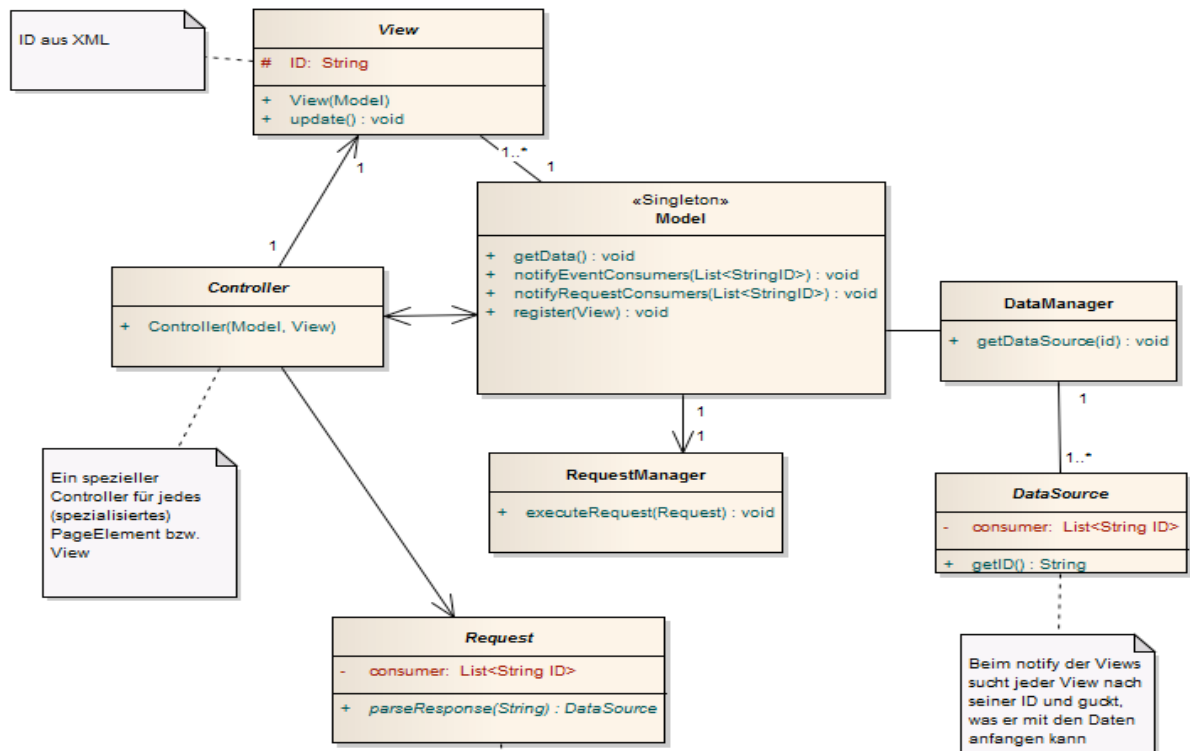


Abbildung 2.03 – UML-Diagramm der MVC-Architektur in ELVIS, Quelle: WISDOM Projektdokumentation

„Das *Model* enthält, nach entsprechenden Anfragen, die darzustellenden Daten. Alle GUI-Elemente sind eigene *Views*, die im *Model* registriert werden und jeweils einen speziellen *Controller* haben. Über den *Controller* werden Benutzeraktionen verarbeitet und beispielsweise Datenanfragen über einen spezifischen *Request* an die Datenbank geschickt. Die angefragten Daten werden im *Model* hinterlegt. Durch *notify*-Methoden werden die *View*-Elemente über die Existenz neuer Daten informiert. Daraufhin konsumieren „interessierte“ *Views* die Daten und zeigen diese an oder verarbeiten sie anderweitig. Die Identifikation interessierter *Views* geschieht über den Abgleich definierter IDs der GUI-Elemente.“²³

²³ WISDOM Projektdokumentation: GUI_Architektur_Beschreibung.doc (Stand: 30.06.2010)

2.4.2 Representational State Transfer (REST)

Als REST (*Representational State Transfer*) wird ein Softwarearchitekturmuster bezeichnet, das für die Entwicklung von verteilten Informationssystemen, beispielsweise im Internet, verwendet wird. Der Architekturstil wurde im Jahr 2000 von Roy T. Fielding in seiner Dissertation beschrieben²⁴ und folgt vier Prinzipien:

Adressierbarkeit: Daten sind Ressourcen, die über URI (*Uniform Resource Identifier*) eindeutig zugeordnet werden können.

Zustandslosigkeit: Weder der Server noch der Client speichern Zustandsinformationen zwischen zwei Nachrichten. In jeder Nachricht sind alle Informationen enthalten, um diese interpretieren zu können.

Wohldefinierte Operationen: alle Ressourcen müssen eine Reihe von wohldefinierten Operationen unterstützen. Im Fall von HTTP gehören dazu GET, POST, PUT und DELETE.

Verwendung von Hypermedia: als Medium für die Repräsentation der Daten kommt Hypermedia wie beispielsweise HTML oder XML zum Einsatz. Ressourcen werden über Hyperlinks miteinander verknüpft.

2.4.3 Muster für grafische Benutzeroberflächen

Neben Softwarearchitekturmustern (siehe zum Beispiel REST) existieren Muster für grafische Benutzeroberflächen, die beschreiben wie bestimmte Aufgaben mit Hilfe von grafischen Komponenten sinnvoll gelöst werden. Beispiele für solche Muster sind unter anderem „Tabs“, „Login“, „Menü“.

Ein Assistent ist ebenfalls eine Bezeichnung für ein solches Muster. Laut der Internetseite *ui-patterns.com* dient ein Assistent dazu ein Aufgabenziel zu erreichen, das aus mehreren, voneinander abhängigen Teilaufgaben besteht („Problem summary: The user wants to achieve a single goal which consists of multiple dependable sub-tasks“²⁵). Es wird verwendet, wenn Nutzer mit den einzelnen Teilschritten zur Erreichung des Gesamtziels nicht vertraut sind und Hilfestellungen benötigen. Die Teilaufgaben müssen in einer bestimmten Reihenfolge durchgeführt werden. Die Navigation zwischen einzelnen Bearbeitungsschritten findet mit Hilfe von „Weiter“- und „Zurück“-Schaltflächen statt.

²⁴ <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (abgerufen am 19.09.2010)

²⁵ <http://ui-patterns.com/patterns/Wizard> (abgerufen am 02.08.2010)

3 „Status quo“ der Datenintegration in ELVIS

In diesem Kapitel wird auf die Rahmenbedingen eingegangen, die durch die bisherigen Entwicklungsarbeiten vom DFD geschaffen wurden und die Grundlage für die folgenden Arbeiten bilden.

3.1 ELVIS Dateninhalte

ELVIS verwaltet heterogene Dateninhalte. **Tabelle 3.1** gibt einen Überblick über die verschiedenen Datentypen, deren Bestandteile, Speicherformate und Standards.

Tabelle 3.01 – ELVIS Dateninhalte

Datentyp		Datensatzelemente	Format (Datei- namenserweiterung)	Standard(s)
Geodaten	Vektor Geodaten	Vektor-Geometrie + Attribute	ESRI Shape (.shp, .prj, .dbf, .shx)	WGEF, ISO 19115, SLD 1.0
		Metadaten	XML (.xml)	
		SLD-Datei	XML (.sld)	
	Raster Geodaten	Raster-Geometrie	Geotiff (.tiff oder .tif)	
		Metadaten	XML (.xml)	
		SLD-Datei	XML (.sld)	
Sensor	Sensoren	Standort, Module	XML (xml)	SOS, O&M, SensorML
	Sensor- messdaten	Messwerte (Zeit, Wert, Sensor, Mess- modul)	CSV (.csv) / XML (.xml)	
Statistik	Statistiken	Werte, Fremdschlüssel	CSV (.csv)	---
Felddaten	Felddaten	Koordinate	WellKnownText (WKT)	---
		4 Fotos (Richtung Nord, Ost, Süd, West)	JPEG (.jpg) / Tiff (.tiff oder .tif)	
		Metadaten	XML (.xml)	
Publi- kation	Publikation	Textdokument	PDF (.pdf)	---
		Bibliographie	XML (.pdf)	

3.2 Automatische Datenintegration

Als automatische Datenintegration wird der Import von externen Daten durch eine Software in das Informationssystem verstanden. Die Software nimmt einen Datensatz in einem bestimmten Format entgegen, extrahiert daraus die zu verarbeitenden Daten, transformiert diese in das Schema und Format der Zieldatenbank und lädt bzw. speichert sie in der Zieldatenbank. Der Begriff *automatisch* bezieht sich auf die Tatsache, dass nach einer Datenübergabe und dem parametrisierten Start der Software keine Benutzerinteraktion mehr stattfindet. Der Prozess wird in Fachliteratur als „Extract / Transform / Load“-Prozess bezeichnet und mit ETL abgekürzt²⁶. Für den Transformationsprozess ist ein „Mapping“ der Daten notwendig. Das Mapping benötigt Informationen an welchem Ort die erwarteten Daten gespeichert sind und in welche Strukturen sie überführt werden sollen. Dafür muss ein Eingabestandard spezifiziert sein, der das Eingabedatenformat festlegt. Für die automatische Integration von Geodatensätzen in ELVIS wurde ein Austauschdateiformat entwickelt, das im Folgenden vorgestellt wird.

3.2.1 WISDOM Geodata Exchange Format (WGEF)

Die „WISDOM Geodata Exchange Format“²⁷ Spezifikation beschreibt die Zusammenstellung und Strukturierung von Geodaten zu einem ELVIS-Geodatensatz, mit dem Ziel die Daten unter den Projektpartnern auszutauschen und automatisch in ELVIS einzuspielen. Bei dem Format handelt es sich um ein proprietäres, vom DFD spezifiziertes, Dateiformat (siehe **Listing 3.01**).

Listing 3.01 – Ordner- und Dateistruktur eines Geodatensatzes nach dem WGEF Standard (Zip-Archiv)

```

01 SubNIAPP_2000_landuse_100K.zip
02 |   metadata.xml
03 |   └── private
04 |       SubNIAPP_2000_landuse_100K_poly.shp
05 |       SubNIAPP_2000_landuse_100K_poly.shx
06 |       SubNIAPP_2000_landuse_100K_poly.dbf
07 |       SubNIAPP_2000_landuse_100K_poly.prj
08 |       SubNIAPP_2000_landuse_100K_poly.sld

```

Bestandteile sind neben Geodaten im Raster- (*GeoTiff*) oder Vektorformat (*ESRI Shapefile*, Zeile 04-07), zugehörige Metadaten nach der ISO 19115 Norm (Zeile 02) und Darstel-

²⁶ Geographic hypermedia: concepts and systems, Stefanakis, E et al., 2006, Springer

²⁷ Spezifikation nach „WISDOM_WGEF_EN_01.doc“, Steffen Gebhardt, DLR, 12.01.2010

lungsvorschriften im SLD-Dateiformat (Zeile 08). Die Dateien werden innerhalb einer bestimmten Ordnerstruktur in einem Zip-Archiv zu einem WGEF-Datensatz gruppiert.

3.2.2 DatenEingangsPortal (DEP)

Das DatenEingangsPortal (DEP) ist eine vom DFD in der Programmiersprache JAVA entwickelte Software für die automatische Datenintegration von Geodaten in ELVIS. Das Programm registriert und verteilt die Bestandteile von Geodatensätzen (siehe **Listing 3.01**, vorherige Seite) in den Datenspeichersystemen des Informationssystems, so dass diese daraus abgerufen, verarbeitet und visualisiert werden können (siehe **Abbildung 3.01**).

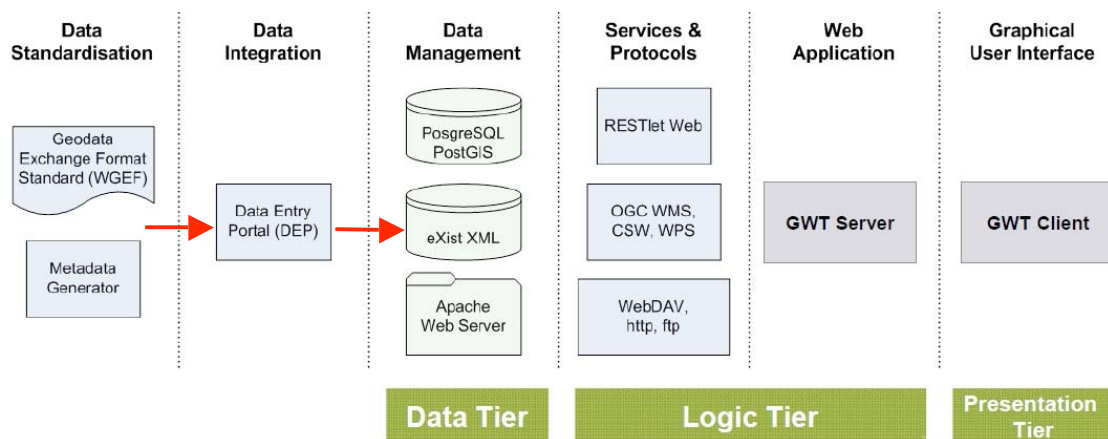


Abbildung 3.01 – ELVIS Systemarchitektur, Quelle: nach WISDOM Projekt Dokumentation

Das DEP nimmt ein WGEF-konformes ZIP-Archiv entgegen und verarbeitet dessen Inhalt. **Abbildung 3.02** auf Seite 28 stellt den Vorgang schematisch dar. Zunächst wird der Inhalt auf Vollständigkeit und standardgemäße Datei- und Ordnerstrukturen überprüft („Validierung“). Trifft dieses zu, so werden die Dateien daraus extrahiert („Entpacken“). Nach einer weiteren Analyse, ob der Dateinhalt gelesen werden kann, findet eine Verarbeitung der Daten statt (z. B. Umprojizieren der Geodaten in das Zielkoordinatensystem, automatisches Auslesen von Metadaten aus den Vektor- bzw. Rasterdateien und Erweiterung der manuell erstellten Metadaten).

In einem letzten Schritt werden die Daten an die jeweiligen Datenspeichersysteme registriert und verteilt. Vektordaten werden in einer PostgreSQL-Datenbank mit PostGIS Erweiterung gespeichert. XML-Dateien werden in einer XML-Datenbank eXist gespeichert. Rasterdateien werden im Dateisystem vorgehalten, da sie nicht effizient innerhalb der Datenbank PostgreSQL abgelegt werden können. Datensätze werden auf einem FTP-Server archiviert.

Das DEP gibt Verarbeitungs-Informationen als Ausgabertext zurück, welche Schritte durchgeführt wurden und ob Fehler dabei aufgetreten sind.

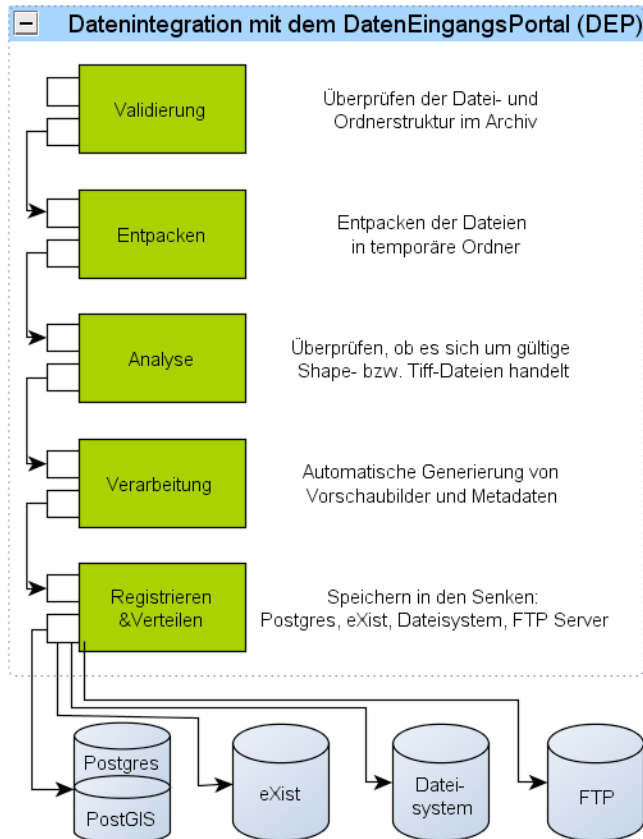


Abbildung 3.02 – Automatische Datenintegration mittels des DatenEingangsPortal (DEP), Quelle: nach WISDOM Projekt Dokumentation

Bedienung

Das Programm besitzt für den Aufruf eine Kommandozeilenschnittstelle, die mit vier Parametern aufgerufen wird (siehe **Listing 3.02**)

Listing 3.02 – Kommandozeilenaufruf des DatenEingangsPortal

```
> java -jar dep.jar -c ../config.xml -i /landuse.wgef -p 7 -g 102
```

Parameter

- c:** Pfad zur Konfigurationsdatei. Die Konfigurationsdatei enthält Schalter zur Steuerung des Informationsflusses, Adressen und Authentifizierungsdaten für die Speichersysteme.
- i:** Datei oder Ordner, der die Quelldaten (WGEF Archiv(e)) enthält.
- p:** Kennziffer für die Organisation des Datenlieferanten („Provider“).
- g:** Kennziffer für die thematische Gruppe des Datensatzes („ThematicGroup“).

Das DEP wurde als Server-Software konzipiert. Aus sicherheitstechnischen Gründen muss das DEP auf dem gleichen System vorliegen, auf dem auch die Datenbank läuft. Des Weiteren müssen die einzuspielenden Datensätze im gleichen Dateisystem wie das DEP vorhanden sein. Der Aufruf geschieht entweder lokal über die Kommandozeile oder entfernt, z. B. über die Netzwerkprotokolle SSH²⁸ oder Telnet²⁹.

Als Zusatzinformationen werden beim Start der Integrationssoftware Kennziffern für die thematische Gruppe (Parameter „-g XX“) und den Provider (Parameter „-p XXX“) übergeben. Die Parameter ermöglichen eine spätere Suche nach Datensätzen bestimmter Datenlieferanten oder Thematiken. Um die Kennziffern zu bestimmen, muss der Datenlieferant eine Internetadresse aufrufen (siehe **Listing 3.03** und **Listing 3.05**), die ein Dokument im JSON-Datenformat³⁰ als Antwort zurückliefert (siehe **Listing 3.04** und **Listing 3.06**). Darin werden allen in der Datenbank vorgehaltenen Organisationen oder thematischen Gruppen eindeutige Kennziffern zugewiesen, die der Benutzer beim Start des DEP als Parameter übergibt.

Listing 3.03 – HTTP Anfrage zur Bestimmung der Kennziffer des Datenanbieters („Provider“)

```
GET http://indus.caf.dlr.de/elvis_rest_dev/providers/json
```

Listing 3.04 – HTTP Antwort zur Bestimmung der Kennziffer des Datenanbieters

```
01  [
02    {
03      "name" : "GFZ" ,
04      "id" : 7 ,
05      "description" : "German Research Center for Geoscience
06    } ,
07    ...
08  ]
```

Listing 3.05 – HTTP Anfrage zur Bestimmung der Kennziffer der thematischen Gruppe

```
GET http://indus.caf.dlr.de/elvis_rest_dev/thematicgroups/json
```

²⁸ http://de.wikipedia.org/wiki/Secure_Shell (Stand: 12.11.2010, 10:50 Uhr)

²⁹ <http://de.wikipedia.org/wiki/Telnet> (Stand: 22.11.2010, 12:09 Uhr)

³⁰ http://de.wikipedia.org/wiki/JavaScript_Object_Notation (Stand: 30.11.2010)

Listing 3.06 – HTTP Antwort zur Bestimmung der Kennziffer der thematisch Gruppe

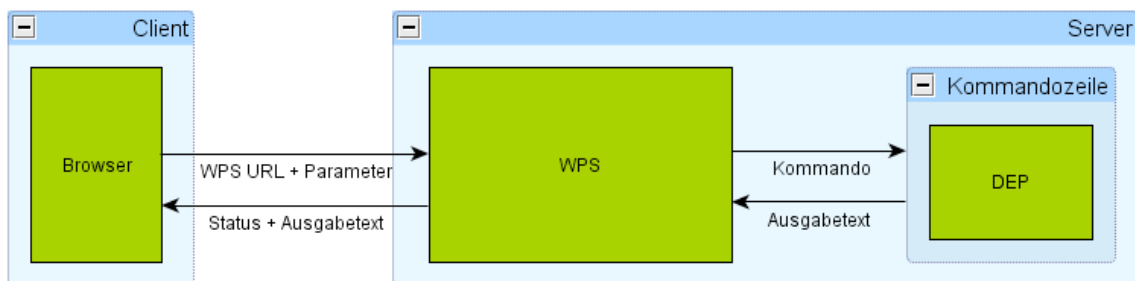
```

01 [
02   {
03     "name": "Inundation scenario from raster",
04     "id": 102
05   },
06   ...
07 ]

```

Kapselung

Um das Kommandozeilenprogramm über das Internet starten zu können, wurde vom DFD ein WPS entwickelt, der die JAVA-Software kapselt und über eine Internetadresse aufrufbar macht. **Abbildung 3.03** veranschaulicht die Kommunikation mit der JAVA Software über den WPS Dienst:

**Abbildung 3.03** – WPS Kapselung des DEP

Der WPS Dienst wird über eine Internetadresse aufgerufen und nimmt die Eingabeparameter entgegen, erstellt daraus ein Kommando und ruft das DEP über die Kommandozeile auf und führt die Integration durch (**Abbildung 3.02**). Der Ausgabebetext und der Status werden an den Aufrufer als XML-Datei zurückgegeben.

Aufruf

Der WPS-Dienst wird über eine HTTP GET Anfrage an eine URL gestartet (**Listing 3.07**).

Listing 3.07 – Aufruf des gekapselten DatenEingangsPortal über das Internet

```

01 http://mekongvm5/cgi-bin/wps.py?
02   datainputs=[
03     dataintegrator:resource=http://mekongvm5/test.wgef;
04     dataintegration:thematic=101;
05     dataintegration:provider=5]&
06   identfier=dataIntegration&
07   request=execute&
08   service=wps&
09   version=1.0.0

```

Voraussetzung:

- Geodatenatz liegt als WGEF-Archiv vor
 1. WGEF Archiv ist über eine Adresse (URL) über das Internet abrufbar
 2. Adresse (URL) ist bekannt
- Kennziffern für die „Thematische Gruppe“ und den „Provider“ sind bekannt
- URL des WPS und Name des Dienstes sind bekannt

Eingaben:

- Zeile 03: Adresse (URL) des WGEF Archivs
- Zeile 04: Identifikationsnummer - Thematische Gruppe
- Zeile 05: Identifikationsnummer - Provider

Verarbeitung:

- Der WGEF Datensatz wird von der übergebenen URL heruntergeladen
- Erstellen eines Kommandos für den Kommandozeilenaufruf des DEP
- Kommandozeilenaufruf

Ausgaben:

- XML-Dokument: Log-File mit Informationen (Ausgabertext) als Integrationsdokumentation

4 Entwicklung eines grafischen Assistenten für die Zusammenstellung und Integration von Geodaten in ELVIS

Die Komplexität der Datenintegrationsaufgabe liegt in der Vorbereitung des Geodaten-satzes, also in der Erstellung der Datensatzbestandteile und Zusammenstellung derer innerhalb eines WGEF-Archivs, als Voraussetzung für den Start des DatenEingangsPortals.

Der grafische Assistent deckt aus diesem Grund den kompletten Arbeitsfluss der Integrationsaufgabe – Erstellung, Zusammenstellung und Integration eines WGEF-Datensatzes – ab, um den Nutzer bestmöglich bei der Datenintegration in ELVIS zu unterstützen, ohne dass dieser sich in die technischen Details der Gesamtaufgabe einarbeiten muss.

Die Entwicklung führt über die Anforderungserhebung, der Entwurfsplanung hin zur Implementierung des Konzeptes. Zunächst werden jedoch die Rahmenbedingungen vorgestellt, die für die Entwicklung vorgegeben sind.

4.1 Rahmenbedingungen

Die Rahmenbedingungen sind durch die bisher vom DFD geleisteten Entwicklungsarbeiten vorgegeben. Eine wichtige Anforderung an den Assistenten ist es, dass sich dieser in die bestehende Oberfläche von ELVIS einfügt. Aus diesem Grund werden im Folgenden die Technologie und die Gestaltung der grafischen Benutzeroberfläche von ELVIS vorgestellt.

4.1.1 Google Web Toolkit (GWT)

Für die Entwicklung des Umweltinformationssystems wird das *Google Web Toolkit* (GWT)³¹ eingesetzt. Es vereinheitlicht den Entwicklungsprozess von Server- und Client-Komponenten, da es eine durchgehende Implementierung von Funktionalitäten in der Programmiersprache JAVA ermöglicht. Dadurch „sollen sich Entwickler nicht mit den Eigenarten der Browser beschäftigen müssen“³². Der Quellcode wird durch einen Java-nach-JavaScript-Compiler verarbeitet, so dass die verschiedenen Browser diesen ausführen können.

³¹ <http://code.google.com/webtoolkit/> (abgerufen am 25.10.2010)

³² <http://www.golem.de/0612/49434.html> (abgerufen am 25.10.2010)

4.1.2 SmartGWT

SmartGWT³³ ist ein Framework, um die Entwicklung von Internetanwendungen zu erleichtern, indem es die beiden Frameworks SmartClient³⁴ (*JavaScript*) und das GWT miteinander verknüpft. SmartGWT erweitert GWT um so genannte „Widgets“ und Konzepte für den Datentransport von Server zum Client und andersherum. Widgets sind in SmartGWT wiederverwendbare grafische Komponenten (z. B. Eingabefelder, Tabellen,...), die mit Datenquellen (*DataSources*) verbunden werden können. Der Inhalt wird automatisch aktualisiert, sobald sich Inhalte der Datenquellen ändern. SmartGWT ist unter verschiedenen Lizenzmodellen verfügbar³⁵. Die unterstützten Merkmale variieren je nach Lizenztyp: In der kostenlosen Version, die unter der OpenSource Lizenz LPGL steht, können ausschließlich client-seitige Funktionen genutzt werden (z. B. Widgets und client-seitige Datenquellen). Höhere Lizenzen sind kostenpflichtig und unterstützen weitere Merkmale, wie z. B. die Kommunikation mit dem Server und Konzepte zur Synchronisation von Datenquellen auf Server und Client. Es ist eine Anforderung an ELVIS, dass es auf OpenSource Komponenten aufbaut. Deshalb steht für die Implementierung des Assistenten nur die LPGL Lizenzversion mit eingeschränkter Funktionalität zur Verfügung.

4.1.3 Grafische Benutzeroberfläche von ELVIS

Die Inhalte von ELVIS sind über die grafische Benutzeroberfläche mittels eines Browser über das Internet abrufbar. Über diese Schnittstelle können Nutzer mit dem System interagieren. Die Oberfläche ist aus Seiten aufgebaut und bedient sich grafischer Komponenten von SmartGWT. Zwischen den Seiten kann über Registerkarten (*Tabs*) navigiert werden. Seiten beinhalten Werkzeugkästen (*Toolboxes*), die den Zugriff auf die Funktionalitäten erlauben. **Abbildung 4.01** auf Seite 34 zeigt die schematische Darstellung einer Seite in ELVIS. Die blauen Elemente sind die Tabs für die Navigation zwischen verschiedenen Seiten (*Navigationsleiste*). Die gezeigte Seite dient der Darstellung von Karten und beinhaltet eine Toolbox für die Ebenenverwaltung (*Layer Toolbox*), die Karte (*Map Toolbox*), den Abruf von Sensormessdaten (*Observations Toolbox*) und statistischen Daten (*Thematic Mapping Toolbox*).

³³ <http://code.google.com/p/smartgwt/> (abgerufen am 25.10.2010)

³⁴ <http://www.smartclient.com/> (abgerufen am 25.10.2010)

³⁵ <http://www.smartclient.com/product/> (abgerufen am 10.9.2010)

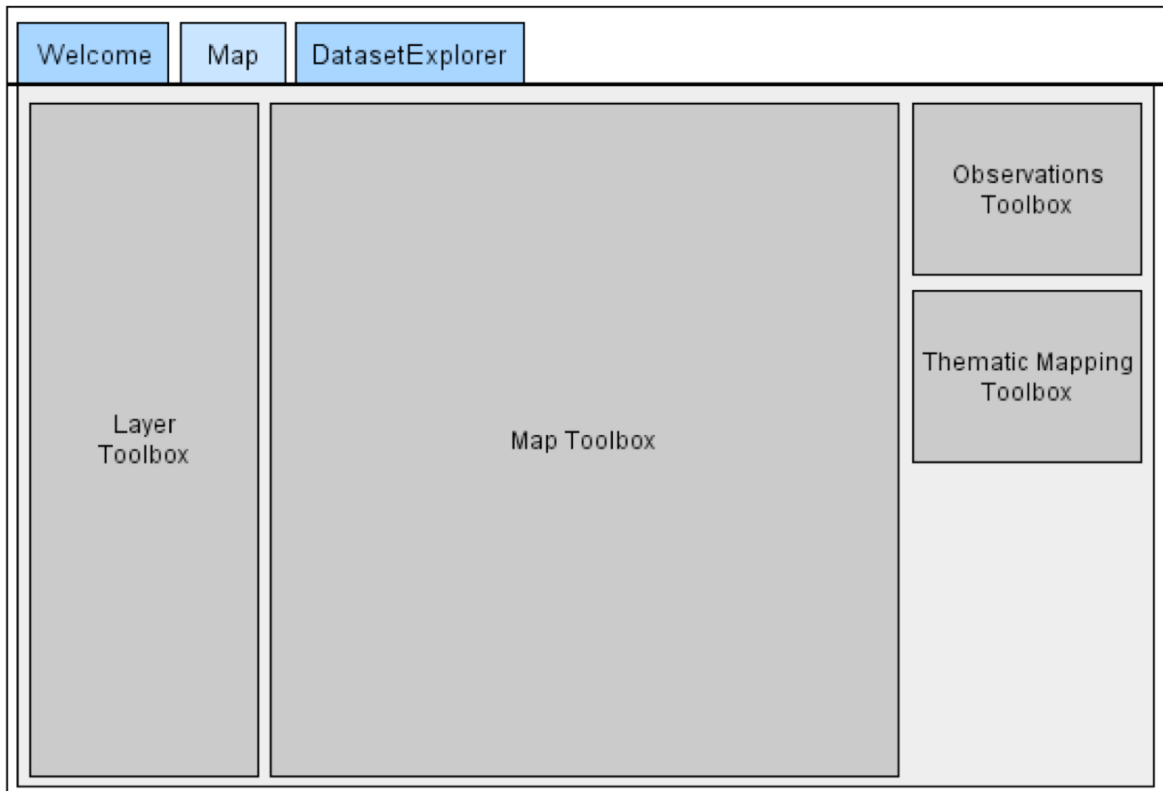


Abbildung 4.01 – ELVIS Layout Gestaltung

Die Anordnung der Toolboxes ist über eine XML-Datei konfigurierbar (siehe Anhang 7.3 auf Seite 98). In der Konfigurationsdatei lassen sich neue Seiten zu ELVIS hinzufügen und ihnen Toolboxes zuweisen.

Toolbox (Werkzeugkasten)

Toolboxes sind Einheiten, die den Zugriff auf zusammengehörige Funktionen ermöglichen. Sie können neben- und untereinander auf einer Seite angeordnet und auf verschiedenen Seiten wiederverwendet werden. Die Toolboxes sind nach dem MVC-Architekturmuster (siehe Kapitel 2.4.1) programmiert, d.h. Datenmodell, Ansicht und Steuerung sind voneinander entkoppelt. Jede Toolbox benachrichtigt bei Ereignissen (z. B. Klick auf eine Schaltfläche) oder bei Änderungen im Datenmodell andere Toolboxes, die innerhalb der Konfigurationsdatei als *EventHandler* oder *DataConsumer* registriert wurden.

4.2 Anforderungserhebung

Der Assistent ist ein Werkzeug zum Erstellen eines gültigen WGEF-Datensatzes und dessen Integration in ELVIS. Zur Lösung der Aufgabe wurden verschiedene Anforderungen an die Software identifiziert, die in Gesprächen mit den Entwicklern von ELVIS³⁶, durch Studium der Dokumentation zu dem WGEF-Datenstandard³⁷ und aus Anleitungen für die Datenintegration^{38,39} erhoben wurden. Die Anforderungen sind in „nichtfunktionale“ (Unterkapitel 4.2.1) und „funktionale“ (Unterkapitel 4.2.2) klassifiziert. Das Präfix „A_“ kürzt im Folgenden die Bezeichnung „Anforderung“ ab. Der Auflistung liegt eine hierarchische Struktur zugrunde, d.h. Anforderungen mit der Bezeichnung A_XX_XX sind durch Anforderung A_XX bedingt bzw. konkretisieren diese.

4.2.1 Nichtfunktionale Anforderungen

„Eine nichtfunktionale Anforderung legt fest, welche Eigenschaften das Produkt haben soll“⁴⁰. **Tabelle 4.01** listet die nichtfunktionalen Anforderungen auf. Im Anschluss an die Tabelle werden diese erläutert.

Tabelle 4.01 – Nichtfunktionale Anforderungen

Identifikator	Anforderung
A_01	Fügt sich in die bestehende Oberfläche des Informationssystems ein.
A_01_01	Nutzt die Programmiersprache JAVA und baut auf den Frameworks Google Web Toolkit und SmartGWT auf.
A_02	Bedienung ist intuitiv
A_03	Erweiterbarkeit

A_01 – Fügt sich in die bestehende Oberfläche des Informationssystems ein

Der Assistent wird in die bestehende ELVIS-Oberfläche eingebunden. Dafür wird innerhalb der ELVIS-Navigationsleiste eine neue Registerkarte für den Start des Assistenten hinzugefügt. Der Assistent wird als Internetanwendung umgesetzt. Dies fördert – durch eine

³⁶ Verena Klinger (DLR), Thilo Wehrmann (DLR), Steffen Gebhardt (DLR)

³⁷ WISDOM Projekt Dokumentation: 2008-04-07_WISDOM_GeodataExchangeStandard_doc_SG.pdf

³⁸ WISDOM Projekt Dokumentation: WISDOM_DEP_EN_01.doc

³⁹ WISDOM Projekt Dokumentation: 2008-07-10_WISDOM_MetadataGenerator_doc_SG.doc

⁴⁰ http://de.wikipedia.org/wiki/Anforderung_Informatik (Stand: 31. Okt. 2010, 20:20)

konsistente grafische Oberfläche – die Bedienbarkeit des Systems. Die Installation und Wartung von proprietärer Einzelplatzsoftware durch den Nutzer wird dadurch vermieden. Aus der Anforderung A_01 ergeben sich die Programmiersprache JAVA und die eingesetzten Frameworks *Google Web Toolkit* und *SmartGWT* für die Implementierung (Anforderung: A_01_01).

A_02 - Bedienung ist intuitiv

Die Nutzer können ohne das Studium externer Dokumentation bzw. Anleitungen das System bedienen und das Aufgabenziel der Datenintegration erfolgreich erreichen. Die Abfolge der Teilaufgaben soll übersichtlich und einprägsam sein, um bei wiederholter Durchführung der Integrationsaufgabe die Effizienz zu steigern.

A_03 – Erweiterbarkeit

Innerhalb dieser Arbeit wird ein Konzept für die Eingabe von Datensätzen vom Typ „Geodaten“ nach dem WGEF-Standard erstellt und umgesetzt. Auf Grundlage dieser Arbeit soll der Assistent für die Integration weiterer Datensatztypen, wie in Kapitel 3.1 auf Seite 25 aufgelistet, erweitert werden können.

4.2.2 Funktionale Anforderungen

„Eine *funktionale Anforderung* legt fest, was das Produkt tun soll.“⁴¹ **Tabelle 4.02** listet die funktionalen Anforderungen auf. Im Anschluss an die Tabelle werden diese erläutert.

Tabelle 4.02 – Funktionale Anforderungen

Identifikator	Anforderung
A_04	Erstellung eines gültigen und vollständigen WGEF Datensatzes
A_04_01	Transfer lokaler Dateien zum Server
A_04_01_01	Transfer von Dateien bis zu einer Größe von 400MB
A_04_02	Editoren zur Eingabe von Datensatzbestandteilen
A_04_02_01	Eingabe von Metadaten
A_04_02_01_01	Temporäre Speicherung der Metadaten
A_04_02_02	Eingabe von Darstellungsregeln für Geodaten
A_04_02_02_01	Temporäre Speicherung der Darstellungsregel

⁴¹ http://de.wikipedia.org/wiki/Anforderung_Informatik(Stand: 31. Okt. 2010, 20:20)

A_04_02_03	Validierung der Dateneingaben
A_05	Start des Integrationsprozesses

A_04 – Erstellung eines gültigen und vollständigen WGEF Datensatzes

Aus den Dateneingaben wird ein WGEF-konformes Archiv erstellt. Der Dienst sammelt die bereitgestellten Daten ein und speichert diese in einer vom „WISDOM Geodata Exchange Standard“ spezifizierten Datei- und Ordnerstruktur in einem ZIP-Archiv. Die Eingaben werden in den Anforderungen A_04_01: Transfer lokaler Dateien zum Server, A_04_02_01: Eingabe von Metadaten und A_04_02_02: Eingabe von Darstellungsregeln für Geodaten beschrieben.

A_04_01 – Transfer lokaler Dateien zum Server

Transfer (Upload) von lokalen Dateien (die im Dateisystem, auf dem Rechner des Datenlieferanten vorliegen) zu einem Server. Es sollen Dateien bis zu einer Größe von 400 MB unterstützt werden (A_04_01_01). Der Benutzer wählt eine Datei vom lokalen Dateisystem aus, die daraufhin zum Server übertragen wird. Während des Vorgangs bekommt der Nutzer Rückmeldung über den Status (Fortschrittsanzeige/Erfolg/Misserfolg). Auf die hochgeladenen Dateien kann mittels einer Internetadresse (URL) zugegriffen werden.

A_04_02 – Editoren für die Datensatzbestandteile

Die Editoren dienen der Dateneingabe mittels Formularmaske. Die eingegebenen Daten werden in XML-Dokumente überführt und zu einem Server übertragen. Auf die übertragene Datei kann über eine Internetadresse (URL) zugegriffen werden.

A_04_02_03 – Validierung der Dateneingaben

Eingaben in Formularmasken werden zunächst auf dem Client auf Vollständigkeit und das erwartete Format überprüft. Die Struktur der übertragenen XML-Dokumente wird gegen ein Schema validiert. Ist die Validierung des XML-Dokuments erfolgreich, so gilt die Eingabe als erfolgreich abgeschlossen.

A_05 – Start des Integrationsprozesses

Der Integrationsprozess kann über die grafische Oberfläche aufgerufen werden. Es wird der in A_04 erstellte WGEF zusammen mit den Startparametern übergeben.

4.3 Entwurfsplanung

Das Ergebnis der Entwurfsplanung ist ein Konzept für die Umsetzung der in Kapitel 4.2 identifizierten Anforderungen. Zunächst werden Teilkonzepte erarbeitet, die im Folgenden zu einem Gesamtkonzept zusammengefügt werden.

4.3.1 Teilkonzepte

Konzept zur Erfüllung von Anforderung „A_01“

Fügt sich in die bestehende Oberfläche des Informationssystems ein

Der Assistent wird als neue Seite in das Informationssystem eingefügt und wird durch Auswahl einer Registerkarte gestartet. Als mögliche Namen für den Kartenreiter kommen zum Beispiel *DataEntryPortal*, *Data Import*, *Import* oder *Data Integration* in Frage. Im Konsens mit dem Entwicklerteam wurde der Begriff *DataEntryPortal* gewählt.

Innerhalb der Seite werden Werkzeugboxen platziert, die die Funktionalitäten implementieren. Das Layout und die Registrierung der Werkzeugboxen wird über die Anpassung der ELVIS Layout Konfigurationsdatei umgesetzt.

Konzept zur Erfüllung von Anforderung „A_01_01“

Nutzt die Programmiersprache JAVA und baut auf den Frameworks Google Web Toolkit und SmartGWT auf.

Für die spätere Implementierung ist eine lokale Entwicklungsumgebung aufzusetzen. Diese hat sich an die im Team eingesetzten Werkzeuge zu orientieren, da dies den Entwicklungsprozess vereinheitlicht.

Es kommen die Entwicklungsumgebung *Eclipse*⁴² und das *Google Plugin for Eclipse*⁴³ zum Einsatz. Der Quellcode wird innerhalb des Versionierungstools *Subversion*⁴⁴ verwaltet und über das Plugin *Subclipse*⁴⁵ importiert und nach erfolgreicher Implementierung zurückgespielt.

Konzept zur Erfüllung von Anforderung „A_02“

Bedienung ist intuitiv

Die nichtfunktionale Anforderung, dass das System „intuitiv“ bedient werden kann, wird innerhalb eines Bedienkonzeptes gelöst (siehe Kapitel 4.3.2, Bedienkonzept). Es ist Be-

⁴² <http://www.eclipse.org/>

⁴³ <http://code.google.com/intl/de-DE/eclipse/>

⁴⁴ <http://subversion.tigris.org/>

⁴⁵ <http://subclipse.tigris.org/>

standteil des Gesamtkonzeptes, das die einzelnen Teilkonzepte aus diesem Kapitel zu einem Ganzen zusammenfügt.

Konzept zur Erfüllung von Anforderung „A_03“

Erweiterbarkeit

Die Anzahl und die Art der Editoren für die Eingabe von Datensatzbestandteilen sind bei verschiedenen Datensatztypen unterschiedlich. Soll der Assistent zu einem späteren Zeitpunkt zum Beispiel für die Integration von Literaturdaten erweitert werden, so muss dieser den Upload von PDF-Dateien ermöglichen und eine Formularmaske zur Eingabe des Literaturnachweises (Autor, Publikationsdatum, ...) bereitstellen. Gemeinsam haben die Integrationsaufgaben für unterschiedliche Datensatztypen, dass sich die Gesamtaufgabe in eine bis beliebig viele Teilaufgaben unterteilen lässt.

Demzufolge ist vor der Initialisierung des Assistenten für die Integration eines Datensatztypen eine Verzweigung einzubauen, die vom Nutzer abfragt, welcher Datensatztyp integriert werden soll. Je nach gewähltem Typ werden die entsprechenden Teilaufgaben geladen und der Nutzer bis zur Erreichung der jeweiligen Gesamtaufgabe geführt.

Konzept zur Erfüllung von Anforderung „A_04“

Erstellung eines gültigen und vollständigen WGEF Datensatzes

Die bereitgestellten Dateien müssen in das *WISDOM Geodata Exchange Format* überführt werden. Voraussetzung dafür ist, dass alle benötigten Bestandteile eingegeben und über das Internet bereitgestellt werden (siehe Anforderungen A_04_01, A_04_02_01, A_04_02_02). Die Verarbeitung übernimmt ein WPS (siehe **Abbildung 4.02**).

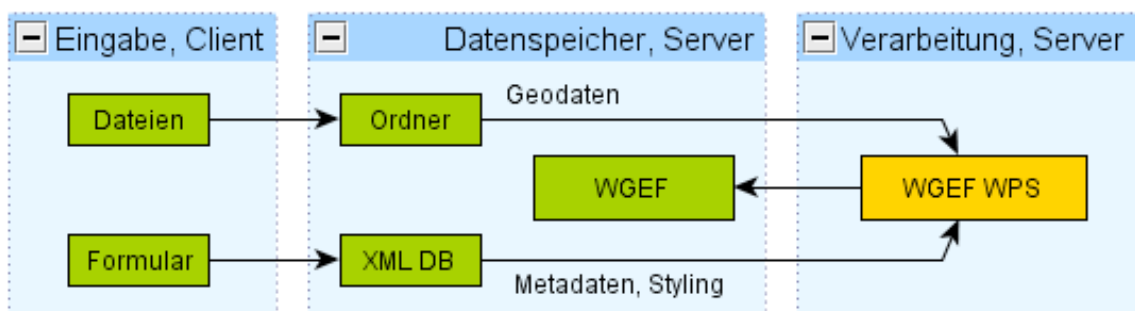


Abbildung 4.02 – WGEF Zusammenstellung durch einen WPS-Dienst

Der WPS-Dienst wird über eine Internetadresse mit Eingabeparametern aufgerufen (siehe **Listing 4.01**). Er lädt die Dateien von den übergebenen Adressen herunter, ordnet sie in die WGEF-Ordnerstruktur ein und erstellt daraus ein ZIP-Archiv. An den Aufrufer wird ein XML-Dokument zurückgegeben, das die Internetadresse des erstellten Archivs enthält.

Listing 4.01 – Beispielaufruf des WPS Dienstes zum Erstellen eines WGEF Archivs

```
01 http://mekongvm5/cgi-bin/wps.py?
02     request=execute&
03     identifier=createwgef&
04     datainputs=[
05         geodata=http://129.247.183.248/geodata;
06         metadata=http://129.247.183.248:8080/metadata.xml;
07         sld=http://129.247.183.248:8080/styling.sld;
08         uuid=CCEA257DF1AE493DB932CEA5094C8C95]&
09     responsedocument=[package:wgef:result=@asreference=true]&
10     service=wps&
11     version=1.0.0
```

Eingaben (*datainputs*):

- Zeile 05: geodata = URL zu Ergebnis aus Anforderung A_04_01
- Zeile 06: metadata = URL zu Ergebnis aus Anforderung A_04_02_01_01
- Zeile 07: sld = URL zu Ergebnis aus Anforderung A_04_02_02_01
- Zeile 08: Universally Unique Identifier (UUID)

Verarbeitung durch den „createWgef“-WPS:

- Download der Geodatendateien mittels des Kommandozeilenprogramms *wgef*
- Kopieren der Eingaben in die erforderlichen Dateistrukturen
- Packen der Ordner und Dateien mittels des Kommandozeilenprogramms *zip*

Ausgaben:

- XML Dokument mit der URL des erstellten WGEF Archivs.

Konzept zur Erfüllung von Anforderung „A_04_01“

Transfer lokaler Dateien zum Server

Geodatendateien werden über den Browser vom lokalen Dateisystem des Nutzers (Client) in einen temporären Ordner auf dem Server übertragen. Der Ordnername auf dem Server muss eindeutig sein, da es sonst bei gleichzeitiger Nutzung durch mehrere Nutzer zur Durchmischung von Daten kommt. Die Eindeutigkeit des Ordnernamens wird durch die Verwendung eines UUID (*Universally Unique Identifier*)⁴⁶ sichergestellt. Für die Implementierung wird recherchiert, ob es Lösungen für den Dateiupload über GWT gibt und ob diese für diese Arbeit angepasst werden können.

⁴⁶ http://de.wikipedia.org/wiki/Universally_Unique_Identifier

Konzept zur Erfüllung von Anforderung „A_04_01“

Eingabe von Darstellungsvorschriften

Darstellungsvorschriften beinhalten Regeln zur Symbolisierung von Geodaten. Die Benutzerfreundlichste Lösung wäre ein grafischer Editor nach dem Prinzip *What You See Is What You Get* (WYSIWYG). Auf Grund der Komplexität eines solchen Editors, ist diese innerhalb dieser Arbeit nicht zu bewerkstelligen. Daher sieht dieses Konzept ein Eingabefeld vor, in das ein Dokument nach dem SLD-Standard⁴⁷ über „Copy & Paste“ eingefügt werden kann. Das SLD-Dokument muss außerhalb des Informationssystems, mit externer GIS-Software, erstellt werden. Der Assistent gibt Hinweise, welche Software dafür verwendet werden kann und verweist auf Hilfetexte.

Konzept zur Erfüllung von Anforderung „A_05“

Start des Integrationsprozesses

Die Datenintegrationssoftware wird unter Angabe der erforderlichen Parameter über einen WPS-Dienst gestartet (siehe Kapitel 3.2.1, Kapselung). Dafür muss die Software die Parameter zusammenstellen und den Dienst über eine Internetadresse aufrufen.

4.3.2 Gesamtkonzept

In diesem Kapitel werden die Teilkonzepte aus Kapitel 4.3.1 in ein Gesamtkonzept überführt. Technische Details werden im technischen Konzept behandelt. Die grafische Oberfläche und die Bedienung des Assistenten werden in einem Bedienkonzept und einem gestalterischen Konzept entworfen.

Die Aufgabe für die Zusammenstellung und Integration eines WGEF-Datensatzes besteht aus einer bekannten Anzahl an Teilaufgaben, die im Folgenden als „Schritte“ (*Steps*) bezeichnet werden. **Tabelle 4.03** gibt einen Überblick über die erforderlichen Schritte für die Zusammenstellung und Integration eines WGEF-Datensatzes in ELVIS.

⁴⁷ <http://www.opengeospatial.org/standards/sld> (abgerufen am 7.8.2010)

Tabelle 4.03 – Schritte für die Zusammenstellung und Integration eines WGEF Datensatzes in ELVIS

Schritt Nr.	Id	Anforderung	Beschreibung
1	Upload	A_04_01	Bereitstellung von Geodaten durch den Nutzer: Dateitransfer vom lokalen Rechner zum Server.
2	Metadaten	A_04_02_01	Eingabe von Metadaten in eine Formularmaske. Erstellen eines XML-Dokuments nach der ISO 19115:2003 Norm. Transfer des erstellten XML-Dokuments zum Server.
3	Styling	A_04_02_02	„Copy & Paste“ eines SLD-Dokuments in ein Eingabefeld. Transfer des XML-Dokuments zum Server.
4	WGEF	A_04	Die Dateien aus Schritt 1, 2 & 3 werden in den WGEF-Standard überführt.
5	Save	A_05	Start der Integrationssoftware

In den Schritten 1, 2 und 3 sind Dateneingaben durch den Nutzer erforderlich. Die Schritte sind geordnet und voneinander abhängig, d.h. ein bestimmter Schritt muss abgeschlossen sein, um mit dem nächsten Schritt fortfahren zu können. Ein Geodatensatz kann z. B. erst in ein WGEF-Archiv überführt werden, wenn die Teilaufgaben *Upload*, *Metadaten* und *Styling* erfolgreich abgeschlossen sind. Des Weiteren kann die Integration erst durchgeführt werden (A_05), wenn ein vollständiges WGEF-Archiv vorliegt (A_04).

Technisches Konzept

Abbildung 4.03 zeigt den Datenfluss des Gesamtprozesses. In Schritt 1, 2 und 3 finden Dateneingaben statt. Die Eingaben werden temporär in Orten auf dem Server gespeichert (Ordner in einem Dateisystem oder in einer Datenbank), auf die mittels Adressen (URL) aus dem Internet zugegriffen werden kann. Formulardaten werden in XML-Dokumente überführt und über HTTP POST in einer XML-Datenbank mit dem Namen *eXist* gespeichert. Neben der REST-Schnittstelle bietet diese Datenbank eine Funktion zum Validieren der eingegebenen XML-Dokumente gegen ein XML-Schema (XSD-Schema). Dies erspart Implementierungsarbeit zum Übertragen und der serverseitigen Überprüfung der Eingaben, da diese Schnittstellen bzw. bereits von der Datenbanksoftware umgesetzt sind.

Ist die Dateneingabe abgeschlossen, so wird in *Schritt 4* aus den Eingaben mittels eines WPS ein WGEF-Archiv erstellt (Anforderung A_04). Das Archiv wird in *Schritt 5* über eine Internetschnittstelle an einen WPS-Dienst übergeben, der das DEP über die Kommandozeile aufruft und die Datensatzbestandteile in den ELVIS-Speichersystemen registriert und speichert (Anforderung A_05, Details in Kapitel 3.2.2 auf Seite 27).

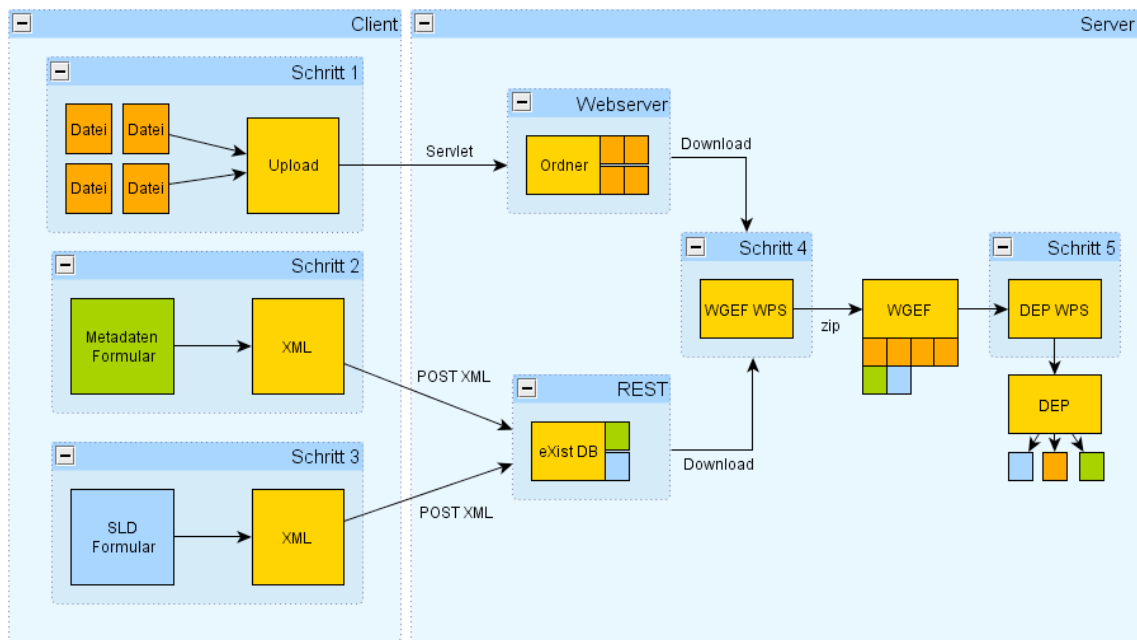


Abbildung 4.03 – Datenfluss

Bedienkonzept

Das Bedienkonzept erweitert das technische Konzept um Benutzerinteraktion und Navigation zwischen den Bearbeitungsschritten. Es muss sich um ein generisches Konzept handeln, das einerseits die Anforderungen aus dem speziellen technischen Konzept für die Erstellung und Integration eines WGEF-Datensatzes abdeckt, andererseits die Erweiterung des Assistenten um neue Datensatztypen berücksichtigt (Anforderung A_03). Des Weiteren liegt der Fokus dieses Konzepts auf der Erfüllung der Anforderung A_02. **Abbildung 4.04** veranschaulicht das Bedienkonzept.

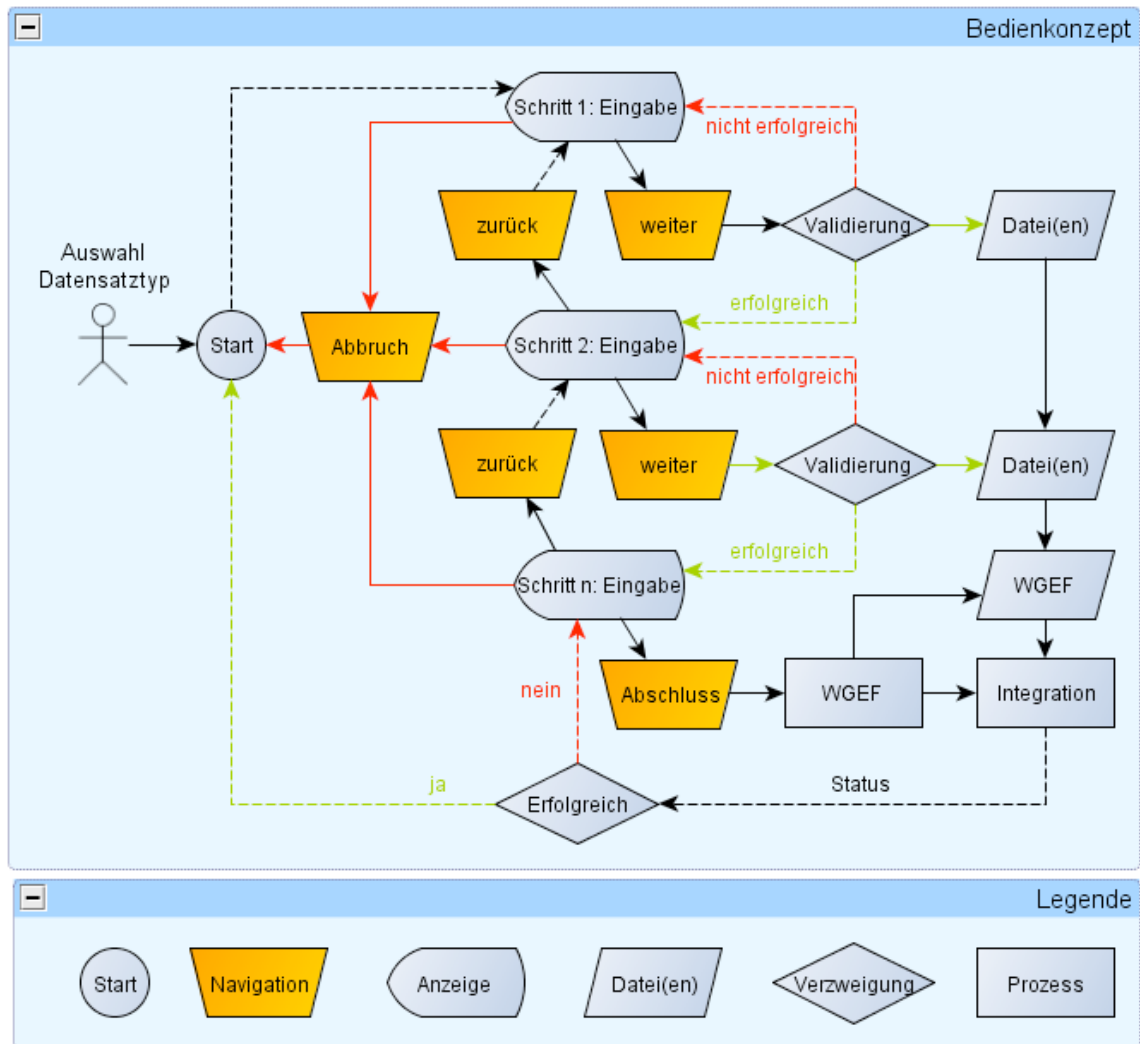


Abbildung 4.04 – Bedienkonzept

Nach dem Aufruf des Assistenten (Auswahl der Registerkarte: *DataEntryPortal*) wählt der Nutzer aus einer Liste verschiedener Datensatztypen denjenigen aus, der in das System importiert werden soll. Daraufhin startet der Assistent (siehe Abbildung: „Start“).

Für den gewählten Datensatztypen werden die entsprechenden Editoren für die Datensatzbestandteile initialisiert („Schritt 1“, „Schritt 2“, ...) Zunächst ist lediglich „Schritt 1“ aktiviert. Dieser wird dem Benutzer angezeigt.

Der Benutzer tätigt die Dateneingabe für „Schritt 1“. Sobald er der Meinung ist, dass er die Eingabe erfolgreich bearbeitet hat, betätigt er eine „Weiter“-Schaltfläche.

Daraufhin wird zunächst die Eingabe clientseitig überprüft (Validierung). Ist die Eingabe vollständig und entspricht dem erwarteten Format, so wird die Eingabe in ein Transportformat überführt und temporär in einer oder mehreren Datei(en) auf dem Server gespeichert. Es findet eine serverseitige Validierung statt. Tritt dabei ein Fehler auf, so wird dem Nutzer ein Hinweis gegeben, welche Eingabe nicht korrekt war. Ist die Validierung jedoch

erfolgreich, so wird der nächste Bearbeitungsschritt freigegeben und angezeigt. Der oben beschriebene Vorgang ist für alle folgenden Schritte einheitlich.

In einem letzten Schritt (siehe Abbildung: „Schritt n“) wird die Gesamtaufgabe abgeschlossen: Durch einen Mausklick auf die *Abschluss*-Schaltfläche wird aus den zuvor eingegebenen Daten ein WGEF-Archiv erstellt und dieses in ELVIS integriert. Der Server gibt Rückmeldung über den Status des Integrationsprozesses. War die Integration erfolgreich, so ist die Gesamtaufgabe erfolgreich abgeschlossen. Dem Nutzer wird daraufhin erneut die Startseite präsentiert, auf der er einen neuen Datensatztypen zur Integration auswählen kann.

Innerhalb jedes Bearbeitungsschrittes kann der Nutzer über eine „Abbrechen“-Schaltfläche die Bearbeitung ohne Datenintegration beenden. Die bisher getätigten Eingaben gehen dabei verloren und temporär gespeicherte Dateien werden gelöscht. Er gelangt dadurch zur Startseite.

Über eine „Zurück“-Schaltfläche kann der Nutzer vorherige Eingaben überprüfen und gegebenenfalls Änderungen vornehmen.

Gestalterisches Konzept

Das gestalterische Konzept schließt die Gesamtkonzeption ab. Innerhalb dieses Unterkapitels wird die Oberfläche des Assistenten in Hinblick auf das Bedienkonzept entworfen.

Layout

Das grafische Oberfläche des DatenEingangsPortals wird als neue Seite in die ELVIS-GUI eingebunden und über die Registerkarte *DataEntryPortal* gestartet (Anforderung: A_01).

Die Anwendung behält durchgehend das Layout aus **Abbildung 4.05** bei. Das Layout besitzt zwei Spalten, in denen drei unterschiedliche Toolboxes angeordnet werden: Eine *Overview*-, eine *DataEntry*- und eine *Control*-Toolbox. Die Aufgaben und Gestaltung der einzelnen Toolboxes werden im Folgenden vorgestellt.

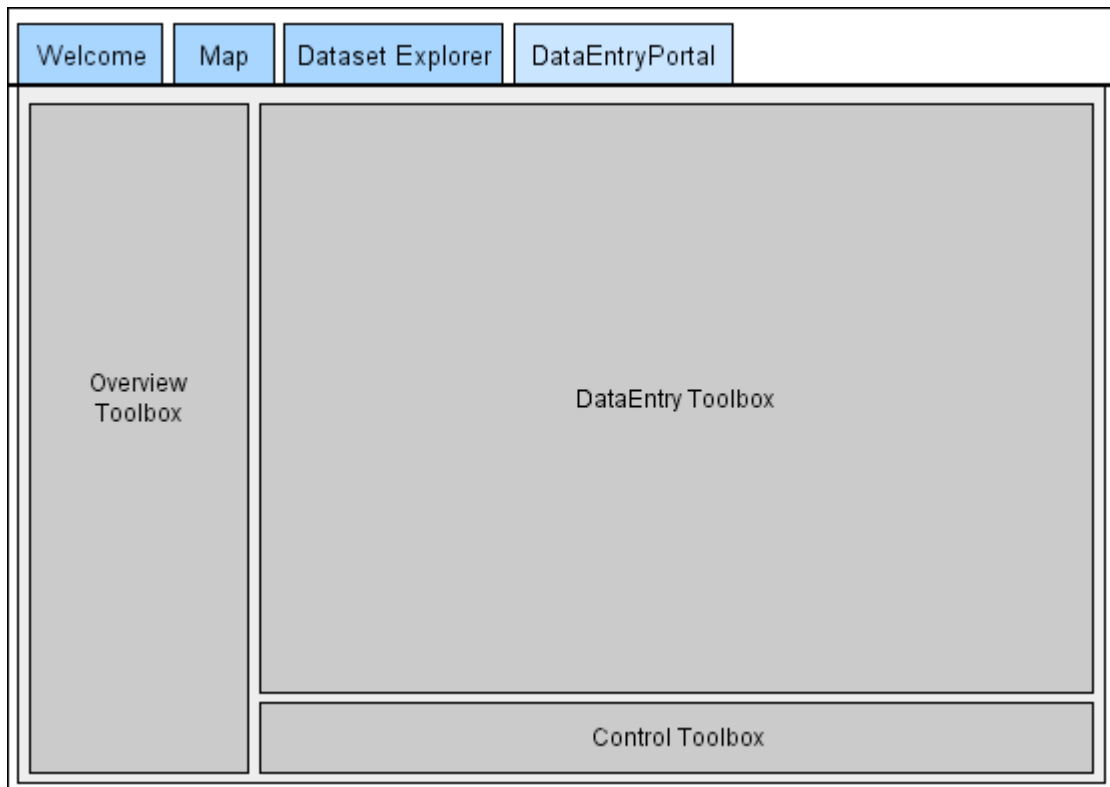


Abbildung 4.05 – Layoutkonzept

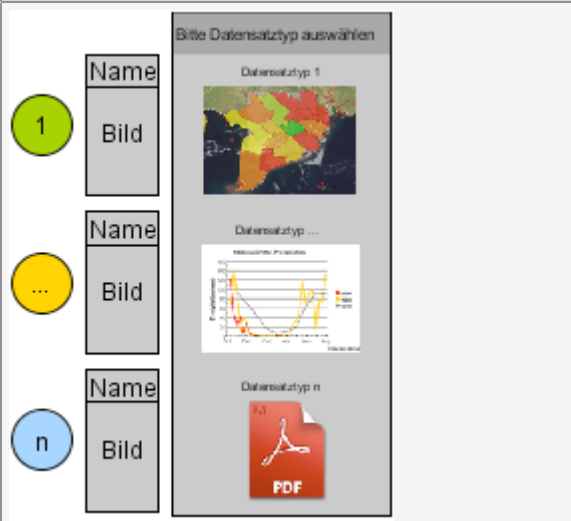
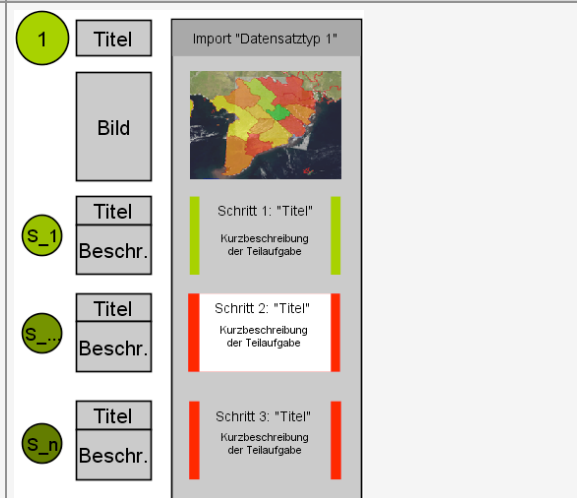
Die Overview-Toolbox nimmt die komplett verfügbare Höhe der linken Spalte ein und besitzt eine feste Breite. Die DataEntry-Toolbox und die Control-Toolbox teilen sich die verfügbare Höhe der rechten Spalte. Die Control-Toolbox besitzt eine festgelegte Höhe. Der Rest der verfügbaren Höhe wird von der DataEntry-Toolbox ausgefüllt, die oberhalb der Control-Toolbox angeordnet ist.

Overview-Toolbox

Die Overview-Toolbox besitzt zwei unterschiedliche Aufgaben. Einerseits dient sie, nach Start des Assistenten, zur Auswahl eines Datensatztypen. Andererseits gibt sie eine Übersicht über die notwendigen Schritte zur Zusammenstellung und Integration eines gewählten Typs.

Beim Start des Assistenten übernimmt die Overview-Toolbox zunächst die Funktion zur Auswahl eines Datensatztypen, der in ELVIS eingespielt werden soll (Bedienkonzept „Start“). Dafür werden mehrere „Kacheln“ bestehend aus einem Namen und einem Bild angezeigt. Die Kacheln repräsentieren verschiedene Datensatztypen. Durch Auswahl (einfacher Klick mit der linken Maustaste) wird die Eingabe für den gewählten Datensatztyp gestartet (**Tabelle 4.04 – Auswahl Datensatztyp**).

Tabelle 4.04 – Overview Toolbox, Auswahl Datensatztyp und Dateneingabe

Auswahl Datensatztyp	Übersicht Dateneingabe
 <ul style="list-style-type: none"> • Es können 1..n unterschiedliche Datensatztypen ausgewählt werden. • Ein Datensatztyp wird von einem Namen und einem Bild repräsentiert. Zusammen bilden diese beiden Daten eine „Kachel“. • Reicht der vertikale Platz nicht für die Darstellung aller Datensatztypen aus, so wird am rechten Rand eine Scrollleiste angezeigt. 	 <ul style="list-style-type: none"> • Wurde ein Datensatztyp ausgewählt, so werden der Titel und das Bild des gewählten Typs angezeigt. (1) • Für die Eingabe und Integration eines Datensatztypen sind 1..n Schritte (S_1 .. S_n) notwendig. • Jeder Schritt besitzt einen Titel und eine Kurzbeschreibung • Es gibt vier Bearbeitungszustände: <p> <input type="button" value="inaktiv"/> <input type="button" value="aktiv"/> <input type="button" value="unvollständig"/> <input type="button" value="vollständig"/> </p>

Nach Auswahl eines Datensatztypen wird eine Übersicht über die Gesamtaufgabe angezeigt, die eine Auflistung der einzelnen Teilaufgaben beinhaltet (**Tabelle 4.04, Übersicht Dateneingabe**). Zusätzlich wird der Bearbeitungsstatus der Teilaufgaben angegeben. Die Übersicht gibt dem Nutzer folgende Informationen:

- Benennung der Gesamtaufgabe („Import Datensatztyp“)
- Beschreibung der einzelnen Schritte, die für die Datenzusammenstellung und Integration notwendig sind (Titel, Kurzbeschreibung)
- Status (aktiv/inaktiv/vollständig/unvollständig)

Aus **Tabelle 4.04** ergeben sich folgende Daten, die für die Gestaltung der Overview-Toolbox benötigt werden:

- Gesamtheit aller Datensatztypen, die in ELVIS integriert werden können
- Für jeden Datensatztyp:

- Name
- Bild
- Für jeden Bearbeitungsschritt:
 - Titel
 - Kurzbeschreibung

DataEntry-Toolbox

Die Dateneingabe, Zusammenstellung und Integration findet innerhalb der DataEntry-Toolbox statt. Nach Auswahl eines Datensatztypen werden alle notwendigen Schritte für die Dateneingabe und Integration geladen. Jeder Schritt wird durch einen Tab repräsentiert, die unter anderem Formularemasken für die Dateneingabe bereitstellen.

Beim Start wird dem Nutzer zunächst eine Einführung in die Benutzung des Assistenten gegeben (siehe **Abbildung 4.06**).

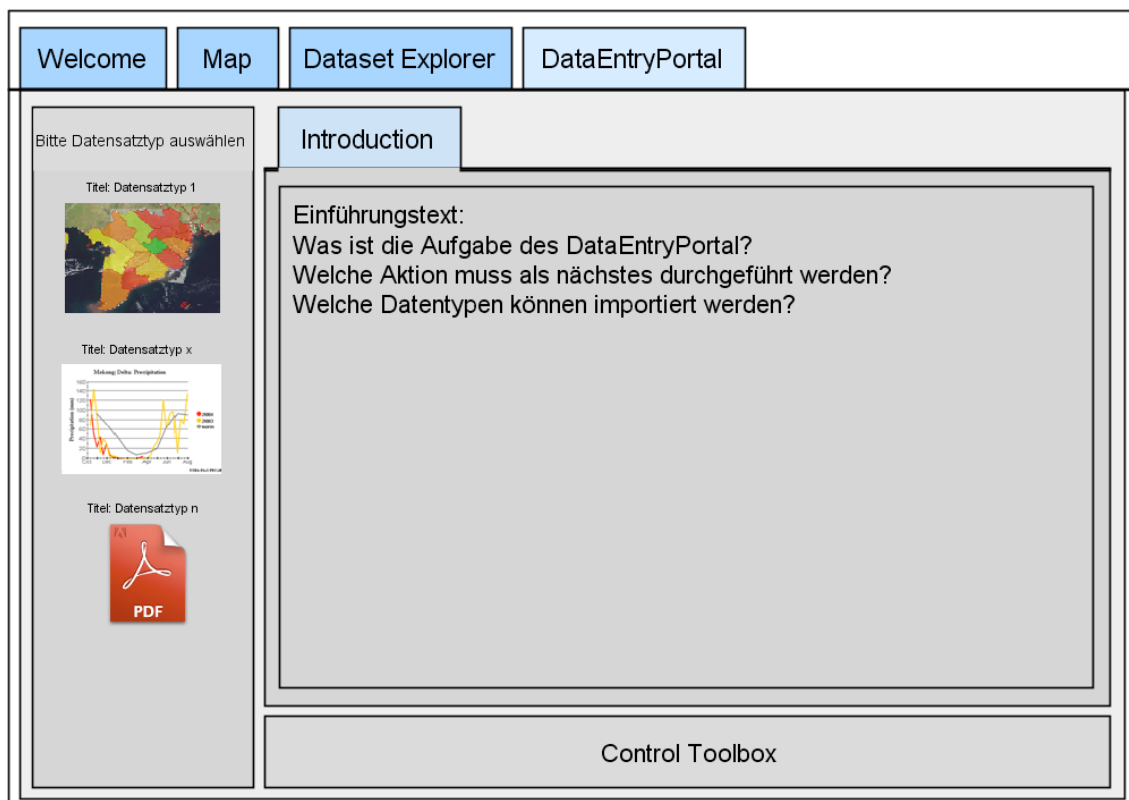


Abbildung 4.06 – Startbildschirm

Innerhalb der DataEntry-Toolbox wird eine Unterseite mit Informationen geladen, die beschreibt, wozu das DEP dient, welche unterschiedlichen Datensatztypen integriert werden können und welche Aktion der Nutzer als nächstes durchführen muss, um fortzufahren. Er wird aufgefordert auf in der Overview-Toolbox einen Datensatztypen auszuwählen, den er in ELVIS importieren will.

Hat der Nutzer einen Datensatztypen für die Integration ausgewählt, so wird der Assistent initialisiert. Es werden innerhalb der DataEntry-Toolbox die Editoren für die einzelnen Schritte geladen. Die Editoren werden als Tabs eingefügt. Die Kartenreiter sind in Bearbeitungsreihenfolge und erhalten als Titel den Identifikator der Teilaufgabe (**Tabelle 4.03**), die darin bearbeitet wird. **Abbildung 4.07** veranschaulicht dies.

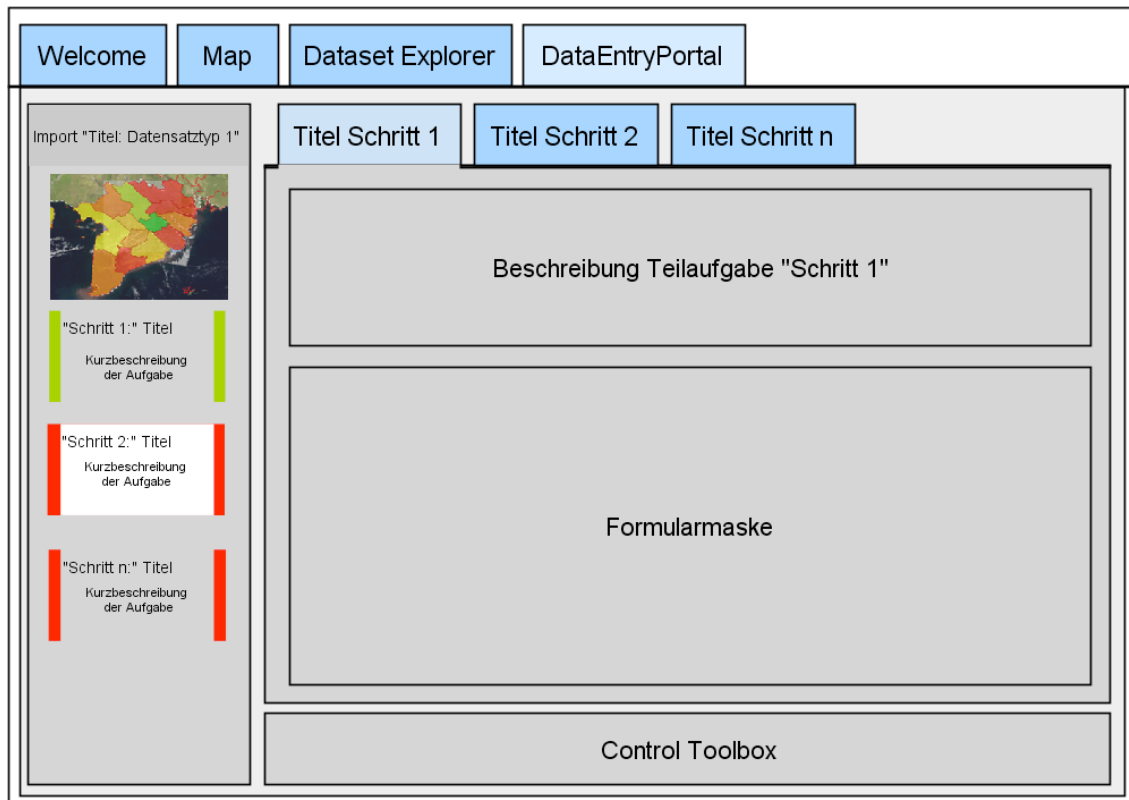


Abbildung 4.07 – Initialisierung der Editoren für Datensatzbestandteile

Eine Teilaufgabe besitzt einen Hilfetext, der beschreibt, was das Ziel der Teilaufgabe ist und was vom Nutzer für die vollständige Bearbeitung verlangt wird. Der Hilfetext wird aus einer externen HTML-Datei geladen. Für die Internationalisierung des Assistenten sind die HTML-Dateien in die jeweilige Sprache zu übersetzen.

Control-Toolbox

Die Control-Toolbox dient der Navigation zwischen einzelnen Bearbeitungsschritten (siehe **Abbildung 4.08**). Der Nutzer kann zum nächsten oder vorherigen Schritt wechseln („Weiter“ und „Zurück“) oder die Bearbeitung abbrechen („Abbrechen“).



Abbildung 4.08 – Schaltflächen für die Navigation

Im letzten Bearbeitungsschritt wird die Gesamtaufgabe abgeschlossen. Die Beschriftung des „Weiter“-Schaltfläche ändert sich nun in „>> Speichern <<“ (siehe **Abbildung 4.09**).



Abbildung 4.09 –Navigationsschaltflächen zum Abschließen der Aufgabe

Die „Weiter“-Schaltfläche ist der zentrale Ort für die Navigation innerhalb des Assistenten. Sie veranlasst bei einfachem Mausklick auf die Schaltfläche die Validierung und die Speicherung der Eingaben der aktuell gewählten Registerkarte und ruft danach den nächsten Bearbeitungsschritt auf. Eine separate „Speichern“-Schaltfläche wird aus diesem Grund nicht benötigt. Dadurch lässt sich der Assistent intuitiv bedienen, weil der Nutzer für die Navigation zwischen den Schritten durchgängig die gleiche Schaltfläche betätigt, die an einem festen Ort platziert wird. Schaltflächen innerhalb der Editoren werden nicht benötigt.

4.4 Implementierung

Bemerkung: Dieses Kapitel kann aus Platzgründen nicht alle Details der Implementierung umfassen. Es wurde eine Auswahl getroffen, die einen Überblick über die Implementierung des entwickelten Konzepts gibt.

4.4.1 Grafische Oberfläche

Für die Umsetzung des - im gestalterischen Konzept entwickelten - Layouts wurde die ELVIS-GUI-Konfigurationsdatei (*/conf/xml/LayoutElvis.xml*) erweitert. Es wurde ein neues `<page>` Element hinzugefügt, das die drei Toolboxes *Overview-Toolbox*, *DataEntry-Toolbox* und *Control-Toolbox* enthält und ihnen unter anderem die jeweilige Position, deren Höhe und die Breite zuweist (siehe Anhang 7.3 auf Seite 98). Die Toolboxes werden über das `<Classname>` Element mit der jeweilige JAVA-Klasse verknüpft, die den Quellcode für die jeweiligen Funktionalitäten beinhalten (**Abbildung 4.10**). Der Begriff *Toolbox* wird dabei mit *TB* abgekürzt. Toolboxes sind nach der MVC-Architektur aufgebaut (siehe Kapitel 2.4.1 auf Seite 22). Der View wird im Folgenden als Toolbox (z. B. *DataEntry-Toolbox* oder *DataEntryTB*) bezeichnet, an die Bezeichnung der Toolbox wird der Begriff Controller (z. B. *DataEntry-Controller*) angehängt, wenn die Controller-Klasse des Views gemeint ist. Das Modell erhält den Anhang *DataSource* (z. B. *DataEntry-DataSource*).

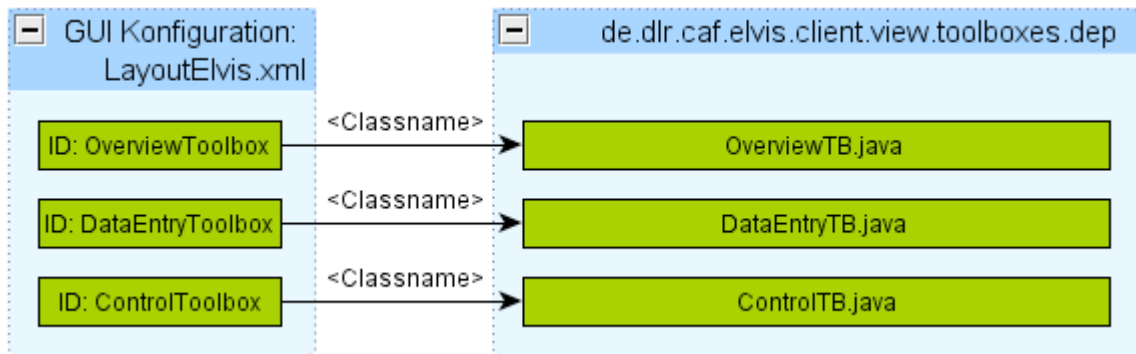


Abbildung 4.10 – Verknüpfung der JAVA-Klassen mit den Toolboxes in der GUI Konfigurationsdatei

Toolboxes

Bei den zugewiesenen Klassen handelt es sich um den *View* der Toolboxes. Beim Start des Assistenten werden die Klassen instanziiert und deren *init()* Methode aufgerufen. Die Methode instanziiert und registriert den zugehörigen Controller und das Modell (*DataSources*) und zeichnet den Startbildschirm der jeweiligen Toolboxes.

Abbildung 4.11 zeigt beispielhaft, den Mechanismus zum Laden der „Einführungs“-Seite in der DataEntry-Toolbox (siehe **Abbildung 4.06** auf Seite 48). Dafür wird innerhalb der Funktion *init()* beim Start die Funktion *updateTabs()* mit dem Parameter „introduction“ aufgerufen, die den Inhalt der Einführungsseite zusammensetzt und zeichnet. In Kapitel 4.4.2 auf Seite 55 wird die Funktion näher erläutert.

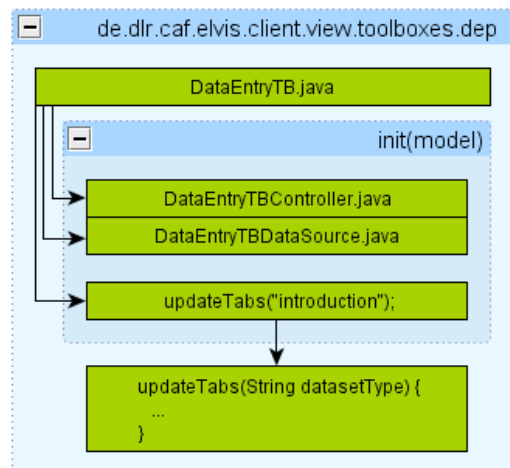



Abbildung 4.11 – Instanziierung der Toolboxes

DataSource

DataSources entsprechen dem Modell der *MVC-Architektur* (siehe Kapitel 2.4.1, Seite 22). Innerhalb einer *OverviewTBDataSource* werden alle benötigten Daten gespeichert, die für die Darstellung der Informationen zu verschiedenen Datensatztypen und die Übersicht über die Integrationsschritte notwendig sind (siehe **Tabelle 4.05**, abgeleitet aus **Tabelle 4.04** auf **Seite 47**).

Tabelle 4.05 – Zu speichernde Daten für einen Datensatztyp

ID: „geodata“	Bild: geodata.png
Name: „Raster/Vektor“ Bild: „geodata.png“ Schritte: “Upload”, “Transfer files from local...” “Metadata”, “Describe data...” “Styling”, “Style your data...” “Save”, “Start integration process...”	

Diese Daten werden mit folgenden Datenstrukturen innerhalb der *OverviewTBDataSource* modelliert (**Listing 4.02**)

Listing 4.02 – Datenmodellierung innerhalb der OverviewTBDataSource

```

01 Datensatztypen (Array)
02 |_ Datensatztyp1 (Objekt vom Typ OverviewTBDataSource)
03     |_ ID (String)
04     |_ Name (String)
05     |_ Bild (String)
06     |_ Schritte (LinkedHashMap)
07         |_Titel (String), Kurzbeschreibung (String)
08         |_Titel (String), Kurzbeschreibung (String)
09 |_ Datensatztyp2 (Objekt vom Typ OverviewTBDataSource)
10     |_ ID
11     |_ ...
12 |_ ...

```

Die Umsetzung in JAVA wird in der Klasse *OverviewTBDataSource.java* vorgenommen (**Listing 4.03**).

Listing 4.03 – Umsetzung der Datenmodellierung der OverviewTBDataSource in JAVA

```

01 public OverviewTBDataSource (
02     String id,
03     String name,
04     String picture,
05
06     LinkedHashMap<String, String> steps) {
07     setId(id);
08     setName(name);
08     setPicture(picture);
09     setSteps(steps);
10 }
11
12 public static OverviewTBDataSource[] getDatasetTypes() {

```

```

13   LinkedHashMap<String, String> geodataIntegrationSteps
14       = new LinkedHashMap<String, String>();
15   geodataIntegrationSteps.put("Upload", "Transfer geodata ...");
16   geodataIntegrationSteps.put("Metadata", "Please describe...");
17   geodataIntegrationSteps.put("Styling", "Control symboliza...");
18   geodataIntegrationSteps.put("Save", "Start the integration");
19
20   OverviewTBDataSource[] datasetTypes
21       = new OverviewTBDataSource [] {
22       new OverviewTBDataSource (
23           "geodata",
24           "Raster/Vector",
25           geodataIntegrationSteps
26       ),
27       new OverviewTBDataSource (
28           ...
29       ),
30       ...
31   };
32   return datasetTypes;
33 }

```

Aktionen/Ereignisse

Toolboxes kommunizieren über Ereignisse (*Events*) miteinander. In der ELVIS-GUI-Konfigurationsdatei werden einer Toolbox andere Toolboxes zugeordnet, die auf Ereignisse reagieren (siehe Anhang 7.3 – Element <EventHandler>). Alle drei Toolboxes konsumieren Ereignisse von sich selbst und der anderen beiden Toolboxes.

Der Benutzer führt Aktionen aus (zum Beispiel ein Mausklick auf eine Schaltfläche), die Ereignisse eines bestimmten Typs hervorrufen. Des Weiteren können Ereignisse auch bei Zustandsänderungen innerhalb des Assistenten programmtechnisch aufgerufen werden. Dadurch werden Abläufe kaskadiert und gekapselt. Die *Views* der registrierten Toolboxes werden durch Aufruf der Funktion `handleEvent(GWTEvent event)` von dem Ereignis benachrichtigt. Sie können - bedingt durch den Ereignistyp - auf die Aktion reagieren. Neue Ereignistypen wurden durch Ableitung von der Klasse `GwtEvent` entworfen.

Tabelle 4.06 gibt einen Überblick über die umgesetzten Ereignistypen, um die Kommunikation zwischen den drei Toolboxes zu realisieren.

Tabelle 4.06 – Ereignistypen für die Kommunikation zwischen Toolboxes

ID	Ereignistyp	Beschreibung
1	StartEvent	Dieses Ereignis wird nach der Auswahl eines Datensatztyps generiert. Der Assistent lädt daraufhin, die für die Eingabe, Zusammenstellung und Integration des gewählten Typs, notwendigen Schritte innerhalb der DataEntry-Toolbox. Innerhalb der Overview-Toolbox wird eine Übersicht über die Teilaufgaben angezeigt.
2	NextEvent	Der Benutzer nimmt an, dass der jetzige Bearbeitungsschritt vollständig ist und erzeugt ein NextEvent durch Betätigen der „Weiter“-Schaltfläche in der Control-Toolbox. Die DataEntry-Toolbox empfängt das Ereignis und leitet die Speichieranfrage an den Controller des aktuell gewählten Tabs weiter.
3	CompleteEvent	Der Tab-Controller hat die Speichieranfrage und die Validierung bearbeitet und übergibt den Status an die Funktion saveConfirm() des DataEntry-Controllers. Es wird ein CompleteEvent erzeugt, das als Attribut den Identifikator und den Status der Ereignisquelle besitzt. War die Speicherung und Validierung erfolgreich, so wird der nächste Bearbeitungsschritt freigegeben. Das Ereignis setzt den Status des bearbeiteten Schrittes innerhalb der Overview-Toolbox auf vollständig oder unvollständig.
4	TabSelectedEvent	Bis auf den ersten Schritt sind bei der Initialisierung zunächst alle anderen Schritte deaktiviert. Wurde ein Schritt zur Bearbeitung freigegeben, so wird ein <i>TabSelectedEvent</i> erzeugt. Dieses Ereignis bewirkt, dass innerhalb der Overview-Toolbox der momentan bearbeitete Schritt hervorgehoben wird.
5	PreviousEvent	Anzeige des vorherigen Bearbeitungsschrittes (falls ein vorheriger Bearbeitungsschritt existiert).
6	CancelRequestEvent	Anfrage zum Abbruch der Bearbeitung durch betätigen der „Abbrechen“-Schaltfläche innerhalb der Control-Toolbox. Es wird eine Rückfrage in Form eines Dialogfelds angezeigt, in dem der Benutzer entscheiden kann, ob er tatsächlich abbrechen, oder ob er mit der Bearbeitung fortfahren möchte. Bestätigt der Nutzer die Rückfrage mit einem „Ja“, so wird ein CancelEvent erzeugt. Bestätigt er mit „Nein“ wird das Dialogfeld geschlossen und keine weitere Aktion ausgeführt.
7	CancelEvent	Abbruch ohne Rückfrage.
8	RecordClickEvent	Das Ereignis wird bei Auswahl eines Datensatztyps in der Overview-Toolbox erzeugt.

4.4.2 Funktionen

Innerhalb der Toolboxen werden verschiedene Funktionen implementiert. Im Folgenden werden die wichtigsten Funktionen erläutert.

Initialisierung der Bearbeitungsschritte

Nach der Auswahl eines Datensatztypen werden der DataEntry-Toolbox verschiedene Registerkarten (Tabs) hinzugefügt, die der Eingabe und Integration von Daten in ELVIS dienen (**Abbildung 4.07** auf Seite 49). Die Anzahl und Art der Tabs ist abhängig vom gewählten Datensatztypen. Tabs besitzen einen View der im Folgenden als Tab (z.B. Metadata-Tab oder MetadataTab) bezeichnet wird und einen Controller, der als Tab-Controller (z.B. MetadataTab-Controller) bezeichnet wird. **Abbildung 4.12** stellt schematisch den Mechanismus dar, um die entsprechenden Tabs in der DataEntry-Toolbox anzuzeigen.

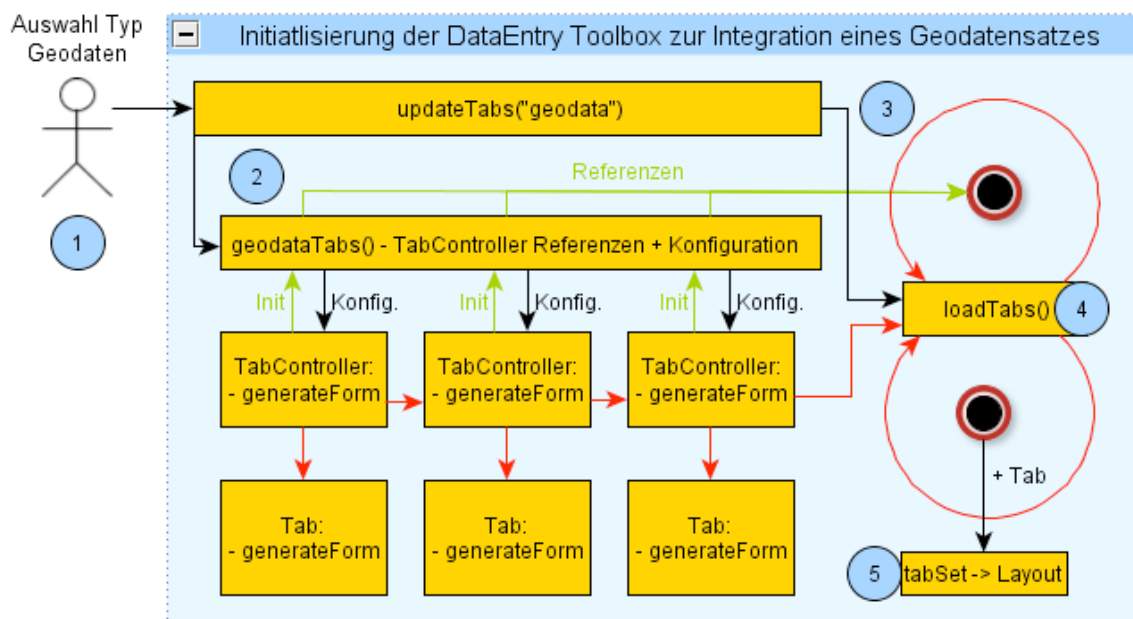


Abbildung 4.12 – Initialisierung der Bearbeitungstabs am Beispiel eines Geodatensatzes

Die Tabs werden nach der Auswahl eines Datensatztypen initialisiert. Dafür wird die Funktion `updateTabs(...)` mit dem Identifikator des gewählten Datensatztyps aufgerufen (siehe **Abbildung 4.13**).

Die Funktion `updateTabs(...)` bestimmt durch den übergebenen Identifikator `datasetType`, welche Funktion für die Initialisierung der Tabs zuständig ist. Im Falle eines Geodatensatzes (`datasetType = „geodata“`) ist dies die Funktion `geodataTabs()`.

Die Funktion instanziiert die entsprechenden Tab Controller und konfiguriert diese. Im Falle eines Geodatensatzes sind dies die Controller: `UploadTabController`, `MetadataTab-`

Controller, *StylingTabController* und *SaveTabController*. Durch die Konfiguration wird eine Wiederverwendung der Tabs ermöglicht. Im Falle des Dateiuploads kann zum Beispiel der *Upload*-Tab konfiguriert werden, welche Dateinamenserweiterungen hochgeladen werden müssen, um den Bearbeitungsschritt erfolgreich abzuschließen. Bei Raster-Geodaten ist dies zum Beispiel eine Datei mit der Endung **.tif*, bei Literaturdaten eine Datei mit der Endung **.pdf*. Dadurch kann der gleiche *Upload*-Tab sowohl für die Integration von Geodaten als auch für die Integration von zum Beispiel von Literaturdatensätzen wiederverwendet werden.

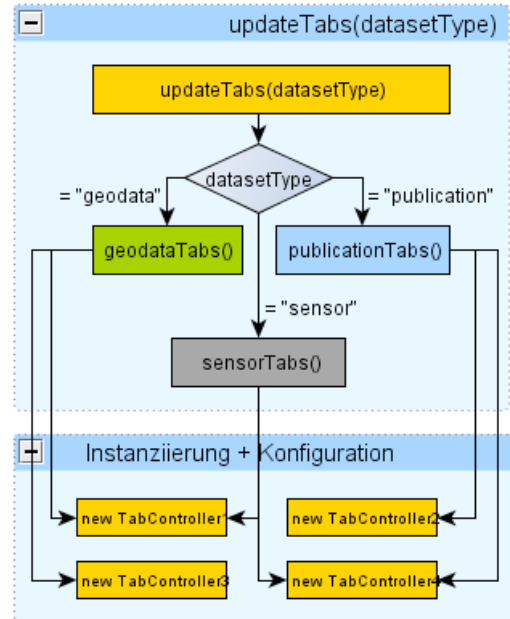


Abbildung 4.13 – Funktion updateTabs(type)

Nach der erfolgreichen Instanziierung und Konfiguration aller Tab Controller für den gewählten Datensatztyp werden die Referenzen als geordnete Liste (LinkedHashMap<Identifikator><Controller>)) an die Funktion *loadTabs(...)* übergeben.

Jeder Tab Controller besitzt per Vertrag (*Interface*) die Funktion *generateForm()*. Die Funktion *loadTabs()* entfernt zunächst alle bereits geladenen Tabs und ruft daraufhin in einer Schleife für jeden Tab Controller die Funktion *generateForm()* auf, die den Inhalt des Tabs generiert und in einem Layout anordnet und zurückgibt. Das Layout wird zu einem Tab hinzugefügt und in ein so genanntes *TabSet* eingehängt. Dieses *TabSet* wird nach Durchlauf durch alle Tab-Controller-Referenzen innerhalb der *DataEntry-Toolbox* angezeigt.

Damit ist die Initialisierung des Assistenten zur Bearbeitung des gewählten Datensatztypen abgeschlossen und der Nutzer kann mit der Eingabe für den ersten Schritt beginnen.

Navigation

Zentraler Ort der Navigation ist die „Weiter“-Schaltfläche innerhalb der Control-Toolbox. Sie stößt die Validierung und Speicherung der aktuellen Eingabe an. **Abbildung 4.14** zeigt die Funktionsverkettung als Sequenzdiagramm.

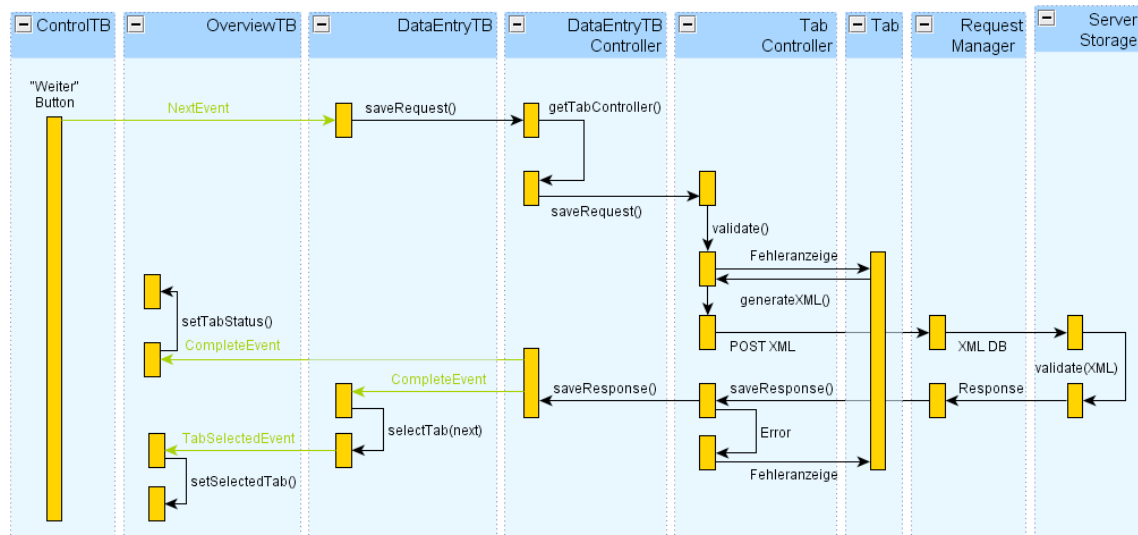


Abbildung 4.14 – Sequenzdiagramm für den Aufruf des nächsten Bearbeitungsschritt

Nach dem Betätigen der „Weiter“-Schaltfläche in der ControlTB wird die DataEntryTB durch ein *NextEvent* über den Benutzerwunsch benachrichtigt, mit dem nächsten Schritt fortfahren zu wollen. Die DataEntryTB verknüpft den Ereignistyp *NextEvent* mit der Funktion *saveRequest()* und ruft diese Funktion im *DataEntry-Controller* auf. Der Controller bestimmt über die Funktion *getTabController()* den Controller des aktuell ausgewählten Tabs (*Tab Controller*) und leitet die Anfrage an dessen *saveRequest()* Funktion weiter. Die Funktion veranlasst zunächst eine clientseitige Validierung der Eingaben (*validate()*). Ist die Validierung erfolgreich, so wird die Eingabe in ein Transportformat XML überführt (Funktion *generateXML()*) und über den *Request Manager* zum Server gesendet (HTTP POST). XML Dokumente werden innerhalb einer XML Datenbank verwaltet. Die Datenbank validiert bei Eingabe das XML-Dokument gegen ein XML Schema und speichert es. Es wird eine Antwort (*Response*) zurückgegeben, ob die Validierung und Speicherung erfolgreich war. Die Antwort wird von der Funktion *saveResponse()* des Tab-Controllers verarbeitet. Enthält die Antwort eine Fehlermeldung, so wird innerhalb des Tabs der Fehler angezeigt. Die Antwort wird an den *DataEntry-Controller* weitergeleitet, der daraus ein *CompleteEvent* generiert. Das *CompleteEvent* benachrichtigt die DataEntry-Toolbox und die OverviewTB. Hat das *CompleteEvent* als Eigenschaft den Statuswert „true“, so war die Speicherung und Validierung erfolgreich und der nächste Bearbeitungsschritt wird in der

DataEntryTB für die Bearbeitung freigegeben und angezeigt. Ist der Statuswert des CompleteEvent „true“, so wird der Schritt als „vollständig“, d.h. mit einer grünen Farbe hinterlegt. Ist der Statuswert des CompleteEvent „false“, so wird der Schritt als „unvollständig“, d.h. mit der Farbe Rot hinterlegt. Außerdem wird durch das erzeugte *TabSelectedEvent* in der OverviewTB angezeigt, dass der nächste Schritt bearbeitet werden kann.

Im Folgenden wird die Implementierung der Funktionskette anhand von Auszügen aus dem Quellcode verdeutlicht. **Listing 4.04** zeigt den Quellcode der „Weiter“-Schaltfläche in der *Control Toolbox (nextButton)*. Zu dieser wird ein *ClickHandler* hinzugefügt (Zeile 04), der bei Betätigung der Schaltfläche ein *NextEvent* (Zeile 06) erzeugt und die anderen Toolboxes davon benachrichtigt (Zeile 07).

Listing 4.04 – „Weiter“-Schaltfläche in der Control Toolbox, NextEvent Ereignis

```

01 nextButton = new IButton("Next");
02 nextButton.setHeight100();
03 nextButton.setWidth("33%");
04 nextButton.addClickHandler(new ClickHandler() {
05     public void onClick(ClickEvent event) {
06         NextEvent nextStepEvent = new NextEvent();
07         controller.notifyHandlers(handlerIDs, nextStepEvent);
08     }
09 });

```

Das Ereignis wird vom *View* der Toolboxes über die Funktion *handleEvent(GwtEvent evt)* entgegengenommen und verarbeitet (**Listing 4.05**). Über *instanceof* wird nach der Art des Ereignisses gefiltert (Zeile 02). Die *DataEntry-Toolbox* informiert den *DataEntry-Controller* über den Speicherwunsch, indem sie die Funktion *saveRequest()* aufruft.

Listing 4.05 – Verarbeitung des NextEvent-Ereignisses durch die DataEntry Toolbox

```

01 public void handleEvent(GwtEvent evt) {
02     if (evt instanceof NextEvent) {
03         controller.saveRequest();
04     }
05     ...
06 }

```

Listing 4.06 zeigt die Verarbeitung durch den DataEntryTB-Controller. Dieser ermittelt den Controller des aktuell gewählten Tabs (Zeile 03) und leitet die Speicheranfrage an diesen weiter (Zeile 04). Per Vertrag (*Interface*) besitzt jeder Tab-Controller die Funktion *saveRequest()*.

Listing 4.06 – Veranlassen der Speicherung der Eingaben durch den DataEntry Controller

```
01 @Override
02 public void saveRequest() {
03     Controller controller = getTabController(selectedTabTitle);
04     controller.saveRequest();
05 }
```

Ist zum Beispiel aktuell der Tab “Metadata” ausgewählt, so wird die Speicheranfrage an den *MetadataTab-Controller* weitergeleitet. **Listing 4.07** zeigt die Verarbeitung der Speicheranfrage durch die Funktion *saveRequest()* des *MetadataTab-Controllers*.

Der Controller führt zunächst die client-seitige Validierung durch (Zeile 03 und 04). Ist diese erfolgreich (Rückgabewert „true“), so werden die Eingaben in das XML Format überführt (Zeile 07, *generateXML()*), eine Anfrage erstellt (Zeile 05+06) und diese an den Server weiterleitet (Zeile 07). Ist die Validierung nicht erfolgreich gewesen, so wird der Benutzer über eine Nachricht davon informiert (Zeile 09).

Listing 4.07 – Validierung und Speicherung innerhalb des MetadataTabController

```
01 @Override
02 public void saveRequest() {
03     ValuesManager valuesManager = getValuesManager();
04     if(valuesManager.validate()) {
05         PutMetadataRequest request = new PutMetadataRequest(this,
06                                                         consumerIDs, handlerIDs);
07         model.sendRequest(generateXml(), request);
08     } else {
09         SC.say("Please complete the form");
10     }
11 }
```

Der Server generiert eine Antwort und ruft die Funktion *saveConfirm()* des *MetadataTab Controller* auf (**Listing 4.08**). Die Funktion *saveConfirm()* wird per Vertrag (*Interface*) von jedem Tab Controller implementiert. Der TabController reicht die Antwort an den DataEntry-Controller weiter.

Listing 4.08 – Weiterleitung der Antwort an den DataEntry-Controller

```
01 @Override
02 public void saveConfirm(boolean status, String errMsg) {
03     controller.saveConfirm(status, errMsg);
04 }
```

Der *DataEntry-Controller* verarbeitet die Antwort innerhalb der Funktion *saveConfirm()* (**Listing 4.09**). Es wird ein *CompleteEvent* erzeugt (**Listing 4.10**, Zeile 04) und an die anderen Toolboxes verteilt (**Listing 4.10**, Zeile 05).

Listing 4.09 – Verarbeitung der Antwort durch den DataEntry-Controller

```
01 @Override
02 public void saveConfirm(boolean status, String response) {
03     tabCompleteEvent(status);
04     if(!response.equals("")) {
05         SC.say(response);
06     }
07 }
```

Listing 4.10 – Generieren eines CompleteEvent: Funktion tabCompleteEvent

```
01 public void tabCompleteEvent(boolean status) {
02     CompleteEvent completeEvent = new CompleteEvent(
03         SelectedTabTitle(),
04         status);
05     notifyHandlers(handlerIDs, completeEvent);
06 }
```

Das *CompleteEvent* wird nun von der *DataEntry-Toolbox* verarbeitet (**Listing 4.11**). Besitzt der übergebene Status den Wert „true“, so wird der nächste Bearbeitungsschritt freigegeben.

Listing 4.11 – Reaktion auf das CompleteEvent durch DataEntryTB.java

```
01 public void tabCompleteEvent(boolean status) {
02     CompleteEvent completeEvent = new CompleteEvent(
03         SelectedTabTitle(),
04         status);
05     notifyHandlers(handlerIDs, completeEvent);
06 }
```

Innerhalb der *Overview-Toolbox* bewirkt das *CompleteEvent*, dass der Status für den Bearbeitungsschritt als *complete* oder *notComplete* angezeigt wird (**Listing 4.12** und **Listing 4.13**).

Listing 4.12 – Reaktion auf das „CompleteEvent“ durch die OverviewTB

```
01 else if(evt instanceof CompleteEvent) {
02     CompleteEvent event = (CompleteEvent) evt;
03     String tabName = event.getTabName();
04     boolean tabStatus = event.getTabStatus();
05     controller.tabCompleteEvent(tabName, tabStatus);
06 }
```

Listing 4.13 – Setzen des Status des Bearbeitungsschrittes in der OverviewTB

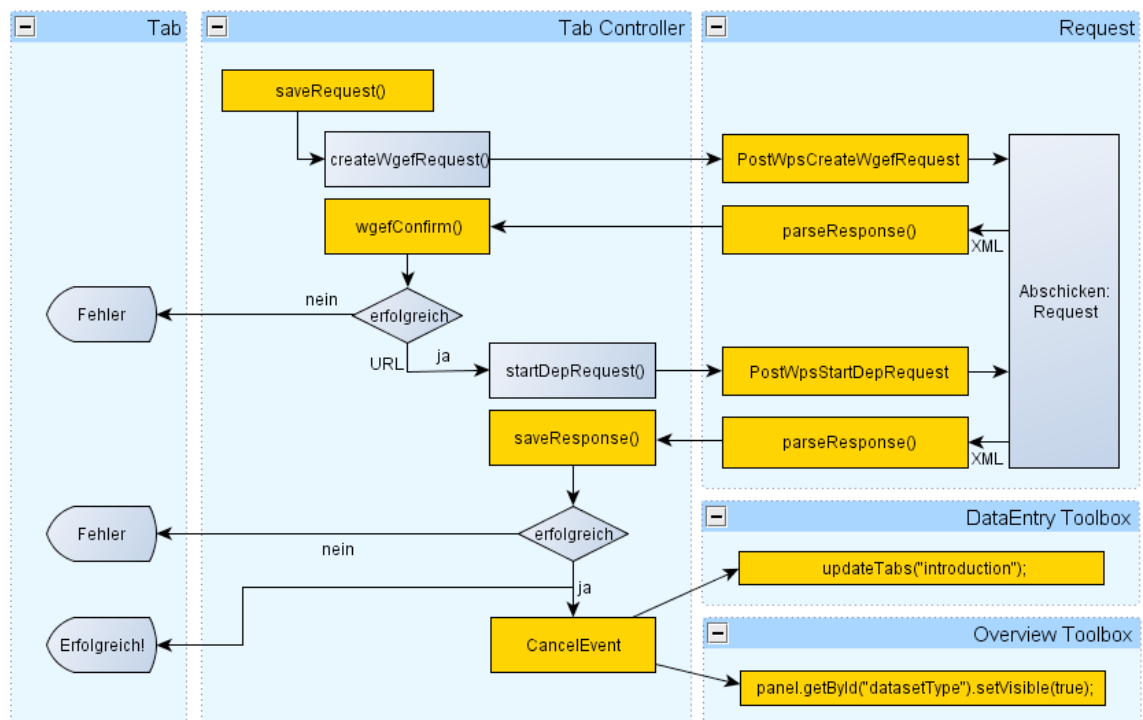
```

01 public void tabCompleteEvent(String completedTab, boolean status)
02 {
03     String stepStatus = "status"+completedTab.toLowerCase();
04     if(status) {
05         Layout.getById(stepStatus).setStyleName("complete");
06     } else {
07         Layout.getById(stepStatus).setStyleName("notComplete");
08     }
09 }

```

Erstellen eines WGEF Archivs und Aufruf der Integrationssoftware

Die Funktionalitäten für die Erstellung eines WGEF-Archivs aus den Eingaben werden innerhalb eines eigenen Tabs *SaveTab.java* umgesetzt. Durch Auswahl der „Weiter“-Schaltfläche wird der Prozess angestoßen. War die Erstellung erfolgreich, so wird die Integration gestartet. **Abbildung 4.15** zeigt mit Hilfe eines Sequenzdiagramms die Verkettung der beteiligten Funktionen des Prozesses.

**Abbildung 4.15** – Erstellung WGEF und Integration

Der DataEntry-Controller ruft die Funktion `saveRequest()` des Tab Controllers auf. Dieser erzeugt zunächst einen *Request* (Funktion `createWgefRequest()`) mit den benötigten Parametern und sendet diesen an den WPS Dienst, der für die Erstellung des WGEF Datensatzes zuständig ist.

Die zurückgelieferte Antwort wird verarbeitet. Ist ein Fehler aufgetreten so wird dieser dem Nutzer gemeldet. Ist die Erstellung erfolgreich gewesen, so kann die Integration gestartet werden.

Dazu wird erneut ein Request aufgebaut und an den Integrations-WPS abgeschickt. Die Antwort wird erneut verarbeitet (*parseResponse()*). Bei einem Fehler wird der Nutzer informiert. Bei Erfolg wird ein Hinweis gegeben, dass die Integration erfolgreich war. Zusätzlich wird ein *CancelEvent* erzeugt, das den Assistenten wieder auf den Ausgangszustand zurücksetzt und somit ein neuer Datensatztyp für eine weitere Integration gewählt werden kann.

XML Validierung

Für die temporäre Speicherung von XML-Dateien wird innerhalb des Projektes die XML-Datenbank *eXist*⁴⁸ eingesetzt. Diese besitzt eine REST-konforme Schnittstelle für die Übertragung von XML-Dokumenten. Sie besitzt des Weiteren die Möglichkeit über die Konfiguration einen XML-Validierungsmechanismus einzuschalten, der die Dokumente bei der Eingabe gegen ein Schema zu überprüfen. Die Schemata können innerhalb der Datenbank verwaltet werden. Ist das Dokument valide, so wird es gespeichert. Ist das Dokument jedoch nicht gültig, so wird es abgelehnt und es wird eine Fehlermeldung zurückgegeben.

⁴⁸ <http://exist.sourceforge.net/> (abgerufen am 05.09.2010)

4.5 Ergebnisse

Die erhobenen Anforderungen dokumentieren die Ansprüche an die entwickelte Software. In den Teilkonzepten und dem Gesamtkonzept wurde deren Umsetzung geplant. Die Konzepte wurden in der Programmiersprache JAVA implementiert.

Das Ergebnis der Implementierung ist eine grafische Oberfläche, die sich in das ELVIS System als neue Seite einfügt und über die Registerkarte „DataEntryPortal“ gestartet wird. Dem Benutzer wird der Startbildschirm präsentiert (**Abbildung 4.16**)

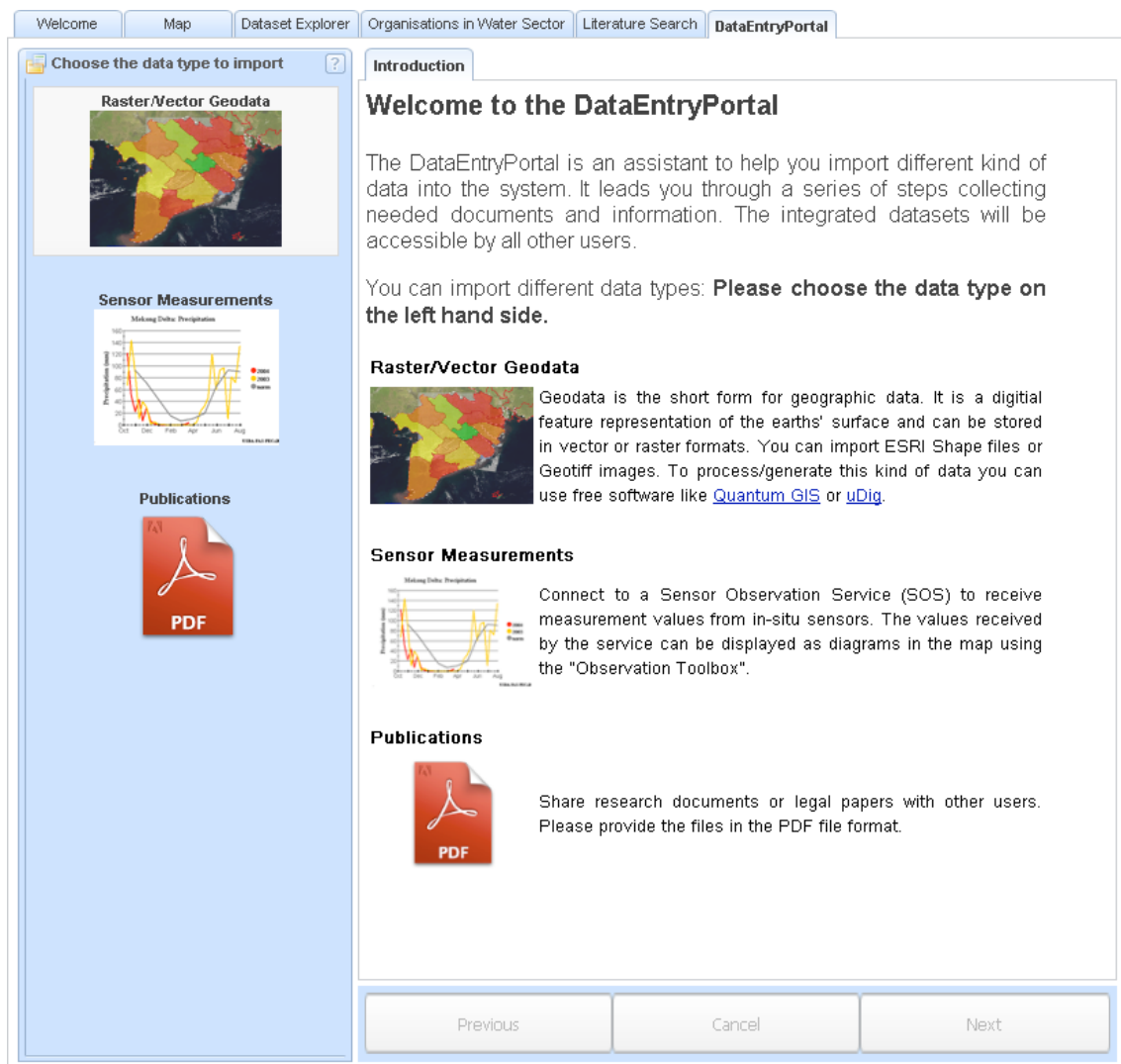


Abbildung 4.16 – Startbildschirm

Das Layout besteht aus zwei Spalten. Die rechte Spalte des Startbildschirms führt den Nutzer in die Bedienung des Assistenten ein und gibt ihm einen Überblick über die verschiedenen Datentypen, die in ELVIS integriert werden können. Um den Integrationsvorgang zu beginnen, wird der Nutzer gebeten in der linken Spalte den gewünschten Datentypen aus einer Liste verschiedener Datentypen auszuwählen. Durch einen einfachen Mausklick mit

der linken Maustaste auf einen Datentyp wird der Assistent für die Integrationsaufgabe initialisiert.

Im Rahmen dieser Diplomarbeit wurde der Assistent für die Integration eines Datensatzes vom Typ *Geodaten* umgesetzt (**Abbildung 4.17**).

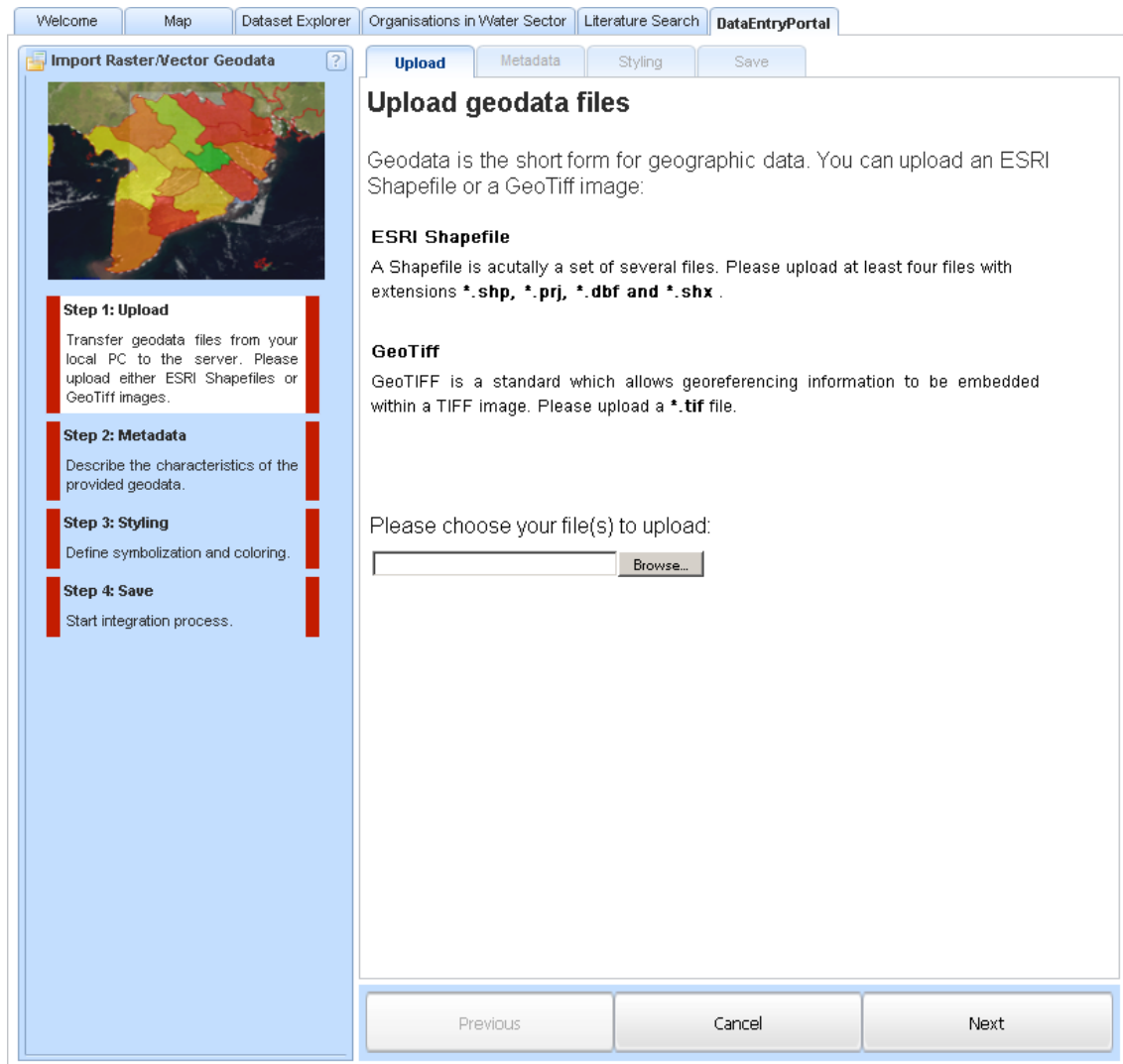


Abbildung 4.17 – Eingabe von Geodaten

Wurde im Startbildschirm der Datensatztyp *Geodaten* ausgewählt, so werden die verschiedenen Schritte für die Eingabe, Zusammenstellung und Integration geladen. In der linken Spalte wird nun eine Übersicht angezeigt, welcher Datensatztyp ausgewählt wurde („Raster/Vector Geodata“), welche Schritte für die Integration notwendig sind („Step 1 – Upload“, „Step 2 – Metadata“, Step 3 Styling“, „Step 4 – Save“ & eine Kurzbeschreibung der einzelnen Schritte).

Die weiße Hinterlegung zeigt den momentan bearbeiteten Schritt an und die Umrandung der Schritte den Bearbeitungsstatus (rot = unvollständig / grün = vollständig).

Zunächst ist lediglich der erste Schritt (*Upload*) für die Bearbeitung freigegeben. Der Nutzer wird gebeten entweder Dateien im *ESRI Shapefile* oder im *GeoTiff* Dateiformat von seinem Computer auf den Server hochzuladen. Über die Schaltfläche „Browse“ kann er die gewünschten Dateien hochladen. Der Nutzer über die Schaltfläche „Next“ in der Steuerungsleiste (rechte Spalte, unten) den Wunsch äußern, mit dem nächsten Bearbeitungsschritt fortzufahren. Es wird zunächst vom Assistenten überprüft, ob alle verpflichtenden Dateien bereitgestellt wurden. Ist dies der Fall, so wird der zweite Schritt (*Metadata*) für die Bearbeitung freigegeben (**Abbildung 4.18**). In der Übersicht wird der erste Schritt durch die grüne Umrandung als vollständig markiert. Als aktuell bearbeiteter Schritt wird durch die weiße Hinterlegung nun „Step 2 – Metadata“ angezeigt.

The screenshot shows the 'DataEntryPortal' interface. At the top, there are navigation tabs: 'Welcome', 'Map', 'Dataset Explorer', 'Organisations in Water Sector', 'Literature Search', and 'DataEntryPortal'. Below these, there are sub-tabs: 'Upload', 'Metadata', 'Styling', and 'Save'. The 'Metadata' sub-tab is active.

On the left side, there is a map titled 'Import Raster/Vector Geodata'. Below the map, there is a vertical list of four steps:

- Step 1: Upload** (green border): Transfer geodata files from your local PC to the server. Please upload either ESRI Shapefiles or GeoTiff images.
- Step 2: Metadata** (white border): Describe the characteristics of the provided geodata.
- Step 3: Styling** (red border): Define symbolization and coloring.
- Step 4: Save** (red border): Start integration process.

The main area on the right contains a form with the following sections:

- Personal Information:** Name, Organisation (dropdown), Position, E-Mail, Website, Phone, Fax, Street, Postal Code, City, Country.
- Dataset Information:** Dataset Title, Alternate Title, Abstract, Attributes, Purpose, Credit.
- Data Generation:** Data Generation Description, Data Source Description.

At the bottom of the form, there are three buttons: 'Previous', 'Cancel', and 'Next'.

Abbildung 4.18 – Metadateneingabe

Der zweite Schritt dient der Eingabe von Metadaten, die den Inhalt der bereitgestellten Dateien beschreiben. Dafür wird ein Formular mit verschiedenen Eingabefeldern angezeigt,

die vom Nutzer auszufüllen sind. Es gibt verpflichtende und optionale Felder. Der Nutzer äußert durch Betätigen der „Next“-Schaltfläche den Wunsch mit dem nächsten Schritt fortzufahren. Es wird überprüft ob alle Pflichtfelder korrekt ausgefüllt wurden. Ist die Eingabe unvollständig, so wird er mit einer Dialogbox darauf aufmerksam gemacht und hat die Möglichkeit die Eingaben zu ergänzen. Sind alle Felder korrekt ausgefüllt, so wird der dritte Schritt („Step 3 - Styling“) für die Bearbeitung freigegeben (**Abbildung 4.19**). Schritt zwei wird als vollständig markiert und Schritt 3 als aktuell bearbeiteter Schritt angezeigt.

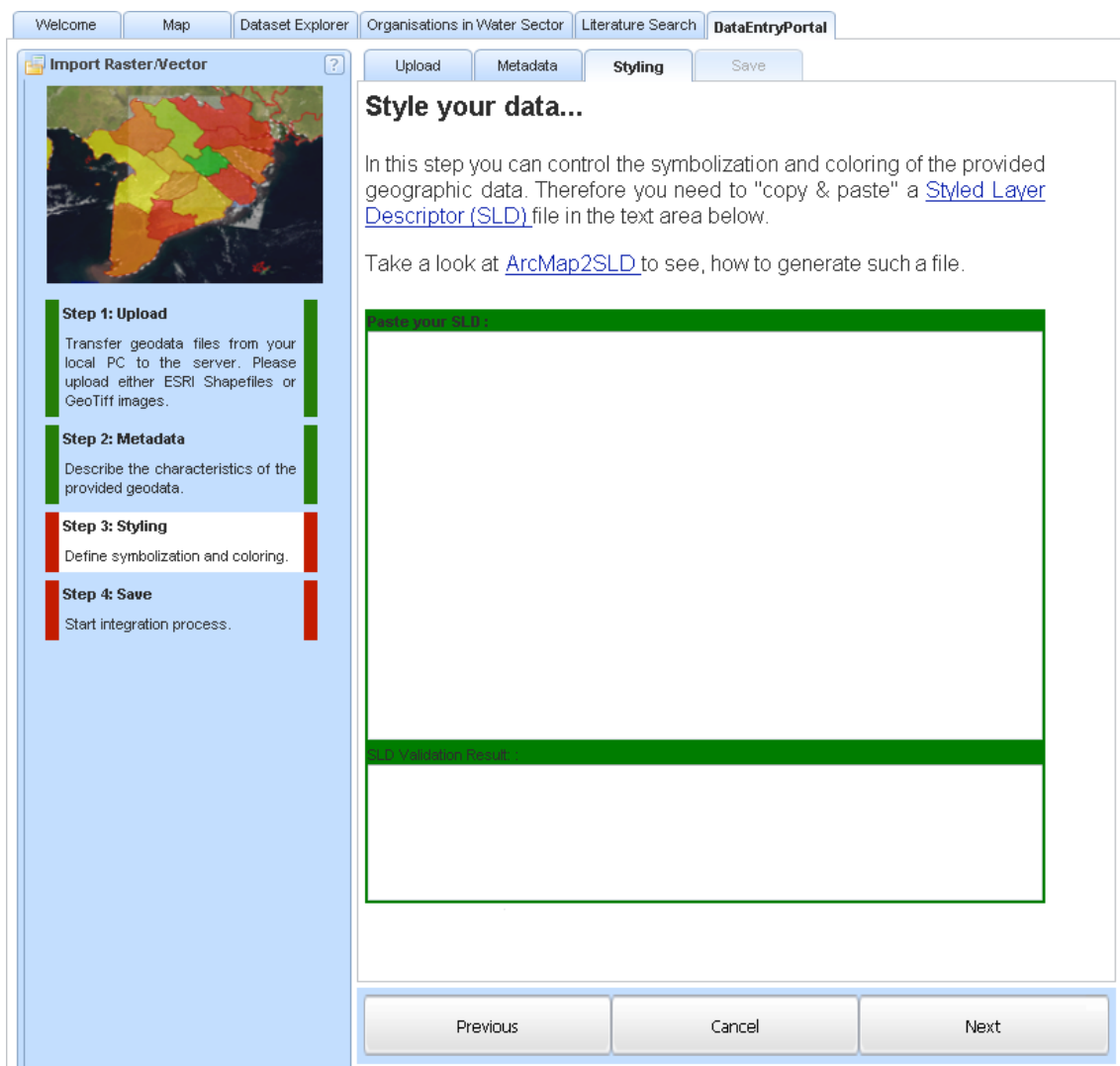


Abbildung 4.19 – Eingabe Darstellungsvorschriften

In Schritt drei wird festgelegt, wie später die Daten in ELVIS angezeigt werden. Dafür wird ein Eingabefeld bereitgestellt, in das der Nutzer ein extern erstelltes SLD-Dokument hineinkopiert. Durch Verweise (*Links*) auf externe Internetseiten und Software wird Hilfestellung gegeben, wie eine SLD-Datei erstellt werden kann. Nach Betätigen der Schaltfläche „Next“ wird die Datei zum Server übertragen und dort validiert, ob es sich um

ein gültiges SLD-Dokument handelt. Ist dies der Fall, so wird der letzte Schritt („Step 4 - Save“) freigegeben und angezeigt (**Abbildung 4.20**).

Im letzten Bearbeitungsschritt ändert die „Next“-Schaltfläche die Beschriftung zu „>>SAVE<<“. Sind alle zuvor getätigten Eingaben vollständig, so werden die Eingaben nach Betätigen der „SAVE“-Schaltfläche verarbeitet. Aus den Eingaben wird ein WGEF-Archiv erstellt und dieses an die Integrationssoftware übergeben. Ist die Integration erfolgreich, so wird der Nutzer über eine Dialogbox darüber informiert. Es wird der Startbildschirm angezeigt, um mit der Integration weiterer Datensätze fortzufahren.

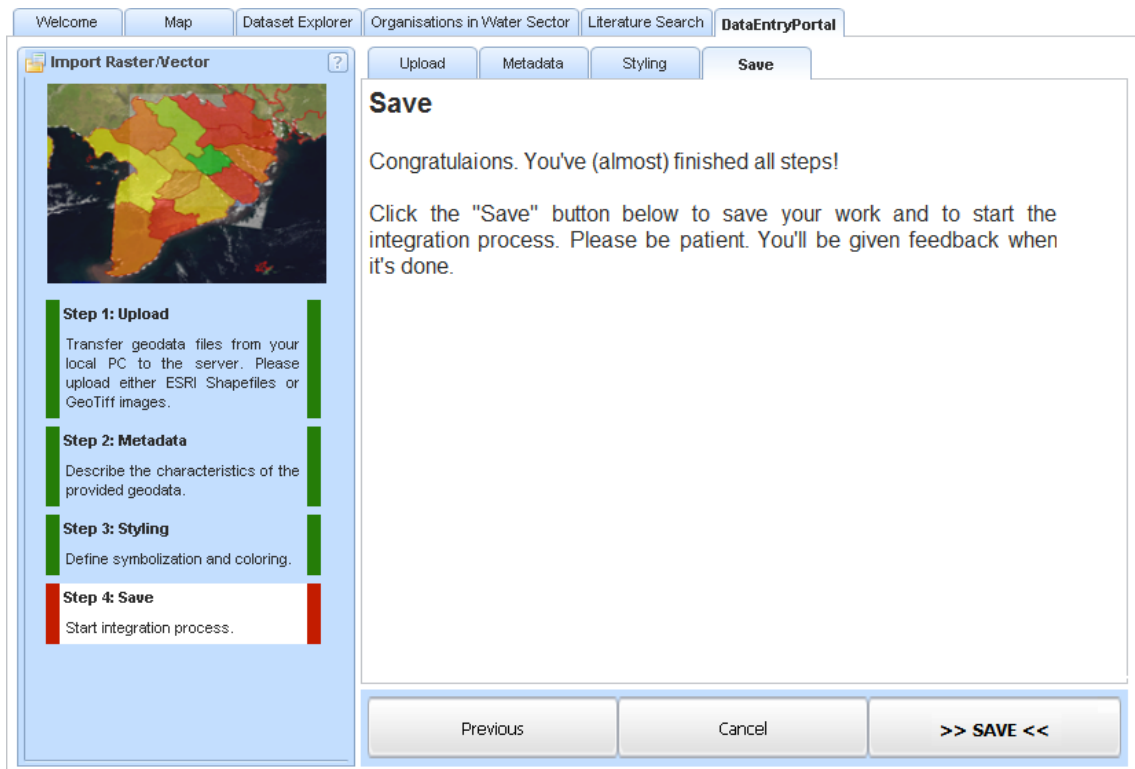


Abbildung 4.20 – Datenintegration

Mit Auswahl der „Previous“-Schaltfläche kann von jedem Schritt aus zum vorherigen Schritt navigiert werden. Die Schaltfläche „Cancel“ bewirkt den Abbruch der Integration. Nach Bestätigung des Abbruch-Wunsches werden getätigte Eingaben gelöscht und der Startbildschirm angezeigt, um mit der Integration eines anderen Datensatzes fortzufahren oder das DatenEingangsPortal zu verlassen.

4.5.1 Diskussion

Der entwickelte Assistent erfüllt technisch die erhobenen Anforderungen. Mit ihm können die Eingaben der Bestandteile vorgenommen und diese in die ELVIS Geodateninfrastruktur integriert werden. Des Weiteren kann der Assistent im Anschluss an diese Arbeit für die Integration weiterer Datensatztypen erweitert werden.

Die Technik wurde während der Entwicklung getestet. Die Umsetzung der Anforderung, dass die „Bedienung intuitiv ist“ beruht auf eigenen Erfahrungswerten und Erfahrungen des Entwicklerteams mit der Integrationsaufgabe. Sie müsste evaluiert werden. Dies konnte auf Grund der begrenzten Zeit nicht durchgeführt werden. Die Evaluierung könnte mit zukünftigen Nutzern des DatenEingangsPortal durchgeführt werden. Dafür wäre ein Test vorstellbar, in dem man potentielle Nutzer ein erstes Mal die Software bedienen lässt und ihnen über die Schulter schaut, ohne Hilfestellungen zu geben. Im Anschluss daran sollte die Eigenwahrnehmung der Probanden mittels eines Fragebogens abgefragt werden. Die Beobachtungen möglicher Irritationen bei der Bedienung sollten in weitere Verbesserungen der Software einfließen.

Des Weiteren ist es bei „Assistenten“ üblich im letzten Schritt (siehe **Abbildung 4.20**) eine Zusammenfassung der zuvor getätigten Eingaben anzuzeigen. Die Anzeige eines Bildes, das die hochgeladenen Geodaten nach den eingegeben Darstellungsregeln visualisiert, bevor der tatsächliche Integrationsvorgang gestartet wird, könnte einen deutlichen Mehrwert bieten. Der Benutzer könnte interaktiv die Darstellung der bereitgestellten Geodaten anpassen, ohne die Daten zunächst in ELVIS einspielen zu müssen.

5 Erweiterung DEP für den Import von Sensormessdaten

Das in Kapitel 3.2.2 auf Seite 27 vorgestellte DatenEingangsPortal (DEP) integriert Geodaten im Raster- (*GeoTiff*) oder Vektor (*ESRI Shapefile*) Datenformat in die ELVIS Geodateninfrastruktur.

In diesem Aufgabenteil wird das DatenEingangsPortal für die Integration von Messdaten von in-situ Sensoren erweitert. Das Hinzufügen neuer Sensoren zu ELVIS wird in dieser Arbeit nicht realisiert.

Im Kapitel 2 Stand der Technik auf Seite 11 wurde der *Sensor Observation Service* (SOS) des *Open Geospatial Consortium* vorgestellt, dessen Schnittstellen und Formate offene Standards für die Verwaltung so genannter Sensornetzwerke bereitstellen. Der SOS standardisiert unter anderem die Kodierung von Sensormesswerten in XML Dokumenten (*Observations&Measurements* (*O&M*)), und beschreibt Schnittstellen für den Datenabruf über die Funktion *GetObservation*, die Registrierung neuer Sensoren mittels *RegisterSensor* und auch den Import von Sensormessdaten über die Funktion *InsertObservation*.

In der Entwurfsphase von ELVIS gab es die Anforderung der einmaligen Integration von Sensoren und Sensormessdaten. Ein kontinuierlicher Datenfluss neuer Sensormessdaten war nicht vorgesehen. Aus dieser Anforderung heraus wurde die Entscheidung getroffen, dass das Hinzufügen und der Abruf von Sensoren und Sensormessdaten nicht über den SOS-Standard umzusetzen, sondern proprietäre Datenmodelle und Schnittstellen zu entwickeln.

Im Zuge der Entwicklung hat sich die Anforderung einer kontinuierlichen Ergänzung von Sensormessdaten ergeben. Für die Umsetzung der Anforderung wäre die Integration von Sensormessdaten über den SOS-Standard wünschenswert. Dies würde aber bedeuten, dass die Funktionalitäten zum Abruf und Visualisierung der Daten neu implementiert werden müssten, da diese auf das proprietäre Datenmodell und Schnittstellen angepasst sind. Dies kann weder im Rahmen dieser Arbeit, noch im Projektkontext zeitnah geleistet werden. Aus diesem Grund wird in dieser Arbeit die Datenintegration in das bestehende Datenmodell angestrebt. Das Datenmodell und das Austauschformat werden in den folgenden Unterkapiteln vorgestellt und darauf aufbauend das DEP für den Import von Sensormessdaten erweitert.

5.1 Rahmenbedingungen

5.1.1 Sensoren und Sensormessdaten in ELVIS

Die Verwaltung und Speicherung von Sensoren und Sensormessdaten wird innerhalb einer relationalen Datenbank vorgenommen (PostgreSQL). Dafür hat das DFD ein eigenes Datenbankmodell entwickelt. Die Modellierung baut auf Begriffen und Beziehungen zwischen den Begriffen auf, die im Folgenden beschrieben werden (semantisches Modell):

Ein Sensor („observer“) ist eine Plattform, auf der ein oder mehrere Messmodule installiert werden. Ein Messmodul („module“) ist für die Messung eines Objektparameters (z. B. Wasserstand, Lufttemperatur) in einer festgelegten Einheit zuständig. Eine Messung („observation“) verknüpft einen Messwert („value“), der zu einem bestimmten Zeitpunkt („time“) gemessen wurde mit dem Modul, das die Messung vorgenommen hat und den Sensor, zu dem das Modul zugehörig ist. Damit sind für eine Messung der Wert, der gemessene Umweltparameter, die Messeinheit („unit“), der Zeitpunkt und der Ort der Messung bekannt.

Aufbauend auf das semantische Modell wurden Datenbankschemata entwickelt (siehe **Abbildung 5.01**).

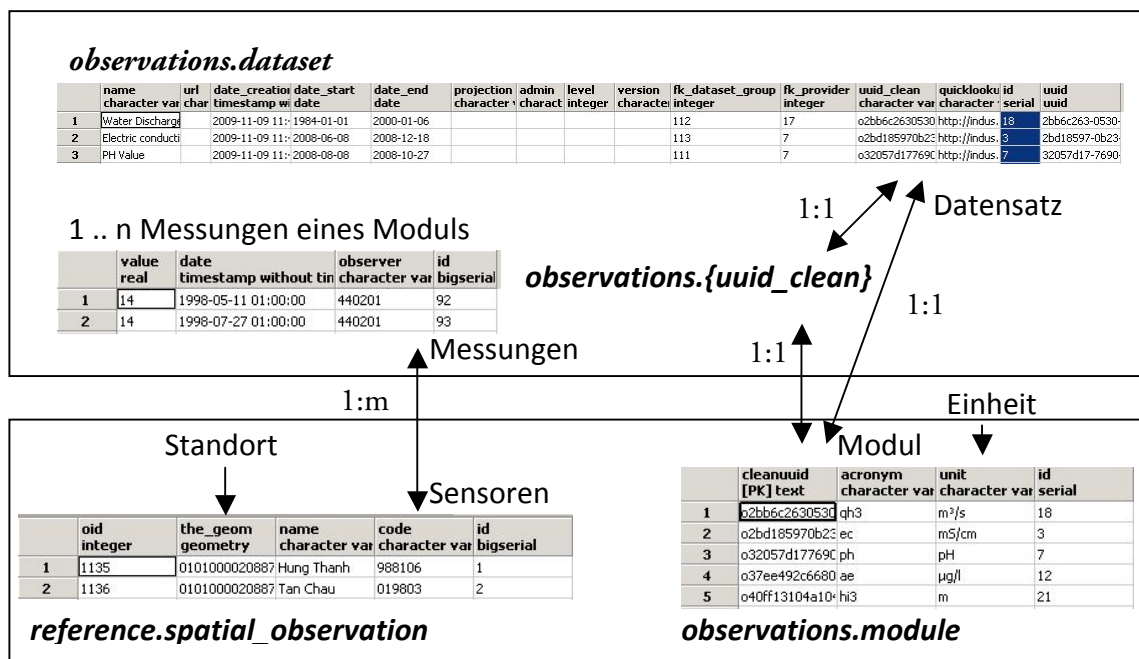


Abbildung 5.01 – Datenbankmodellierung für die Verwaltung von Sensoren und Sensormesswerten

Messungen werden als Datensätze in der Tabelle *observations.dataset* verwaltet. Ein Datensatz besteht aus 1 .. n Messungen eines Modultypen, die von einem oder mehreren Sensoren vorgenommen wurden. Er besitzt einen eindeutigen Identifizierungsschlüssel (uuid).

Der Datensatz wird über die Spalte *uuid_clean* einerseits mit einem Modul (Tabelle *observations.module*, Spalte *cleanuuid*), andererseits mit einer Tabelle *observations.{uuid_clean}*, die die Messungen speichert, verknüpft. Eine Messung besteht aus einem Identifikator (*id*), einem Zeitpunkt (*date*), einem gemessenen Wert (*value*) und einem Fremdschlüssel zu einem Sensor (*observer*). Der Fremdschlüssel verweist auf die Spalte *code* in der Tabelle *reference.spatial_observations*. Eine Integritätsbedingung (Fremdschlüssel-„Constraint“) legt fest, dass in die Tabelle *observations.{uuid}* nur Messungen eingefügt werden können, deren Sensor bereits in der Tabelle *reference.spatial_observation* registriert sind. Anderenfalls wird beim Einfügen eine Fehlermeldung zurückgegeben.

Kodierung von Sensormessdaten

Für den Austausch von Sensormessdaten wurde ein Format festgelegt, das die Messdaten kodiert (siehe **Listing 5.01**). Das Listing zeigt beispielhaft vier Messungen eines Moduls (Zeile 02 – Zeile 04).

Projektpartner liefern Messdaten als kommagetrennte Dateien (*Comma Separated Values*, CSV-Dateiformat). Diese beinhaltet vier Spalten: Den Identifikator des Sensors (Spalte 1: *observer*), den Messzeitpunkt (Spalte 2: *time*) im Format „YYYY-MM-DD hh:mm:ss.f“ (Y=Jahr, M=Monat, D=Tag, h=Stunde, m=Minute, s=Sekunde), den numerischen Messwert (Spalte 3: *value*) und das Akronym des Sensormoduls (Spalte 4: *module*), das den Wert gemessen hat. Das Akronym „wl“ steht in diesem Beispiel für die Bezeichnung „waterlevel“, also ein Sensormodul, das den Pegelstand zum Beispiel eines Flusses misst und ein Modul in Tabelle *observations.module* (Spalte „acronym“) referenziert. Der angegebene Sensor verweist auf einen, in *reference.spatial_observation* (Spalte „code“) gespeicherten, Sensor. Innerhalb der Spaltenwerte können Hochkommas und Leerzeichen vorkommen.

Listing 5.01 – Kodierung von Sensormessdaten im kommagetrennten CSV-Format.

```

01 observer, time, value, module
02 GFZ_G1, 2008-06-01 01:00:00.0, -0.08, wl
03 GFZ_G1, 2008-06-01 02:00:00.0, -0.07, wl
04 GFZ_G1, 2008-06-01 03:00:00.0, 0.02, wl
05 GFZ_G1, 2008-06-01 04:00:00.0, 0.29, wl
06 ...

```

5.2 Anforderungserhebung

5.2.1 Nichtfunktionale Anforderungen

„Eine *nichtfunktionale Anforderung* legt fest, welche Eigenschaften das Produkt haben soll“⁴⁹. **Tabelle 5.01** listet die nichtfunktionalen Anforderungen auf. Im Anschluss an die Tabelle werden diese erläutert.

Tabelle 5.01 – Nichtfunktionale Anforderungen

Identifikator	Anforderung
A_01	Erweitert das bestehende DatenEingangsPortal
A_01_01	Kann über das Internet ausgeführt werden

A_01 – Erweitert das bestehende DatenEingangsPortal

Der Mechanismus für den Import der Messdaten wird in das bestehende DatenEingangsPortal (DEP) eingebunden. Nach der Erweiterung kann beim Aufruf des DEPs - alternativ zum WGEF Archiv, das Geodaten enthält - eine Datei übergeben werden, die die Messdaten beinhaltet.

A_01_01 – Kann über das Internet ausgeführt werden

Der Dienst kann über das Internet ausgeführt werden. Dafür muss eine Schnittstelle geschaffen werden, die die Messdaten über das Internet entgegennimmt und ein Kommandozeilenprogramm für die Integration aufruft.

5.2.2 Funktionale Anforderungen

„Eine *funktionale Anforderung* legt fest, was das Produkt tun soll.“⁵⁰ **Tabelle 4.02** listet die funktionalen Anforderungen auf. Im Anschluss an die Tabelle werden diese erläutert.

Tabelle 5.02 – Funktionale Anforderungen

Identifikator	Anforderung
A_02	Identifizierung des übergebenen Datensatztypen
A_03	Auslesen der Inhalte einer kommasetrennten Textdatei (CSV Format). Überführen der Spalteninhalte in JAVA-Objekte.
A_03_01	Validierung der Dateneingaben

⁴⁹ http://de.wikipedia.org/wiki/Anforderung_Informatik (Stand: 31. Okt. 2010, 20:20)

⁵⁰ http://de.wikipedia.org/wiki/Anforderung_Informatik(Stand: 31. Okt. 2010, 20:20)

A_04	Bestimmen des Namens der Zieltabelle
A_05	Speichern der Daten in der Datenbank

A_02 – Identifizierung des übergebenen Datensatztypen

Anforderung A_02 folgt aus A_01. Beim Aufruf des DEP muss ein Einstiegspunkt (Funktionsaufruf) für die Integration des jeweiligen Datensatztypen bestimmt werden. Dafür wird zunächst der Datensatztyp identifiziert. Eine Weiche ruft nach der Erkennung des Datensatztypen die entsprechende Startfunktion auf.

A_03 – Auslesen der Inhalte einer kommagetrennten Textdatei (CSV-Format). Überführen der Spalteninhalte in JAVA-Objekte.

Für die Verarbeitung und Validierung der Eingabedaten wird der Inhalt der CSV-Datei in JAVA-Objekte/Variablen eingelesen.

A_03_01 – Validierung der Dateneingaben

Da die übergebenen Messdaten Fehler enthalten können, die zu einer Inkonsistenz in der Datenbank führen könnten, muss überprüft werden, ob der Inhalt der CSV-Datei definierten Erwartungen entsprechen. Dafür müssen die Erwartungen festgelegt und diese mit dem Inhalt der CSV verglichen werden. Entspricht der Inhalt nicht den Erwartungen, so werden die Daten nicht in der Datenbank gespeichert.

A_04 – Bestimmen des Namens der Zieltabelle

Die Messungen werden zu einem Datensatz hinzugefügt. Da die Messwerte eines Datensatzes in einer Zieltabelle *observations.{uuid}* gespeichert werden, muss zunächst über das - in der CSV Datei angegebene - Modul-Akronym der Identifikator des Datensatzes bestimmt werden. Dafür wird eine SQL-Anfrage an die Datenbank gesendet. Das zurück gelieferte Ergebnis entspricht dem Tabellennamen der Zieltabelle, in der die Messungen gespeichert werden.

A_05 – Speichern der Daten in der Datenbank

Die eingelesenen Messwerte werden in der, in Anforderung A_04, bestimmten Zieltabelle der Datenbank dauerhaft gespeichert.

5.3 Entwurfsplanung

5.3.1 Teilkonzepte

Konzept zur Erfüllung von Anforderung „A_01“

Erweitert das bestehende DatenEingangsPortal

Für die Integration von Sensormessdaten wird das DatenEingangsPortal in der gleichen Art und Weise aufgerufen, wie bei der Integration von Geodaten (siehe Kapitel 3.2.2). Anstatt des WGEF-Archivs wird eine Datei übergeben, die die Sensormessdaten enthält. Das DatenEingangsPortal bestimmt aus der übergebenen Datei den Datensatztypen (Anforderung A_02) und startet durch einen Funktionsaufruf den Import der Messdaten (Anforderung A_01_01).

Konzept zur Erfüllung von Anforderung „A_01_01“

Kann über das Internet aufgerufen werden

In Kapitel 3.2.2 auf Seite 30 wurde die WPS Kapselung des DEP vorgestellt. Der WPS stellt eine Schnittstelle für den Kommandozeilenaufruf der JAVA-Software über das Internet bereit. Um die Modularität des DEP zu erhöhen wird das Programm für die Sensormessdatenintegration ebenfalls über einen WPS gekapselt. Dieser wird durch das DEP aufgerufen (siehe **Abbildung 5.02**).

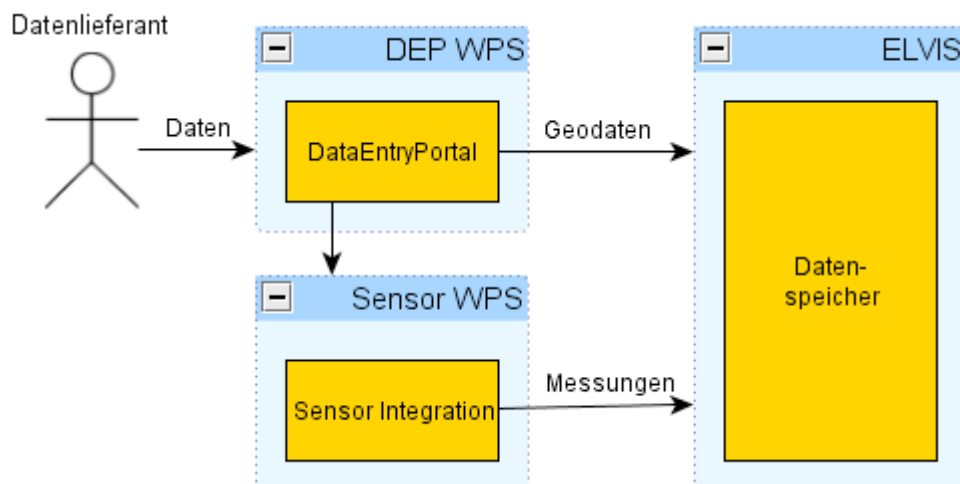


Abbildung 5.02 – WPS Kapselung des DatenEingangsPortals

Konzept zur Erfüllung von Anforderung „A_02“

Identifizierung des übergebenen Datensatztypen

Der Typ eines Datensatzes kann über mehrere Methoden identifiziert werden. Einerseits besteht die Möglichkeit diese über eine generische Methode durch das DEP zu bestimmen:

- Ein Datensatz besteht aus einer oder mehreren Dateien. Jede Datei besitzt eine Dateinamenswertweiterung. Die Kombination verschiedener Erweiterungen kann für die Identifizierung des Datensatztypen herangezogen werden. Z. B. charakterisiert die Kombination *.shp, *.prj, *.shx, *.dbf, *.xml, *.sld einen Geodatensatz. Ein Sensordatensatz besteht aus einer einzigen *.csv Datei. Die Kombination wird mit einer vorgehaltenen Liste abgeglichen. Allerdings kann nicht sichergestellt werden, dass diese Kombination für jeden Typ eindeutig ist.

Andererseits kann von außen die Information über den Datensatztypen beim Start durch den Aufrufer mitgegeben werden:

- Beim Aufruf des DatenEingangsPortals wird ein weiteres Argument übergeben, das den Datensatztyp spezifiziert („-type sensor“ bzw. „-t sensor“). Das DatenEingangs-Portal wertet das Argument aus.

Der Datensatztyp wird innerhalb der Dateinamenserweiterung kodiert:

- Die Datei(en) eines Datensatzes werden in einem Archiv komprimiert. Das Archiv erhält eine eindeutige Dateinamenserweiterung, durch die der Datensatztyp bestimmt wird. Analog zur Dateinamenserweiterung *.wgef wird die CSV-Datei in einem Archiv gespeichert, das die Endung *.wsef besitzt (WISDOM Sensor Exchange Format).

Die letztgenannte Methode ist für die Erfüllung der Anforderung gut geeignet. Sie erweitert das bisherige Konzept für die Kodierung der Datensatztypen (siehe WGEF). Die Dateinamenserweiterung ist eindeutig und die Methode ermöglicht die gleichzeitige Integration mehrerer Datensätze („Batch“-Import), da die Information innerhalb der Datei kodiert wird.

Konzept zur Erfüllung von Anforderung „A_03“

Auslesen der Inhalte einer kommagetrennten Textdatei (CSV Format). Überführen der Spalteninhalte in JAVA Objekte.

Die, in der CSV-Datei gespeicherten Daten werden in der Tabelle *observations.{uuid}* gespeichert. Vor der Speicherung muss zunächst der Identifikator der Zieltabelle (uuid) bestimmt werden (Anforderung A_04). Dafür ist es notwendig die Daten aus der CSV-Datei zunächst in JAVA-Objekte einzulesen und über die Spalte „module“ die Zieltabelle zu bestimmen.

Es wird eine Klasse „Measurement“ entworfen, die einzelne Messungen modelliert (**Abbildung 5.03**). Sie besitzt die Attribute „observer“, „value“ und „time“ und hat Methoden zum Auslesen der Attributwerte (getObserver()). Die CSV-Datei wird in einer Klasse DataExtractor

DataExtractor	Measurement	Module
file <File> readCsvFile()	observer <String> value <float> time <Date>	name uuid measurements sqlInsert
	getObserver() getValue() getTime()	getUuid() addMeasurement() save()

Abbildung 5.03 – JAVA-Klassen

in der Funktion readCsvFile() eingelesen und enthaltene Messungen zeilenweise in Measurement-Objekte überführt und verschiedenen Module-Objekten eingehängt.

Konzept zur Erfüllung von Anforderung „A_03_01“

Validierung der Dateneingaben

Das Ziel der Validierung ist es die Konsistenz der Datenbank zu wahren. Konsistenz bedeutet, dass die Daten innerhalb der Datenbank widerspruchsfrei sind. Eine Inkonsistenz würde z. B. dadurch entstehen, dass Werte in der CSV Datei falsche Datentypen beinhalten, Spalten fehlen oder Referenzen auf (in der Datenbank gespeicherte) Module oder Sensoren nicht korrekt sind.

Es müssen Erwartungen an die Formatierung der Daten gestellt und diese in JAVA formuliert werden. Des Weiteren müssen Mechanismen entwickelt werden, den Inhalt mit den Erwartungen zu vergleichen.

Erwartungen

- Spalten sind durch Kommata getrennt (Trennzeichen: „ , “)
- Zeilen sind durch einen Zeilenumbruch getrennt (Steuerzeichen „\n“)
- Die erste Zeile enthält vier Spaltenüberschriften (*observer*, *time*, *value*, *module*). Sie liegen in dieser Reihenfolge vor.
- Jede folgende Zeile entspricht einer Messung. Eine Messung besteht aus vier Spalten mit folgenden Datentypen:
 - 1. Spalte (*observer*): Datentyp String
 - 2. Spalte (*time*): Datentyp Date, Format: YYYY-MM-DD hh:mm:ss.f
 - 3. Spalte (*value*): Datentyp Float
 - 4. Spalte (*module*): Datentyp String

Validierung

Die Validierung findet nach dem Auslesen und hauptsächlich vor der Überführung in JAVA Objekte statt. Dafür werden die Spaltenwerte aus der CSV-Datei mit den Erwartungen über Zeichenkettenvergleiche, Bedingungen, Typumwandlung validiert. Eine weitere Art der Validierung sind die in den Datenbanktabellen modellierten Integritätsbedingungen:

- Das Vorhandensein der Spaltenüberschriften wird über einen Zeichenkettenvergleich durchgeführt. Es wird überprüft, ob die erste Zeile die Spaltenüberschriften „observer“, „time“, „value“, „module“ besitzen und in dieser Reihenfolge vorliegen (Spalte1.equals(„observer“), Spalte2.equals(„...“), ...).
- Überprüfen, ob alle Zeilen (Überschrift, Messungen) vier Spalten besitzen (Spalten.length == 4)
- Für das Programm sind alle eingelesenen Daten zunächst vom Typ String. Über eine Typumwandlung kann überprüft werden, ob der Inhalt den Erwartungen entspricht. Validierung der Spaltenwerte:
 - *time*: Der Spalteninhalt wird vom Typ String in einen Date-Datentyp umgewandelt. Schlägt die Konvertierung fehl, so ist der Inhalt der Spalte ungültig.
 - *value*: Schlägt die Konvertierung vom Typ String in den Typ float fehl, so ist der Inhalt der Spalte ungültig.
 - *observer*: Der Spalteninhalt wird ohne Typumwandlung als String eingelesen. Bei der Ermittlung der Zieltabelle (siehe Anforderung A_04) über eine SQL-Anfrage wird ein leeres Ergebnis zurückgeliefert, falls der referenzierte Sensor in der Tabelle *reference.spatial_observation* nicht vorhanden ist. Ist dies der Fall, so ist der Inhalt der Spalte ungültig.
 - *module*: Wird ohne Typumwandlung als String eingelesen. Falls das referenzierte Modul nicht in der Tabelle *observations.module* vorhanden ist (Integritätsbedingung der Datenbanktabelle, Fremdschlüssel- „Constraint“-Verletzung), wird beim Durchführen der INSERT-Anweisung ein Fehler zurückgegeben. Ist dieses der Fall, so ist der Inhalt der Spalte ungültig.

Konzept zur Erfüllung von Anforderung „A_04“

Bestimmen des Namens der Zieltabelle

Für die Speicherung der Messungen, muss der Name der Zieltabelle bestimmt werden. Die Zieltabelle wird über das, in der Messungen angegebene, Modul-Akronym (**Listing 5.01** auf Seite 71, Zeile 02-05, Spalte 4) mittels einer SQL-Anfrage ermittelt (**Listing 5.02**). Das Ergebnis ist der Identifikator des zugehörigen Datensatzes (UUID) und entspricht dem Namen der Zieltabelle, in der die Messungen gespeichert werden (UUID: o2bb6c26305304a87b3bcb9e4a8967eac -> Zieltabelle: *observations.o2bb6c26305304a87b3bcb9e4a8967eac*).

Listing 5.02 – Bestimmen der UUID anhand des Modul Akronyms über die Tabelle observations.module

```
01 SELECT cleanuuid FROM observations.module WHERE acronym = 'wl';
```

Wird eine leere Ergebnismenge zurückgebenden, so ist das referenzierte Modul nicht in der Datenbank registriert. Die Messung wird nicht weiter verarbeitet.

Konzept zur Erfüllung von Anforderung „A_06“

Speichern der Daten in der Datenbank

Die Speicherung der Messungen erfolgt in der, in Anforderung A_04, ermittelten Tabelle *observations.{uuid}*. Die Struktur von CSV Dateien, ist der Datenstruktur der Datenbanktabellen ähnlich (Zeilen und Spalten). In manchen Datenbankmanagementsystemene gibt es Anweisungen für den direkten Import von Dateien im CSV-Format in die Datenbank (MySQL: „LOAD DATA INFILE“, PostgreSQL: „COPY“, siehe **Listing 5.03**). Diese Anweisungen sind Datenbanksystem spezifisch und bieten eine höhere Geschwindigkeit beim Import über einzelne INSERT-Anweisungen⁵¹.

Listing 5.03 – Speicherung der Werte in der Tabelle observations.{uuid} über „Copy“ (PostgreSQL)

```
01 COPY tablename [ ( column [, ...] ) ]
02     FROM { 'filename' | STDIN }
03     [ [ WITH ]
04         ...
05         [ CSV [ HEADER ]
06             [ QUOTE [ AS ] 'quote' ]
07             [ ESCAPE [ AS ] 'escape' ]
08             [ FORCE NOT NULL column [, ...] ]
```

⁵¹ <http://jakub.wartak.pl/blog/?p=93> (abgerufen am 1.11.2010)

Listing 5.03 – Beispielbefehl

```
01 COPY observations.o9352c9d2b03c40b58e48d9cace7158c0 (observer,
02 date, value) FROM 'C:/data/measurements_ph.csv' WITH CSV HEADER
```

Die Spaltenanzahl innerhalb der CSV-Datei muss dabei der Spaltenanzahl der Zieltabelle entsprechen. Dies ist jedoch bei vorliegender Spezifikation des Austauschformats für Sensormessdaten (**Listing 5.01** auf Seite 71) nicht der Fall. Vor dem Ausführen des COPY Befehls müsste die Spalte „module“ aus der CSV Datei entfernt werden.

Des Weiteren können in einer CSV-Datei Messungen verschiedener Module eines Sensors enthalten sein. Messungen unterschiedlicher Module werden in verschiedenen Zieltabellen gespeichert. Damit ist die Speicherung der Messdaten mit der COPY Anweisung in diesem Fall nicht umsetzbar.

Aus diesem Grund wird die Speicherung über INSERT-Statements durchgeführt. Die verschiedenen Messmodule werden als Module-Objekte modelliert, in die die verschiedenen Messungen einsortiert werden. Die Methode save() der Klasse Module speichert die enthaltenen Messungen in der Datenbank.

Listing 5.04 – Speicherung der Werte in der Tabelle observations.{uuid} über eine INSERT-Anweisung

```
01 INSERT INTO observations.{uuid} VALUES
02 (observer, time, value),
03 (observer, time, value);
04 ...
```

5.4 Implementierung

5.4.1 Erweiterung

Das DatenEingangsPortal (DEP) dient als zentraler Einstiegspunkt für die Datenintegration in ELVIS. Der DEP-WPS wird über eine HTTP GET Anfrage an eine URL gestartet (Kapitel 3.2.2, Seite 30). Der Dienst ruft über die Kommandozeile das DEP auf und übergibt eine ZIP-Datei mit der Endung .wsef, die die CSV Datei mit den Messungen enthält. Über die Dateiendung des Archivs wird der Datensatztyp bestimmt und die zugehörige Startfunktion aufgerufen (siehe **Listing 5.05**).

Listing 5.05 – Weiche zum Aufruf der Startfunktion

```
01 if(file.endsWith(wsef)) {
02     sensorIntegration(file);
03 } else if(file.endsWith(wgef) {
04     geodataIntegration(file);
05 }
```

Die Startfunktion *sensorIntegration(file)* erzeugt eine URL zum Aufruf des Sensor WPS und führt eine HTTP GET Anfrage an die URL durch (**Listing 5.06**)

Listing 5.06 – Adresse des Sensor WPS Dienstes

```

01 http://mekongvm5/cgi-bin/wps.py?
02     request=execute&
03     identifier=sensorIntegration&
04     datainputs=[
05         geodata=http://xxx.xxx.xxx.xxx/sensorMeasurement.wsef ]
06     responsedocument=[package:wsef:result=@asreference=true]&
07     service=wps&
08     version=1.0.0

```

Der Sensor WPS ruft wiederum über die Kommandozeile die JAVA-Software für die Integration von Sensormessdaten auf, die im Folgenden beschrieben wird.

5.4.2 Verarbeitung

Die Methode *readCsvFile()* der Klasse *DataExtractor* liest die Messungen Zeilenweise aus der CSV-Datei ein und validiert diese. Die Werte werden in „Measurement“-Objekte überführt (siehe Kapitel 5.4.3).

Abhängig von dem angegebene Modul-Akronym wird ein Objekt der Klasse *Module* instanziiert (falls dieses nicht schon vorhanden ist) und zu einer *HashMap modules* hinzugefügt. Bei der Instanziierung wird über das Modul-Akronym die Zieltabelle aller enthaltenen Messungen ermittelt. Dafür wird eine Anfrage an die Datenbank gestellt (siehe **Listing 5.02**). Aus dem Ergebnis der Anfrage (UUID) wird der Name Zieltabelle für die Speicherung bestimmt. Die Messung wird durch Aufruf der Objektmethode *addMeasurement(...)* zu dem jeweiligen Modul-Objekt hinzugefügt.

Nach dem Einlesen aller Zeilen wird für jedes Modul in *modules* die Objektmethode *save()* aufgerufen. Diese transformiert alle im *Module*-Objekt enthaltenen *Measurement*-Objekte in ein INSERT-Statement (siehe **Abbildung 5.04**) und speichert die Messungen in der, bei der Instanziierung ermittelten Zieltabelle.

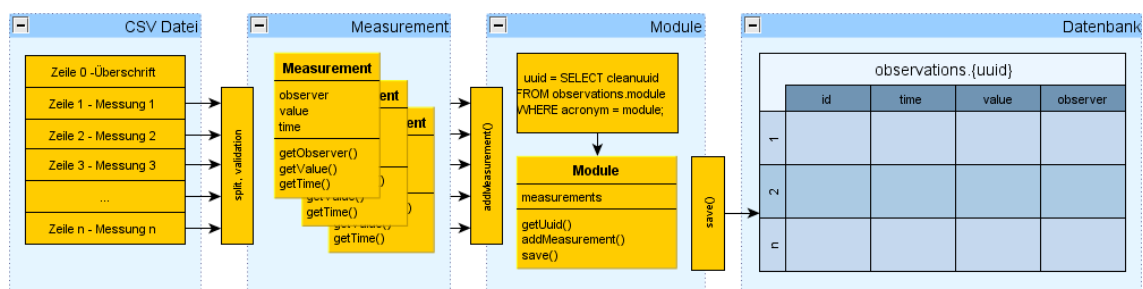


Abbildung 5.04 – Verarbeitung der Sensormessdaten

5.4.3 Einlesen & Validierung

Einlesen

Für das Einlesen der Messungen wird zunächst ein Objekt der Klasse *DataExtractor* instanziiert (**Listing 5.07**, Zeile 01).

Listing 5.07 – Einlesen der Dateien

```
01 DataExtractor dataExtractor = new DataExtractor(file);
02 ArrayList<Module> modules = dataExtractor.readCsvFile(...);
```

Dem Konstruktor wird eine Dateireferenz auf die CSV-Datei übergeben. Durch Aufruf der Funktion *readCsvFile()* werden die Messwerte in Measurement-Objekte überführt, zu Module-Objekten hinzugefügt und Referenzen auf die Module-Objekte als ArrayListe zurückgegeben.

Measurement-Objekte besitzen die Attribute *observer*, *value*, *time* (siehe **Listing 5.08**) und bieten Funktionen zum Lesen der Attributwerte (z. B. Zeile 12, *getObserver()*).

Listing 5.08 – Klasse Measurements

```
01 public class Measurement {
02     String observer;
03     float value;
04     Date time;
05
06     Measurement(String observer, Date time, float value)
07     {
08         this.observer = observer;
09         this.value = value;
10         this.time = time;
11     }
12
13     public String getObserver() {
14         return observer;
15     }
16     ...
17 }
```

Validierung

Abbildung 5.05 verdeutlicht schematisch die Validierung des Dateiinhalts. Die erste Zeile der CSV Datei enthält die Spaltenüberschriften. Beim Einlesen werden die Zeilen durch das Trennzeichen Komma in Spalten getrennt (*columns = Zeile.split(„,“)*). Die einzelnen Spalten werden in einem Array „*columns*“ gespeichert. Über die Größe des Arrays wird die Spaltenanzahl bestimmt (*columns.length()*). Diese muss der Zahl Vier entsprechen.

Die Erwartung ist, dass das erste Element des Arrays die Zeichenkette „observer“, die Zweite „time“, die Dritte „value“ und die vierte „module“ enthält. Der Vergleich erfolgt über die Funktion „`column[0].equals(„observer“)`“, die bei Übereinstimmung einen Boolean-Wert „true“ zurückliefert, anderenfalls ein „false“. Vor dem Vergleich werden zunächst alle Hochkommata und Leerzeichen entfernt. Sind alle oben genannten Bedingungen „true“, so sind die Spaltenüberschriften korrekt. Andernfalls wird eine Fehlermeldung ausgegeben und die Verarbeitung aller weiteren Messungen abgebrochen.

Alle folgenden Zeilen der CSV-Datei enthalten Messungen. Die Validierung geschieht entweder über Typumwandlung (**Listing 5.09**). Die Messungen werden ebenfalls über die „`split()`“-Funktion in Spalten getrennt und Hochkommata und Leerzeichen aus den Werten entfernt.

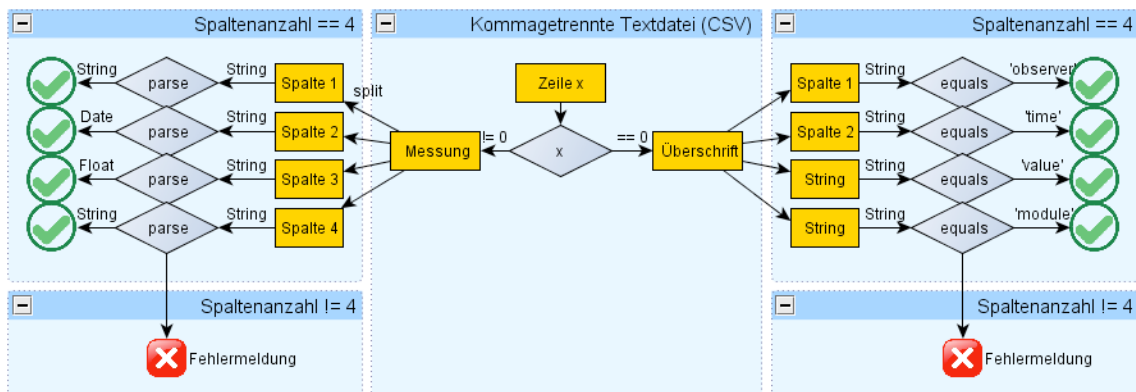


Abbildung 5.05 – Validierung der Dateneingaben

Listing 5.09 zeigt beispielhaft die Validierung der zweiten Spalten „time“. Zunächst wird in Zeile 08 die Erwartungen an das Format der Werte festgelegt. In Zeile 10 erfolgt die Typumwandlung von String zu dem Datentyp Date. Hochkommata werden zuvor über `replace()` ersetzt. Schlägt die Konvertierung fehl, so wird eine Fehlermeldung ausgegeben (siehe Zeile 12 bis 16) und der Initialwert der Variable „time“ bleibt „null“.

Listing 5.09 – Typumwandlung von String zu Date der Werte in der Spalte „time“

```

01 Date time = null;
02
03 switch(x) {
04     case 0: {
05         ...
06     }
07     case 1: {
08         try {
09             DateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
10                 HH:mm:ss.f");

```

```

11     time = (Date) formatter.parse(columns[x].replace("'", ""));
12     } catch (ParseException e) {
13         logger.error(formatter.format(new Date()
14             + " File: "+file.getName()
15             + ", row "+(lineCount+1)+",
16             + column "+(x+1)+" (" +csvHeader[x]+").
17                 Necessary format: yyyy-MM-dd HH:mm:ss.f");
18     }
19 }
20 case 2: {
21     ...
22 }
23 case 3: {
24     ...
25 }
26 break;
27 }
28 }

```

Die Überführung in Measurement-Objekte findet nur statt, wenn alle Typumwandlungen erfolgreich durchgeführt werden können (**Listing 5.10**).

Listing 5.10 – Überführung der Ausgelesenen Werte in Measurement-Objekte

```

01  HashMap<String, Module> modules = new HashMap<String, Module>();
02
03  if( observer != null
04      && time != null
05      && value != null
06      && module != null
07      && error != null) {
08
09      Measurement measurement=new Measurement(observer, time, value);
10      module.addMeasurement(measurement);
11  }
12  ...
13  return measurements

```

Eine Validierung findet nur statt, wenn beim Einlesen eine Typumwandlung stattfindet.

Spalte 1, case 0 (observer):	String -> String	(keine Validierung)
Spalte 2, case 1 (time):	String -> Date	(Validierung)
Spalte 3, case 2 (value):	Datentyp -> Float	(Validierung)
Spalte 4, case 3 (module):	String -> String	(keine Validierung)

Da in Spalte 1 & 4 keine Typumwandlung stattfindet werden diese bei Durchführung der INSERT-Anweisung validiert. Falls der referenzierte Sensor („observer“) nicht in der Tabelle *reference.spatial_observation* vorhanden ist (Fremdschlüssel-„Constraint“ in der Tabelle *observations.{uuid}*), wird ein Fehler von der Datenbank zurückgegeben (**Listing 5.11**)

Listing 5.11 – Fehlermeldung bei Verletzung des Fremdschlüssel-Constraint

```

01 FEHLER: Einfügen oder Aktualisieren in Tabelle
02 »o32057d17769041f69a24da93fe42736e« verletzt Fremdschlüssel-
03 Constraint »fk32_observer«

```

Speicherung

Die Funktion `readCsvFile()` zum Lesen der CSV Datei hat als Rückgabewert eine HashMap mit allen Referenzen auf alle Modul-Objekte, die die Measurement-Objekte beinhalten. Für alle Module wird die Funktion `save()` aufgerufen, die die Messungen in eine INSERT-Anweisung transformiert und in der Datenbank speichert (siehe **Listing 5.12**).

Verletzt die Sensor-Referenz (*observer*) beim Speichern die Fremdschlüssel-Bedingung, so wird eine Fehlermeldung zurückgegeben.

Listing 5.12 – Methode `save()` der Klasse `Module.java`

```

01 void save() {
02     java.util.Iterator<Measurement> iterator=measurements.iterator();
03     while(iterator.hasNext()) {
04         Measurement measurement = (Measurement) iterator.next();
05
06         if(!uuid.equals("")) {
07             sqlInsert += "("+measurement.getObserver()+",'+"+
08                 measurement.getTime()+",'+"+
09                 measurement.getValue()+"'";
10         if(iterator.hasNext()) {
11             sqlInsert += ",";
12         } else {
13             sqlInsert += ";";
14
15         try {
16             this.dbTemplate.execute(sqlInsert);
17         } catch (Exception e){
18             System.out.println("Exception: "+e);
19         }
20     }
21 }
22 }
23 }

```

5.5 Ergebnisse

5.5.1 Aufruf

Die Integrationssoftware wird in der gleichen Art und Weise aufgerufen, wie bei der Integration von Geodaten, allerdings wird statt dem WGEF-Archiv eine CSV-Datei übergeben (**Listing 5.13**). Die Parameter für den „Provider“ und die „ThematicGroup“ (Kapitel 3.2.2, Seite 27) werden für die Integration von Sensormessdaten nicht benötigt.

Listing 5.13 – Aufruf der Erweiterung

```
01 java -jar sensor.jar -c wisdom_configuration.xml -i data/wl.csv
```

Parameter:

- c: Konfigurationsdatei. Enthält Anmeldedaten für die Datenbank.
- i: Sensormessdatei im CSV-Dateiformat

5.5.2 Softwaretest

Für den Test der Software wurde ein Testdatensatz erstellt (siehe **Listing 5.14**).

Listing 5.14 – Beispieldatensatz Sensormessdaten (measurements.csv)

```
01 observer,time,value,module
02 U_B1, '2008-07-17 23:19:41',0.3438194, 'wl'
03 U_B1, '2008-07-17 23:04:34',0.3538194, 'wl'
04 U_B1, '2008-07-17 23:34:48',0.34308416, 'wl'
05 U_B1, '2008-07-17 23:49:55',0.34279609, 'wl'
06 U_B1, '2008-07-18 00:05:02',0.3425537, 'wl'
07 U_B1, '2008-07-18 00:20:10',0.33890206, 'wl'
08 U_B1, '2008-07-18 00:35:17',0.33207196, 'wl'
09 U_B1, '2008-07-18 00:50:24',0.32816511, 'wl'
10 U_B1, '2008-07-18 01:05:31',0.32145578, 'wl'
11 U_B1, '2008-07-18 01:20:38',0.31738895, 'wl'
```

Die Ausgabe wird in **Listing 5.15** dargestellt. Für die Integration von 10 Messungen benötigt die Erweiterung etwas weniger als eine Sekunde (Zeitvergleich: Zeile 01 & Zeile 22), wobei die Verarbeitung der Testdatei in JAVA in etwa die gleiche Zeit benötigt, wie die Speicherung in der Datenbank (Zeitdifferenz: Zeile 01 & Zeile 07, Zeile 07 & Zeile 22).

Listing 5.15 – Beispieldatensatz Sensormessdaten (measurements.csv)

```
01 16:43:23,869 INFO - 2010/11/16 16.43.23 -
02     Starting sensor measurements integration: measurements.csv
03
04 16:43:23,947 INFO - Loading XML bean definitions from file
05     [C:\eclipse_workspace\dep_new_work\wisdom_configuration.xml]
06
07 16:43:24,072 INFO - Loaded JDBC driver: org.postgresql.Driver
08
09     INSERT INTO observations.o9352c9d2b03c40b58e48d9cace7158c0
10     (observer, date, value) VALUES
11         ('U_B1', 'Thu Jul 17 23:19:41 CEST 2008', '0.3438194'),
12         ('U_B1', 'Thu Jul 17 23:04:34 CEST 2008', '0.3538194'),
13         ('U_B1', 'Thu Jul 17 23:34:48 CEST 2008', '0.34308416'),
14         ('U_B1', 'Thu Jul 17 23:49:55 CEST 2008', '0.3427961'),
15         ('U_B1', 'Fri Jul 18 00:05:02 CEST 2008', '0.3425537'),
16         ('U_B1', 'Fri Jul 18 00:20:10 CEST 2008', '0.33890206'),
17         ('U_B1', 'Fri Jul 18 00:35:17 CEST 2008', '0.33207196');
18         ('U_B1', 'Fri Jul 18 00:50:24 CEST 2008', '0.3281651'),
19         ('U_B1', 'Fri Jul 18 01:05:31 CEST 2008', '0.32145578'),
20         ('U_B1', 'Fri Jul 18 01:20:38 CEST 2008', '0.31738895');
21
22 16:43:24,588 INFO Inserted 10 measurements
23
24 16:43:24,588 INFO - 2010/11/16 16.43.24 ---
25     Stopping sensor measurements integration: measurements.csv
```

Es wurden weitere Tests mit fehlerhaft strukturierten Datensätzen durchgeführt. Diese sind im Anhang dokumentiert.

6 Schlussbetrachtung

6.1 Zusammenfassung

Die Entwicklungsarbeiten fanden unter vorgegebenen **Rahmenbedingungen** statt. Diese wurden durch die bisher vom DFD geleisteten Arbeiten vorgegeben. Daraus resultieren die verwendeten Technologien **GWT**, **SmartGWT** und **JAVA** für die Entwicklung des Assistenten. Aus diesem Grund wurden zunächst die **bestehenden Softwarekomponenten** in dieser Arbeit **analysiert und beschrieben**.

Darauf aufbauend wurde ein grafischer **Assistent** für die **Eingabe von Geodatendateien**, **Metadaten** und **Darstellungsregeln** und den **Aufruf der Integrationssoftware** entwickelt. Dafür wurden zunächst - zusammen mit dem Entwicklerteam - **funktionale und nicht-funktionale Anforderungen** an den Assistenten erhoben. Diese Anforderungen wurden zunächst in **Teilkonzepten** einzeln betrachtet und mittels eines **Gesamtkonzeptes**, bestehend aus einem technischen, einem gestalterischen und einem Bedienkonzept, miteinander in Verbindung gesetzt. Die Konzeption ist wiederum die Grundlage für die darauf folgende programmtechnische **Implementierung** der Funktionalitäten, die zu den beschriebenen **Ergebnissen** führten.

Neben der grafischen Oberfläche zur Eingabe durch einen Datenlieferanten als Vorbereitung für die Integration von Geodaten, wurde im zweiten Teil der Arbeit die **bestehende Integrationssoftware** zur automatischen Integration erweitert. Dafür wurde zunächst das bestehende DatenEingangsPortal als Softwarekomponente für die automatische Integration von Geodaten vorgestellt.

Die Erweiterung beinhaltet das **Hinzufügen von Messdaten** von in-situ Sensoren zu bereits registrierten Sensormessstationen und Messmodulen. Dafür liest die Erweiterung übergebene Daten aus einer CSV-Datei ein, validiert die Eingaben und überführt sie in JAVA-Objekte. Die Messwerte werden in einer bestimmten Datenbanktabelle gespeichert. Die Entwicklung führte analog zur vorherigen Vorgehensweise über eine **Anforderungserhebung**, der **Konzeptionierung** hin zur **Implementierung**.

6.2 Fazit

Durch den zweigeteilten Aufbau der Arbeit wurde die Integration von Geodaten in das Umweltinformationssystem aus zwei verschiedenen Blickwinkeln betrachtet. Mit der Entwicklung des Assistenten wurde ein benutzerfreundliches Werkzeug geschaffen, um Daten in ein Austauschformat zu überführen. Die Erweiterung für den Import von Sensormessdaten hingegen hatte als Ziel, Daten von einem Austauschformat entgegenzunehmen und dessen Inhalt in die Speicherstrukturen zu überführen.

Dreh und Angelpunkt der Datenintegration sind die Austauschformate. Die Komplexität der Integration steigt durch die Vielzahl unterschiedlicher Formate und die verschiedenen Aspekte, die Daten zugeordnet werden können. Bei Geodaten gibt es in ELVIS die Aspekte der Geometrie, der Attributdaten, der Visualisierung, der Zeit und der Dokumentation mittels Metadaten. Diese werden (teilweise) in ein eigenes, proprietäres Format „verpackt“. Eine Reduktion der verschiedenen Formate könnte z.B. durch die Verwendung von GML bzw. KML erreicht werden, da diese innerhalb einer Beschreibungssprache (XML) die verschiedenen Aspekte, wie z.B. Geometrie, Darstellung und die Zeit modellieren.

Dies könnte einerseits die Integration vereinfachen, andererseits auch die Datenverarbeitung innerhalb von ELVIS.

Quellenverzeichnis

Die Nummern innerhalb der Fußnoten verweisen auf die Nummern in diesem Quellenverzeichnis. Quellen, die der WISDOM Projektdokumentation entnommen wurden und bisher nicht öffentlich verfügbar sind, wurden auf der beiliegenden CD-ROM archiviert (siehe Anhang 7.1.1 und README.txt im Stammverzeichnis der beigelegten CD-ROM). Bei den entsprechenden Quellenangaben wird darauf verwiesen.

1 – Projekt WISDOM

Titel: Water Related Information System for the Sustainable Development of the Mekong
Homepage: http://www.wisdom.caf.dlr.de/intro_en.html

2 – Projekt CAWa

Titel: CAWa - Central Asian Water
Homepage: <http://www.cawa-project.net/>

3 / 10 – Geodateninfrastruktur Deutschland

Titel: **Übersetzung des ISO 19115 - Metadaten vom Englischen in deutsche Sprache**
URL: http://www.gdi-de.org/de_neu/thema/2009/c_thema_iso_uebersetzung.html
Abruf: 7.8.2010

4 – Open Geospatial Consortium

Titel: **Styled Layer Descriptor**
URL: <http://www.opengeospatial.org/standards/sld>
Abruf: 7.8.2010

5 – Deutsche Wikipedia

Titel: **World Geodatic System 1984**
URL: http://de.wikipedia.org/wiki/World_Geodetic_System_1984
Stand: 17.11.2011, 21:19 Uhr

6 – ESRI

Titel: **ESRI Shapefile Technical Description**
URL: <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>

7 – Stefan Steiniger

Titel: **An Overview of Free & Open Source Desktop GIS (FOS-GIS)**
URL: http://spreadsheets.google.com/ccc?key=0Albk_XRkhVkzdGxyYk8tNEZvLUp1UTUzTFN5bjlLX2c&hl=en
Stand: 2010
Abruf: 08.10.2010

8 – Manfred Mittelböck, Paul Schreilechner

Titel: **Metadaten: ISO – konformes Profil als Schritt in die Praxis in Österreich**
URL: http://ispace.researchstudio.at/downloads/2004/mittlboeck_schreilechner_2004_agit_iso_profil_at.pdf
Stand: 2004

9 – Deutsche Wikipedia

Titel: **Styled Layer Descriptor**

URL: http://de.wikipedia.org/wiki/Styled_Layer_Descriptor (Stand: 03.08.2009, 09:17 Uhr)

11 – Open Source Geospatial Foundation

Titel: **MapServer**

URL: <http://mapserver.org/>

Abruf: 29.11.2010

12 – GeoServer

Titel: **Welcome**

URL: <http://geoserver.org/display/GEOS/Welcome>

Abruf: 29.11.2010

13 – GeoTools

Titel: **Frequently Asked Questions**

URL: <http://docs.geotools.org/stable/userguide/faq.html>

Abruf: 29.11.2010

14 – Open Geospatial Consortium (OGC)

Titel: **The OGC Announces Styled Layer Descriptor & Symbol Encoding Specifications**

URL: <http://www.opengeospatial.org/pressroom/pressreleases/761>

Abruf: 25.11.2010

15 – Deutsche Wikipedia

Titel: **Webservice**

URL: <http://de.wikipedia.org/wiki/Webservice>

Stand: 15.10.2010, 20:24 Uhr

16 – World Wide Web Consortium (W3C)

Titel: **Web Services Architecture - W3C Working Group Note 11 February 2004**

URL: <http://www.w3.org/TR/ws-arch/#introduction>

Abruf: 29.10.2010

17 – Deutsche Wikipedia

Titel: **Hypertext Transfer Protocol**

URL: http://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Stand: 05.07.2010, 12:15

18 – Deutsche Wikipedia

Titel: **Web Processing Service**

URL: http://de.wikipedia.org/wiki/Web_Processing_Service

Stand: 27.03.2010, 17:19

19 – Open Geospatial Consortium (OGC)

Titel: **OpenGIS Sensor Observation Service**

URL: <http://www.opengeospatial.org/standards/sos>

Abruf: 02.07.2010

20 – Deutsche Wikipedia

Titel: **Sensor Observation Service**
URL: http://de.wikipedia.org/wiki/Sensor_Observation_Service
Stand: 06.08.2010, 17:57

21 – Lexikon der Kartographie und Geomatik - Müller, A. in Bollmann, J., Koch, W.

Titel: **Geoinformationssystem**
Seite: 304/305
Band: Erster Band
Verlag: Spektrum Akademischer Verlag
Ort: Heidelberg
Datum: 2001.

22 – Cunningham & Cunningham, Inc.

Titel: **Model View Controll History**
URL: <http://c2.com/cgi/wiki?ModelViewControllerHistory>
Abruf: am 05.09.2010

23 – Fielding Roy Thomas, 2000

Titel: **Architectural Styles and the Design of Network-based Software Architectures**
URL: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
Stand: 2000
Abruf: 19.09.2010

24 – WISDOM Projekt-Dokumentation

Titel: **ELVIS GUI Architektur**
Dateiname: GUI_Architektur_Beschreibung.doc (Anlage: CD-ROM)
Stand: GUI_Architektur_Beschreibung.doc

25 – ui-patterns.com

Titel: **Wizard**
URL: <http://ui-patterns.com/patterns/Wizard>
Abruf: 02.08.2010

26 – Stefanakis, E et al.

Titel: **Geographic hypermedia: concepts and systems**
Verlag: Springer
Stand: 2006

27 – WISDOM Projekt-Dokumentation

Titel: **WGEF Spezifikation**
Dateiname: WISDOM_WGEF_EN_01.doc (Anlage: CD-ROM)
Version: 0.1

28 – Deutsche Wikipedia

Titel: **Secure Shell**
URL: http://de.wikipedia.org/wiki/Secure_Shell
Stand: 12.11.2010, 10:50 Uhr

29 – Deutsche Wikipedia

Titel: **Telnet**
URL: <http://de.wikipedia.org/wiki/Telnet>
Stand: 22.11.2010, 12:09 Uhr

30 – Deutsche Wikipedia

Titel: **JavaScript Object Notation**
URL: http://de.wikipedia.org/wiki/JavaScript_Object_Notation
Stand: 30.11.2010

31 – Google

Titel: **Google Web Toolkit**
URL: <http://code.google.com/webtoolkit/>
Abruf: 25.10.2010

32 – Golem.de, Klaß & Ihlenfeld Verlag GmbH

Titel: **Google Web Toolkit nun Open Source**
URL: <http://www.golem.de/0612/49434.html>
Abruf: 25.10.2010

33 – Smart GWT

Titel: **Smart GWT - GWT API's for SmartClient**
URL: <http://code.google.com/p/smartgwt/>
Abruf: 25.10.2010

34 / 35 – Smart Client

Titel: **Ajax RIA system**
URL 1: <http://www.smartclient.com/>
Abruf: 25.10.2010
URL 2: <http://www.smartclient.com/product/>
Abruf: 10.9.2010

36 – Verena Klinger (DLR), Thilo Wehrmann (DLR), Steffen Gebhardt (DLR)

Kontakt: Verena Klinger: Verena.Klinger@dlr.de
Kontakt: Thilo Wehrmann: Thilo.Wehrmann@dlr.de
Kontakt: Steffen Gebhardt: Steffen.Gebhardt@dlr.de

37 – WISDOM Projekt-Dokumentation

Titel: **Bedienungsanleitung DataEntryPortal**
Dateiname: **WISDOM_DEP_EN_01.doc** (Anlage: CD-ROM)
Version: 0.1

38 – WISDOM Projekt-Dokumentation

Titel: **WGEF Spezifikation**
Dateiname: **WISDOM_WGEF_EN_01.doc** (Anlage: CD-ROM)
Version: 0.1

39 – WISDOM Projekt-Dokumentation

Titel: Anleitung Metadaten Erstellung
Dateiname: WISDOM_MetadataGenerator_EN_01.doc (Anlage: CD-ROM)
Version: 0.1

40 / 41 / 50 / 51 – Deutsche Wikipedia

Titel: Anforderung (Informatik)
URL: http://de.wikipedia.org/wiki/Anforderung_Informatik
Stand: 31. Okt. 2010, 20:20

42 – Eclipse Foundation

Titel: Eclipse IDE
URL: <http://www.eclipse.org/>

43 – Google

Titel: Google Plugin for Eclipse
URL: <http://code.google.com/intl/de-DE/eclipse/>

44 – Tigris.org, Open Source Software Engineering Tools

Titel: Subversion
URL: <http://subversion.tigris.org/>

45 – Tigris.org, Open Source Software Engineering Tools

Titel: Subclipse
URL: <http://subclipse.tigris.org/>

46 – Deutsche Wikipedia

Titel: Universally Unique Identifier
URL: http://de.wikipedia.org/wiki/Universally_Unique_Identifier
Stand: 6. Okt. 2010, 17:04

47 – Open Geospatial Consortium (OGC)

Titel: Styled Layer Descriptor
URL: <http://www.opengeospatial.org/standards/sld>
Abruf: 7.8.2010

48 – Wolfgang Meier

Titel: Open Source Native XML Database
URL: <http://exist.sourceforge.net/>
Abruf: 05.09.2010

49 – Vnull's blog

Titel: Benchmarking PostgreSQL's INSERT/COPY data loading performance
URL: <http://jakub.wartak.pl/blog/?p=93>
Abruf: 1.11.2010

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Oberpfaffenhofen, den 7. Dezember 2010

Malte Ahrens, Matrikelnummer: 21779

7 Anhang

7.1 CD-ROM

7.1.1 Inhalt

Weitere Informationen sind auf der CD-ROM in der Datei README.txt enthalten.

- Diplomarbeit im PDF Dateiformat
- WISDOM Projektdokumentation
- Quellcode

Diplomarbeit im PDF Dateiformat

Verzeichnis: / (Hauptverzeichnis)

Dateiname: MalteAhrens_Diplomarbeit_DataEntryPortal.pdf

WISDOM Projektdokumentation (siehe Quellenverzeichnis)

Verzeichnis: /WISDOM_Projektdokumentation

Dateiname 1: WISDOM_DEP_EN_01.pdf

Dateiname 2: WISDOM_HandsOn_EN_01.pdf

Dateiname 3: WISDOM_MetadataGenerator_EN_01.pdf

Dateiname 4: WISDOM_WGEF_EN_01.pdf

Dateiname 5: GUI_Architektur_Beschreibung.doc

Dateiname 6: GUI_How_to_code_a_toolbox.pdf

Quellcode

Auf der beigefügten CD-ROM befindet sich der Quellcode der in Kapitel 4 und Kapitel 5 entwickelten Programme. Dieser ist im Verzeichnis „Quellcode“ in Eclipse-Projekten abgelegt, auf die mittels eines Dateibrowsers und Texteditors zugegriffen werden kann (oder über den Import in Eclipse [Import – Existing Projects into Workspace]).

Der Assistent kann momentan nur innerhalb des DLR-Intranets ausgeführt werden, da die zugrundeliegenden Dienste (WPS: createWgef.py, startDep.py) und Datenspeichersysteme (eXist und PostgreSQL Datenbanken) durch eine Firewall vom DLR nach außen abgeschottet sind. Im Diplom-Kolloquium wird die Bedienung der Programme vorgeführt.

Der Quellcode ist in Paketen (Ordnern) und JAVA-Dateien strukturiert:

Projektverzeichnis: /Quellcode/DataEntryPortal_Assistant_Gui/

src/de/dlr/caf/elvis/client/view/toolboxes/dep

- ControlTB.java
- DepTB.java
- DepTBController.java
- OverviewTB.java
- OverviewTBController.java
- UUID.java

src/de/dlr/caf/elvis/client/view/toolboxes/dep/event

- CancelEvent.java
- CancelRequestEvent.java
- CompleteEvent.java
- NextEvent.java
- PreviousEvent.java
- SaveEvent.java
- StartEvent.java

src/de/dlr/caf/elvis/client/view/toolboxes/dep/dataSources

- DatasetProviderDataSource.java
- DatasetThemeDataSource.java
- DepTBDataSource.java
- OverviewTBDataSource.java

src/de/dlr/caf/elvis/client/view/toolboxes/dep/requests

- DeleteDepFolderRequest.java
- GetDatasetProviderRequest.java
- GetDatasetThemeRequest.java
- PostWpsCreateWgefRequest.java
- PostWpsStartDepRequest.java
- PutMetadataRequest.java
- PutSldRequest.java

src/de/dlr/caf/elvis/client/view/toolboxes/dep/tabs

- ErrorForm.java
- ErrorFormController.java
- Tab.java
- TabController.java
- IntroductionTab.java
- IntroductionTabController.java
- MetadataTab.java
- MetadataTabController.java
- SaveTab.java
- SaveTabController.java
- SldTab.java
- SldTabController.java
- UploadTab.java
- UploadTabController.java

Projektverzeichnis: /Quellcode/DataEntryPortal_Sensor_Extension/

src/de/dlr/caf/elvis/system/dep

- DataEntryPortal.java
- DataExtractor.java
- Measurement.java
- Module.java
- DepConfigurationManager.java
- DepConfigurationSettings.java
- DEPException.java

Projektverzeichnis: /Quellcode/WPS/

- createWgef.py, startDep.py

7.2 Beispiel einer SLD Datei

```
<sld:StyledLayerDescriptor xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:sld="http://www.opengis.net/sld" version="1.0.0">
<sld:NamedLayer xmlns="http://www.opengis.net/sld">
<sld:Name>Inundation</sld:Name>
<sld:UserStyle>
  <sld:Name>Inundation depth</sld:Name>
  <sld:Title>Inundation depth</sld:Title>
```

```

<sld:Abstract />
  <sld:FeatureTypeStyle>
    <sld:Name>Inundation depth</sld:Name>
    <sld:Title>Inundation depth</sld:Title>
  </sld:FeatureTypeStyle>
</sld:Abstract />
<sld:SemanticTypeIdentifier>
  generic:geometry
</sld:SemanticTypeIdentifier>
<sld:Rule>
  <sld:Name>Inundation depth</sld:Name>
  <sld:Title>Inundation depth</sld:Title>
<sld:RasterSymbolizer>
  <! --
    Colour ramp for 8 bit
  -->
  <sld:Opacity>0.8</sld:Opacity>
  <sld:ColorMap>
    <sld:ColorMapEntry color="#FFCCE1" quantity="1" opacity="1.0" />
    <sld:ColorMapEntry color="#F7A8C5" quantity="2" opacity="1.0" />
    <sld:ColorMapEntry color="#ED82AA" quantity="4" opacity="1.0" />
    <sld:ColorMapEntry color="#E05E90" quantity="6" opacity="1.0" />
    <sld:ColorMapEntry color="#D43B78" quantity="8" opacity="1.0" />
    <sld:ColorMapEntry color="#C70063" quantity="10" opacity="1.0" />
  </sld:ColorMap>
</sld:RasterSymbolizer>
</sld:Rule>
</sld:FeatureTypeStyle>
</sld:UserStyle>
</sld:NamedLayer>
</sld:StyledLayerDescriptor>

```

7.3 Layout-Konfiguration. ElvisLayout.xml

```

<wisdom:Container>
...
<Page>
  <Title>DataEntryPortal</Title>
  <ID>page5</ID>
  <Orientation>H</Orientation>
  <PageElementContainer>
    <Orientation>H</Orientation>
    <Height>100%</Height>
    <Width>100%</Width>
    <ID>integrationContainer</ID>
    <PageElementContainer>
      <Orientation>H</Orientation>
      <Height>100%</Height>
      <Width>250</Width>
      <ID>xyz</ID>
      <Toolbox xsi:type="wisdom:Toolbox">
        <Classname>
          de.dlr.caf.elvis.client.view.toolboxes.dep.IntegrationStepsTB
        </Classname>
        <ID>integrationSteps</ID>
        <Height>100%</Height>
        <Width>100%</Width>
        <EventHandler>depTB</EventHandler>
        <EventHandler>controlTB</EventHandler>
        <EventHandler>integrationSteps</EventHandler>
        <Icon>true</Icon>
        <Minimizable>>false</Minimizable>
      </Toolbox>
    </PageElementContainer>
  </PageElementContainer>
</Page>

```

```

    <Movable>false</Movable>
    <Resizable>false</Resizable>
    <StartMinimized>false</StartMinimized>
  </Toolbox>
</PageElementContainer>
<PageElementContainer>
  <Orientation>V</Orientation>
  <Height>100%</Height>
  <Width>*</Width>
  <ID>contentTB</ID>
  <Toolbox xsi:type="wisdom:Toolbox">
    <Classname>
      de.dlr.caf.elvis.client.view.toolboxes.dep.DepTB
    </Classname>
    <ID>depTB</ID>
    <Height>*</Height>
    <Width>100%</Width>
    <DataConsumer>depTB</DataConsumer>
    <EventHandler>depTB</EventHandler>
    <EventHandler>controlTB</EventHandler>
    <EventHandler>integrationSteps</EventHandler>
    <Icon>>true</Icon>
    <Minimizable>false</Minimizable>
    <Movable>false</Movable>
    <Resizable>false</Resizable>
    <StartMinimized>false</StartMinimized>
  </Toolbox>
  <Toolbox xsi:type="wisdom:Toolbox">
    <Classname>
      de.dlr.caf.elvis.client.view.toolboxes.dep.ControlTB
    </Classname>
    <ID>controlTB</ID>
    <Height>60</Height>
    <Width>100%</Width>
    <EventHandler>depTB</EventHandler>
    <EventHandler>controlTB</EventHandler>
    <EventHandler>integrationSteps</EventHandler>
    <Icon>>true</Icon>
    <Minimizable>false</Minimizable>
    <Movable>false</Movable>
    <Resizable>false</Resizable>
    <StartMinimized>false</StartMinimized>
  </Toolbox>
</PageElementContainer>
</PageElementContainer>
</Page>
</wisdom:Container>

```

7.4 WPS – Erstellung WGEF Archiv (Python)

```
001 from pywps.Process.Process
002 import WPSProcess
003 import logging.config, os, tempfile, types
004 import urllib
005 import configuration
006
007 logging.config.fileConfig(configuration.WPS_LOGFILE,
008 disable_existing_loggers=False)
009 logger = logging.getLogger("createWgef")
010
011 class Process(WPSProcess):
012     def __init__(self):
013         WPSProcess.__init__(
014             self,
015             identifier=" createWgef", #the same as the file name
016             title="Generate WGEF package for DEP",
017             abstract="Preparing data for data integration",
018             version="1.0",
019             storeSupported="true",
020             statusSupported="true",
021             grassLocation=False
022         )
023
024         self.geodataResourceIn = self.addLiteralInput(
025             identifier="geodata",
026             title="URL to geodata content",
027             # some optional parameters
028             abstract="web folder to geodata", # default is empty
029             metadata=[{"wis": "dom"}], # default is empty
030             type=types.StringType, # default value
031             maxOccurs=1 # default maximum size
032         )
033
034         self.identifierIn = self.addLiteralInput(
035             identifier="uuid",
036             title="identifier",
037             metadata=[{"wis": "dom"}],
038             type=types.StringType,
039             maxOccurs=1
040         )
041
042         self.metadataResourceIn = self.addComplexInput(
043             identifier="metadata",
044             title="URL to metadata",
045             # some optional parameters
046             abstract="Metadata file", # default is empty
047             metadata=[{"wis": "dom"}], # default is empty
```

```

047     formats=[{"mimeType":"text/xml"}], # default value
048     maxOccurs=1,
049     maxmegabites="500" # default maximum size
050 )
051
052 self.sldResourceIn = self.addComplexInput(
053     identifier="sld",
054     title="Url to sld file",
055     # some optional parameters
056     abstract="SLD file", # default is empty
057     metadata=[{"wis":"dom"}], # default is empty
058     formats=[{"mimeType":"text/xml"}], # default value
059     maxOccurs=1,
060     maxmegabites="500" # default maximum size
061 )
062
063 self.resourceOut = self.addComplexOutput(
064     identifier="package:wgef:result",
065     title="Identified resource.",
066     formats=[{"mimeType":"text/xml"}] # default value
067 )
068
069 def execute(self):
070     percent = 0
071     uuid = self.identifierIn.getValue()
072     geodataUrl = self.geodataResourceIn.getValue()
073     localDir = "/local/wps/temp/"+uuid+"/private/"
074     geodataUrl = " http://129.247.183.248/geodata/"+uuid+"/"
075
076     downloadCmd ="wget -r --level=2 -nd -A '.xml, .shp, .shx,
077                                     .dbf, .prj, .sbn,
078                                     .sld, .tif, .txt' -P"
079
080     downloadCmd += localDir
081     downloadCmd += geodataUrl
082
083     self.cmd(downloadCmd)
084     self.cmd("mkdir geodata")
085     self.cmd("mkdir geodata/public")
086     self.cmd("touch geodata/public/.empty")
087     self.cmd("cp -R " + localDir + " / geodata/")
088     self.cmd("cp " + self.metadataResourceIn.getValue() + "
089             geodata/" + self.metadataResourceIn.getValue() + ".xml")
090     self.cmd("cp " + self.sldResourceIn.getValue() + " geodata/" +
091             self.sldResourceIn.getValue() + ".sld")
092     self.cmd("zip -r geodata.zip geodata")
093     logger.info(os.listdir("."))
094     self.resourceOut.setValue("geodata.zip")

```

7.5 WPS – Aufruf DataEntryPortal (Python)

```
001 from pywps.Process.Process
002 import WPSProcess
003 import logging.config, os, tempfile, types
004 import configuration
005
006 logging.config.fileConfig(configuration.WPS_LOGFILE,
007 disable_existing_loggers=False)
008 logger = logging.getLogger("startDep")
009
010 class Process(WPSProcess):
011     def __init__(self):
012         WPSProcess.__init__(self,
013             identifier="test_startDep", #the same as the file name
014             title="Start DataEntryPortal",
015             abstract="Data integration",
016             version="1.0",
017             storeSupported="true",
018             statusSupported="true",
019             grassLocation=False
020         )
021
022         self.inputWgetIn = self.addComplexInput(
023             identifier="dataintegration:resource",
024             title="URL to unzipped content",
025             # some optional parameters
026             abstract="C5 text file containing rules", # default is empty
027             metadata=[{"wis": "dom"}], # default is empty
028             formats=[{"mimeType": "application/zip"}], # default value
029             maxOccurs=1,
030             maxmegabites="500" # default maximum size
031         )
032
033         self.providerIn = self.addLiteralInput(
034             identifier="dataintegration:provider",
035             title="Sensor: tsx",
036             # some optional parameters
037             abstract="C5 text file containing rules", # default empty
038             metadata=[{"wis": "dom"}], # default is empty
039             type=types.IntType,
040             maxOccurs=1 # default maximum size
041         )
042
043         self.thematicIn = self.addLiteralInput(
044             identifier="dataintegration:thematic",
045             title="Sensor: tsx",
046             # some optional parameters
047             abstract="C5 text file containing rules", # default is empty
```

```

047     metadata=[{"wis":"dom"}], # default is empty
048     type=types.IntType,
049     maxOccurs=1 # default maximum size
050 )
051
052 self.resourceOut = self.addComplexOutput(
053     identifier="dataintegration:result",
054     title="Identified resource.",
055     formats=[{"mimeType":"text/plain"}])
056
057 def execute(self):
058     percent = 0
059     out = ""
060     logger.info("WGEF import into database started...")
061     self.status.set(msg="Starting ", percentDone=percent)
062     depConfiguration =
063         "http://mekongvm5/xml/wamapro/wisdom_configuration.xml"
064
065     path = configuration.WPS_EXTERNAL_FUNCTIONS_DIR + "java/"
066     inputfile = self.inputWgetIn.getValue()
067     inputfile = inputfile[inputfile.find("/") + 1:]
068
069     cmd = "echo java -jar " + path + "2010-04-19_wisdom-dep.jar"
070     cmd += " -c " + depConfiguration
071     cmd += " -g " + str(self.thematicIn.getValue())
072     cmd += " -p " + str(self.providerIn.getValue())
073     cmd += " -i " + inputfile
074
075     logger.info(os.listdir("."))
076     self.cmd("cp " + self.inputWgetIn.getValue() + " /tmp")
077     logger.info("Cmd: " + str(cmd))
078     out = self.cmd(cmd)
079     logger.info("Cmd output: " + str(out))
080     outfile = open("out.txt", "w")
081     outfile.write(out)
082
083     self.resourceOut.setValue("out.txt")

```

Softwaretest: Fehlerprotokolle - Sensormessdatenintegration

7.5.1 Fehlermeldungen Spalten vertauscht

Eingabe

```
time,observer,value,module  
'2008-07-17 23:19:41', U_B1, 0.35029736, 'wl'
```

Ausgabe

```
16:35:20,575 ERROR DataExtractor:129 - 2010/11/16 16.35.20  
File: measurements_wl_full.csv, row 1, column 1 ('time') needs to be  
'observer'. Abort.
```

```
16:35:20,575 ERROR DataExtractor:129 - 2010/11/16 16.35.20  
File: measurements.csv, row 1, column 2 ('observer') needs to be  
'time'. Abort.
```

```
16:35:20,575 ERROR SensorEntryPortal:93 - 2010/11/16 16.35.20 ABORTION  
Error in file: C:\eclipse_workspace\dep_sensor\data\measurements.csv
```

7.5.2 Fehlende Spalte "value"

Eingabe

```
U_B1, '2008-07-17 23:19:41', 'wl'
```

Ausgabe

```
16:22:15,178 ERROR DataExtractor:118 - 2010/11/16 16.22.15  
File: measurements.csv, row 32. Column missing. Skip this measurement.
```

7.5.3 Falsches Datumsformat

Eingabe

```
U_B1, '2008-07-28',0.48647872, 'wl'
```

Ausgabe

```
16:23:21,805 ERROR DataExtractor:81 - 2010/11/16 16.23.21  
File: measurements.csv, row 32, column 2 (time).  
Necessary format: yyyy-MM-dd HH:mm:ss.f
```

7.5.4 Falsche Sensor-Referenz

Eingabe

```
U_B, '2008-07-17 23:19:41',0.48647872, 'wl'
```

Ausgabe

```
FEHLER: Einfügen oder Aktualisieren in Tabelle  
»o9352c9d2b03c40b58e48d9cace7158c0« verletzt Fremdschlüssel-Constraint  
»fk11_observer«
```


7.5.5 Falsche Modul-Referenz

Eingabe

```
UU_B1, '2008-07-28 10:49:26',0.48647872, 'w'
```

Ausgabe

```
16:31:14,655 ERROR No table matches the referenced module in csv  
file... org.springframework.dao.EmptyResultDataAccessException: Incor-  
rect result size: expected 1, actual 0
```

7.5.6 Leerer Spaltenwert

Eingabe

```
--- U_B1, '2008-07-17 23:19:41',, 'w1'---
```

Ausgabe

```
16:26:03,731 ERROR DataExtractor:90 - 2010/11/16 16.26.03  
Error in file measurements.csv - line 311, column 2 ('value'): empty  
String. Field must contain a float value. Skip.
```