Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät für Geomatik
Studiengang Kartographie und Geomatik

Diplomarbeit

Naturereignisinformationssystem

Entwicklung eines WebGIS auf Open-Source-Basis zur Dokumentation von Naturereignissen in Schutzgebieten

eingereicht von
Friedjoff Trautwein

betreut von
Prof. Dr. Detlef Günther-Diringer
(Hochschule Karlsruhe)

Prof. Dr. Robert Weibel
(Universität Zürich)

Dipl. Geogr. Ronald Schmidt
(Wildnispark Zürich)

Dipl. Geogr. Ruedi Haller
(Schweizerische Nationalpark)

Zürich, den 21. September 2009

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

Diplomarbeit

für Herrn Friedjoff Trautwein

Thema: Naturereignisinformationssystem - Entwicklung eines WebGIS auf Open-Source-Basis zur Dokumentation von Naturereignissen in Schutzgebieten

In Schutzgebieten - wie beispielsweise im Schweizerischen Nationalpark oder im Wildnispark Zürich - die weitgehend dem Lauf der Natur überlassen werden, kommt es fortlaufend zu verschiedensten Naturereignissen. Mögliche Naturereignisse sind vielfältig und können von Steinschlag oder Murgängen über Lawinen bis hin zu Überschwemmungen oder Windbruch reichen. Allgemein ist es für Schutzgebiete sowohl aus wissenschaftlicher Sicht als auch aus Sicht des Managements interessant, diese Ereignisse zu dokumentieren. Bisher steht allerdings noch kein offenes System zur effizienten Erfassung und Nutzung von Daten über Naturereignisse in Schutzgebieten zur Verfügung.

Die Problemstellungen bei der Dokumentation von Naturereignissen beginnen bei einer geeigneten Erfassungsmethode von Ereignissen. Diese sollte einerseits schnell und einfach sein, andererseits aber auch eine vollständige und korrekte Dokumentation der räumlichen und zeitlichen Dimension des Ereignisses gewährleisten. Eine weitere Herausforderung ist die dauerhafte Dokumentation der gesammelten Informationen und die Bereitstellung von Funktionen für eine sinnvolle und nachhaltige Nutzung der Daten.

Aufbauend auf der Masterarbeit „Naturereignisdokumentation“ von Jonas Büchel am Geographischen Institut der Universität Zürich (2009) soll in Zusammenarbeit mit dem Schweizerischen Nationalpark und dem Wildnispark Zürich ein WebGIS auf Open-Source-Basis, zur Dokumentation von Naturereignissen in Schutzgebieten entwickelt werden.

Zentrale Aufgabenteile sind:
- Anforderungserhebung mit Hilfe des Schweizerischen Nationalpark und dem Wildnispark Zürich
- Untersuchung und Vergleich von möglichen Softwarepaketen für die Implementierung
- Entwurf der Datenhaltung, des Fachkonzepts und der graphischen Oberfläche
- Implementierung des entworfenen Systems
- Evaluation des Systems mit Hilfe des Schweizerischen Nationalpark und dem Wildnispark Zürich

Bearbeitungszeit: 4 Monate

Ausgabedatum: 22. Mai 2009

Abgabetermin: 21. September 2009

Prof. Dr. Detlef Günther-Diringer
Betreuer Hochschule Karlsruhe, 1. Prüfer

Prof. Dr. Robert Weibel
Betreuer Universität Zürich, 2. Prüfer

Erklärung

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.

Zürich, den 21. September 2009

Friedjoff Trautwein

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
BLOB	Binary Large Object
BSD	Berkeley Software Distribution
CSS	Cascading Style Sheets
DBMS	Datenbankmanagementsystem
DBS	Datenbanksystem
EXIF	Exchangeable Image File Format
GIS	Geoinformationssystem
GML	Geography Markup Language
GPL	GNU General Public License
GPS	Global Positioning System
GSP	Groovy Server Pages
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
LGPL	GNU Lesser General Public License
MVC	Model, View, Controller
OGC	Open Geospatial Consortium
OOA	objektorientierte Analyse
OOP	objektorientierte Programmierung
PDF	Portable Document Format
ORM	Object-relational Mapping
REST	Representational State Transfer
SNP	Schweizerischer Nationalpark
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TCP/IP	Transmission Control Protocol / Internet Protocol
TMS	Tile Map Service
WFS	Web Feature Service

WGS 84	World Geodetic System 1984
WKT	Well Known Text
WKB	Well Known Binary
WMS	Web Map Service
WPZ	Wildnispark Zürich
XML	Extensible Markup Language

1 Einleitung

Nach einer Beschreibung der Motivation werden in **Kapitel 1.2** die zentralen Ziele der Arbeit formuliert. Anschließend wird in **Kapitel 1.3** ein genereller Überblick zum Aufbau der Arbeit gegeben, bevor in **Kapitel 1.4** die Abgrenzung - insbesondere zu der Masterarbeit von Jonas Büchel - erläutert wird.

1.1 Motivation

Naturereignisse spielen in Form von Naturgefahren schon seit Anbeginn der Menschheit eine wichtige Rolle für das menschliche Leben. Zunächst stand das Naturereignis als Gefahr im Mittelpunkt des wissenschaftlichen Interesses und ist auch weiterhin ein wichtiger Forschungsschwerpunkt. Durch ein intensiveres Verständnis der Prozesse von Naturereignissen möchte man Schutzmaßnahmen, Vorhersagen und Raumplanung in den betroffenen Gebieten verbessern. Die flächendeckende Dokumentation der Ereignisse bildet dabei die Grundlage zur Erstellung von Gefahrenkarten sowie -analysen und ist gerade in alpinen Regionen schon weitestgehend eingeführt.

Mit der Etablierung von Nationalparks und Naturschutzgebieten hat sich allerdings die Sichtweise auf Naturereignisse wesentlich erweitert und berücksichtigt schon länger nicht mehr nur Ereignisse mit direkter Gefahr für den Menschen. Auch für das Verständnis von Prozessen in der Natur stellt die Dokumentation von Naturereignissen eine wichtige Grundlage dar.

In Schutzgebieten - wie beispielsweise im Schweizerischen Nationalpark oder im Wildnispark Zürich - die weitgehend dem Lauf der Natur überlassen werden, kommt es fortlaufend zu verschiedensten Naturereignissen. Mögliche Naturereignisse sind vielfältig und können von Steinschlag oder Murgängen über Lawinen bis hin zu Überschwemmungen oder Windbruch reichen.

Allgemein ist es für Schutzgebiete sowohl aus wissenschaftlicher Sicht als auch aus Sicht des Managements interessant, diese Ereignisse zu dokumentieren. Während der Schweizerische Nationalpark schon seit 1987 solche Ereignisse schriftlich und fotografisch dokumentiert, wurden im Wildnispark Zürich Naturereignisse bisher nicht dokumentiert.

Bislang fehlt es allerdings auch an einem geeigneten Softwaresystem zur effizienten Dokumentation von Ereignissen in Schutzgebieten. An diesem Punkt setzt die vorliegende Arbeit an, mit dem Ziel, eine offene sowie einfach zu bedienende Anwendung zur Erfassung, Speicherung, Anzeige und Verteilung von Informationen über Naturereignisse zu entwickeln.

Die Probleme bei der Dokumentation von Naturereignissen beginnen bei einer geeigneten Erfassungsmethode von Ereignissen. Diese sollte einerseits schnell und einfach sein, andererseits aber auch eine vollständige und korrekte Dokumentation des Ereignisses gewährleisten. Ein weiteres Problem ist die dauerhafte Dokumentation der gesammelten Informationen und die Bereitstellung von Funktionen für eine sinnvolle und nachhaltige Nutzung der Daten. Das zu entwickelnde Naturereignisinformationssystem versucht eine Lösung für diese Probleme zu bieten.

Die oben genannten Motive und nachfolgende Fragestellungen stehen im Rahmen dieser Diplomarbeit im Mittelpunkt:

- Welche Schnittstellen soll das Informationssystem zwischen Benutzer und Datenbank zur Verfügung stellen?
- Wie kann die Dateneingabe implementiert werden, um eine einfach und schnelle, aber auch korrekte und umfassende Erfassung von Naturereignissen zu ermöglichen?
- Wie kann eine möglichst einfache, informative und vielseitige Ausgabe der Daten erreicht werden?
- Wie können die eingegebenen Daten innerhalb eines Plausibilitätstests validiert werden?
- Welche Systemarchitektur und Softwareumgebung eignet sich am besten für die geforderten Rahmenbedingungen?

1.2 Zielsetzungen

Als Ergebnis dieser Diplomarbeit wird ein Konzept und eine möglichst komplette Implementierung eines Informationssystem zur Erfassung, Speicherung, Verbreitung und Anzeige von Informationen über Naturereignissen in Schutzgebieten angestrebt. Die Datenbank, auf der das Informationssystem aufbaut, soll das Datenmodell aus der Masterarbeit von Jonas Büchel (2009) implementieren. Insgesamt soll das System möglichst offen, flexibel und erweiterbar sein, um eine vielseitige Nutzung zu ermöglichen. Mit Hilfe des Schweizerischen Nationalparks und des Wildnisparks Zürich soll die praktikable Nutzung des Systems überprüft werden.

Im Bereich der Benutzerschnittstellen hat ein *Webinterface* mit Funktionen zum Erstellen, Anzeigen und Aktualisieren von Naturereignissen höchste Priorität. Dieses Benutzeroberfläche soll darüber hinaus Abfragefunktionen zur thematischen, räumlichen und zeitlichen Eingrenzung von Naturereignissen anbieten. Außerdem wird eine Schnittstelle zu GIS-Software wie ArcMap angestrebt, um auch GIS-Experten Zugriff auf die Datenbank zu ermöglichen. Geringere Priorität hat die Integration von mobilen Endgeräten, welche aber zumindestens bei der Konzeption der Systemarchitektur berücksichtigt werden sollen.

1.3 Aufbau der Arbeit

In einem ersten Schritt sollen zunächst die Voraussetzungen und Anforderungen für das Naturereignisinformationssystem ermittelt werden. Im Rahmen einer Anforderungsanalyse werden vom Schweizerischen Nationalpark und vom Wildnispark Zürich die potentiellen Nutzer des Systems zu den Erfordernissen befragt. Die Ergebnisse der Befragungen sollen anschließend ausgewertet werden und bilden die Rahmenbedingungen für das Naturereignisinformationssystem.

Basierend auf dem relationalen Datenbankmodell von Jonas Büchel (2009) sowie den Ergebnissen der Befragungen, soll eine Analyse mit dem Ziel eines objektorientierten Datenmodells erstellt werden. Außerdem soll der Ablauf und die Struktur von zentralen Funktionen der Anwendung mit Hilfe der Analyse ermittelt werden.

Aufbauend auf den Ergebnissen der Anforderungsanalyse und dem objektorientierten Datenmodell sollen unterschiedliche Softwareumgebungen zur Realisierung des Systems untersucht werden. Anschließend soll mit derjenigen Softwarelösung, welche sich als geeignetste erweist, eine Anwendung entwickelt werden. Die Anwendung soll im Rahmen einer Evaluation von den zukünftigen Nutzern getestet und bewertet werden.

1.4 Abgrenzung

Diese Arbeit baut bewusst auf den Ergebnissen der Masterarbeit von Jonas Büchel, die als grüne Elemente in Abbildung 1.1 dargestellt sind, auf und kann in diesem Sinne als Fortsetzung seiner Arbeit gesehen werden. Insbesondere werden die Klassifizierung von Naturereignissen und das Datenbankmodell übernommen und gegebenenfalls dem System angepasst. Abhängig vom Aufwand wird sich die Implementierung eventuell nur auf eine Untermenge der definierten Ereignisse beschränken.

Außerdem konzentriert sich die Arbeit - wie der Titel schon andeutet - auf die Dokumentation von Naturereignissen in Schutzgebieten, d.h. Naturereignisse werden unabhängig von ihrer Gefahr für den Menschen betrachtet und das System wird in erster Linie für eine kleinräumige Region sowie für einen eingeschränkten Benutzerkreis konzipiert.

Zur Realisierung der Anwendung, deren Bestandteile als blaue Elemente in Abbildung 1.1 visualisiert sind, soll auf Open Source Projekte aufgebaut werden, sodass eine möglichst freie Nutzung der Anwendung ermöglicht wird.

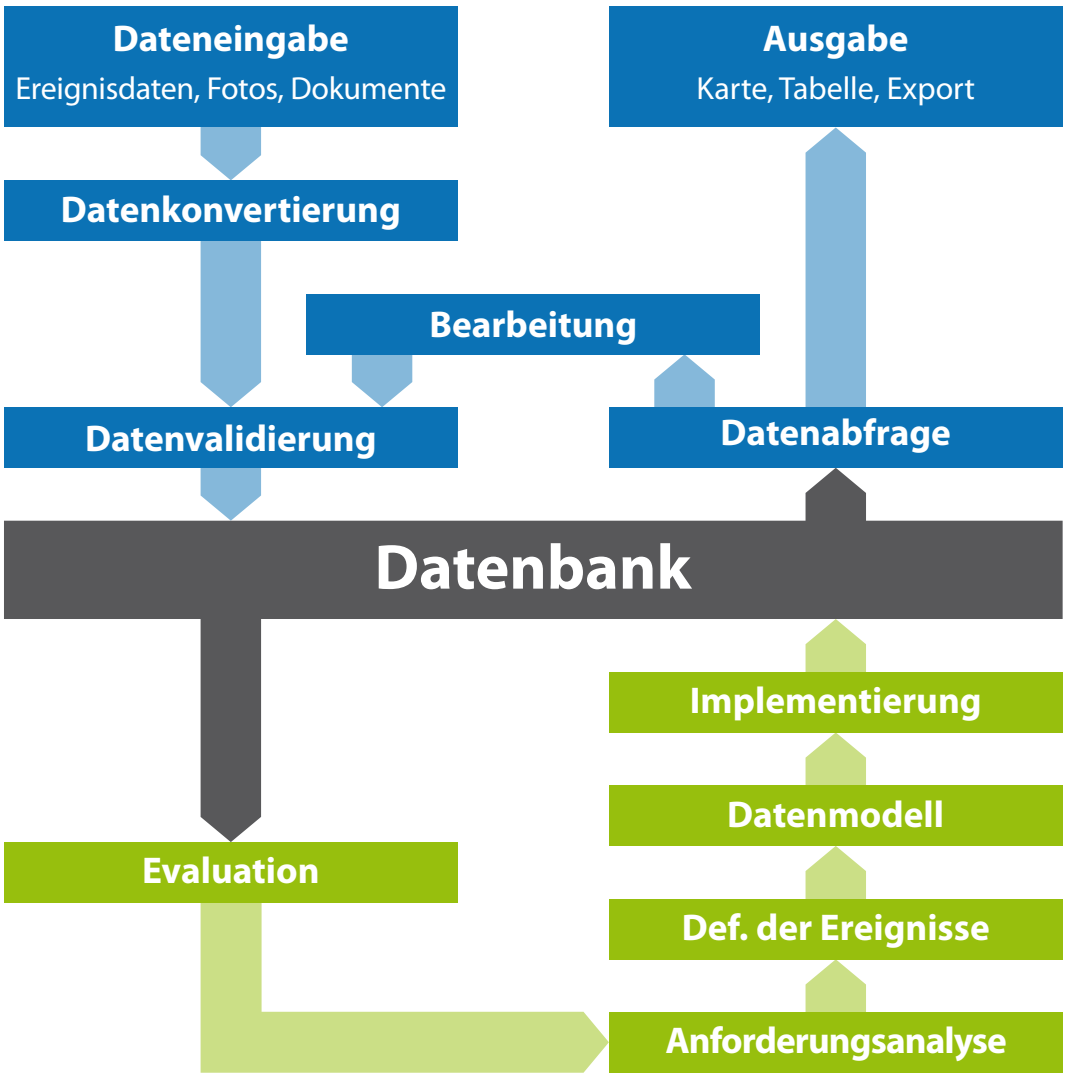


Abbildung 1.1: Gesamtkonzept zum Projekt Naturereignisdokumentation

2 Grundlagen

Zunächst wird in **Kapitel 2.1** ein Überblick über den aktuellen Forschungsstand und die Ergebnisse der Masterarbeit von Jonas Büchel gegeben. Im darauf folgenden Kapitel werden Anwendungsgebiete im Allgemeinen und der Schweizerische Nationalpark sowie der Wildnispark Zürich im Speziellen beschrieben. Anschließend wird in **Kapitel 2.3** auf technische Grundlagen, die für die Entwicklung von WebGIS-Anwendungen von Bedeutung sind, eingegangen; gefolgt von einer Zusammenfassung wichtiger Vorgehensmodelle, Analysemethoden und Softwarearchitekturmuster, die im weiteren Verlauf der Arbeit eine Rolle spielen.

2.1 Forschungsstand

2.1.1 Dokumentation von Naturereignissen

In der Vergangenheit wurde die Dokumentation von Naturereignissen vorrangig im Hinblick auf Naturgefahren entwickelt und betrieben. Anstoß für diese Entwicklung war unter anderem die *Dekade zur Reduzierung von Naturkatastrophen* (IDNDR) in den 1990er Jahren von den *Vereinten Nationen*. Ziel war die Reduzierung der sozialen, ökologischen und ökonomischen Schäden und Folgen von Naturkatastrophen. Als Grundlage für Frühwarnsysteme sowie Pläne zur Katastrophenvorbeugung und zum Katastrophenschutz wurde die Erfassung aller Gefährdungen durch natürliche Extremereignisse angestrebt. Inzwischen werden die Ziele im Rahmen der *Internationalen Strategie zur Reduzierung von Katastrophen* (ISDR) weiterverfolgt.

Die erwähnten Initiativen hatten zur Folge, dass auch der Dokumentation von Naturereignissen eine zunehmend wichtigere Bedeutung zukam. Verschiedene Projekte wie DOMODIS, DIS-ALP, PLANALP und StorMe haben sich in den letzten Jahren mit diesem Thema auseinander gesetzt und sollen im weiteren Verlauf kurz vorgestellt.

Das Projekt *Documentation of Mountain Disasters* (DOMODIS) wurde von Hans Kienholz (Universität Bern) mit dem Ziel initiiert, Richtlinien für eine standardisierte Unwetterdokumentation und entsprechende organisatorische Strukturen zu entwickeln. Beteiligt an dem Projekt waren der *International Council for Science, Committee on Disaster Reduction* (ICSU-CDR), die *International Association of Geomorphologists* (IAG) und die *Internationale Forschungsgesellschaft Interpraevent*. Auf insgesamt vier Workshops in den Jahren 1998 bis 2000 wurden von den Teilnehmern aus Alpenländern und weiteren Bergregionen Grundsätze für die Unwetterdokumentation erarbeitet, die 2002 in einer englischen und 2006 auch in einer deutschen Fassung veröffentlicht wurden (Hübl et al., 2006b). Inhalt dieser Veröffentlichung sind Hinweise und Empfehlungen für eine sinnvolle Dokumentation von Naturereignissen sowie konkrete Hilfsmittel wie Checklisten, Formulare und ein Vorschlag zur kartographischen Darstellung von Naturereignissen. Damit bildet das Projekt eine wichtige Informationsgrundlage für das Naturereignisinformationssystem.

Die Ergebnisse von DOMODIS wurden in verschiedenen nationalen und regionalen Projekten umgesetzt. In Bayern wurde in den Jahren 2004 bis 2006 an der *historische Analyse von Naturgefahren* (HANG) gearbeitet, mit dem Ziel Naturereignisse zwischen der Gegenwart und der möglichst weit zurückliegenden Vergangenheit zu dokumentieren (Becht et al., 2006). In Österreich ist seit 2005 eine digitale Ereignisdatenbank (WLK) verfügbar, die über eine Web-Plattform auch externen Personen die Dokumentation von Naturereignissen ermöglicht (DIS-ALP, 2007). In diesem Zusammenhang ist auch das Projekt *StorMe* aus der Schweiz zu nennen (siehe unten). In Südtirol werden Informationen abhängig vom Naturereignistyp entweder in der Datenbank *ED30*, in einem Murenkataster (IFFI) oder in einem Lawinenregister gesammelt (DIS-ALP, 2007).

Das Projekt *Disaster and Information Systems of Alpine Regions* (DIS-ALP) hatte zum Ziel, basierend auf den genannten regionalen und nationalen Projekten, die Dokumentation von Naturereignissen zu verbessern und eine Implementierung als WebGIS zu entwickeln (DIS-ALP, 2007). Entstanden ist das Projekt im Rahmen von *INTERREG III B*, einer Initiative der europäischen Gemeinschaft zur

Stärkung der Zusammenarbeit im Alpenraum. Forschungsschwerpunkte waren eine vereinheitlichte Methodologie, die Entwicklung von Hilfsmitteln zur Dokumentation und die Implementierung einer WebGIS-Plattform für die nachhaltige Nutzung der Informationen.

Ein wichtiges Ergebnis von DIS-ALP war die *Feldanleitung zur Dokumentation von Naturereignissen*, welche durch die *Plattform Naturgefahren der Alpenkonvention* (PLANALP) veröffentlicht wurde. Ziel der Publikation war eine einheitliche Grundlage für die Spurensicherung von abgelaufenen Prozessen im Feld, sowohl für die Ausbildung als auch für Referenzzwecke. Die verschiedenen Ereignistypen wurden kurz erläutert und durch zahlreiche Fotos, Grafiken und Dokumentationsvorschläge ergänzt (PLANALP, 2006).

In der Schweiz wurde schon 1991 durch das *Bundesgesetz über den Wald* und das *Bundesgesetz über den Wasserbau* eine gesetzliche Grundlage geschaffen, welche die Kantone zur Dokumentation von Naturereignissen verpflichtet. Die gesammelten Informationen sollen im Rahmen eines Gefahreninformationssystems bei der Identifizierung von potentiellen Gefahrenbereichen und der Abschätzung der Wiederkehrdauer von Naturgefahren helfen. Für diese Aufgabe wird vom *Bundesamt für Umwelt* (BAFU) die webbasierte Datenbank *StorMe* den kantonalen Naturgefahrenfachstellen zur Verfügung gestellt (BAFU, 2008).

Auch in Österreich ist die Dokumentation von Naturereignissen inzwischen etabliert. So wird beispielsweise ein zertifizierter Universitätslehrgang in Kooperation mit der *Universität für Bodenkultur Wien*[1], dem *Bundesforschungs- und Ausbildungszentrum für Wald, Naturgefahren und Landschaft*[2] und der *Wildbach- und Lawinenverbauung*[3] zum *Ereignisdokumentar* angeboten, der auf den Projekten DIS-ALP und DOMODIS aufbaut (FORSTnet, 2009). Insgesamt scheint vor allem in alpinen Regionen die Dokumentation von Naturereignissen eine zunehmend wichtigere Rolle zu spielen, gerade im Hinblick auf Naturgefahren.

Im Gegensatz zu den bislang genannten Projekten und Initiativen, sind in Schutzgebieten Naturereignisse unabhängig von ihrer Gefahr für den Menschen und die Infrastruktur von Interesse. Als Schutzgebiet hat der *Schweizerische Nationalpark* (SNP) eine Vorreiterrolle im Bereich der Dokumentation von Naturereignissen eingenommen; schon seit 1987 werden von Parkwächtern Ereignisse dokumentiert und seit 2004 auch digital erfasst (siehe Kapitel 2.2.2). In anderen Schutzgebieten in der Schweiz werden bisher nur die von den Behörden vorgeschriebenen Ereignistypen aufgenommen. Auf europäischer Ebene wird im französischen *Park National des Ecrins* an einem Projekt zur Ereignisdokumentation gearbeitet, während im *Nationalpark Hohe Tauern* in Österreich ein solches Projekt noch in Planung ist. Der *Nationalpark Berchtesgaden* in Deutschland führt langfristige Umweltbeobachtungen durch, beispielsweise zum Thema Borkenkäfer. In amerikanischen Nationalparks wird die Naturereignisdokumentation durch das Beobachtungsprogramm des *National Park Service* abgedeckt (Büchel, 2009).

Insgesamt wird deutlich, dass die Dokumentation von Naturereignissen erst seit wenigen Jahren eine weite Verbreitung gefunden hat und in Schutzgebieten vielerorts noch nicht aktiv betrieben wird. Obwohl es verschiedene Initiativen zur Entwicklung von standardisierten Verfahren und Erfassungssystem gab, müssen sich diese noch in der Praxis etablieren.

1 http://www.boku.ac.at/ [Abruf: 2.9.2009]

2 http://bfw.ac.at/ [Abruf: 2.9.2009]

3 http://www.die-wildbach.at/ [Abruf: 2.9.2009]

2.1.2 Naturereignisse

Dieses Kapitel setzt sich mit einer Definition des Begriffes Naturereignis und einer Klassifizierung der verschiedenen Arten von Naturereignissen auseinander. Als Grundlage dient dabei in erster Linie die Masterarbeit von Jonas Büchel, welche bei weiterem Informationsbedarf hinzugezogen werden sollte (Büchel, 2009).

Der Begriff *Naturereignis* setzt sich aus den Wörtern *Natur* und *Ereignis* zusammen. Beide Wörter lassen sich getrennt definieren und erklären zusammen das Konzept von Naturereignissen.

Der Begriff *Natur* umfasst ursprünglich alle Objekte, aus denen die Welt besteht, hat sich inzwischen aber auf Einzelbegriffe wie Landschaft und Umwelt aufgeteilt. In der Regel werden der Mensch und vom Menschen erschaffene Objekte nicht miteinbezogen (Leser, 2001).

Der Ausdruck *Ereignis* in Bezug auf die Natur wurde im Rahmen des Projektes DIS-ALP (siehe Kapitel 2.1.2) prägnant beschrieben. Danach ist ein Ereignis „die Summe der Wirkungen von einem oder mehreren Prozessen, die in räumlichem und zeitlichem Zusammenhang stehen“ (Hübl et al., 2006b, S. 4). Des Weiteren hat ein Ereignis nach dieser Definition einen klar definierten Zeitraum mit einem auslösenden Prozess als Beginn und der Unterschreitung des für den jeweiligen Prozess üblichen Schwellenwertes aller beteiligter Prozesse als Ende. Teilweise ist die Unterscheidung zwischen Prozess und Ergebnis problematisch, daher hat Jonas Büchel in seiner Masterarbeit den Begriff Ereignis im Sinne einer morphologischen Veränderung verwendet.

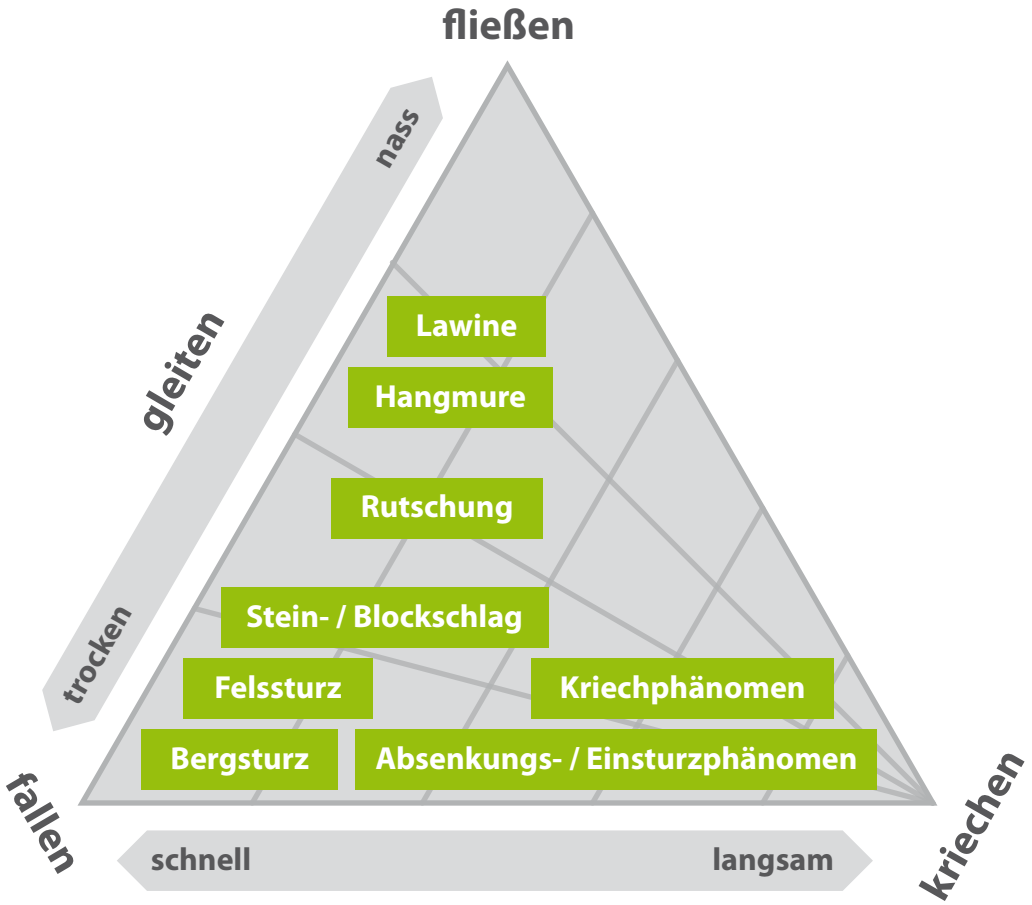


Abbildung 2.1: Klassifikation von Massenbewegungstypen nach Büchel (2009)

Ein Naturereignis wird zu einer Naturgefahr, sobald der Mensch oder vom Menschen geschaffene Objekte von einem Naturereignis betroffen sind. Dieser Gesichtspunkt bleibt im Rahmen der Arbeit allerdings unberücksichtigt, da der Mensch in Schutzgebieten keine zentrale Rolle einnimmt. Außerdem stehen in diesem Bereich inzwischen Systeme und Ergebnisse aus abgeschlossenen Projekten zur Verfügung (siehe Kapitel 2.1.2).

Die folgende Klassifizierung wurde aus Büchel (2009) übernommen und mit einer kurzen Zusammenfassung für jedes Ereignis ergänzt. Zu beachten ist, dass sich diese Klassifizierung auf bewaldete und alpine Regionen beschränkt und daher keinen Anspruch auf Vollständigkeit hat. Das zu entwickelnde System wird sich zunächst auf diese Klassifizierung beschränken, prinzipiell aber um neue Ereignistypen erweiterbar sein.

Massenbewegungen

Naturereignisse aus der Klasse *Massenbewegungen* sind in alpinen Regionen besonders häufig und offensichtlich. Nach Leser (2001, S. 496) werden unter diesem Begriff „alle Bewegungen von gleitendem, rutschendem und stürzendem Boden-, Hangschutt- und Gesteinsmaterial unter ausschließlichem Einfluss der Schwerkraft“ zusammengefasst. Wie die Ereignisse in dieser Klasse im Zusammenhang stehen, wird in Abbildung 2.1 veranschaulicht.

Rutschung: Auch bekannt als *Hangrutschung*, bezeichnet die hangabwärts gerichtete Bewegung von Lockergesteinen oder Böden auf einer Gleitfläche unter Einfluss der Gravitation (Leser, 2001).

Hangmure: Schnelles Abfahren von Feststoffen wie Gestein, Boden oder Vegetation zusammen mit Wasser ohne Vorhandensein oder Ausbildung einer Gleitfläche (BUWAL, 1998).

Kriechphänomen: Langsame, kriechende Bewegungen von Erd- oder Lockergesteinsmassen. Weitere Bezeichnungen sind *Hangkriechen*, *Erdkriechen*, *Schuttkriechen* und *Bodenkriechen* (Baumhauer, 2006).

Absenkungs- und Einsturzphänomene: Abwärtsbewegungen der Erdoberfläche, entweder langsam und kontinuierlich (Absenkung) oder schnell und abrupt (Einsturz), können infolge der Auslaugung eines löslichen Untergrundes oder aufgrund unterirdischer Hohlräume auftreten (BUWAL, 1997).

Stein- und Blockschlag, Fels- und Bergsturz: Prozesse der Bewegungsart Fallen oder Stürzen werden unter diesem Ereignistyp zusammengefasst. *Steinschlag* (ø < 50 cm) und *Blockschlag* (ø > 50 cm) treten in erster Linie als Folgeerscheinung von Verwitterung auf. Bei *Felstürzen* (Volumen < 1 Mio. m^3, Geschwindigkeit < 40 m/s) und *Bergstürzen* (Volumen > 1 Mio. m^3, Geschwindigkeit > 40 m/s) löst sich eine größere Gesteinsmasse aus einem Bergmassiv und stürzt ab (BUWAL, 1997). In der Regel ist das Volumen für die Einordnung ausschlaggebend, da die Geschwindigkeit nur schwer rekonstruiert werden kann.

Lawine: Plötzliche und schnelle hangabwärts gerichtete Bewegung von Schnee und Eis. Weiterhin können Feststoffe wie Gestein oder Vegetation involviert sein. Das Ereignis kann als gleitende, fließende oder rollende Masse oder als aufgewirbelte Schneewolke an Hängen und Wänden mit einer Sturtzbahn von über 50 Meter Länge auftreten (BUWAL, 1998).

Wasserereignisse

Naturereignisse, bei denen fluviale Prozesse im Vordergrund stehen, werden in der Klasse *Wasserereignisse* zusammengefasst.

Überschwemmung: Wird hervorgerufen durch den erhöhten Abfluss von Wasser, entweder aufgrund von anhaltendem Niederschlag oder aufgrund von Schneeschmelze, und hat zur Folge, dass das Wasser aus seinem natürlichen oder künstlichen Bett austritt. Wenn das Wasser in flachem Gelände langsam über die Ufer tritt, spricht man von einer *statischen Überschwemmung*, während von einer *dynamischen Überschwemmung* die Rede ist, wenn das Wasser in geneigtem Gelände und bei hoher Fließgeschwindigkeit ausufert. Die Ablagerung von Feststoffen außerhalb des Wasserbettes wird *Übersarung* genannt und kann Ausgangspunkt eines Murgangs sein.

Murgang: Auch bekannt als *Mure*, *Rüfe*, *Schlamm-* und *Schuttstrom*, bezeichnet ein „schnell fließendes Gemisch aus Wasser und einem hohen Anteil an Feststoffen (Steine, Blöcke, Geröll oder Holz)“ (WSL, 2006, S. 1). Teilweise wird aufgrund der Korngröße von den Feststoffen zwischen *Schlammstrom*, *Murgang* und *Schuttstrom* unterschieden. Nach Ahnert (2003) sind für das Auftreten von Murgängen eine umfangreiche Akkumulation von Feststoffen, seltene und stoßweise Zufuhr von Wasser sowie ein großes Gefälle Voraussetzung. Obwohl es sich um eine Übergangsform zwischen Massenbewegung und Wasserereignis handelt, wurde dieser Ereignistyp von Jonas Büchel den Wasserereignissen zugeordnet, da er viele Gemeinsamkeiten mit dynamischen Überschwemmung aufweist.

Seitenerosion: Das Nachbrechen der Uferböschung wird als *Seitenerosion* bezeichnet und ist bei Wildbächen und Gebirgsflüssen besonders ausgeprägt. Im flachen Gelände sind exponierte Stellen wie Prallhänge, Engstellen und Hindernisse im Abflussbereich betroffen.

Quelle: Orte, wo der Grundwasserspiegel oberhalb der Erdoberfläche liegt und folglich Wasser austritt. Eine Quelle kann versiegen, wenn sich der Grundwasserspiegel absenkt.

Forstschädlinge

Ereignisse im Bereich der *Forstschädlinge* treten auf, wenn die Populationsgröße von Insekten, Nagetieren oder Wild ihr natürliches Gleichgewicht übersteigt. Als Gründe für ein Ungleichgewicht kommen Veränderungen der klimatischen Bedingungen, des Räuber-Beute-Verhältnisses oder anthropogene Faktoren in Frage.

Insekten: Sind die größte Schädlingsgruppe und haben eine entscheidende Bedeutung für die Entwicklung eines Waldes. Obwohl Beeinträchtigungen und Zusammenbrüche von Waldbeständen zur natürlichen Walddynamik gehören, sind diese in einem Schutz- oder Wirtschaftswald ab einem gewissen Ausmaß unerwünscht und als Naturereignisse zu betrachten. Zu den schädlichen Insektenarten gehören *Borken-*, *Rüssel-* und *Bockkäfer* aus der Insektenordnungen *Käfer*, *Wickler* und *Holzbohrer* aus der Insektenordnung *Schmetterlinge* sowie verschiedene Wespen- und Läusearten aus den Insektenordnungen *Hautflügler* und *Pflanzensauger*. Für die Bestimmung der Arten wird von Jonas Büchel die Verwendung eines Praxishandbuches oder die Einholung einer Expertenmeinung empfohlen.

Nagetiere: Zu den Nagetieren, die als Forstschädlinge auftreten, gehören insbesondere Eichhörnchen, Siebenschläfer sowie Mäuse aus den Familien Langschwanzmäuse und Wühlmäuse. Abhängig von der Nagerart und der Jahreszeit kommen als Nahrungsgrundlage Knospen, Blätter, junge Triebe, Samen, Keimlinge, Rinde und Wurzeln von Kräutern, Sträuchern und Bäumen in Frage. Für die Artenbestimmung wird ebenfalls ein Praxishandbuch oder eine Expertenmeinung empfohlen.

Wild: Aus der Klasse *Wild* tritt als Forstschädling an erster Stelle Rotwild, darüber hinaus aber auch Damm-, Reh-, Schwarz-, Stein- und Gamswild auf. Diese Tiere hinterlassen durch Abnagen, Abreißen und Aufäsen von Rindenteilen, auch bekannt als Schälen, sowie Verbiss an Waldfrüchten, Knospen und Trieben erhebliche Spuren im Wald. Die durch das Schälen verursachten Stammverletzungen steigern die Wahrscheinlichkeit eines Befalls von Pilzarten, welche wiederum die Wahrscheinlichkeit von Schneebruch und Windwurf erhöhen. Ähnliche Spuren wie das Schälen, aber von wesentlich geringerem Ausmaß, hinterlässt das Fegen (Entfernung des Bastes vom Geweih) und Schlagen (Markier- und Imponierverhalten) von männlichen Hirschen. Obwohl verschiedenste Spuren bei der Identifizierung der Wildart helfen, bleibt die Unterscheidung anhand von Verbiss- und Schälspuren schwierig.

Vegetationsschädigung

Ereignisse, welche eine Veränderung der Vegetation zur Folge haben, werden in der Klasse *Vegetationsschädigung* zusammengefasst. Ausschlaggebend sind extreme Klimabedingungen, die erheblichen Einfluss auf die Vegetation ausüben können.

Brand: Abhängig vom Brennmaterial, der Temperatur, der Luftfeuchtigkeit und dem Wind hat ein *Brand* unterschiedliche Auswirkungen auf die Vegetation. So kann ein langsamer Brandherd beispielsweise zum Vegetationstod führen. Als Ursache ist in erster Linie fahrlässige Brandstiftung zu nennen, aber auch vorsätzliche Brandstiftung und natürliche Ursachen wie Blitzschlag können Auslöser sein.

Dürre: Dürreperioden bringen unterschiedliche Veränderungen in der Vegetation mit sich. Junge Pflanzen sterben ab, Bäume verlieren ihre Blätter, die Fruchtbildung ist vermindert, Wuchsstörungen und erhöhte Empfindlichkeit gegenüber Krankheiten treten auf.

Frost: Als Folge von *Frost* können in der Vegetation Frostrisse und der Kältetod, als schwerwiegendste Auswirkung auftreten. Beim Kältetod handelt es sich um das Absterben von Pflanzen oder Pflanzenteilen aufgrund niedriger Temperatur. Anzeichen für Frostschädigungen sind das fehlende Austreiben im Frühling und eine Rotverfärbung bei Nadelhölzern.

Schneebruch: Als *Schneebruch* wird das Abbrechen von Ästen, Zweigen und Bäumen aufgrund des Gewichts der aufliegenden Schneemasse bezeichnet. Besonders betroffen sind davon Nadelhölzer, weil diese im Vergleich zu den kahlen Laubbäumen eine größere Auflagefläche aufweisen. Die Gefahr von Schneebruch ist im Winter bei nassem Schnee, Temperaturen um den Gefrierpunkt und schwachen Windverhältnissen am höchsten. Wenn die Äste, Zweige und Bäume nicht abbrechen, sondern nur niedergedrückt werden, spricht man von *Schneedruck*.

Windwurf: Die Folgen von Stürmen treten in der Vegetation als *Windwurf* oder *Windbruch* auf. Während beim Windwurf die Bäume mit Wurzel ausgerissen und umgeworfen werden, wird beim Windbruch der Stamm von Bäumen geknickt. Das Ausmaß wird von Faktoren wie Sturmdauer, Sturmfolge, Sturmrichtung, Bodenfeuchtigkeit, Jahreszeit und Topographie bestimmt.

Blitzschlag: Ein *Blitzschlag* kann in der Vegetation Spuren wie Risse, Spalten und Löcher hinterlassen. Bei besonders starken Blitzen könne ganze Bäume gespalten oder in Ausnahmefällen Waldbrände verursacht werden. Von Blitzschlag betroffene Bäume sind anfälliger für Pilz- und Insektenbefall.

Andere Ereignisse

Mit Ausnahme vom Fallwild werden in dieser Klasse nicht natürliche Ereignisse, sondern anthropogene Eingriffe in die Natur gesammelt.

Fallwild: *Fallwild* ist die Bezeichnung für gefallenes Wild, welches nicht durch die Jagd getötet wurde. Als Ursache kommen Krankheiten, Unfälle und am häufigsten der altersbedingte Tod in Frage. Eine Ursache anthropogener Herkunft sind Tötungsfälle durch den Strassenverkehr.

Anthropogene Veränderung: Geomorphologische Änderungen der Umwelt, die vom Menschen verursacht wurden, werden unter dem Ereignistyp *anthropogene Veränderung* zusammengefasst. In erster Linie sind Bauten und Terrainveränderungen zu nennen, wie beispielsweise Brücken oder Wege.

Lebensraumbeeinträchtigung: Weitere menschlich verursachte Schädigungen an der Flora und Fauna werden durch den Ereignistyp *Lebensraumbeeinträchtigung* abgedeckt. Beispiele hierfür sind der Bau einer Strasse, der den Lebensraum von Wildtieren aufgrund von Lärmemissionen verringert, oder die Besucher eines Schutzgebietes, welche die Natur durch Verlassen der Wege und Verschmutzung beeinträchtigen.

Zusammenfassend kann man sagen, dass die Masterarbeit von Jonas Büchel eine ausgezeichnete Grundlage für die Dokumentation von Naturereignissen in Schutzgebieten bildet. Nicht nur die Ereignisklassifikation, sondern auch definierte Parameter und zahlreiche Hinweise zu den Ereignistypen vereinfachen die Arbeit. Für die Anwendung in Schutzgebiete außerhalb von alpinen oder bewaldeten Regionen besteht allerdings weiterhin Forschungsbedarf.

2.1.3 Richtlinien zur Naturereignisdokumentation

Neben einer Klassifizierung der Naturereignisse spielen Richtlinien zur Naturereignisdokumentation eine wichtige Rolle bei der Erfassung. Auch diesem Thema hat sich Jonas Büchel in seiner Masterarbeit angenommen und für jeden Ereignistyp Richtlinien zur Dokumentation entwickelt. Diese Richtlinien enthalten für die verschiedenen Ereignistypen eine Beschreibung der Attribute und Parameter sowie bei Bedarf Hinweise zur Erfassung der Parameter.

Zusammen mit der Klassifizierung haben die Richtlinien entscheidend bei der Entwicklung der Webanwendung geholfen. Für ausführliche Informationen zu den Richtlinien sei auf die Masterarbeit von Jonas Büchel (2009) verwiesen.

Ein zentraler Gesichtspunkt der entwickelten Richtlinien ist die Unterteilung der Dokumentation in drei Erfassungsstufen. In der ersten Erfassungsstufe, die auch von Laien ausgeführt werden kann, wird neben dem Aufnahmeort lediglich eine Ortsbezeichnung und eine Beschreibung erfasst. Die zweite Erfassungsstufe berücksichtigt Daten wie Ereignisgröße und Ereigniszeitpunkt, die für alle Ereignistypen gültig sind, aber nur von geschulten Mitarbeitern erfasst werden sollen. In der dritten Erfassungsstufe, die von Experten ausgeführt werden soll, werden Daten, die spezifisch für einen Ereignistyp sind, aufgenommen.

2.2 Anwendungsgebiet und Datengrundlage

2.2.1 Schutzgebiete

Ein Schutzgebiet, auch bekannt als Schutzbereich oder Schutzzone, ist ein „exakt abgegrenzter Raum, der aufgrund seiner natürlichen Beschaffenheit, seiner ökologischen, ästhetischen oder historischen Erhaltungswürdigkeit oder auch seiner wirtschaftlichen und sozialen Funktion durch Gesetz oder Verordnung unter besonderen Schutz gestellt worden ist und in der Regel nur unter besonderen Bedingungen genutzt und verändert werden darf“ (Leser, 2001, S. 758). Diese Definition trifft gut auf die angestrebte Anwendungsgebiete für das Naturereignisinformationssystem zu. Für eine sinnvolle und nachhaltige Nutzung bestehen allerdings noch weitere Voraussetzungen.

Die wichtigste Voraussetzung sind entsprechend geschulte Mitarbeiter, damit überhaupt Informationen in die Datenbank gelangen können. Die Anzahl der beteiligten Mitarbeiter ist abhängig von der Größe des Schutzgebietes, der Häufigkeit von Naturereignissen und der angestrebten Vollständigkeit der Erfassung. Weiterhin muss eine technische Infrastruktur vorliegen, um das Informationssystem betreiben zu können. Minimalausstattung ist ein Computer, auf dem sowohl die Server- als auch Clientanwendung läuft; seine Vorteile kann das System aber erst in einem Netzwerk bestehend aus einem Server und mehreren Clients ausspielen. Zuletzt sind für eine sinnvolle Nutzung auch Geodaten wie topographische Karten, Luftbilder und ein digitales Geländemodell empfehlenswert, um die Lage der erfassten Ereignisse richtig einordnen zu können.

Im weiteren Verlauf werden mit dem Schweizerischen Nationalpark und dem Wildnispark Zürich zwei Schutzgebiete vorgestellt, die während der Analyse und Entwicklung als Testgebiete gedient haben.

2.2.2 Schweizerischer Nationalpark

Der Schweizerische Nationalpark (SNP), 1914 als erster Nationalpark der Alpen gegründet, liegt im Kanton Graubünden an der östlichen Grenze der Schweiz. Motivation für die Gründung war es, der zunehmenden Naturzerstörung entgegenzusteuern und einen möglichst ursprünglichen Teil des Landes vor den Eingriffen des Menschen zu bewahren. Der SNP ist mit 172 km^2 das größte Schutzgebiet in der Schweiz und ein Reservat der IUCN-Kategorie I (Badman & Bomhard, 2008). Seit 1979 ist der Park auch als UNESCO-Biosphärenreservat anerkannt. Laut dem Nationalparkgesetz (1980, S. 1) ist der Zweck des Schutzgebietes, dass „die Natur vor allen menschlichen Eingriffen geschützt und namentlich die gesamte Tier- und Pflanzenwelt ihrer natürlichen Entwicklung überlassen wird“. Zudem soll er „Gegenstand dauernder wissenschaftlicher Forschung sein“.

Insgesamt werden im Nationalpark die drei Ziele Naturschutz, Forschung und Information verfolgt (SNP, 2009a). Wie im Gesetz festgelegt, wird das Gebiet komplett dem Lauf der Natur überlassen und soll sich so zurück zu seinem ursprünglichen Zustand entwickeln. Auch die wissenschaftliche Forschung ist gesetzlich vorgeschrieben und trägt in Form von Langzeitforschungen zum Verständnis von natürlichen Prozessen und Veränderungen im Nationalpark bei. Eine zunehmend wichtigere Rolle spielt der Nationalpark als Bildungseinrichtung, um einen Beitrag zum Naturverständnis der Besucher zu leisten. Im Zuge dieser Bemühung wurde 2008 ein neues Besucherzentrum in Zernez eröffnet.

Der Park befindet sich in Höhenlagen zwischen 1400 m und 3174 m, das Klima ist trocken, rau und von starker Sonneneinstrahlung sowie geringer Luftfeuchtigkeit geprägt. Die Hälfte der Parkfläche befindet sich oberhalb der Vegetationszone, die übrige Fläche verteilt sich jeweils etwa zur Hälfte auf Nadelwald und alpine Wiesen. Im Park leben insbesondere Steinadler, 250 bis 450 Steinböcke, in den Sommermonaten 1800 bis 2000 Rothirsche und seit 1991 werden Bartgeier in der Val da Stabelchod wieder angesiedelt (SNP, 2009b).

Im Rahmen der Naturereignisdokumentation wurden im Schweizerischen Nationalpark in den Jahren 1987 bis 2003 insgesamt 584 Ereignisse und 786 Fotos zu Naturereignissen in analoger Form erfasst. Seit 2004 werden die Informationen direkt in einer Filemaker-Datenbank aufgenommen. Das analoge Archiv wurde 2007 dokumentiert und in eine GIS-Datenbank überführt. Diese Daten stehen als Grundlage für diese Arbeit zur Verfügung und sollen später in das zukünftige System importiert werden. Darüber hinaus wird im SNP eine umfangreiche Geodaten-Sammlung mit vielfältigen Raster- und Vektordatensätzen verwaltet. In der ArcSDE-Datenbank sind neben topographischen Karten, Luftbildern und digitalen Geländemodellen noch viele weitere Geodaten vorhanden.

2.2.3 Wildnispark Zürich

Der Wildnispark Zürich (WPZ) liegt rund 20 km südlich von Zürich und besteht aus dem Waldreservat Sihlwald und dem Wildpark Langenberg. Der Sihlwald ist mit 11 km^2 der größte zusammenhängende Laubmischwald im Schweizer Mittelland und diente über 500 Jahre der Holznutzung, die in den 1980er Jahren stark reduziert wurde. Seit neun Jahren ist die Nutzung eingestellt und der Wald damit der natürlichen Entwicklung zurück zu einem Naturwald überlassen (Prozessschutz). Am 28. August 2009 hat der Sihlwald vom Bundesamt für Umwelt die Anerkennung als Naturerlebnispark erhalten und ist damit ein Park von nationaler Bedeutung (WPZ, 2009).

Schon seit längerer Zeit ist der WPZ ein bedeutendes Forschungsgebiet für Hochschulen und Institute aus der Umgebung. Im Fordergrund der Erforschung steht die langfristige Landschafts- und Vegetationsentwicklung, die durch den Prozessschutz ermöglicht wird. In diesem Zusammenhang ist auch die Dokumentation von Naturereignissen, die bislang noch nicht stattfindet, von großem Interesse.

Das Waldgebiet eignet sich nicht nur zur Beobachtung von Ereignissen der Kategorien Forstschädlinge und Vegetationschädigung, sondern ist auf Grund einer hohen Reliefenergie von nahezu 500 Metern und einem differenzierten Gewässernetz auch für Massenbewegungen und Wasserereignisse ein interessantes Forschungsgebiet. Durch die ausgesetzte Lage des Albisgrates, kommt es hin und wieder auch zu Windwurfereignissen.

2.3 Technische Grundlagen

2.3.1 GIS im Internet

Bis heute sind die Begriffe rund um das Thema Internet und Geoinformationssystem nicht eindeutig definiert und einem steten Wandel unterworfen. Im Folgenden soll der aktuelle Stand kurz umrissen und eindeutige Begriffen für die verschiedenen Varianten von GIS im Internet festgelegt werden.

Allgemein werden unter dem Begriff *Informationssystem* alle System zusammengefasst, welche die Erfassung, Speicherung, Verarbeitung und Darstellung von Informationen unterstützen. Abhängig von den Zielsetzungen haben die Systeme unterschiedliche Schwerpunkte in ihrem Funktionsumfang oder begrenzen sich auf eine bestimmte Form von Daten.

Ein *Geoinformationssystem* (GIS) ist beispielsweise spezialisiert auf die Erfassung, Speicherung, Verarbeitung und Darstellung von räumlichen Daten und findet in zahlreichen Bereichen eine Anwendung, vom Vermessung- und Katasterwesen über Verkehr, Marketing und Logistik bis hin zu Umweltschutz und Geologie (Brinkhoff, 2008). Das Zusammenspiel der vier Aufgabenbereiche eines Geoinformationssystems **E**rfassung, **V**erwaltung, **A**nalyse und **P**räsentation (EVAP) lässt sich anhand der Abbildung 2.2 veranschaulichen.

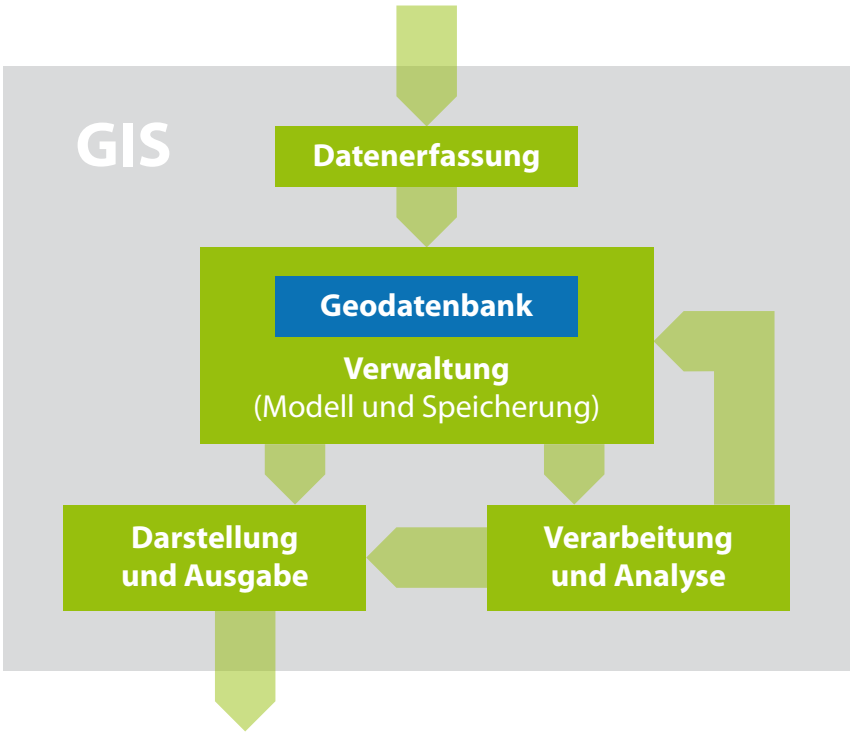


Abbildung 2.2: Komponenten eines GIS nach Brinkhoff (2008)

Lange Zeit war die Entwicklung von Geoinformationssystemen unter anderem auf Grund von technischen Einschränkungen von proprietären und geschlossenen Lösungen geprägt. Die fehlende Interoperabilität führte teilweise zu technologisch rückständigen Systemen oder Monopolstellungen der Hersteller und führte im Jahr 1994 schließlich zu der Gründung des *Open Geospatial Consortium*

(OGC). Ziel der in dieser Organisation zusammengeschlossenen GIS-Anbieter und Anwender war die Spezifizierung und Etablierung von offenen Geodatenformaten, -modellen und -schnittstellen, um die Interoperabilität von Geoinformationssystemen zu unterstützen (Brinkhoff, 2008). Einige OGC-Standards, die in Kapitel 2.3.2 näher beschrieben werden, bilden heute eine wichtige Grundlage für netzwerkbasierte Geoinformationssysteme.

In der Regel geht man bei netzwerkbasierten Informationssystemen, d.h. auch bei Geoinformationssystemen in einem Netzwerk, von einer Client-Server-Struktur aus. Der Client, zum Beispiel ein Webbrowser, sendet eine Anfrage an einen bestimmten Dienst eines Servers, welcher die Anfrage verarbeitet und das Ergebnis als Antwort an den Client zurückschickt (Gumm & Sommer, 2004).

Im Allgemeinen werden netzwerkbasierte Geoinformationsdienste als Internet GIS, Web GIS oder Distributed GIS bezeichnet (Shekar & Xiong, 2008). Diese Begriffe werden entweder synonym verwendet oder, abhängig von der Art des Netzwerkes, nach Fitzke (1999) folgendermaßen differenziert. Ein Distributed GIS ist die allgemeine Bezeichnung für netzwerkbasierte Geoinformationsdienste unabhängig von der Art des Netzwerkes, ein Internet GIS beschränkt sich auf den Datenaustausch über die Internetprotokollfamilie TCP/IP und ein Web GIS ist auf die Kommunikation über das Webprotokoll HTTP begrenzt.

Schaut man sich die Architektur von netzwerkbasierten Informationssystemen im Allgemeinen und GIS im Speziellen etwas genauer an, findet man normalerweise eine Drei-Schichten-Architektur vor (Shekar & Xiong, 2008). Die Präsentationsschicht visualisiert die Daten und stellt eine Schnittstelle zwischen dem System und dem Benutzer zur Verfügung, die Logikschicht beinhaltet die angebotenen Dienste und Prozesse, in der Datenhaltungsschicht werden die Daten gespeichert und verwaltet.

Die Drei-Schichten-Architektur kann in verschiedenen Kombinationen auf das Client-Server-Modell angewandt werden (siehe Abbildung 2.3). Wenn der Client nur die Präsentationsschicht übernimmt, spricht man von einem Thin-Client; wenn der Server nur die Datenhaltungsschicht zur Verfügung stellt und der Client neben der Präsentationsschicht auch die Logikschicht enthält, wird der Thin-Client zu einem Fat-Client.

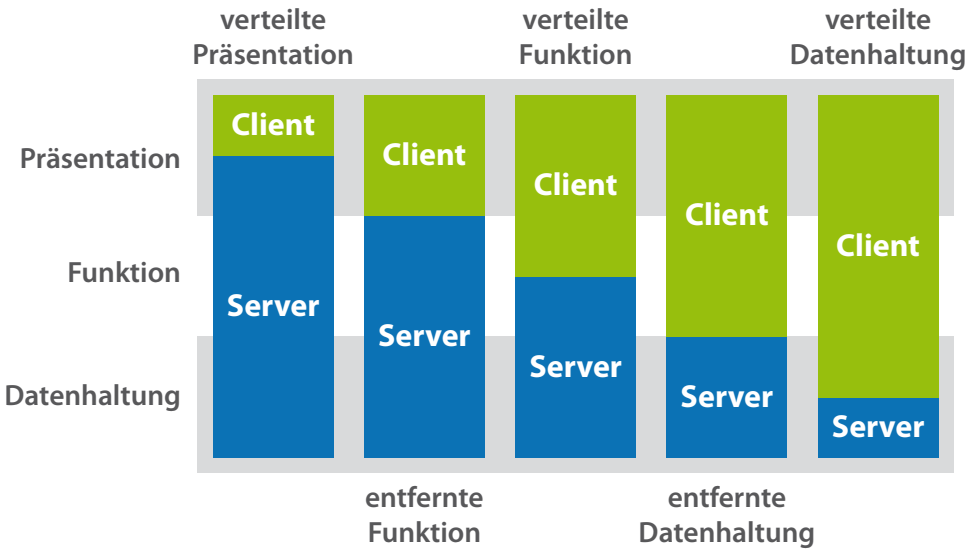


Abbildung 2.3: Mögliche Client-Server-Strukturen von Informationssystemen

Wenn man die Drei-Schichten-Architektur von netzwerkbasierten Geoinformationssystemen betrachtet, bieten die zur Verfügung stehenden Dienste auf der Seite des Servers eine hilfreiche Differenzierungsmöglichkeit. Insgesamt kann man von vier zentralen Diensten sprechen, die den Aufgabenbereichen eine Geoinformationssystems gleichkommen: Erfassung und Verwaltung, Präsentation, Abfrage und Analyse. Abhängig von den Diensten unterscheidet Fitzke (1999) folgende netzwerkbasierten Geoinformationssysteme, die in Abbildung 2.4 dargestellt sind:

Geodaten-Server: Bildet die Grundlage für einen Fat-Client und übernimmt einzig die Erfassung und Verwaltung der Geodaten.
Karten-Server: Stellt entweder dynamisch oder statisch visualisierte Geodaten, sprich Karten zur Verfügung.
Auskunftssystem: Bietet über die kartographische Visualisierung hinaus thematische und raumbezogene Abfragemöglichkeiten an.
Internet GIS: Dem Benutzer wird ein vollständiges Geoinformationssystem auf einem Internet Server zur Verfügung gestellt.
Funktions-Server: Bietet einzig raumbezogene Analyse- sowie Abfragemöglichkeiten an und benötigt eine Client-Anwendung, um sinnvoll eingesetzt werden zu können.

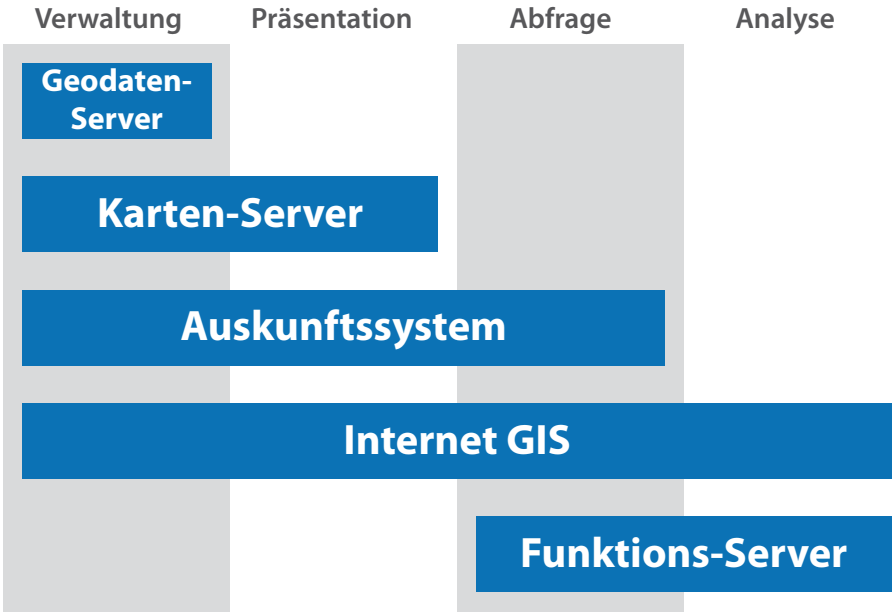


Abbildung 2.4: Unterscheidung von netzwerkbasierten GIS nach Fitzke (1999)

Insgesamt bieten die Art des Netzwerkes, die Client-Server-Struktur und die zur Verfügung stehenden Dienste ausreichend Differenzierungsmöglichkeiten für netzwerkbasierte Geoinformationssysteme. Das in dieser Arbeit zu entwickelnde System kann ohne Schwierigkeiten der Kategorie *Web GIS* zugeordnet werden, da es über einen normalen Internet Browser und das Webprotokoll HTTP bedient werden soll. Die Client-Server-Struktur wird voraussichtlich weitgehend ausgewogen zwischen Client und Server gestaltet sein, während die angestrebten Dienste eine Einordnung in die Kategorie *Auskunftssystem* nahelegen.

2.3.2 Netzwerkschnittstellen für Geodaten

Wie bereits in Kapitel 2.3.1 angedeutet, spielen offene Schnittstellen eine wichtige Rolle für die Interoperabilität und Etablierung von Internetangeboten auf Basis von Geodaten. Das Open Geospatial Consortium (OGC) hat für diesen Zweck eine Reihe von sogenannten Web Services (OWS) entwickelt, die im weiteren Verlauf kurz beschrieben werden. Alle erwähnten Schnittstellen sind für den Einsatz im Internet, genauer gesagt für HTTP konzipiert und nutzen zum Datenaustausch HTTP `POST` und HTTP `GET` (Shekar & Xiong, 2008).

Am bekanntesten und vermutlich auch am weitesten verbreitet ist der *Web Map Service* (WMS), der zum Austausch von georeferenzierten Rasterdaten dient. Der Server generiert aus unterschiedlichen Datenquellen eine Kartendarstellung und sendet das Ergebnis als Rasterbild an den Client (Shekar & Xiong, 2008). Damit eignet sich WMS beispielsweise zur Integration als Basiskarte in einer komplexen Kartendarstellung.

Der Datenaustausch von räumlichen Vektordaten wird vom Web Feature Service (WFS) unterstützt. Als Austauschformat kommt das auf XML basierende GML-Format zum Einsatz (siehe Kapitel 2.3.3). Der Dienst lässt sich nicht nur zur Anzeige, sondern auch zum Erstellen, Bearbeiten und Löschen von Datensätzen verwenden und wird dann als dann als Transaction WFS (WFS-T) bezeichnet (Shekar & Xiong, 2008).

Für Anwendungen, die umfangreiche GIS-Funktionalitäten benötigen, stehen weitere OGC-Dienste wie Web Processing Service (WPS) und Web Coverage Service (WCS) zur Verfügung.

Eine Alternative zu WMS für die Implementierung von Basiskarten ist der *Tile Map Service* (TMS). Die Spezifikation für diesen Dienst wurde im Rahmen der *Open Source Geospatial Foundation* (OSGeo) entwickelt (OSGeo, 2008). Im Gegensatz zu den OGC-Diensten, die eine entsprechende Software auf dem Server voraussetzen, wird TMS auf der Ebene des Dateisystems realisiert. Die verfügbaren Karten, welche in einer XML-Datei beschrieben sein müssen, werden in festgelegten Maßstäben als Kacheln über eine REST-Schnittstelle bereitgestellt.

2.3.3 Geodatenformate im Internet

Neben offenen Schnittstellen sind für den Austausch von Geodaten über das Internet außerdem standardisierte Datenformate notwendig. Im Folgenden sollen einige, im Internet weit verbreitete Datenformate für den Austausch von räumlichen Vektordaten beschrieben werden.

Ein häufig verwendetes Format ist die *Geographic Markup Language* (GML), welches auf XML basiert und zuletzt in Version 3.2 als OGC-Spezifikation sowie als ISO 19136 veröffentlicht wurde (OGC, 2004). Aufgrund seiner Vielseitigkeit bildet GML zudem die Grundlage von anwendungsspezifischen Formaten wie beispielsweise CityGML. Darüber hinaus wurden zur Einschränkung der Komplexität unterschiedliche GML Profile entwickelt.

Interessant für Webanwendungen, die ein weniger umfangreiches Austauschformat benötigen, ist das GML *Simple Features Profile* (OGC, 2006a). Das Geometriemodell entspricht dem *Simple Features for SQL* (SF SQL) Standard und unterstützt die Geometrietypen Punkt, Linie, Fläche und die entsprechenden Multi-Varianten. Die Komplexität des Simple Feature Profile wird in drei Stufen, vom einfachsten Level-0 bis zum umfangreichsten Level-2 unterteilt. Während das Level-0 Profil lediglich einfach Datentypen und 1:1-Beziehungen zur Verfügung stellt, können mit den Level-1 und Level-2 Profilen auch komplexe Datentypen und 1:N-Beziehungen realisiert werden.

Kein Dateiformat im klassischen Sinn, aber nichtsdestoweniger ein Format zum Austausch von Geometriedaten ist *Well-known text* (WKT), welches im Rahmen der *Simple Feature Access* OGC-Spezifikation beschrieben wird. Das Format stellt sozusagen den kleinsten gemeinsamen Nenner dar und eignet sich zum Austausch von Punkten, Linien, Flächen und den entsprechenden Multi-Varianten. Allerdings können keine Metadaten, wie beispielsweise Informationen über das verwendete Koordinatensystem, ausgetauscht werden.

Ein weiteres interessantes Austauschformat, welches sich in Bezug auf die Komplexität zwischen WKT und GML einordnen lässt, ist GeoJSON (Butler et al., 2008). Im Gegensatz zu GML basiert es nicht auf XML, sondern auf *JavaScript Object Notation* (JSON), und eignet sich daher optimal für die Verarbeitung im Browser. Unterstützt werden die Geometrietypen von WKT sowie Metadaten zu den Geometriedaten.

2.4 Grundlagen der Software-Entwicklung

2.4.1 Vorgehensmodelle

Eine zentrale Fragestellung bei der Entwicklung von Software ist die Vorgehensart, nach der das Projekt bearbeitet werden soll. Im Laufe der Zeit haben sich verschiedenste Vorgehensmodelle entwickelt, welche im weiteren Verlauf kurz vorgestellt werden.

Ende der 1960er Jahre kam man im Zuge der sogenannten Softwarekrise zu der Erkenntnis, das die Entwicklung von Software nach Methoden, die mit anderen Ingenieursdisziplinen vergleichbar sind, angegangen werden sollte (Partsch, 1998). Mit dem Ziel der Entwicklung von zuverlässiger Software entstand die Disziplin des *Software-Engineerings* sowie zahlreiche Vorgehensmethoden, die von Partsch (1998) auch als „Methoden-Dschungel“ bezeichnet werden.

Ein frühes und nach wie vor weit verbreitete Vorgehensmethode stellt das Wasserfall-Modell dar, welches charakteristisch für die Klasse der *Phasenmodelle* ist. Typisch für diese Modelle ist die Einteilung des Projekts in aufeinander folgende Phasen, mit jeweils klar definierten Vorgaben und Ergebnissen, die im Folgenden kurz beschrieben werden (Gumm & Sommer, 2004). Dieser Ansatz ermöglicht eine systematische Projektaufteilung und eine ergebnisorientierte Arbeitsweise.

Problemanalyse und Anforderungsdefinition: Zunächst wird der Gegenstandsbereich des Projekts analysiert und eingegrenzt. Anschließend werden die Anforderungen an das System definiert und die Funktionalitäten grob festgelegt.
Fachlicher Entwurf: Aufbauend auf der Anforderungsdefinition werden die Funktionen aus fachlicher Sicht spezifiziert. Gewöhnlich wird dafür ein Modell erstellt, welches die Strukturen und Beziehungen aller relevanten Objekte enthält.
Software-technischer Entwurf: Das fachliche Modell wird im Rahmen des technischen Entwurfs auf die konkrete Softwarearchitektur abgebildet. Dazu wird das System in Komponenten, Module und Schnittstellen aufgeteilt.
Programmierung und Modultest: Die Spezifikationen werden in Programmcode umgesetzt und mit Hilfe von Modultests systematisch überprüft.
System-Integration und Systemtest: Die fertigen Module werden zum Gesamtsystem zusammengesetzt und getestet. Der Entwicklungsprozess ist damit abgeschlossen.

Installation, Betrieb und Weiterentwicklung: Das fertiggestellte System wird in der Zielumgebung installiert und vom Auftraggeber in Betrieb genommen. In den meisten Fällen ist die Entwicklung von Software damit nicht abgeschlossen, weil es immer Änderungs-, Verbesserungs- und Erweiterungswünsche gibt.

In der Praxis sind bei der Anwendung von Phasenmodellen, trotz der genannten Vorteile, immer wieder Probleme aufgetreten. Zu den wesentlichen Kritikpunkten zählen realitätsferne und mangelnde Flexibilität, Software-Bürokratie, eine fehlende Weiterentwicklungs-Strategie und die Trennung von Anwender- und Entwickler-Welt (Gumm & Sommer, 2004). Um diese Probleme in den Griff zu bekommen, wurden so genannte *nichtsequentielle Vorgehensmodelle* entworfen, mit dem Ziel, stabilere Anforderungen zu erhalten und flexibler auf sich ändernde Anforderungen reagieren zu können.

Zu den nichtsequentiellen Vorgehensmodellen gehören beispielsweise das Prototyping, das Spiralmodell und andere inkrementelle Entwicklungsmethoden. In jüngster Zeit haben sich zudem verschiedene Methoden entwickelt, die unter dem Begriff *agile Softwareentwicklung* zusammengefasst werden und auf den Prinzipien von Beck et al. (2001) aufbauen. Als Entwicklungsmethoden werden in unterschiedlichem Maße testgetriebene Entwicklung, Paarprogrammierung, ständige Refaktorisierung und schnelle Codereviews eingesetzt.

2.4.2 Analysemethoden

Bei der Entwicklung von Softwareprodukten helfen Analysemethoden bei der Anforderungsdefinition, Problemanalyse und dem fachlichen Entwurf; bezogen auf das Wasserfall-Modell sind also die ersten zwei Entwicklungsphasen betroffen. Ähnlich wie bei den Vorgehensmodellen gibt es auch in diesem Bereich eine unüberschaubare Anzahl unterschiedlicher Methoden. Die Analysemethode im Rahmen dieser Arbeit orientiert sich im Wesentlichen an der objektorientierten Analyse, wie sie beispielsweise von Balzert (2004) und Oestereich (2009) beschrieben wird. Eine Zusammenfassung der wichtigsten Konzepte und Begriffe, die sich an Balzert (2004) orientiert, soll als Grundlage zum Verständnis von Kapitel 3 dienen.

Im Zuge der Etablierung von objektorientierten Programmiersprachen hat die objektorientierte Analyse (OOA) eine weite Verbreitung gefunden. Dieser Ansatz versucht die Vorteile der objektorientierten Programmierung (OOP) auf die Analyse und Spezifikation der Softwareentwicklung zu übertragen (Jackson, 1995). Ziel dieser Methode ist es, Objekte der realen Welt durch Abstraktion in ein objektorientiertes Modell zu überführen. Grundsätzlich wird dabei zwischen einem statischen und dynamischen Modell unterschieden; beide Modelle hängen eng miteinander zusammen und helfen bei der gegenseitigen Validierung (Balzert, 2004).

Grundlegend für die objektorientierte Analyse sind die Konzepte *Klasse*, *Objekt*, und *Operation*. Eine Klasse definiert für eine Sammlung von Objekten deren Struktur, Verhalten und Beziehungen. Ein Objekt ist ein individuelles, eindeutig identifizierbares Exemplar einer Klasse und besitzt einen Zustand, bestehend aus Attributwerten und Beziehungen zu anderen Klassen, sowie ein festgelegtes Verhalten zu seiner Umwelt. Attribute beschreiben Daten eines Objektes und können einem bestimmten Datentyp zugeordnet werden. Alle Objekte einer Klasse besitzen dieselben Attribute, in der Regel jedoch unterschiedliche Attributwerte. Das Verhalten von Objekten sind die Ergebnisse von Operationen bzw. Funktionen, welche auf die internen Attributwerte zugreifen können.

Das statische Modell ist vor allem für Datenbankanwendungen von zentraler Bedeutung. Es beschreibt die Klassen und Generalisierungsstrukturen des Systems sowie die Assoziationen zwischen

den Klassen. Ähnlich wie ein Objekt Exemplar einer Klasse ist, handelt es sich bei einer Objektbeziehung um ein Exemplar einer Assoziation. Eine Assoziation beschreibt also die Beziehungen zwischen Klassen und ist vergleichbar mit der Relationship des Entity-Relationship-Modells. Die Generalisierung beschreibt die Beziehung zwischen einer Basisklasse und einer spezialisierten Klasse, es wird also eine Klassenhierarchie definiert. Die spezialisierte Klasse erweitert die Basisklasse mit eigenen Attributen, Operationen und Assoziationen; man spricht auch von Vererbung. Die Elemente eines Systems können in Paketen zusammengefasst und auf diese Weise auf einem höheren Abstraktionsniveau beschrieben werden. Das Ergebnis eines statischen Modells besteht überwiegend aus einem Klassenmodell, in der Regel in Form eines UML-Klassendiagramms.

Das dynamische Modell beschreibt das Verhalten des zu entwickelnden Systems und basiert auf den Konzepten *Aktivität*, *Szenario* und *Use-Case*. Die kleinste ausführbare Funktionseinheit eines Systems wird als Aktion bezeichnet, in Aktivitäten zusammengefasst und in einem Aktivitätsdiagramm modeliert. Mehrere Verarbeitungsschritte werden in Szenarien gesammelt und unter bestimmten Bedingungen ausgeführt. Die Visualisierung erfolgt mit Hilfe von Sequenz- oder Kommunikationsdiagrammen. Eine Kollektion von Szenarien wird in einem Use-Case gebündelt und beschreibt die Arbeitsabläufe der Benutzer mit dem System auf einer höheren Abstraktionsebene.

Die Reihenfolge und Schwerpunkte der Analyseschritte sind nicht genau festgelegt, sondern immer abhängig von einem bestimmten Anwendungsfall. Eine generische Vorgehensweise der Modellbildung wird von Balzert (2004) durch den Makroprozess beschrieben.

Die objektorientierte Analyse (OOA) kann allerdings auch eine Reihe von Problemen mit sich bringen, da die Objekte der realen Welt nicht immer mit Hilfe der oben genannten Konzepte und Begriffe modelliert werden können (Jackson, 1995). Ein Problem ist die Veränderung von Individuen in der realen Welt, beispielsweise von einer Raupe zum Schmetterling, die in einem objektorientierten Modell nur schwierig abgebildet werden kann, da ein Objekt immer an eine bestimmte Klasse gebunden ist. Weiterhin können Objekte oft einer Vielzahl von Klassen zugeordnet werden, die objektorientierte Generalisierung erlaubt hingegen nur die Vererbung von einer Klasse. In der realen Welt werden Objekte außerdem häufig selbstständig und spontan aktiv, die Objekte der OOA handeln allerdings nur passiv. Nichtsdestotrotz kann die OOA gerade in Kombination mit objektorientierten Programmiersprachen erfolgreich eingesetzt werden, wenn man die genannten Probleme berücksichtigt. Grundsätzlich sollte man beachten, dass die Analyse in der Software-Entwicklung immer eine „Gratwanderung zwischen Formalismus und Formlosigkeit“ (Balzert, 2004, S. 130) ist; entweder die Kreativität wird gehemmt oder das Chaos gefördert.

2.4.3 Softwarearchitekturmuster

Ein weit verbreitetes Softwarearchitekturmuster ist die *Model-View-Controller*-Architektur (MVC), welche Ende der 1970er für Benutzungsoberflächen in der Programmiersprache Smalltalk entwickelt und von Krasner und Pope (1988) erstmals beschrieben wurde (Gamma et al., 1995). Inzwischen wird das Architekturmuster nicht mehr nur für grafische Oberflächen von Desktop-Anwendungen verwendet, sondern auch innerhalb von Web-Anwendungen eingesetzt. Ziel des Musters ist eine flexible Softwarestruktur, die eine spätere Anpassung erleichtert und die Wiederverwendbarkeit von einzelnen Teilen der Anwendung erlaubt.

Die vielfältigen Einsatzbereiche haben im Laufe der Zeit zu unterschiedlichsten Interpretationen geführt, die sich teilweise stark voneinander unterscheiden. Im Folgenden wird das Konzept in Bezug auf typische Webanwendungen, so wie es auch in zahlreichen *Web Application Frameworks* zum Einsatz kommt (siehe Kapitel 4.3), kurz zusammengefasst.

Das *Modell* entspricht der Datenebene und repräsentiert die Objekte einer Anwendung, ist also auch für die Persistenz der Daten zuständig. Der *Controller*, oft auch als Steuerung bezeichnet, reagiert auf Benutzereingaben und führt entsprechende Aktionen aus. Für die Darstellung der Daten ist die Präsentationsschicht (*View*) zuständig, welche außerdem die Benutzereingaben entgegen nimmt und an den Controller weiterleitet. Die Daten werden entweder in einem *Push*-Verfahren an die Präsentationsschicht geschickt oder in einem *Pull*-Verfahren von der Präsentationsschicht abgefragt. Die MVC-Architektur kann mit Hilfe unterschiedlicher Entwurfsmuster, die in Gamma et al. (1995) beschrieben sind, umgesetzt werden.

Als *Representational State Transfer* (REST) wird ein Softwarearchitekturmuster bezeichnet, welches für die Entwicklung von verteilten Informationssystem - beispielsweise im Internet - verwendet werden kann. Im Gegensatz zu Ansätzen wie SOAP, wird keine zusätzliche Transportschicht eingeführt, sondern existierende Protokolle wie HTTP verwendet. Der Begriff wurde von Fielding (2000) geprägt und umfasst die folgenden vier Prinzipien:

Adressierbarkeit: alle Daten werden als Ressourcen zur Verfügung gestellt, die über *Uniform Resource Identifier* (URI) eindeutig zugeordnet werden können.
Zustandslosigkeit: weder der Server noch der Client speichert Zustandsinformationen zwischen zwei Nachrichten. Das bedeutet, dass in jeder Nachricht alle Informationen enthalten sein müssen, 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. Dadurch ist keine zentrale Registrierungsdatenbank notwendig, sondern die Ressourcen werden über Hyperlinks miteinander verknüpft.

3 Analyse der Anforderungen

Zu Beginn der Anforderungsanalyse wurden Gespräche mit den zuständigen Mitarbeitern vom Schweizerischen Nationalpark und vom Wildnispark Zürich geführt, die in **Kapitel 3.1** zusammengefasst werden. Aus den Ergebnissen wird im folgenden Kapitel ein Pflichtenheft erstellt, welches die erforderlichen Leistungen des zu erstellenden Systems beschreibt. Aufbauend auf den Gesprächen und dem Pflichtenheft wird in **Kapitel 3.3** in drei Iterationen eine objektorientierte Analyse durchgeführt, die als Grundlage für die Realisierung dient.

3.1 Gespräche

Für einen ersten Überblick und um eine Vorstellung von dem gewünschten System zu bekommen, wurden schon im August 2008 - als ich mich das erste Mal mit dem Thema auseinander gesetzt habe - Gespräche mit den zuständigen Mitarbeitern Ruedi Haller vom Schweizerischen Nationalpark und Ronald Schmidt vom Wildnispark Zürich geführt. Die Resultate der Gespräche sind im Folgenden zusammengefasst und dienen als Grundlage für die weitere Anforderungsanalyse. Insbesondere das Pflichtenheft wurde auf Grundlage dieser Gespräche erstellt.

Insgesamt wird von beiden Befragten ein System angestrebt, welches Eingabe- und Ausgabemöglichkeiten beinhaltet und in einer Anwendung zusammenführt. Analysefunktionen haben zunächst eine geringe Priorität, sind zukünftig aber durchaus interessant. Ziel ist eine in erster Linie eine dauerhafte Dokumentation und nachhaltige Nutzung der gesammelten Informationen.

Im Wildnispark Zürich wurden bisher noch keine Ereignisse erfasst, allerdings stehen Quellen wie beispielsweise Waldwirtschaftspläne, Jahresberichte und Parkwächter zur Verfügung, um Ereignisse nachträglich rekonstruieren zu können. Im Schweizerzischen Nationalpark wurden die Ereignisse in einem Papierformular erfasst und anschließend in eine Filemaker-Datenbank eingetragen. Zusätzlich stehen in beiden Schutzgebieten umfangreiche Basisdaten als Hilfsmittel für die Ein- und Ausgabe zur Verfügung (siehe Kapitel 2.2).

Die Dateneingabe soll zukünftig möglichst benutzerfreundlich, d.h. schnell, einfach und weitestgehend automatisiert erfolgen. Als optimale Lösung wird von beiden Befragten ein digitales, mobiles Gerät zur Erfassung vor Ort angesehen. Weiterhin sollten eine Web- und eine GIS-Schnittstelle als zusätzliche Eingabemöglichkeiten zur Verfügung stehen. Zunächst hat für beide Schutzgebiete allerdings eine Webanwendung mit einer Formularmaske für die Sachdateneingabe und einer Karte für die räumliche Positionierung Vorrang. Von beiden Seiten wird eine Vorschaufunktion als wünschenswert angesehen. Für den Schweizerischen Nationalpark ist weiterhin noch eine Importfunktion interessant, um die bestehenden Daten in die Datenbank zu übernehmen.

Bei der Dateneingabe hatte sich die Idee einer mehrstufigen Erfassung, welche Jonas Büchel in einem Expertengespräch mit Mario Negri, Leiter Betrieb im SNP, erarbeitet hatte, als erstrebenswerte Lösung herauskristallisiert. Von beiden Seiten wurde daher vorgeschlagen, eine einfache Eingabemaske für Besucher und Mitarbeiter sowie spezialisierte Eingabemasken für geschulte Mitarbeiter anzubieten. In einem weiteren Expertengespräch von Jonas Büchel mit Dr. Christoph Hegg wurde festgestellt, dass „aufgrund der Erfahrungen bei StorMe die Einfachheit der ersten Stufen unbedingt gewährleistet werden muss“ (Büchel, 2009, S. 42).

Ähnlich wie bei der Dateneingabe, wurde auch für die Datenausgabe ein mehrgleisiges System vorgeschlagen, allerdings mit anderen Zielgruppen. Für einfache Informationsabfragen, beispielsweise von Besuchern oder Mitarbeitern, sollte eine intuitive Webanwendung zur Verfügung stehen, während für Wissenschaftler und Experten weitere Schnittstellen mit direktem Zugriff auf die Daten wünschenswert sind. Für die Webanwendung wird von beiden Gesprächspartnern ein Datenblatt zur Ausgabe von einem Ereignis bevorzugt, welches alle Informationen wie Karte, Tabelle, Fotos und ähnliches auf einer Seite integriert. Während Ruedi Haller eine kartographische Darstellung der

Daten für geeignet hält, findet Ronald Schmidt - zumindest bei mehreren Ereignissen - eine tabellarische Darstellung übersichtlicher.

Die Abfrage von Ereignissen soll nach räumlichen, zeitlichen und thematischen Gesichtspunkten möglich sein. Vor allem unterschiedliche räumliche Abfragen scheinen von Interesse zu sein. Insgesamt hat die Ausgabe als HTML-Seite im Browser und eine GIS-Schnittstelle zur Datenbank erste Priorität. Andere Ausgabeformate wie PDF, KML, Shapefile oder OGC-Dienste wie WMS und WFS sind zweitrangig, aber auch von Interesse.

Ein möglicherweise interessantes Anwendungsgebiet wären unterschiedlichste Analysen auf Grundlage der erfassten Naturereignisdaten. Als Beispiele werden von den Befragten die Berechnung des Windwurfpotentials und die Identifizierung von potentiell betroffener Infrastruktur genannt. Gerade im Bereich der Forschung ließen sich noch viele Möglichkeiten aufzählen.

Obwohl Analysefunktionen als interessante Zusatzoptionen angesehen werden, ist die einhellige Meinung, dass diese nur von Spezialisten eingesetzt werden und in ihrer Ausprägung so vielfältig sind, dass sie sich nicht für eine Integration in die Webanwendung eignen. Ein generelles Problem sieht Ruedi Haller in der fehlenden Abdeckung von abgelegenen Gebieten, weshalb die Aussagekraft einer Analyse schwer zu bestimmen sein wird. Insgesamt wird von beiden Befragten eine Kombination aus Abfrage- und Exportfunktionen als Grundlage für Analysen innerhalb der Webanwendung bevorzugt.

Die Anzahl der Benutzer und Ereignisse konnte von beiden Befragten nur grob geschätzt werden, so wird in beiden Fällen mit 10 bis maximal 20 Mitarbeitern gerechnet, die auf das System zugreifen. Das Einschätzen der möglichen Besucherzahlen war noch schwieriger und wurde von Ruedi Haller mit maximal 100 Gästen pro Monat beziffert. Des Weiteren wird im Schweizerischen Nationalpark zusätzlich zu den rund 600 bereits existierenden Datensätzen mit 50 bis 100 Erfassungen pro Jahr gerechnet, im Wildnispark Zürich mit deutlich weniger.

3.2 Pflichtenheft

Das Pflichtenheft beschreibt in abstrakter Form, was das zu realisierende System leisten soll und dient als Ausgangsbasis für die weitere Anforderungsanalyse. Darüber hinaus eignet es sich auch als Einstiegsdokument für zukünftige Projektbeteiligte. Die Gliederung orientiert sich an dem Beispiel von Balzert (2004).

3.2.1 Zielbestimmung

Muss-Kriterien

Zentrale Kriterien, die mit dem System erreicht werden sollen, sind die Implementierung des Datenbankschema aus der Masterarbeit von Jonas Büchel, die dreistufige Erfassung von Naturereignisdaten und die Verwaltung von Naturereignisdatensätzen, also erstellen, anzeigen, bearbeiten und löschen derselben. Weiterhin sind Abfragemöglichkeiten nach Zeit, Kategorie und geographischer Lage, sowie eine Exportmöglichkeit der Daten Voraussetzung für den produktiven Einsatz der Anwendung.

Kann-Kriterien

Von geringerer Priorität, aber trotzdem von Interesse, sind eine Schnittstelle zu einer mobilen Anwendung, Benutzer- und Rechteverwaltung sowie Backup- und Importfunktion. Außerdem ließen sich weitere Abfragemöglichkeiten implementieren, um Datensätze genauer eingrenzen zu können.

Abgrenzungskriterien

Im Vergleich zu anderen Webanwendungen wird das zu realisierende System zunächst nur als interne Anwendung für die Mitarbeiter von Schutzgebieten konzipiert sein. Außerdem wird das System als reine Client-Server-Anwendung implementiert und ist nicht als Einzelplatzanwendung gedacht.

3.2.2 Einsatz

Anwendungsbereiche

Die Anwendung soll zur Dokumentation von Naturereignissen in Schutzgebeiten, speziell im Schweizerischen Nationalpark (SNP) und im Wildnispark Zürich (WPZ) eingesetzt werden. Im Vordergrund steht die Erfassung von neuen Naturereignissen. Eine sinnvolle und nachhaltige Nutzung der Daten soll möglich sein.

Zielgruppen

Die Anwendung ist zunächst als interne Anwendung für die Mitarbeiter eines Schutzgebietes wie dem SNP oder dem WPZ konzipiert.

Betriebsbedingungen

Der Betrieb der Anwendung soll in einem weitgehend wartungsfreien Dauerbetrieb als Internetdienst möglich sein. Die Konfiguration und Datensicherung müssen hingegen manuell von einem Administrator ausgeführt werden.

3.2.3 Umgebung

Software

Auf der Seite des Anwenders wird ein aktueller, standardkonformer Internetbrowser wie beispielsweise *Firefox 3*, *Internet Explorer 8*, *Opera 9* oder *Safari 4* vorausgesetzt.

Auf dem Server wird eine relationale Datenbank, welche mit räumlichen Daten umgehen kann, wie *PostgreSQL* oder *Oracle*, sowie eine Laufzeitumgebung für Webanwendungen wie *Java*, *PHP*, *Python* oder *Ruby* notwendig sein. Weitere Anforderungen werden im Laufe der Softwareuntersuchung (siehe Kapitel 4) festgelegt.

Hardware

Der Anwender benötigt lediglich einen Desktop-Computer mit Internetanschluss. Die Anforderungen an die Rechenleistung sollten entsprechend der von typischen Kartenanwendungen im Internet sein.

Der Server muss, wie nicht anders zu erwarten, ebenfalls über eine Internetverbindung verfügen. Die benötigte Rechenleistung und Speicherkapazität ist abhängig von der Anzahl an Benutzern und Datensätzen.

3.2.4 Funktionalität

Die Arbeitsabläufe folgen im Wesentlichen der dreistufigen Erfassung und beinhalten in erster Linie die Erstellung von Naturereignisdatensätzen. Darüber hinaus sind Funktionalitäten zum Bearbeiten und Löschen der Datensätze notwendig. Für die Datenausgabe sind eine Datenblattansicht sowie Listen- und Kartendarstellung der Naturereignisdatensätzen notwendig. Des Weiteren sollen die Daten exportiert und nach den Kriterien Zeit, geographische Lage und Kategorie eingegrenzt werden können.

3.2.5 Daten

Pro Jahr sind grob geschätzt 100 bis 200 neue Erfassungen von Naturereignissen realistisch. Außerdem sollen im SNP rund 600 bestehende Naturereignisdatensätze importiert werden.

3.3 Analyse der Naturereignisdokumentation

3.3.1 Überblick

Die bisherige Anforderungsanalyse zeigt deutlich, dass zunächst die Erfassung von Naturereignissen im Mittelpunkt der Anwendung steht. Die Arbeit wird sich daher weniger mit den Nutzungsmöglichkeiten der Naturereignisdaten beschäftigen, sondern vielmehr die Realisierung einer Plattform zur Erfassung von Naturereignisdaten zum Ziel haben. Gleichwohl sollen zukünftige Nutzungsmöglichkeiten bei der Konzeption und Implementierung berücksichtigt werden.

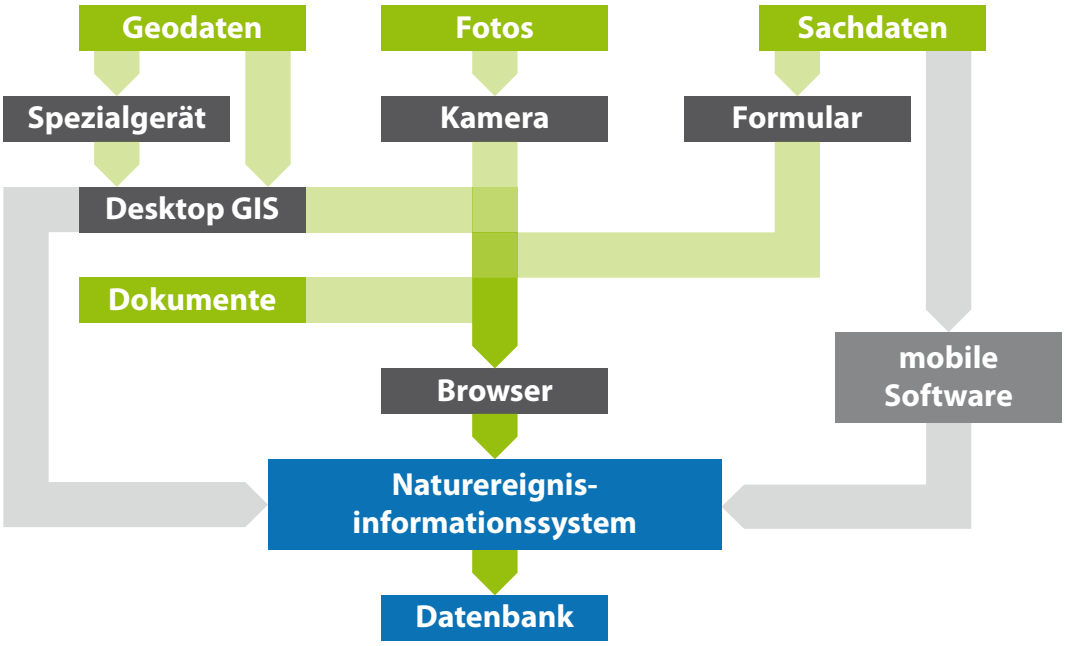


Abbildung 3.1: Wege der Datenerfassung

Für das Gesamtverständnis der Naturereignisdokumentation ist an erster Stelle ein Überblick über die verschiedenen Wege der Datenerfassung und die involvierten Datentypen wichtig. Dieser wird in Abbildung 3.1 als Diagramm visualisiert. Die grünen Rechtecke stellen zusammen mit den hellgrünen Verbindungslinien die möglichen Datentypen und Erfassungswege dar. Als dunkelgraue Rechtecke werden die verschiedenen Werkzeuge und Hilfsmittel visualisiert. Die mehrstufige Erfassung ist in dieser Abbildung absichtlich außen vor gelassen und wird im Anschluss behandelt.

Unter *Geodaten* werden Geometriedaten mit direktem Raumbezug zusammengefasst. Als Geometrietypen kommen sowohl Punkte und Linien als auch Polygone in Frage. Während in der ersten Erfassungsstufe nur Punktdaten zur Positionierung von Ereignissen und Fotos erhoben werden und in der zweiten Erfassungsstufe keine Geodaten auftauchen, kommen in der dritten Erfassungsstufe zu den Punkten sowohl Linien als auch Polygon zur exakten räumlichen Abgrenzung und Positionierung der Ereignisse hinzu. Die Erfassung von Geodaten erfolgt entweder an einem Desktop-Computer mit Hilfe eines GIS oder vor Ort mit speziellen Geräten wie Tachymeter, Laser Distanzmessgerät oder GPS-Gerät. Unabhängig von der Erfassungsart müssen die erhobenen Daten in einem Desktop-GIS voraussichtlich nachbearbeitet, eventuell in das gewünschte Koordinatensystem konvertiert und als GML-Datei exportiert werden. Die daraus resultierende Datei kann anschließend mit Hilfe eines Internetbrowsers in das System importiert werden.

Für die Erfassung von Fotos wird ein digitaler Fotoapparat vorausgesetzt. Analog aufgenommene Bilder müssen in einem Zwischenschritt noch digitalisiert werden, aber selbst die Fotos der bisherigen Naturereignisdokumentation im Schweizerischen Nationalpark liegen inzwischen in digitaler Form vor. Die Fotos werden entweder im Voraus georeferenziert, enthalten also GPS-Koordinaten in den EXIF-Daten, oder werden innerhalb der Webanwendung während der Erfassung interaktiv georeferenziert.

Dokumente, wie zum Beispiel PDFs oder Excel-Tabellen, müssen ebenfalls in digitaler Form vorliegen. Unabhängig vom Dateityp können die Dokumente, ähnlich wie Fotos, in das System hochgeladen werden.

Die Kategorie der Sachdaten macht den größten Teil der erfassten Daten aus und setzt sich aus allen drei Erfassungsstufen zusammen. Hinzu kommen Metadaten über Fotos und Dokumente. Die Daten werden von Mitarbeitern der Schutzgebiete mit Hilfe einer mobilen Anwendung oder auf Papierformularen aufgenommen. Im Fall der digitalen Erfassung, können die Daten - sobald eine Verbindung zur Serveranwendung besteht - direkt in die Datenbank übertragen werden. Die analog erfassten Daten müssen an einem Desktop-Computer mit Hilfe einer Webanwendung in die Datenbank eingetragen werden. Obwohl eine digitale Erfassung der Sachdaten vom Schweizerischen Nationalpark und dem Wildnispark Zürich angestrebt wird, beschränkt sich diese Arbeit auf die analoge Erfassung mit Hilfe von Papierformularen. Die Implementierung einer mobilen Anwendung zusätzlich zu der Serveranwendung würde den Rahmen dieser Arbeit sprengen, nichtsdestotrotz wird dieser Erfassungsweg bei der Konzeption und Realisierung berücksichtigt.

Abhängig von den zukünftig verwendeten mobilen Geräten wäre es auch möglich, die Erfassung von Geodaten und Fotos in die mobile Anwendung zu integrieren. Zu Gunsten der Übersicht wurde diese Option jedoch nicht visualisiert. Außerdem hat sich während der Gespräche mit Ronald Schmidt (WPZ) und Ruedi Haller (SNP) herausgestellt, dass aufgrund der vielfältigen Möglichkeiten der Geodatenerfassung diese Funktion besser nicht in die mobile Anwendung integriert werden sollte. Diese Einschränkung betrifft jedoch nur die dritte Erfassungsstufe, während der Aufnahmeort von Grunddaten und Fotos problemlos mit Hilfe eines integrierten GPS-Empfängers erfasst werden könnte.

Wie bereits im Kapitel 2.1.3 erläutert wurde, spielt die Unterteilung in drei Erfassungsstufen eine wichtige Rolle für die Naturereignisdokumentation. Die drei Stufen werden zusammen mit den entsprechenden Objekten und Akteuren in Abbildung 3.2 visualisiert und bilden zusammen mit dem geschilderten Ablauf der Erfassung das grundlegende Konzept der Naturereignisdokumentation.

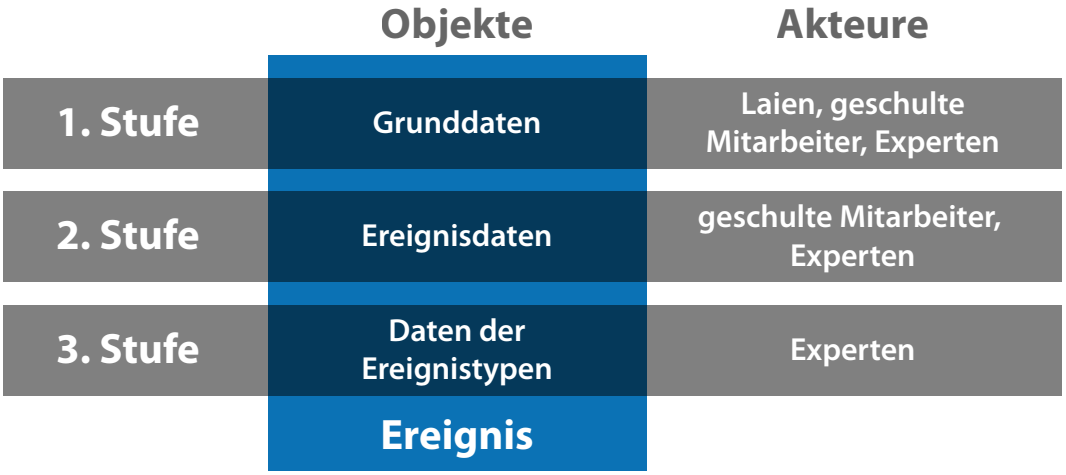


Abbildung 3.2: Objekte und Akteure der dreistufigen Erfassung

Die weitere Analyse wird, angelehnt an die drei Erfassungsstufen, in drei Iterationen aufgeteilt. Grundlage sind das Pflichtenheft (siehe Kapitel 3.2) und die Masterarbeit von Jonas Büchel (2009), insbesondere das Entity-Relationship-Modell. Zu beachten ist, dass sich Änderungen in übergeordneten Erfassungsstufen auch auf untergeordnete Stufen auswirken können. Diese Änderungen werden jedoch nicht in den untergeordneten Stufen dokumentiert, das endgültige Analyseergebnis für eine Erfassungsstufe setzt sich also aus allen drei Iterationen zusammen.

3.3.2 Erste Iteration

In der ersten Iteration sind die Objekte und möglichen Aktionen bewusst auf eine überschaubare Menge und Komplexität begrenzt. Dadurch sollen die ersten Schritte im Entwurf und in der Implementierung vereinfacht und eine solide Grundlage für weitere Iterationen geschaffen werden.

Voraussetzung für ein funktionsfähiges, webbasiertes Informationssystem, welches den Anforderungen der ersten Erfassungsstufe gerecht wird, sind zum einen die Grundfunktionalitäten von typischen Webanwendungen, wie Anmeldung, Abmeldung und die Verwaltung von Benutzerkonten. Darüber hinaus sind Funktionen zur Verwaltung von Grunddatensätzen und Fotos notwendig. Der Begriff Verwaltung fasst in diesem Zusammenhang Funktionen zum Erstellen, Bearbeiten und Löschen von Datensätzen zusammen.

Die genannten Funktionen der ersten Erfassungsstufe lassen sich in drei Use-Cases zusammenfassen: die Erfassung von Grunddaten von einem Naturereignis, die Erstellung eines Benutzerkontos von einem Administrator und das Hinzufügen eines Fotos zu einem bestehenden Grunddatensatz. Alle drei Anwendungsfälle werden, angelehnt an die Use-Case-Schablone aus Balzert (2005), im Folgenden beschrieben und in Abbildung 3.3 als einfaches Use-Case-Diagramm zusammenfassend visualisiert. Im Vordergrund der Use-Cases steht die Erfassung, während das Bearbeiten und Löschen von Datensätzen als Spezialfälle der Erfassung angesehen werden.

Use-Case 1: Grunddaten erfassen

Ziel: Grunddaten eines Naturereignisses in das System eingeben
Vorbedingung: Grunddaten eines Naturereignisses müssen vorliegen
Nachbedingung bei Erfolg: Grunddaten im System gespeichert
Nachbedingung bei Fehlschlag: System im ursprünglichen Zustand
Akteure: Alle registrierten Benutzer des Systems
Auslösendes Ereignis: Grunddaten im Feld erfasst oder erhalten

Beschreibung:
1 Der Benutzer meldet sich im System an.
2 Der Benutzer legt einen neuen Grunddatensatz an.
3 Der Benutzer gibt die Grunddaten ein.
4 Der Benutzer gibt die Position des Ereignisses an.
5 Der Benutzer speichert den Grunddatensatz.
Alternativen:
1a Der Benutzer bittet den Administrator um eine Registrierung.
2a Der Benutzer überprüft einen bestehenden Grunddatensatz.
3a Der Benutzer aktualisiert gegebenenfalls einen bestehenden Grunddatensatz.
4a Der Benutzer aktualisiert gegebenenfalls die Position eines bestehenden Grunddatensatzes.
5a Der Benutzer löscht gegebenenfalls einen bestehenden Grunddatensatz.

Use-Case 2: Foto hinzufügen

Ziel: Neues Foto zu einem Grunddatensatz hinzufügen
Vorbedingung: Foto zu einem Naturereignis mit Informationen über das Foto müssen vorliegen
Nachbedingung bei Erfolg: Foto im System gespeichert, zu einem Grunddatensatz hinzugefügt
Nachbedingung bei Fehlschlag: System im ursprünglichen Zustand
Akteure: Alle registrierten Benutzer des Systems
Auslösendes Ereignis: Foto zu einem Grunddatensatz aufgenommen oder erhalten

Beschreibung:
1 Der Benutzer meldet sich im System an.
2 Der Benutzer prüft, ob das Foto schon im System erfasst ist.
3 Der Benutzer sucht den entsprechende Grunddatensatz.
4 Der Benutzer fügt einen neuen Fotodatensatz zu dem entsprechenden Grunddatensatz hinzu.
5 Der Benutzer gibt die Informationen über das Foto ein.
6 Der Benutzer speichert den Fotodatensatz.
Erweiterungen:
2e Der Benutzer notiert sich die ID, wenn das Foto schon im System erfasst ist.
3e Der Benutzer erstellt einen neuen Grunddatensatz, wenn die Grunddaten noch nicht im System erfasst sind (siehe Use-Case 1).
5e Der Benutzer gibt Informationen über die Position des Fotos ein, wenn diese nicht im Foto enthalten sind.

Alternativen:

1a Der Benutzer bittet den Administrator um eine Registrierung.

4a Der Benutzer fügt einen bestehenden Fotodatensatz zu einem Grunddatensatz hinzu, wenn das Foto schon im System erfasst ist.

5a Der Benutzer aktualisiert gegebenenfalls die Informationen des bestehenden Fotodatensatzes.

6a Der Benutzer löscht gegebenenfalls einen bestehenden Fotodatensatz.

Use-Case 3: Benutzerkonto erstellen

Ziel: Neues Benutzerkonto im System erstellen
Vorbedingung: Bedarf nach einem neuen Benutzerkonto
Nachbedingung bei Erfolg: Benutzerkonto im System erstellt
Nachbedingung bei Fehlschlag: System im ursprünglichen Zustand
Akteure: Alle registrierten Benutzer des Systems mit der Qualifikation „Administrator“
Auslösendes Ereignis: Anfrage nach einem Benutzerkonto

Beschreibung:

1 Der Administrator meldet sich im System an.

2 Der Administrator legt ein neues Benutzerkonto an.

3 Der Administrator gibt die Informationen über den Benutzer ein.

4 Der Administrator speichert das Benutzerkonto.

Alternativen:

2a Der Administrator überprüft ein bestehendes Benutzerkonto.

3a Der Administrator aktualisiert gegebenenfalls die Informationen des bestehenden Benutzers.

4a Der Administrator löscht gegebenenfalls das Benutzerkonto.

Insgesamt decken diese drei Anwendungsfälle - zusammenfassend visualisiert als Use-Case-Diagramm in Abbildung 3.3 - alle zentralen Funktionen des Informationssystems ab, die für die erste Erfassungsstufe notwendig sind. Die Beschreibung der Funktionen ist sicherlich noch sehr abstrakt, wird aber in den nächsten Schritten weiter differenziert und ausgebaut.

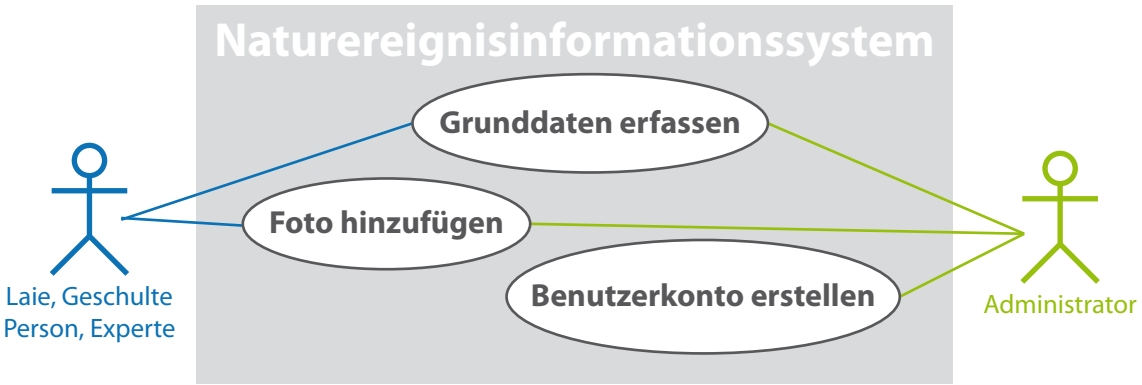


Abbildung 3.3: Klassen-Diagramm der ersten Iteration

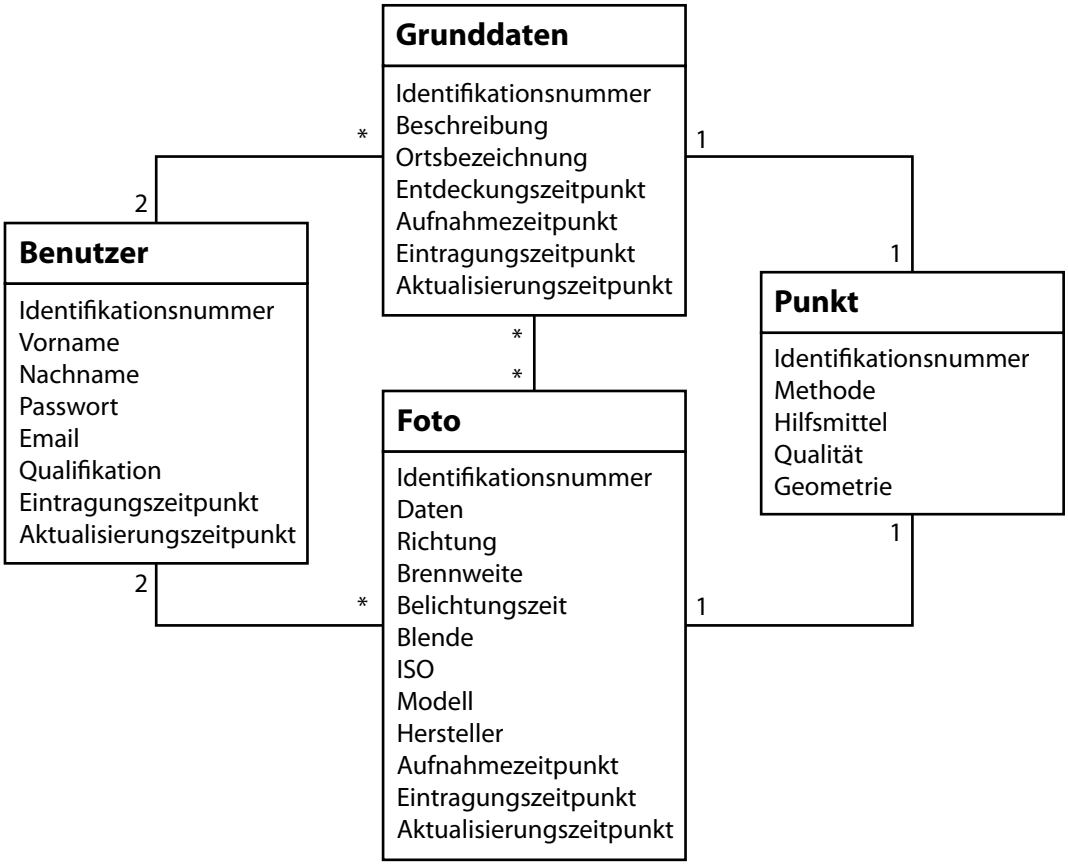


Abbildung 3.4: Klassen-Diagramm der ersten Erfassungsstufe

Die geschilderten Anwendungsfälle und das ER-Diagramm von Jonas Büchel dienen nun als Grundlage für das statische Modell, welches die Klassen und Assoziationen des Systems beschreibt. Insgesamt wurden aus diesen Informationen vier Klassen modelliert und in Abbildung 3.4 als Klassendiagramm visualisiert.

Allgemeine Klassenattribute

Einige Attribute sind - abgesehen von wenigen Ausnahmen - für alle Klassen gültig und werden deshalb nur einmal zu Beginn beschrieben. Nichtsdestotrotz werden die Attribute in den Klassendiagrammen vollständig visualisiert. Die Ausnahmen werden in den entsprechenden Klassen erwähnt.

Identifikationsnummer: Dient zur eindeutigen Identifizierung der Objekte und entspricht im ER-Diagramm dem Entitätstyp *[Obj]Nr*; *[Obj]* steht für die dreistellige Abkürzung des Entitätstyp.
Aufnahmezeitpunkt: Zeitpunkt, zu dem die Informationen im Feld erhoben wurden.
Eintragungszeitpunkt: Zeitpunkt, zu dem die Informationen in die Datenbank eingegeben wurden. Wird automatisch vom System verwaltet.
Aktualisierungszeitpunkt: Zeitpunkt, zu dem die Informationen zuletzt aktualisiert wurden. Wird automatisch vom System verwaltet.

Klasse Benutzer

Die Klasse *Benutzer* ist nicht nur in Use-Case 3 von zentraler Bedeutung, sondern spielt auch in den übrigen Use-Cases als Akteur eine wichtige Rolle. Sie entspricht im ER-Diagramm dem Entitätstyp *Erfasser* und wird ergänzt durch einige systemrelevante Attribute mit Informationen über das Benutzerkonto. Das allgemeine Klassenattribut *Aufnahmzeitpunkt* entfällt für diese Klasse.

Im Gegensatz zu dem Datenbankmodell von Jonas Büchel, das eine 1:N-Beziehung zwischen Erfasser und anderen Entitäten vorsieht, wird für die Klasse Benutzer eine 2:N-Beziehung vorgesehen. Auf diese Weise kann nicht nur die Erfassung, sondern auch die letzte Änderung auf einen Benutzer zurückverfolgt werden. Zusätzlich zu der Beziehung zwischen Erfasser und Grunddaten wird eine weitere 2:N-Beziehung zu der Klasse Foto aufgebaut.

Vorname: Der Vorname des Benutzers; entspricht dem Attribut *ErfVorn*.
Nachname: Der Nachname des Benutzers; entspricht dem Attribut *ErfName*.
Passwort: Das Passwort für das Benutzerkonto, welches zum Anmelden im System benötigt wird; neu hinzugefügt.
Email: Die Emailadresse des Benutzers, welche als Kontaktmöglichkeit und für die Anmeldung im System gebraucht wird; neu hinzugefügt.
Qualifikation: Die Qualifikation eines Benutzer entspricht dem Attribut *ErfQual* und dient in erster Linie der Rechteverwaltung. Sie richtet sich nach den Erfassungsstufen von 1 = Laie (erste Stufe), 2 = geschulte Person (erste und zweite Stufe), 3 = Experte (alle drei Stufen) bis 4 = Administrator (alle drei Stufen und Benutzerverwaltung).

Klasse Grunddaten

Die Klasse *Grunddaten* steht bei Use-Case 1 im Vordergrund und entspricht weitgehend dem gleichnamigen Entitätstyp aus dem ER-Diagramm. Eine größere Änderung gab es nur in Bezug auf den Entitätstyp *GruPoi*, der als Klasse *Punkt* ausgelagert wurde und nun auch in Beziehung zur Klasse *Foto* steht. Neu hinzugekommen sind die Attribute *Beschreibung*, *Entdeckungszeitpunkt* und *Aktualisierungszeitpunkt*.

Beschreibung: Allgemeine Informationen zum Naturereignis; neu hinzugefügt.
Ortsbezeichnung: Lagebezeichnung für den Erfassungsort; entspricht dem Attribut *GruOrts*.
Entdeckungszeitpunkt: Zeitpunkt, zu dem das Naturereignis entdeckt wurde; neu hinzugefügt.

Klasse Foto

Die Klasse *Foto* ist für den Use-Case 2 von zentraler Bedeutung und enthält Informationen sowie die eigentlichen Bilddaten von einem Foto. Im Gegensatz zum ER-Diagramm wurde der Entitätstyp *FotPoi* ausgelagert und zusammen mit dem ebenfalls ausgelagerten *GruPoi* vom Entitätstyp *Grunddaten* als allgemeine Klasse *Punkt* modelliert. Hinzugekommen sind einige Attribute mit Informationen zu Aufnahmeparametern sowie eine Beziehung zur Klasse *Benutzer*, um die Herkunft des Fotos nachvollziehen zu können.

Daten: Die eigentlichen Bilddaten des Fotos; neu hinzugefügt.
Richtung: Angabe zur Aufnahmerichtung; entspricht dem Attribut *FotRich*.
Brennweite: Die Brennweite des Fotos entspricht dem Abstand zwischen Brennpunkt und der Hauptebene einer Linse; neu hinzugefügt.
Belichtungszeit: Die Belichtungszeit entspricht der Zeitspanne, der ein Sensor bzw. Film bei der Aufzeichnung dem Licht ausgesetzt war; neu hinzugefügt.
Blende: Zusammen mit der Belichtungszeit bestimmt die Blendenöffnung die Lichtmenge, der ein Sensor bzw. Film bei der Aufzeichnung ausgesetzt war; neu hinzugefügt.
ISO: Lichtempfindlichkeit des Sensors bzw. Filmmaterials nach ISO-5800-Norm; neu hinzugefügt.
Modell: Die Bezeichnung für das Kameramodell kann zusammen mit dem Attribut *Hersteller* als Grundlage für weitere Recherchen zu dem Foto dienen; neu hinzugefügt.
Hersteller: Der Name des Kameraherstellers; neu hinzugefügt.

Klasse Punkt

Wie oben schon erwähnt, wurden die Informationen über die geographische Lage von Grunddaten und Foto aus den Entitätstypen *GruPoi* und *FotPoi* in die Klasse *Punkt* ausgelagert und vereinheitlicht. Auf diese Weise fallen im Datenbankmodell redundante Spalten weg und die Programmierung wird vereinfacht. Die allgemeinen Klassenattribute sind, abgesehen von der Identifikationsnummer, für diese Klasse nicht notwendig, weil diese in einer assoziierten Klasse gespeichert werden.

Methode: Die Erfassungsmethode unterscheidet zwischen der Erfassung des Standortes am Standort selbst (1 = Felderfassung), der Erfassung des Standortes im Büro (2 = Büroerfassung) und einer Berechnung des Systems zur Bestimmung des Mittelpunktes von mehreren Geometrieobjekten (3 = Berechnung); entspricht dem Attribut *GruPoiMeth*.
Hilfsmittel: Als Hilfsmittel zur Bestimmung des Standortes können folgende Attribute angegeben werden: 1 = Meldung, 2 = Karte, 3 = Orthofoto, 4 = Messband, = 5 Laser, 6 = GPS, 7 = DGPS und 8 = Fernerkundung. Die Berechnung des Mittelpunktes von mehreren Geometrieobjekten entspricht dem Wert 9.
Qualität: Die Genauigkeit der Erfassung kann qualitativ mit 1 = hoch, 2 = mittel und 3 = niedrig bewertet werden.
Geometrie: Speicherung der räumlichen Daten als Koordinatenpaar.

Eine detailliertere, teilweise auch übersichtlichere Darstellung der Use-Cases kann mit Hilfe von Aktivitätsdiagrammen erreicht werden. Die oben beschriebenen Use-Cases werden in zwei Aktivitätsdiagrammen visualisiert. Diese sind nicht nur für die beschriebenen Objekte und Klassen gültig, sondern lassen sich teilweise auch auf andere Klassen anwenden.

Die in Use-Case 1 geschilderte Erstellung, Bearbeitung und Löschung, kurz die Verwaltung von Grunddatensätzen, wird in Abbildung 3.5 möglichst ausführlich als Aktivitätsdiagramm visualisiert. Ziel der Abbildung ist es, möglichst alle in Frage kommenden Aktionen, die sich auf Grunddatensätze anwenden lassen, zu berücksichtigen.

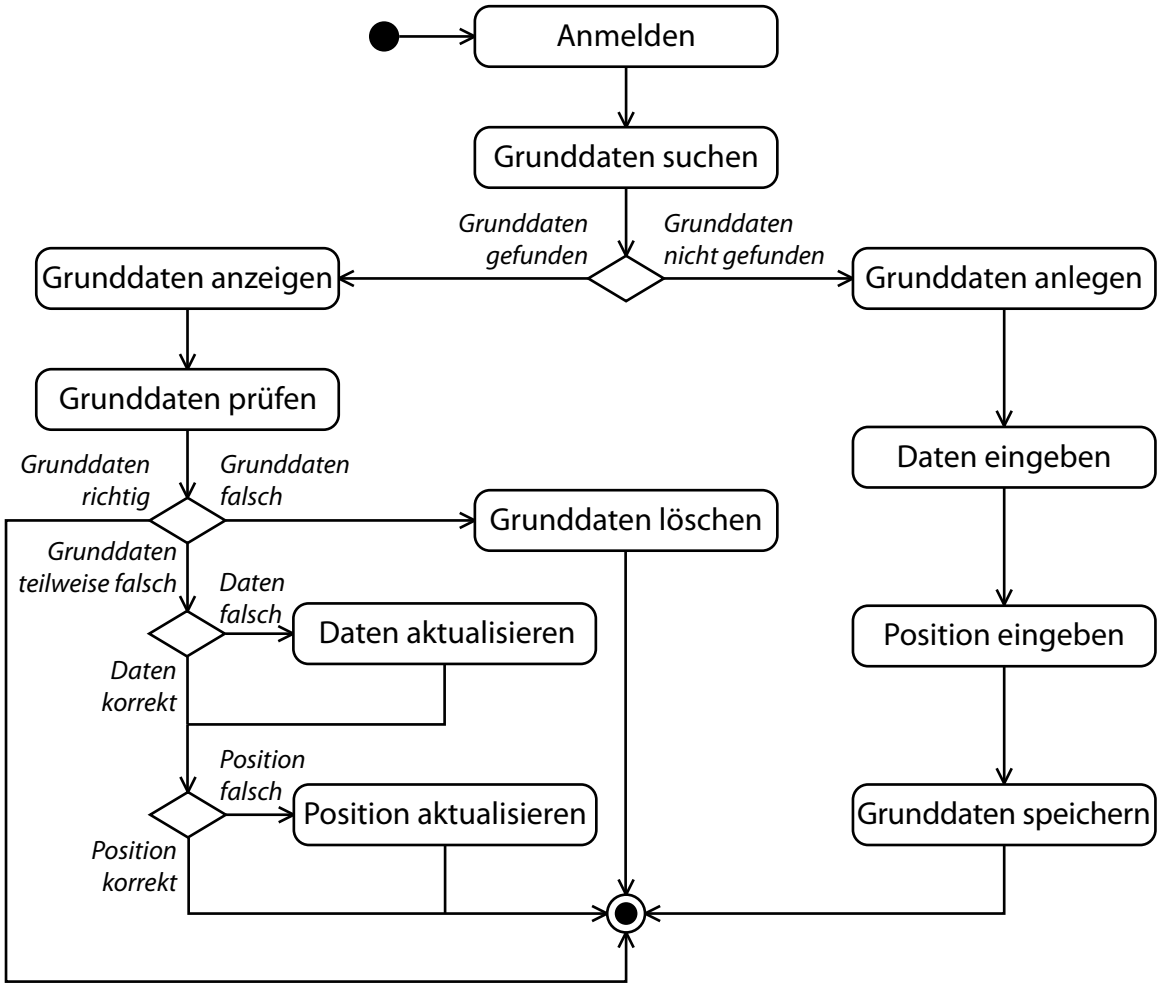


Abbildung 3.5: Aktivitätsdiagramm der Verwaltung von Grunddatensätzen

Die Verwaltung von Fotodatensätzen, beschrieben in Use-Case 2, wird in Abbildung 3.6 als Aktivitätsdiagramm visualisiert. Neben den üblichen und weitgehend selbsterklärenden Funktionen, wie sie beispielsweise auch für Grunddatensätzen zur Verfügung stehen, kann mit Hilfe der Funktion *Foto hinzufügen* ein Fotodatensatz zu einem weiteren Grunddatensatz hinzugefügt werden. Mit Ausnahme der Funktionen, welche räumliche Daten betreffen, gilt dieses Diagramm ebenfalls für die Klasse Dokument (siehe Kapitel 3.3.2).

Auf eine Visualisierung von Use-Case 3 wird verzichtet und auf Abbildung 3.5 verwiesen. Die Benutzerkontenverwaltung entspricht weitgehend der Verwaltung von Grunddatensätzen, mit der Ausnahme, dass nur Administratoren als Akteure in Frage kommen und keine Positionsinformationen erfasst werden müssen.

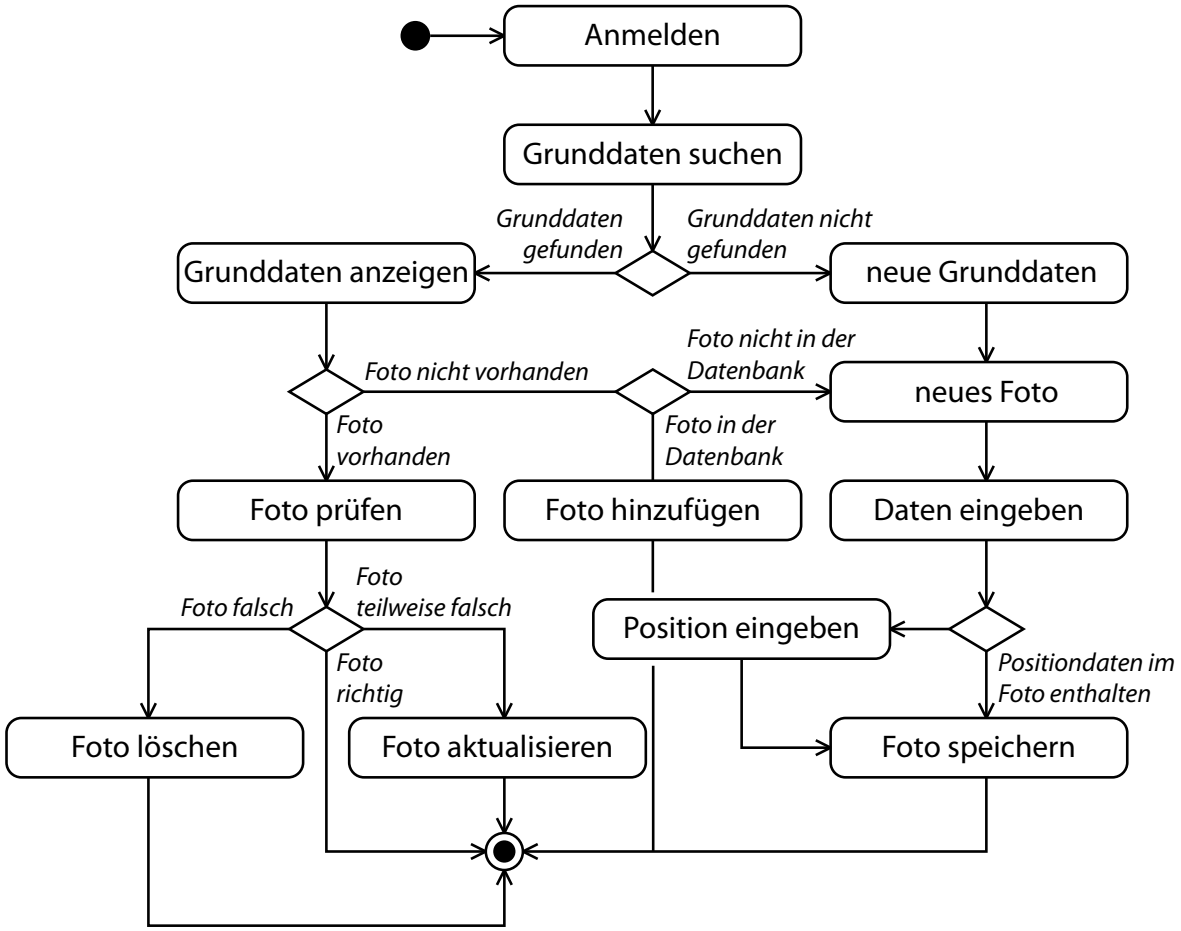


Abbildung 3.6: Verwaltung von Fotodatensätzen als Aktivitätsdiagramm

3.3.2 Zweite Iteration

In der zweiten Iteration kommen die Entitätstypen *Ereignis* und *Dokument* aus dem ER-Diagramm ins Spiel. Das Informationssystem muss also um die Handhabung von allgemeinen Ereignisdaten und Dokumenten erweitert werden; folglich müssen auch die Abfragemöglichkeiten ergänzt werden. Außerdem ist eine differenziertere Verwaltung der Benutzerrechte notwendig, da nur Benutzer mit der entsprechenden Qualifikation Daten der zweiten Erfassungsstufe bearbeiten dürfen. Damit bleiben die funktionalen Erweiterungen in einem überschaubaren Rahmen.

Die neuen Objekte *Ereignisdaten* und *Dokument* bringen jeweils einen Anwendungsfall mit sich. Die Erfassung und Bearbeitung von allgmeinen Ereignisdaten wird in Use-Case 4 beschrieben, während das Hinzufügen von einem Dokument zu einem Ereignisdatensatz in Use-Case 5 erläutert wird. Um den Überblick über die Anwendungsfälle der ersten und zweiten Erfassungsstufe zu unterstützen, werden Use-Case 1 bis 5 in Abbildung 3.7 als Use-Case-Diagramm zusammenfassend visualisiert.

Use-Case 4: Ereignisdaten erfassen

Ziel: Ereignisdaten eines Naturereignisses in das System eingeben
Vorbedingung: Ereignisdaten eines Naturereignisses müssen vorliegen
Nachbedingung bei Erfolg: Ereignisdaten im System gespeichert
Nachbedingung bei Fehlschlag: System im ursprünglichen Zustand
Akteure: Alle registrierten Benutzer des Systems ab der Qualifikationsstufe „Geschulte Person"
Auslösendes Ereignis: Ereignisdaten im Feld erfasst oder erhalten

Beschreibung:
1 Der Benutzer, ab der Qualifikationsstufe „Geschulte Person", meldet sich im System an.
2 Der Benutzer sucht den entsprechenden Grunddatensatz.
3 Der Benutzer fügt einen neuen Ereignisdatensatz dem entsprechenden Ereignis hinzu.
4 Der Benutzer gibt die Ereignisdaten ein.
5 Der Benutzer speichert den Ereignisdatensatz.
Erweiterungen:
2e Der Benutzer erstellt einen neuen Grunddatensatz, wenn die Grunddaten nicht im System erfasst sind (siehe Use-Case 1).
Alternativen:
1a Der Benutzer bittet den Administrator um eine Registrierung bzw. Änderung der Qualifikation.
3a Der Benutzer überprüft einen bestehenden Ereignisdatensatz.
4a Der Benutzer aktualisiert gegebenenfalls einen bestehenden Ereignisdatensatz.
5a Der Benutzer löscht gegebenenfalls einen bestehenden Ereignisdatensatz.

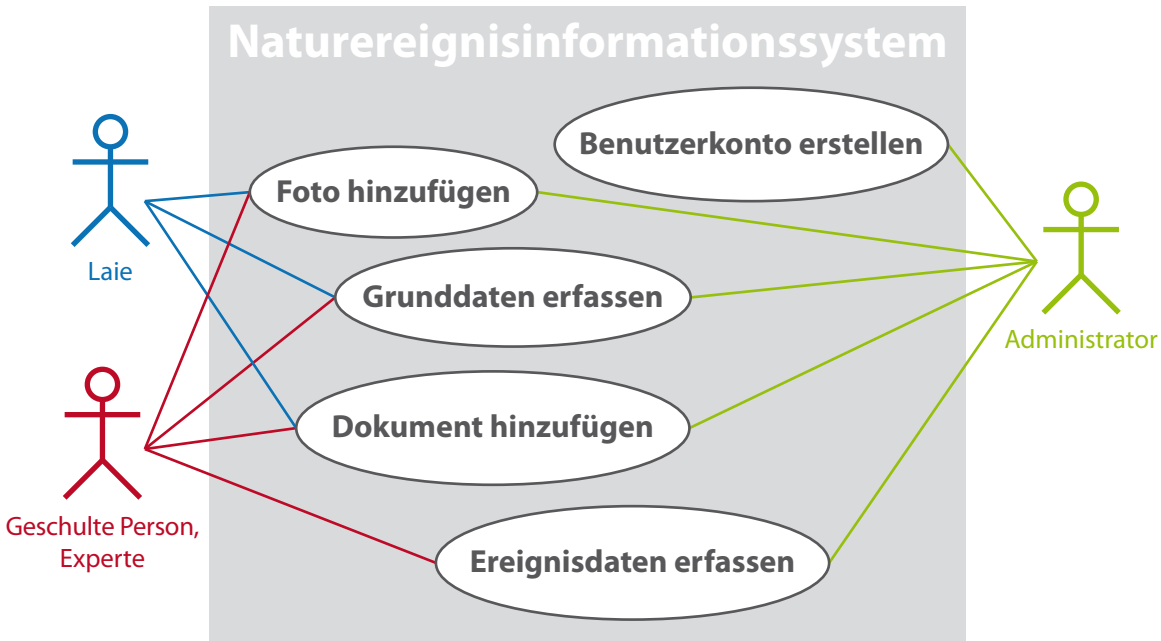


Abbildung 3.7: Use-Case-Diagramm der ersten und zweiten Iteration

Use-Case 5: Dokument hinzufügen

Ziel: Neues Foto zu einem Grunddatensatz hinzufügen
Vorbedingung: Foto zu einem Naturereignis mit Informationen über das Foto müssen vorliegen
Nachbedingung bei Erfolg: Foto im System gespeichert und zu einem Ereignis hinzugefügt
Nachbedingung bei Fehlschlag: System im ursprünglichen Zustand
Akteure: Alle registrierten Benutzer des Systems
Auslösendes Ereignis: Foto zu einem Grunddatensatz aufgenommen oder erhalten

Beschreibung:
1 Der Benutzer meldet sich im System an.
2 Der Benutzer prüft, ob das Dokument schon im System erfasst ist.
3 Der Benutzer sucht den entsprechenden Ereignisdatensatz.
4 Der Benutzer fügt einen neuen Dokumentdatensaz zu dem entsprechenden Ereignis hinzu.
5 Der Benutzer gibt die Informationen über das Dokument ein.
6 Der Benutzer speichert den Dokumentdatensaz.
Erweiterungen:
2e Der Benutzer notiert sich die ID, wenn das Dokument schon im System erfasst ist.
3e Der Benutzer erstellt einen neuen Ereignisdatensatz, wenn die Ereignisdaten noch nicht im System erfasst sind (siehe Use-Case 4).
Alternativen:
1a Der Benutzer bittet den Administrator um eine Registrierung.
4a Der Benutzer fügt einen bestehenden Dokumentdatensaz zu einem Ereignisdatensatz hinzu, wenn das Dokument schon im System erfasst ist.
5a Der Benutzer aktualisiert ggf. die Informationen des bestehenden Dokumentdatensaz.
6a Der Benutzer löscht gegebenenfalls einen bestehenden Dokumentdatensaz.

Im Gegensatz zu den funktionalen Erweiterungen sind die strukturellen Änderungen und Erweiterungen in der zweiten Iteration von größerer Bedeutung. Während der Programmierung wurde deutlich, dass eine Klasse notwendig ist, welche alle Informationen von einem Ereignis verknüpft und so den Zugriff auf die Informationen erleichtert. Diese Klasse und die Klassen der Entitätstypen *Ereignis* und *Dokument* sollen kurz beschrieben werden. Wiederrum gelten die allgemeinen Klassenattribute, wenn nicht anders erwähnt.

Klasse Ereignis

Die Klasse Ereignis entspricht nicht dem Entitätstyp *Ereignis*, sondern dient als Verbindungsknoten für alle Daten und Informationen zu einem Naturereignis. Ohne eine solche Klasse müssten die entsprechenden Objekte von einem Ereignis einzelnd abgefragt und verarbeitet werden. Einzige Aufgabe der Klasse Ereignis ist es also, alle Informationen zu einem Ereignis zusammenzuführen und auf diese Weise die Verarbeitung zu erleichtern und den Informationszugriff auf einen Punkt zu konzentrieren. Während die Objekte der Grund- und Ereignisdaten innerhalb der Klasse repräsentiert werden, sind Fotos und Dokumente über Assoziationen mit dem Ereignis verknüpft. Abgesehen von der Identifikationsnummer sind die allgemeinen Klassenattribute nicht für diese Klasse gültig.

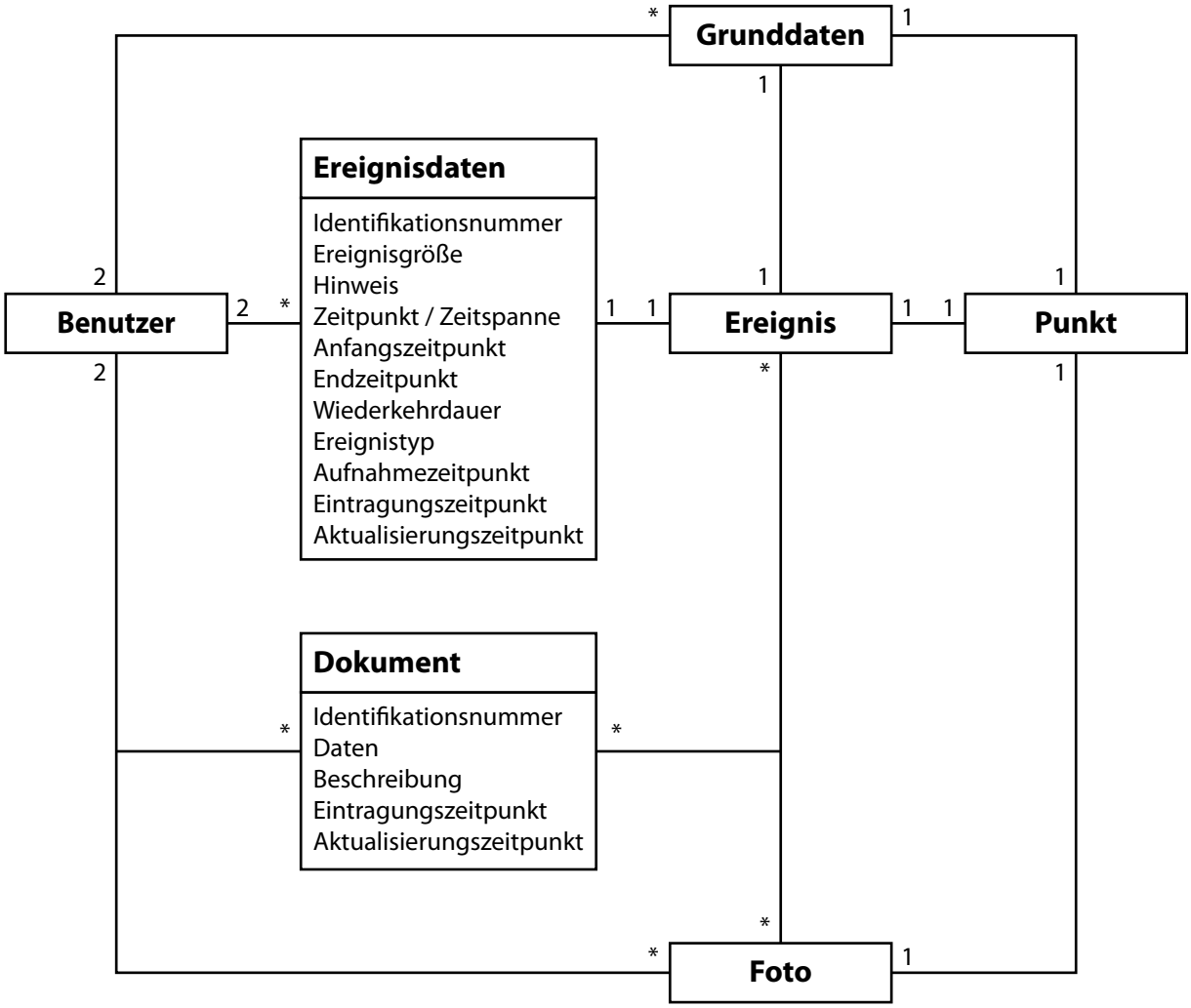


Abbildung 3.8: Klassen-Diagramm der zweiten Erfassungsstufe

Klasse Ereignisdaten

Der Entitätstyp *Ereignis* spiegelt sich in der Klasse *Ereignisdaten* wieder und enthält - ähnlich wie die Klasse Grunddaten - Attribute, welche sämtliche Ereignisse gemeinsam haben. Die Trennung zwischen Ereignis- und Grunddaten begründet sich in der Anforderung, dass bestimmte Informationen nur von geschulten Mitarbeitern in einem zweiten Schritt erfasst werden sollen. Diese Informationen werden von der Klasse Ereignisdaten repräsentiert und kommen in Use-Case 4 zur Anwendung.

Einige Attribute in dieser Klasse werden durch den so genannten MAXO-Code um eine Angabe zur qualitativen Genauigkeit ergänzt. Zu Gunsten der Übersicht wird dieses Attribut nicht in dem Klassendiagramm dargestellt. Zur Auswahl stehen:

M = Messung bzw. detaillierte, methodisch klar dokumentierte Feststellung (1)

A = Annahme bzw. grobe Schätzung durch eine geschulte Person (2)

X = Wert unklar, noch zu erheben (3)

O = Wert nicht bestimmbar (4)

Ereignisgröße: Zuordnung einer Größenordnung zwischen 1m (1), 10m bzw. 100m^2 (2), 100m bzw. 10 000m^2 (3), 1km bzw. 1km^2 (4) und 10km bzw. 100km^2 ohne aufwendige Messung oder Berechnung. Entspricht dem Attribut *EreGros*.
Hinweis: Für die Aufname der Raumdaten in der dritten Erfassungsstufe können dem zukünftigen Erfasser Hinweise zur Aufnahmemethode gegeben werden. Entspricht dem Attribut *EreHinw*.
Ereigniswiederkehr: Ein gleichartiges Ereignis an derselben Stelle wird als wiederkehrendes Ereignis verstanden. Die Wiederkehrdauer kann entweder täglich (1), wöchentlich (2), monatlich (3), jährlich (4), alle 30 Jahre (5) oder alle 100 Jahre (6) sein. Wenn das Ereignis kein wiederkehrendes Ereignis ist, wird der Wert 0 angegeben. Entspricht dem Attribut *EreWkehr* und wird ergänzt durch einen MAXO-Code.
Zeitpunkt / Zeitspanne: Gibt an, ob die Attribute Anfangszeitpunkt und Endzeitpunkt als direkte Beobachtung oder als Zeitspanne verstehen zu sind. Wird durch einen MAXO-Code ergänzt und entspricht dem Attribut *EreZeit*.
Beginnzeitpunkt: Entweder wird ein konkreter Zeitpunkt des Beginns oder der Beginn einer Zeitspanne angegeben.
Endzeitpunkt: Entweder wird ein konkreter Endzeitpunkt oder das Ende einer Zeitspanne angegeben.
Ereignistyp: Das Ereignis soll einem der folgenden Ereignistypen zugeordnet werden: Rutschung (1), Sturzbewegung (2), Lawine (3), Überschwemmung/Murgang (4), Seitenerosion (5), Quelle neu/versiegt (6), Insekt (7), Nagetier (8), Wild (9), Brand (10), Dürre (11), Frost (12), Schneebruch (13), Windwurf (14), Blitzschlag (15), Anthropogene Veränderung (16), Fallwild (17), Lebensraumbeeinträchtigung (18). Wird druch einen MAXO-Code ergänzt und entspricht dem Attribut *ErePraz*.

Klasse Dokument

Die Klasse Dokument kann als „kleine Schwester" der Klasse Foto betrachtet werden. Sie weist wesentlich weniger Attribute auf, so fehlen ihr beispielsweise das allgemeine Klassenattribut *Aufnahmezeitpunkt* sowie zahlreiche Metadaten, und sie baut keine Beziehung zur Klasse Punkt auf, da Dokumente in der Regel nicht direkt georeferenziert werden können. Ansonsten sind Handhabung, beschrieben in Use-Case 5 und visualisiert in Abbildung 3.6, sowie Assoziationen der beiden Klassen vergleichbar.

Beschreibung: Kurze inhaltliche Beschreibung des Dokuments; entspricht dem Attribut *DokPraz*.
Daten: Die eigentlichen Binärdaten des Dokuments; entspricht dem Attribut *DokDok*.

Eine Visualisierung der Verwaltung von Ereignisdatensätzen - und damit von Use-Case 4 - als Aktivitätsdiagramm entspricht der Abbildung 3.9. Es lassen sich einerseits viele Ähnlichkeiten zu dem Aktivitätsdiagramm der Verwaltung von Grunddatensätzen feststellen. Andererseits finden sich ebenso entscheidende Unterschiede. So entfällt die Erfassung der geographischen Lage und das Vorhandensein von Grunddaten ist als Bedingung hinzugekommen.

Ein Aktivitätsdiagramm von Use-Case 5 entspricht weitgehend dem Aktivitätsdiagramm der Verwaltung von Fotodatensätzen (siehe Abbildung 3.6) und wird daher als überflüssig angesehen. Einzig die Aufnahme der geographischen Lage entfällt für die Erfassung von Dokumenten.

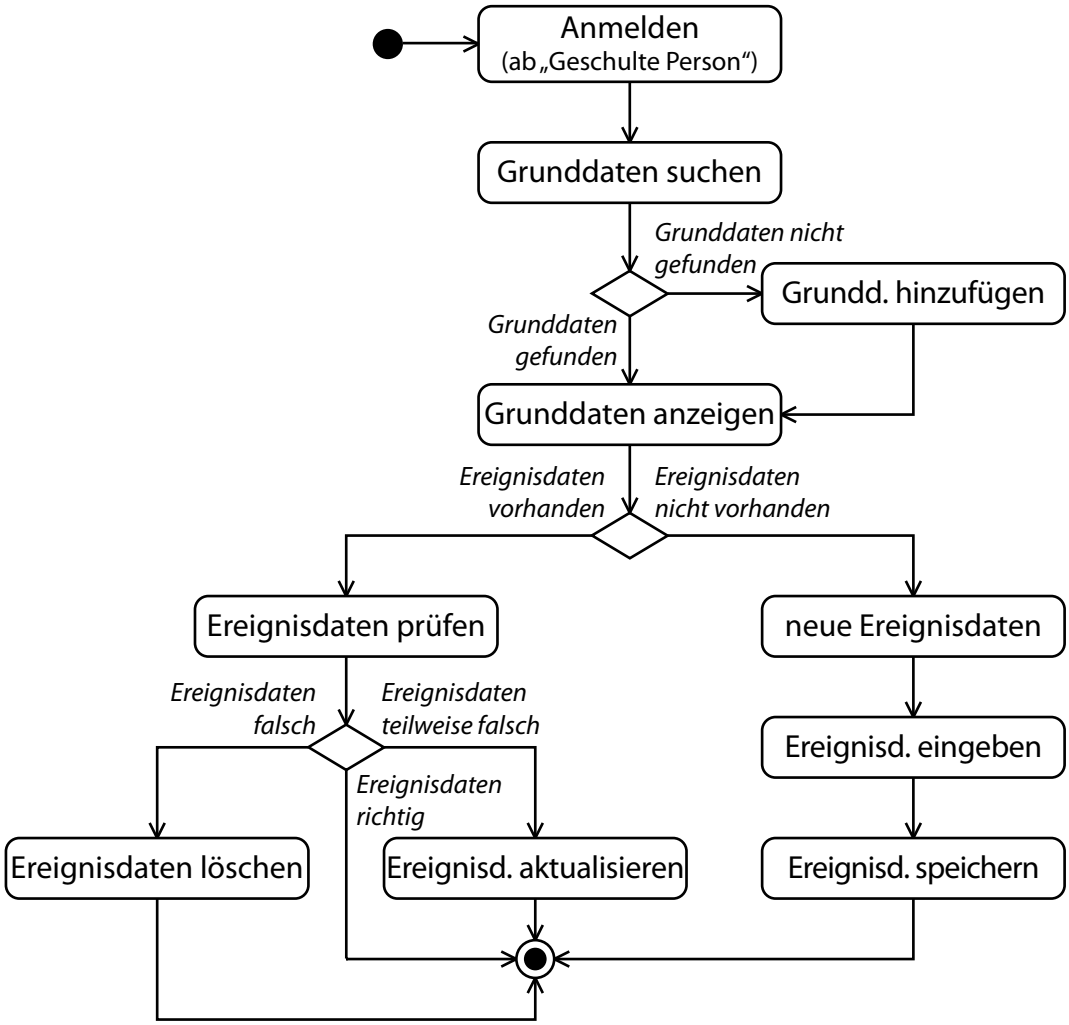


Abbildung 3.9: Aktivitätsdiagramm der Verwaltung von Ereignisdatensätzen

3.3.3 Dritte Iteration

Die dritte Iteration betrifft die Aufnahme von Ereignisdaten, die spezifisch für einen Ereignistyp sind und nur von Benutzern mit der Qualifikation *Experte* aufgenommen werden. Die Anzahl der zu erfassenden Attribute ist für die dritte Erfassungsstufe mit Abstand am höchsten und während der Realisierung mit einem entsprechenden Aufwand verbunden. Deshalb beschränkt sich die Analyse und Realisierung auf den Ereignistyp Lawine. Weitere Ereignisypen lasssen sich später auf die gleiche Weise hinzufügen.

Use-Case 6: Lawinendaten erfassen

Ziel: Lawinendaten eines Naturereignisses in das System eingeben
Vorbedingung: Lawinendaten eines Naturereignisses müssen vorliegen
Nachbedingung bei Erfolg: Lawinendaten im System gespeichert
Nachbedingung bei Fehlschlag: System im ursprünglichen Zustand
Akteure: Alle registrierten Benutzer des Systems, ab der Qualifikationsstufe „Experte“
Auslösendes Ereignis: Lawinendaten im Feld erfasst oder erhalten

Beschreibung:

1 Der Benutzer, ab der Qualifikationsstufe „Experte“, meldet sich im System an.
2 Der Benutzer sucht den entsprechenden Ereignisdatensatz.
3 Der Benutzer fügt einen neuen Lawinendatensatz dem entsprechenden Ereignis hinzu.
4 Der Benutzer gibt die Lawinendaten ein.
5 Der Benutzer erfasst die geographische Lage der Lawine.
6 Der Benutzer fügt gegebenenfalls von der Lawine betroffene Objekte hinzu.
7 Der Benutzer speichert den Lawinendatensatz.

Erweiterungen:

2e Der Benutzer erstellt einen neuen Ereignisdatensatz, wenn die Ereignisdaten nicht im System erfasst sind (siehe Use-Case 4).

Alternativen:

1a Der Benutzer bittet den Administrator um eine Registrierung oder eine Änderung der Qualifikation.
3a Der Benutzer überprüft einen bestehenden Lawinendatensatz.
4a Der Benutzer aktualisiert gegebenenfalls einen bestehenden Lawinendatensatz.
5a Der Benutzer aktualisiert gegebenenfalls die geographische Lage der Lawine.
7a Der Benutzer löscht gegebenenfalls einen falschen Lawinendatensatz.

Alle sechs Use-Cases werden abschließend in Abbildung 3.10 als Use-Case-Diagramm zusammengefasst. Obwohl bei Weitem nicht alle Funktionen in dieser Abbildung enthalten sind, werden zumindestens die Schwerpunkte und zentralen Aufgaben der Anwendung in den Vordergrund gestellt.

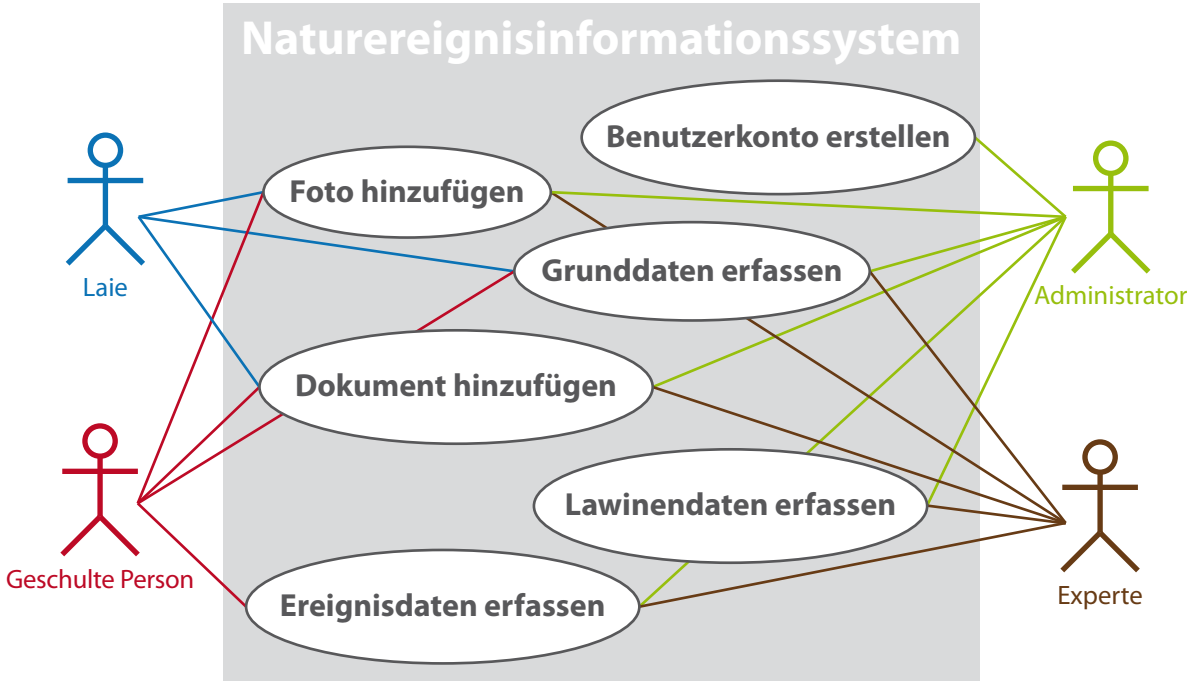


Abbildung 3.10: Use-Case-Diagramm der ersten bis dritten Erfassungsstufe

Die Klassen der dritten Erfassungsstufe werden im Folgenden ausführlich beschrieben und in Abbildung 3.11 als Klassen-Diagramm dargestellt. Neu hinzugekommen sind neben der Klasse La-

winendaten, die Klassen Linie und Fläche sowie die Klasse Betroffenes Objekt mit den Unterklassen Infrastruktur, Lebewesen und Baum hinzugekommen.

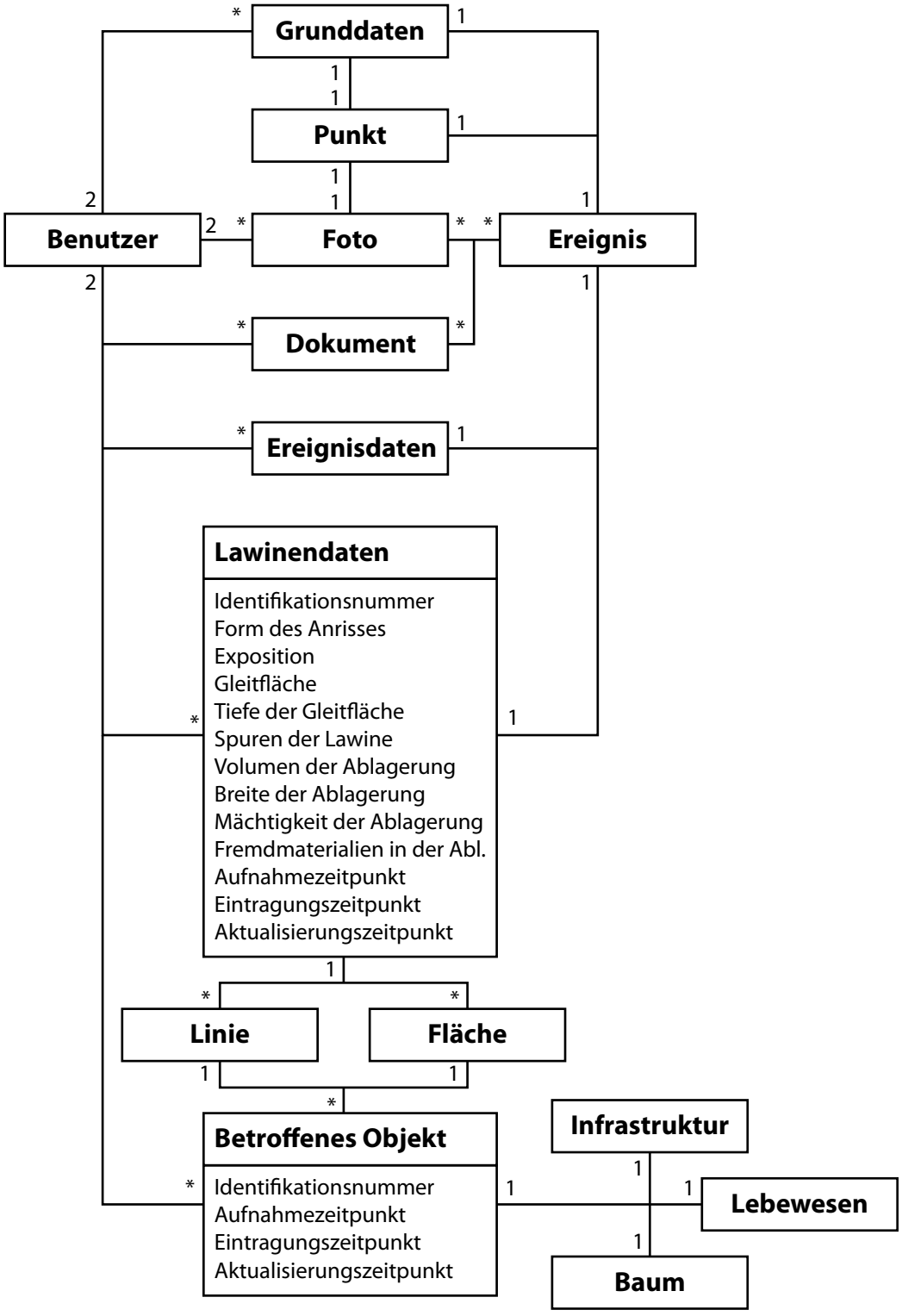


Abbildung 3.11: Klassen-Diagramm der dritten Erfassungsstufe am Beispiel Lawine

Klasse Lawinendaten

Die Klasse Lawinendaten umfasst alle Daten, die spezifisch für den Ereignistyp Lawine sind. Ähnlich wie die Klassen Grunddaten und Ereignisdaten steht die Klasse Lawinendaten in Beziehung zu den Klassen Benutzer und Ereignis. Darüber hinaus besteht eine 1:N-Beziehung zu den Klassen Linie und Fläche; auf diese Weise wird eine exakte räumliche Abgrenzung des Ereignis ermöglicht.

Form des Anrisses: Die Form des Anrisses kann als punktförmig (1) oder linienförmig (2) angegeben werden. Entspricht dem Attribut *LawForA*.
Exposition: Die Hangexposition des Anrissgebeites wird in Grad zwischen 0 und 359 beschrieben. Wird ergänzt durch einen MAXO-Code und entspricht dem Attribut *LawExpo*.
Gleitfäche: Unterscheidet zwischen Existenz (1) und Nichtexistenz (0) einer Gleitfläche; entspricht dem Attribut *LawGlei*.
Tiefe der Gleitfäche: Von der Oberfläche gemessene oder geschätzte Tiefe der Gleitfläche in Meter. Wird ergänzt durch einen MAXO-Code und entspricht dem Attribut *LawTief*.
Spuren der Lawine: Beschreibung von Spuren der Lawine, z.B. Fließspuren. Entspricht dem Attribut *LawSpur*.
Volumen der Ablagerung: Volumen der Ablagerung in Quadratmeter; entspricht dem Attribut *LawVolu* und wird durch einen MAXO-Code ergänzt.
Breite der Ablagerung: Maximale Breite der Ablagerung in Meter; entspricht dem Attribut *LawBrei* und wird ebenfalls durch einen MAXO-Code ergänzt.
Mächtigkeit der Ablagerung: Maximale Mächtigkeit der Ablagerung in Meter unter Verwendung eines MAXO-Codes; entspricht dem Attribut *LawMach*.
Fremdmaterialien der Ablagerung: Beschreibung von Fremdmaterial in der Ablagerung, wie Bäume, Sträucher oder Steine. Entspricht dem Attribut *LawFreM*.

Klasse Linie und Klasse Polygon

Die Klassen Linie und Polygon entsprechen, abgesehen von ihren Beziehungen zu anderen Klassen und dem Geometrietyp, der Klasse Punkt. Im Gegensatz zu der Klasse Punkt, stehen Linien und Polygone in einer 1:N Beziehung zu der Klasse Lawine und der Klasse Betroffenes Objekt.

Klasse Betroffenes Objekt

Ein betroffenes Objekt kann entweder ein Baum, ein Lebewesen oder die Infrastruktur in Form von Gebäuden oder Verbindungen sein und wird einer Linie oder einem Polygon zugeordnet. Die Klasse Betroffenes Objekt dient einzig als Verbindung zwischen den Raumdaten und den Klassen der entsprechenden Objekte, daher sind abgesehen von den allgemeinen Klassenattributen keine weitere Attribute vorhanden. Auf eine detaillierte Beschreibung oder Visualisierung der Klassen Baum, Lebewesen und Infrastruktur wird verzichtet, weil diese auf Grund zeitlicher Einschränkung in der Realisierung nicht berücksichtigt werden.

Aktivitäten, die auf Grundlage von Lawinendatensätzen ausführbar sein sollen, werden in Abbildung 3.12 als Aktivitätsdiagramm visualisiert.

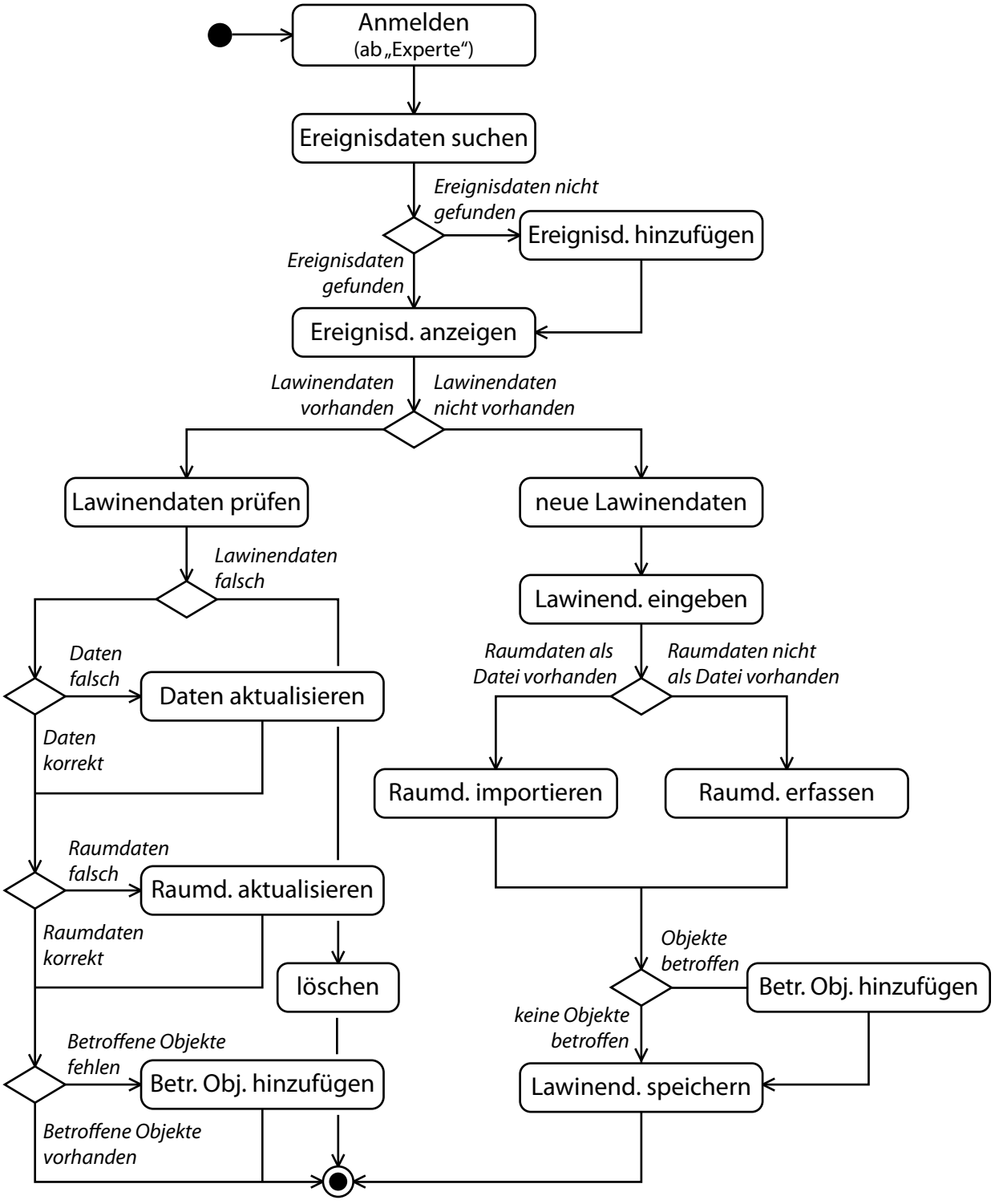


Abbildung 3.12: Aktivitätsdiagramm der Verwaltung von Lawinendatensätzen

4 Softwareuntersuchung

Im folgenden Kapitel soll Open Source Software, die für die Realisierung des Naturereignisinformationssystems in Frage kommt, identifiziert, beschrieben und gegebenenfalls verglichen werden. Die Untersuchung wurde in vier Kategorien aufgeteilt, die jeweils in einem separaten Kapitel behandelt werden.
Neben einer kurzen Einführung in Geodatenbanksysteme und einem Überblick über kommerzielle Lösungen werden in **Kapitel 4.1** zwei der fortgeschrittensten Open Source Geodatenbanksysteme gegenübergestellt. Anschließend werden in **Kapitel 4.2** Programmbibliotheken für die Handhabung von Geodatenstrukturen und die Anbindung an Geodatenbanksysteme untersucht. Im folgenden Kapitel sollen für die weit verbreiteten Entwicklungsplattformen von Webanwendungen Projekte identifiziert werden, die als Grundlage für die Entwicklung der WebGIS-Anwendung dienen können. Abschließend werden in **Kapitel 4.4** verschiedene Möglichkeiten zur Visualisierung von Geodaten im Browser untersucht.

4.1 Geodatenbanksysteme

4.1.1 Einleitung

Im Allgemeinen dienen *Datenbanksysteme* (DBS) der persistenten, also dauerhaften Speicherung von Daten und bestehen aus zwei Komponenten, die in Abbildung 4.1 dargestellt sind. Zum einen ist das die eigentliche *Datenbank* (DB), eine Sammlung der strukturiert und persistent gespeicherten Daten. Das *Datenbankmanagementsystem* (DBMS) funktioniert als Schnittstelle zwischen dem Anwender und der Datebank und ermöglicht die Verwaltung der Daten (Brinkhoff, 2008).

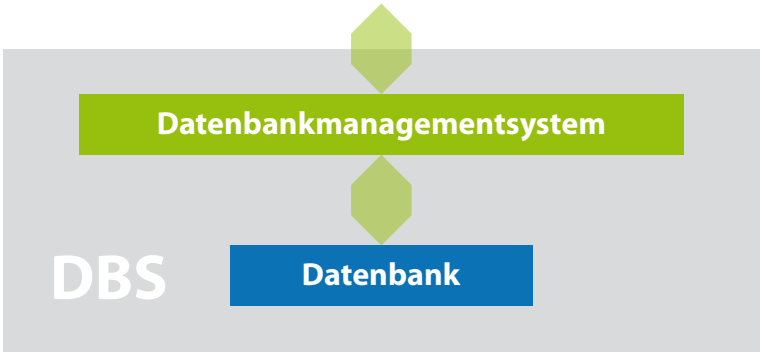


Abbildung 4.1: Komponenten eines Datebanksystems nach Brinkhoff (2008)

Im Gegensatz zu Datei-basierten Speicherungsmethoden bieten Datenbanksysteme nach Brinkhoff (2008) eine Reihe von Vorteilen. Durch *Datenunabhängigkeit* sind Änderungen hinsichtlich der Speicherung der Daten möglich, ohne die Datenbankanwendung zu beeinflussen. Eine *zentrale Datenhaltung* auf einem Server stellt allen Benutzern die Daten in gleicher Aktualität zur Verfügung und ermöglicht durch *Mehrbenutzerbetrieb* unterschiedlichen Benutzern zeitgleich Lesenden oder Schreibenden Zugriff auf einen Datenbestand. Dabei können für einzelne Benutzer oder ganze Benutzergruppen Systemberechtigungen vergeben und so eine sehr genaue Zugriffskontrolle erreicht werden. Das *Transaktionskonzept* (siehe unten) gewährleistet einen konsistenten Zustand der Daten vor und nach einer Datenbankaktion. Weiterhin vermeidet ein Datenbanksystem, ein entsprechendes Datenbankmodell vorausgesetzt, die redundante, also mehrfache Speicherung derselben Information. Mit Hilfe von *Integritätsbedingungen* kann sichergestellt werden, dass bei der Erfassung und Fortführung der Daten keine Widersprüche im Datenbestand entstehen.

Das Transaktionskonzept kann mit Hilfe der so genannten ACID-Eigenschaften, welche für Datenbanksystemen wünschenswert sind und erstmals von Theo Härder und Andreas Reuter (1984) erwähnt wurden, beschrieben werden. Die Abkürzung steht für die Anfangsbuchstaben der Begriffe Atomicity, Consistency, Isolation und Durability; zu deutsch Atomizität, Konsistenz, Isolation und Dauerhaftigkeit (Elmasri & Navathe, 2005).

Atomizität: Eine Transaktion wird als atomare Verarbeitungseinheit entweder ganz oder gar nicht ausgeführt werden.
Konsistenz: Die Datebank wird durch eine Transaktion von einem konsistenten Zustand in einen anderen überführt.

Isolation: Eine Transaktion wird unabhängig von anderen Transaktionen ausgeführt.
Dauerhaftigkeit: Die Änderungen einer Transaktion werden in der Datenbank dauerhaft festgehalten.

Gleichzeitig werden von Brinkhoff (2008) aber auch einige Nachteile genannt. So ist die zentrale Datenhaltung besonders anfällig für Datenverlust, eine Backupstrategie ist daher zentraler Bestandteil einer Datenbankumgebung. Ein weiteres Problem ist die Abhängigkeit von bestimmten Datenbanksystemen, die trotz zahlreicher Standards nicht komplett vermieden werden kann. Außerdem sind Datenbanksysteme mit hohen Investitionskosten in das entsprechende Fachwissen und teilweise auch Lizenzkosten verbunden.

Im Gegensatz zu anderen Softwarebereichen beschränkt sich das Angebot von Geodatenbanksystem sowohl im proprietären als auch im open-source Bereich auf eine Hand voll Anbieter. Die folgende Untersuchung von Geodatenbanksystemen wird sich auf Open Source Software konzentrieren. Da das zu entwickelnde System jedoch anstrebt möglichst unabhängig von einem bestimmten Datenbanksystemen zu sein, sollen proprietäre System nicht vollkommen unberücksichtigt bleiben und als nächstes kurz beschrieben werden.

4.1.2 Kommerzielle Geodatenbanksysteme

Die Firma *IBM* hat zwei Datenbanksysteme im Portfolio und bietet für beide Systeme Erweiterungen für räumliche Anwendungen an. Für *Informix*[1] werden die Erweiterung *Spatial DataBlade* mit OGC-konformer Unterstützung für geographische Datentypen und Methoden sowie *Geodetic DataBlade* zum Speichern von geographische Koordinaten angeboten. Für das Datenbanksystem *DB2*[2] sind mit dem OGC-konformen *Spatial Extender* und dem *Geodetic Data Management Feature* vergleichbare Erweiterungen im Angebot. Beide Systeme unterstützen außerdem räumliche Indizes.

Microsoft unterstützt mit dem *SQL Server 2008*[3] ebenfalls OGC-konforme geometrische Datentypen und einen räumlichen Index. Diese Funktionen sind in das Datenbanksystem integriert und müssen nicht als eine Erweiterung hinzugekauft werden. Ein aktueller und ausführlicher Vergleich des Funktionsumfangs für räumliche Anwendungen mit Open-Source-Systemen findet sich unter (Boston GIS, 2009).

Von der Firma *Oracle* wird für das gleichnamige Datenbanksystem die Erweiterung *Oracle Spatial*[4] angeboten. Diese Erweiterung folgt ebenfalls den OGC-Spezifikationen und stellt umfangreiche geographische Datentypen und Methoden zur Verfügung.

1 http://www.ibm.com/software/data/informix/ [Abruf: 1.9.2009]
2 http://www.ibm.com/software/data/db2/ [Abruf: 1.9.2009]
3 http://www.microsoft.com/germany/sql/2008/ [Abruf: 1.9.2009]
4 http://www.oracle.com/technology/products/spatial [Abruf: 1.9.2009]

4.1.3 Open-Source Geodatenbanksysteme

Im Open-Source Bereich sind die Datenbanksysteme *PostgreSQL*[5] und *MySQL*[6] am weitesten verbreitet. Weiterhin sind noch *SQLite*[7] und *Ingres*[8] erwähnenswert, für die ebenfalls Erweiterungen für räumliche Anwendungen verfügbar sind. Nach einer kurzen Beschreibung dieser Systeme, sollen PostgreSQL und MySQL miteinander verglichen werden, um eine Entscheidung für eines der beiden Produkte zu ermöglichen.

Eine der frühesten Entwicklungen eines relationalen Datenbanksystems begann 1973 an der *University of California at Berkeley* (UCB) mit dem Projekt Ingres. Nachdem der Quellcode in den 1980er und 1990er von verschiedensten Firmen kommerziell vertrieben wurde, ist er heute unter der GPL-Lizenz wieder frei verfügbar und wird von der Firma *Ingres Corporation* weiterentwickelt. Inzwischen wird auch an einer Open Source Erweiterung für räumliche Anwendungen gearbeitet, welche allerdings noch nicht vergleichbar mit dem Funktionsumfang und der Stabilität von anderen Lösungen ist (Ingres, 2009). Nichtsdestotrotz sollte es sich lohnen, die weitere Entwicklung von diesem Projekt im Auge zu behalten.

Als Nachfolger von Ingres wurde im Jahr 1986 ebenfalls an der UBC die Entwicklung am objektrelationalen Datenbanksystem *Postgres* begonnen und bis Mitte der 1990er fortgeführt. Auch Informix (siehe oben), welches 2001 von IBM aufgekauft wurde, hat seine Wurzeln in diesem Umfeld. Im Jahr 1996 wurde die Entwicklung von Programmierern, welche das Potenzial von der Software erkannten, außerhalb der Universität aufgenommen und unter dem Namen PostgreSQL fortgesetzt (PostgreSQL, 2009).

Bis heute wird das Projekt, anders als beispielsweise bei MySQL, von einer internationalen Gruppe von Entwicklern und Unternehmen organisiert und unterstützt. Die Software wird für Linux, Unix-Derivate und Windows unter einer BSD-Lizenz veröffentlicht und hat am 1. Juli 2009 Version 8.4 erreicht. Weil die BSD-Lizenz auch die kommerzielle Verwendung des Quellcodes erlaubt, gibt es inzwischen einige Anbieter von kostenpflichtigen Versionen wie zum Beispiel *EnterpriseDB*[9], welche in der Regel auf einen bestimmten Anwendungsfall spezialisiert sind.

Während die Standardinstallation keine Unterstützung für räumliche Funktionen und Datentypen enthält, können diese mit Hilfe der Erweiterung *PostGIS*[10] nachträglich hinzugefügt werden. Die Entwicklung der Erweiterung wird maßgeblich von der kanadischen Firma *Refractions Research* vorangetrieben und steht aktuell bei Version 1.4, die am 24. Juli 2009 veröffentlicht wurde. Im Gegensatz zu PostgreSQL wird keine BSD-Lizenz, sondern die GPL verwendet.

Die Entwicklung von MySQL begann im Jahr 1995 und hat am 27. November 2008 die Version 5.1 erreicht. Zusammen mit der Programmiersprache *PHP* hat das Datenbanksystem eine weite Verbreitung auf Webservern erreicht und wird von vielen großen Websites eingesetzt. Das Projekt wird sowohl als propritäre und kostenpflichtige Software mit Kundenservice als auch unter der GPL

5 http://www.postgresql.org/ [Abruf: 1.9.2009]
6 http://www.mysql.com/ [Abruf: 1.9.2009]
7 http://www.sqlite.org/ [Abruf: 1.9.2009]
8 http://www.ingres.com/ [Abruf: 1.9.2009]
9 http://www.enterprisedb.com/ [Abruf: 1.9.2009]
10 http://postgis.refractions.net/ [Abruf: 1.9.2009]

als Open Source Software angeboten. Der Kundenservice und weitgehend auch die Entwicklung werden von der schwedischen Firma *MySQL AB* geleisetet, die im Jahr 2008 durch die Firma *Sun Microsystems* übernommen wurde. Aktuell befindet sich Sun Microsystems in einem Übernahmeprozess durch Oracle; damit würde Oracle sein Portfolio durch ein weit verbreitetes Open Source Datenbanksystem ergänzen.

Räumliche Funktionen und Datentypen werden mit Hilfe der *Spatial Extensions* implementiert und orientieren sich an der OGC-Spezifikation *Simple Features Specifications for SQL* (OGC 2006b).

Ein Datenbanksystem, das im Hinblick auf eine mobile Anwendung interessant ist und ebenfalls eine Erweiterung für die Verarbeitung von räumlichen Daten aufweist, ist die Programmbibliothek *SQLite*[11]. Im Gegensatz zu den bisher beschriebenen Projekten ist SQLite keine eigenständige Software, sondern für die Integration in eigene Anwendungen vorgesehen. Außerdem sind die Anforderungen an die Hardware sehr gering und die Zahl der unterstützten Betriebssystem hoch, sodass dem Einsatz auf mobilen Geräten nichts im Wege stehen sollte.

Die Erweiterung *SpatiaLite*[12] ergänzt die Programmbibliothek um räumliche Datentypen sowie Funktionen und braucht sich in Bezug auf den Funktionsumfang nicht vor anderen Datenbanksystemen zu verstecken. Für den Einsatz als Datenbanksystem für Webanwendungen sind SQLite und SpatiaLite hingegen nicht konzipiert und kommen daher nicht für die weitere Untersuchung in Frage.

Das Projekt *CouchDB*[13] verfolgt, verglichen mit den bisher beschriebenen Projekten, neue Ansätze, die sich an den Anforderungen von aktuellen Webanwendungen orientieren. Das Ziel von CouchDB ist es, das einfache Datenmodell einer dokumentenorientierten Datenbank mit der Skalierbarkeit und Schnelligkeit von relationalen Datenbank zu verknüpfen. Mit der aktuellen Version 0.9.1 vom 26. Juli 2009 wird die Software allerdings noch nicht für den produktiven Einsatz empfohlen.

Die einfache Replikation von Datenbanken ließe sich eventuell auch für den Einsatz auf mobilen Geräten nutzen. Allerdings ist fraglich, ob CouchDB mit den geringen Hardwareressourcen und dem eingesetzten Betriebssystem zusammenarbeitet.

Nicht unerwähnt sollte ein kleines, im Entwicklungsstadium befindliches Projekt bleiben, welches CouchDB um räumliche Abfragen ergänzt[14]. Für den räumlichen Index kommt SpaitaLite zum Einsatz.

4.1.4 Vergleich

Die weitere Untersuchung konzentriert sich auf die Datenbanksysteme MySQL und PostgreSQL. Beide Systeme werden im Hinblick auf Softwareinteroperabilität und Verarbeitung von räumlichen Daten miteinander verglichen werden. Obwohl ein möglichst vollständiger Vergleich erzielt werden soll, kann an dieser Stelle kein Anspruch auf Vollständigkeit erhoben werden.

11 http://www.sqlite.org/ [Abruf: 1.9.2009]

12 http://www.gaia-gis.it/spatialite/ [Abruf: 1.9.2009]

13 http://couchdb.apache.org/ [Abruf: 1.9.2009]

14 http://gitorious.org/geocouch/ [Abruf: 1.9.2009]

Softwareinteroperabilität

In Bezug auf Softwareinteroperabilität soll untersucht werden, welche Software in bestimmten Aufgabenbereichen mit den Kandidaten zusammenarbeitet. Ein wichtiger Aufgabenbereich ist der Import von Geodaten in die Datenbank. Für die Entwicklung von Datenbankanwendung sind Programmbibliotheken, die das *Object-Relational Mapping* (ORM) übernehmen (siehe auch Kapitel 4.2.3 Geodatenbankanbindung), von Interesse. Darüber hinaus kommen natürlich noch kompatible Geoinformationssysteme, im speziellen WebGIS für die Untersuchung in Betracht.

	MySQL	**PostgreSQL**
Software für Datenimport	FME[14], OGR2OGR[15], shp2mysql	ArcGIS, FME, Manifold[16], shp2pgsql OGR2OGR, QuantumGIS[17]
ORM-Software	GeoAlchemy[18], HibernateSpatial[19]	GeoAlchemy, HibernateSpatial, NhibarnateSpatial[20]
DesktopGIS-Software	GvSig[21]	ArcGIS, GvSig, Manifold, ZigGIS[22], OpenJump[23], QuantumGIS, uDig[24]
WebGIS-Software	GeoServer[25], Mapserver[26], MapGuide Open Source[27]	ArcGIS, FeatureServer[28], GeoServer, Manifold, MapDotNet[29], MapGuide Open Source, Mapserver

Tabelle 4.1: Vergleich der Softwareinteroperabilität

Verarbeitung von räumlichen Daten

Die Möglichkeiten zur Verarbeitung von räumlichen Daten innerhalb der Datenbank sind von besonderem Interesse für die Entwicklung von WebGIS Anwendungen. Zum einen soll festgestellt werden, welcher räumliche Index zum Einsatz kommt und welche Geometrietypen zur Verfügung stehen. Von Belang ist außerdem die Möglichkeit zur Koordinatentransformation. Weiterhin soll festgestellt werden, welche Eingabe- und Ausgabeformate zur Verfügung stehen und welche Funktionen zur Untersuchung der Geometrietopologie implementiert sind. Außerdem sind Möglichkeiten zur Vermessung und linearen Referenzierung von Geometrieobjekten von Interesse. Nicht zuletzt sollen auch Funktionen zur Aggregation von mehreren Datensätzen in Betracht gezogen werden.

15 http://www.safe.com/ [Abruf: 1.9.2009]
16 http://gdal.org/ogr2ogr.html [Abruf: 1.9.2009]
17 http://www.manifold.net/ [Abruf: 1.9.2009]
18 http://www.qgis.org/ [Abruf: 1.9.2009]
19 http://www.geoalchemy.org/ [Abruf: 1.9.2009]
20 http://www.hibernatespatial.org/ [Abruf: 1.9.2009]
21 http://www.codeplex.com/NHibernateSpatial [Abruf: 1.9.2009]
22 http://www.gvsig.gva.es/ [Abruf: 1.9.2009]
23 http://www.obtusesoft.com/ [Abruf: 1.9.2009]
24 http://www.openjump.org/ [Abruf: 1.9.2009]
25 http://udig.refractions.net/ [Abruf: 1.9.2009]
26 http://geoserver.org/ [Abruf: 1.9.2009]
27 http://mapserver.org/ [Abruf: 1.9.2009]
28 http://mapguide.osgeo.org/ [Abruf: 1.9.2009]
29 http://featureserver.org/ [Abruf: 1.9.2009]
30 http://www.mapdotnet.com/ [Abruf: 1.9.2009]

	MySQL	PostgreSQL
Räumlicher Index	R-Tree	GiST
Geometrietypen (2D, 3D, 4D)	Polygon, Point, LineString, MultiPoint, MultiPolygon,MultiLineString, GeometryCollection	Polygon, Point, LineString, MultiPoint, MultiPolygon, MultiLineString, GeometryCollection, CircularString, CompoundCurve, CurvePolygon, MultiCurve, MultiSurface
Transformation	-	ja (2D und 3D)
Ausgabeformate	Binär, Text	Binär, Text, SVG, GML, KML, GeoJSON, EWKT, HexEWKB
Eingabeformate	Text, WKB	Text, WKB
Topologie-funktionen	Nur in Bezug auf eine Bounding Box: Intersects, Contains, Equal, Overlaps, Touches, Within	Contains, Difference, Disjoint, Equals, Intersects, Overlaps, Relate, SymDifference, Touches, Within
Messung	Distanz[1], Fläche, Länge[1]	Distanz[2], Fläche, Länge[2], Umfang
lineare Referenzierung	-	Line Interpolate Point, Line Substring, Line Locate Point, Locate along Measure, Locate between Measure
Aggregations-funktionen	-	Extent, Collect, Union, Accum, MakeLine, Polygonize

Tabelle 4.2: Vergleich der Verarbeitung von räumlichen Daten

4.1.5 Schlussfolgerung

Das Angebot an Geodatenbanksystemen im Open Source Umfeld ist mit zwei Optionen, die für den Einsatz auf Webservern gedacht sind, sehr übersichtlich. Sowohl in der Softwareinteroperabilität (siehe Tabelle 4.1) als auch im Funktionsumfang (siehe Tabelle 4.2) kann sich PostgreSQL deutlich gegenüber MySQL durchsetzen.

Ziel der Entwicklung wird es sein, möglichst unabhängig von einer spezifischen Datenbanklösung zu sein. Aufgrund des hohen Funktionsumfangs, der besseren Softwareinteroperabilität und insbesondere der guten Dokumentation, wird PostgreSQL zusammen mit PostGIS als Entwicklungsplattform dienen.

Einziger Wermutstropfen ist die geringere Verbreitung von PostgreSQL auf Webservern. Während MySQL häufig schon vorinstalliert ist, muss PostgreSQL in der Regel nachinstalliert werden. Angesichts der Zielgruppe, die mit der zu entwickelnden Anwendung angesprochen werden soll, ist dieser Nachteil jedoch vernachlässigbar.

31 nur für kartesische Koordinaten

32 sowohl für kartesische als auch ellipsoidische Koordinaten

4.2 Software zur Geodatenverarbeitung

4.2.1 Einleitung

Ein unerlässlicher Bestandteil von Anwendungen, die mit räumlichen Daten umgehen, sind Programmbibliotheken, welche räumliche Datentypen und Funktionen zur Verfügung stellen. Während in Kapitel 4.1 einzig die Ebene der räumlichen Datenbanken betrachtet wird, sind auf der Anwendungsebene entsprechende Anforderungen zu finden, um die räumlichen Informationen aus der Datenbank verarbeiten zu können. Die Anforderungen an die Programmbibliotheken können vielfältiger Natur sein, deshalb wird die weitere Untersuchung in drei Kategorien erfolgen.

Eine essentielle Aufgabe im Bereich der Geodatenverarbeitung ist die Bereitstellung von Datenstrukturen für räumliche Daten und darauf aufbauende, grundlegende Funktionen. Projekte mit diesem Schwerpunkt werden in Kapitel 4.2.2 näher untersucht.

Für Software, die nach dem Konzept der objektorientierten Programmierung entwickelt wird, sind außerdem diejenigen Programmbibliotheken interessant, die als Schnittstelle zwischen relationalen Datenbanken und objektorientierten Datenstrukturen eingesetzt werden können. Diesem Aufgabenbereich widmet sich Kapitel 4.2.3.

Sobald eine Anwendung komplexe Funktionen wie den Import und Export von Geodaten oder OGC-Dienste wie WMS oder WMS anbieten soll, steigen die Anforderungen an den Funktionsumfang der Programmbibliotheken erheblich. Im Kapitel 4.2.4 werden Projekte begutachtet, die entsprechend umfangreiche Aufgaben bewältigen können.

Aufgrund der zahlreichen Open Source Projekte in diesem Bereich beschränkt sich die vorliegende Untersuchung auf frei verfügbare Software. Ein direkter Vergleich der Projekte, wie er für Geodatenbanksysteme in Kapitel 4.1 vorgenommen wurde, ist meiner Ansicht nach nicht sinnvoll. Häufig handelt es sich bei den Projekten um eine Portierung auf eine andere Programmiersprache, oder aber die angesprochenen Zielgruppen sind grundverschieden. Die Entscheidung für eine bestimmte Programmbibliothek ist daher nicht nur vom Funktionsumfang, sondern vielmehr von der verwendeten Programmiersprache und anderen Faktoren abhängig.

4.2.2 Geodatenstrukturen

Eine unscheinbare, aber zentrale Voraussetzung für die Entwicklung von Anwendungen, die mit räumlichen Daten umgehen soll, sind entsprechende Datenstrukturen und grundlegende geometrische Funktionen. Als Richtlinie für die Implementierung des Geometriemodells dient den meisten Projekten die OGC-Spezifikation *Simple feature access* (OGC, 2006b) bzw. die ISO-Norm 19125. Die Spezifikation beschreibt ein Objektmodell für die Geodatentypen Punkt, Linie und Fläche sowie den entsprechenden Multi-Varianten. Außerdem können mit den Programmbibliotheken Berechnungen auf Grundlage der geometrischen Topologie ausgeführt und räumliche Indizes erstellt werden.

Eine der frühesten Implementierung der genannten Spezifikation ist die *Java Topology Suite*[33] (JTS), deren Entwicklung im Jahr 2000 begann. Die Software wird in der Programmiersprache Java entwickelt und unter der LGPL - aktuell in Version 1.10 - veröffentlicht. Bis heute wird die Software aktiv weiterentwickelt und in zahlreichen Java-Anwendungen eingesetzt. Zudem ist das Projekt auch Vorlage für Portierungen in andere Programmiersprachen.

Zwei der erwähnten Portierungen in andere Programmiersprachen nennen sich *Geometry Engine Open Source*[34] (GEOS) und *NetTopologySuite*[35] (NTS), die beide dem Funktionsumfang von JTS entsprechen. Während GEOS in C++ implementiert ist und Schnittstellen zu den Programmiersprachen C, C++ und Python anbietet, ist NTS in C# implementiert und damit für den Einsatz auf der .NET-Plattform gedacht. Eine zentrale Aufgabe übernimmt die Programmbibliothek GEOS in den Projekten PostGIS (siehe Kapitel 4.1) und Shaply (siehe Kapitel 4.2.4).

	Java Topology Suite (JTS)	***namenloses Projekt***
Programmier-sprache	Java	Java
Portierung	GEOS (C++), NTS (C#)	-
Geometrie-modell	Simple Feature Access	GML
Topologie-funktionen	contains, covers, crosses, disjoint, equals, intersects, overlaps, relate, touches, within	contains, crosses, disjoint, equals, intersects, touches, within, within distance
	Boundary, Buffer, Centroid, Convex Hull, Difference, Envelope, Intersection, Union	Buffer, Centroid, Convex Hull, Difference, Envelope, Intersection, Union
Messungen	Distanz, Fläche, Länge	Distanz, Fläche, Länge
räumliche Indizes	bintree, chain, intervalrtree, quadtree, strtree, sweepline	-

Tabelle 4.3: Vergleich von Programmbibliotheken für Geodatenstrukturen

Eine noch sehr junges, von den Entwicklern der Projekte GeoTools und deegree (siehe Kapitel 4.2.4) initiiertes und bisher namenloses Vorhaben richtet sich nicht mehr nach der OGC-Spezifikation *Simple feature access*, sondern nach dem Geometriemodell der GML-Spezifikation (OGC, 2004). Bisher sind allerdings nur die Programmschnittstellen festgelegt[36]; langfristig wird dieses Projekt aber vermutlich JTS in der Java-Welt ablösen.

33 http://tsusiatsoftware.net/jts/main.html [Abruf: 2.9.2009]

34 http://trac.osgeo.org/geos/ [Abruf: 2.9.2009]

35 http://code.google.com/p/nettopologysuite/ [Abruf: 2.9.2009]

36 http://svn.osgeo.org/geotools/trunk/spike/geometry/ [Abruf: 2.9.2009]

4.2.3 Geodatenbankanbindung

Anwendungen mit einem hohen Datenaufkommen sind ohne eine Datenbankanbindung nur schwer zu realisieren. Für weit verbreitete Programmiersprachen stehen für diese Aufgabe Schnittstellen zur Verfügung, die den Zugriff auf relationale Datenbanken erheblich vereinfachen. Als Beispiel ist für die Java-Plattform die Schnittstelle *Java Database Connectivity* (JDBC) zu nennen, die mit Hilfe entsprechender Treiber eine Verbindung zu den meisten relationalen Datenbanken aufbauen und SQL-Abfragen verarbeiten kann. Obwohl der Programmieraufwand durch die Verwendung dieser Schnittstellen deutlich reduziert wird, bleiben einige Probleme ungelöst. Die Abstraktionsstufe dieser Schnittstellen ist sehr niedrig, sodass die Abfrage und Speicherung von Datensätzen auf Ebene der Datenbanken erfolgen muss. Gerade die Vorteile der objektorientierten Programmierung gehen dabei verloren oder sind nur schwer umzusetzen.

Dieses Problem kann mit Hilfe der objektrelationalen Abbildung, auch bekannt als *object-relational mapping* (ORM), gelöst werden, eine Technik zur Abbildung von Objekten in relationalen Datenbanken. Verschiedene ORM-Programmbibliotheken sind verfügbar, die dem Entwickler weitgehend die Interaktion mit der Datenbank abnehmen. Besondere Anforderungen an die ORM-Programmbibliotheken stellen wiederum Anwendungen, die räumlichen Daten verarbeiten. So sollen nicht nur die Geodatenstrukturen in der Datenbank abgebildet werden, sondern auch räumliche Abfragen der Daten und Topologiefunktionen ermöglicht werden. Lösungen für diese Aufgaben sollen im Folgenden beschrieben werden.

Auf der Java-Plattform ist *Hibernate*[37] eine weit verbreitete ORM-Programmbibliothek, die einer Implementierung der *Java Persistence API* (JPA) entspricht. Allerdings werden räumliche Daten nur mit Hilfe der Erweiterung *Hibernate Spatial*[38] unterstützt, die sich nach der OGC-Spezifikation *Simple feature access* (OGC, 2006b) richtet und für die Abbildung der Geometrieobjekte die Programmbibliothek JTS (siehe Kapitel 4.2.2) verwendet. Als Geodatenbanksysteme können MySQL, Oracle oder Postgresql zum Einsatz kommen.

Ein vergleichbares Projekt für die .NET-Plattform ist *NHibernate.Spatial*[39], eine Erweiterung für *NHibernate*[40], die für das Geometriemodell die Programmbibliothek NetTopologySuite einsetzt und damit den gleichen Funktionsumfang wie Hibernate Spatial anbietet. Als Geodatenbanksysteme werden MySQL, PostgreSQL und SQL Server 2008 unterstützt.

Für Anwendungen, die in der Programmiersprache Python geschrieben sind, übernimmt *SQLAlchemy*[41] die Aufgabe von Hibernate. Wenn ein Geodatenbanksystem zum Einstatz kommt, steht die Erweiterung *GeoAlchemy*[42] zur Verfügung, die ebenfalls der OGC-Spezifikation Simple feature access folgt und mit den Datenbanken MySQL, PostgreSQL und SQLite zusammenarbeitet.

37 http://www.hibernate.org/ [Abruf: 2.9.2009]
38 http://www.hibernatespatial.org/ [Abruf: 2.9.2009]
39 http://nhforge.org/wikis/spatial/default.aspx [Abruf: 2.9.2009]
40 http://nhforge.org/ [Abruf: 2.9.2009]
41 http://www.sqlalchemy.org/ [Abruf: 2.9.2009]
42 http://www.geoalchemy.org/ [Abruf: 2.9.2009]

	Hibernate Spatial	Nhibernate.Spatial	GeoAlchemy
Programmier-sprache	Java	C#	Python
ORM	Hibernate	NHibernate	SQLAlchemy
Geometrie-modell	Simple Feature Access Specification		
Geometrie-bibliothek	JTS	NTS	-
Datenbanken	MySQL, Oracle, Postgresql	MySQL, PostgreSQL, SQL Server 2008	MySQL, PostgreSQL, SQLite
Abfrage-kriterien	contains, crosses, disjoint, equals, intersects, overlaps, touches, within	contains, crosses, disjoint, equals, intersects, overlaps, touches, within	contains, covers, crosses, disjoint, distance, equals, intersects, overlaps, touches, within

Tabelle 4.4: Vergleich von Programmbibliotheken zur Geodatenbankanbindung

4.2.4 Schlussfolgerung

Zusammenfassend entsteht der Eindruck, dass die meisten Open Source Programmbibliotheken im Geomatikbereich für Java entwickelt werden und diese häufig in andere Programmiersprachen portiert werden. Obwohl die Auswahlmöglichkeiten begrenzt sind, steht für die populären Programmiersprachen wenigstens eine Lösung bereit. Zudem ist der Funktionsumfang über die Programmiersprachen hinweg vergleichbar, daher müssen diese Programmbibliotheken keinen entscheidenden Einfluss auf die Wahl der Entwicklungsplattform nehmen.

4.3 Software für Webanwendungen

4.3.1 Einleitung

Die Erkenntnis, dass bei der Programmierung immer wieder gleich oder sehr ähnliche Aufgabenstellungen auftreten, hat nicht nur zu der Entwicklung von Softwarearchitekturmustern geführt (siehe Kapitel 2.4.3), sondern auch den Softwarebereich der so genannten *Frameworks* hervorgebracht. Häufig entstehen Frameworks durch die Entfernung von anwendungsspezifischen Teilen aus einer Software; das verbleibende Programmgerüst steht anschließend für andere Projekte zur Verfügung (Gumm & Sommer, 2004).

Für die Entwicklung von Webanwendungen haben sich *Web Application Frameworks* etabliert, die im Gegensatz zu den bisher untersuchten Softwarebereichen in einer fast unüberschaubaren Anzahl verfügbar sind. Selbst die Auswahlmöglichkeiten im Rahmen einer populären Web-Programmiersprache sind schwer zu überblicken. Lediglich der Wikipedia-Artikel *Comparison of web application frameworks* (Wikipedia, 2009) verzeichnet - ohne Anspruch auf Vollständigkeit - zehn

Python-Frameworks, 26 PHP-Frameworks und 30 Frameworks basierend auf der Java-Plattform. Eine intensive Untersuchung der verschiedenen Auswahlmöglichkeiten ist allein durch die große Anzahl nicht realistisch. Gleichwohl sollen in diesem Kapitel eine Auswahl an Web Application Frameworks, getrennt nach Programmiersprache, zusammenfassend beschrieben und in Bezug auf ihre Eignung für WebGIS-Anwendungen untersucht werden.

Zunächst gilt es, Programmiersprachen zu identifizieren, die eine weite Verbreitung bei der Entwicklung von Webanwendungen aufweisen. Ein üblicher Index für die Popularität von Programmiersprachen ist der *TIOBE Programming Community Index* (TIOBE, 2009). Dieser berücksichtigt allerdings nicht das Anwendungsgebiet der Programmiersprachen. Obwohl die Entwicklung von Webanwendungen einen immer größeren Anteil ausmachen, können mit dieser Quelle keine direkten Rückschlüsse zur Popularität von Programmiersprachen im Bereich der Webanwendungsentwicklung gemacht werden.

In dieser Hinsicht ist eine Veröffentlichung mit dem Title *Comparing Web Development Platforms Through the Eyes of Professional Developers* von Hardy (2007) aufschlussreicher. Insgesamt 268 professionelle Entwickler von Webanwendungen sollten ihnen bekannte Programmiersprachen nach zwölf Kriterien vergleichen und einordnen. Mit Abstand am häufigsten wurde PHP genannt, gefolgt von Perl, Python, Java, Ruby und .NET. Auch wenn die Ergebnisse nicht repräsentativ sind, deuten sie doch auf eine gewisse Beliebtheit und Verbreitung der genannten Plattformen hin.

Insgesamt lässt sich aus der Anzahl der verfügbaren Web Application Frameworks, dem TIOBE Index und Hardy (2007) deutlich Java, PHP und Python als die beliebtesten Entwicklungsplattformen für Webanwendungen identifizieren. Darüber hinaus haben noch Ruby, Perl und die .NET-Plattform eine größere Bedeutung für die Webentwicklung.

Im Vordergrund der Beschreibung soll einerseits die Unterstützung der bisher erwähnten Programmbibliotheken und Geodatenbanksysteme stehen. Andererseits soll auch die Berücksichtigung von Entwicklungsprinzipien wie *Convention over Configuration* (CoC) und *Don't Repeat Yourself* (DRY) erwähnt werden (siehe Kapitel 2.4.4). Nicht zuletzt spielen Softwarearchitekturmuster und Hilfsmittel wie *Scaffolding* eine Rolle. Mit Hilfe der Entwicklungsmethode Scaffolding wird auf Basis eines benutzerdefinierten Objektmodells automatisch eine standardisierte Anwendungslogik und Benutzeroberfläche generiert, die rudimentäre Funktionen zum Erstellen, Anzeigen, Bearbeiten und Löschen von Datensätzen bereitstellt. Auf diese Weise kann das Objektmodell ohne großen Aufwand getestet werden.

4.3.2 Java-Plattform

Die Java-Plattform gehört zu den ersten und bis heute weit verbreiteten Entwicklungsumgebungen von Webanwendungen. Damit erklärt sich vermutlich auch die bereits erwähnte hohe Zahl an Web Application Frameworks. Die folgende Zusammenfassung beschränkt sich nicht nur auf Java als Programmiersprache, sondern als Plattform und umfasst also auch Projekte, die nicht in der Programmiersprache Java implementiert, aber auf der Java-Plattform zuhause sind.

Beispiele für Java-Frameworks, im Sinne von Java als Programmiersprache, sind *Wicket*[43], *Tapestry*[44] und *Struts*[45], die alle dem MVC-Architekturmuster folgen. Während Wicket noch ein vergleichbar junges Projekt ist, haben Tapestry und Struts bereits im Jahr 2000 das Licht der Welt

43 http://wicket.apache.org/ [Abruf: 2.9.2009]

44 http://tapestry.apache.org/ [Abruf: 2.9.2009]

45 http://struts.apache.org/ [Abruf: 2.9.2009]

erblickt und sind von Prinzipien wie Convention over Configuration oder DRY weniger beeinflusst als jüngere Projekte.

Einen anderen Ansatz verfolgen die Frameworks *Google Web Toolkit*[46] (GWT) und *ZK*[47], die beide für Webanwendungen mit komplexen Benutzeroberflächen - so genannte *Rich Internet Application* (RIA) - konzipiert sind. Die Entwicklung von Client- und Serveranwendung wird nach diesem Konzept in einem Framework vereint und die Benutzeroberfläche der Webanwendung nicht mehr in HTML und JavaScript entwickelt. Während im Fall von GWT für diesen Zweck Java zur Anwendung kommt, steht für ZK mit *ZUML* eine eigene Markupsprache zur Verfügung.

Eine interessante Alternative zu den bisher genannten Projekten sind Frameworks, die in anderen Programmiersprachen für die Java-Plattform entwickelt werden. Für die Scriptsprache *Groovy*, welche stark von Ruby beeinflusst wurde, ist das Framework *Grails*[48] verfügbar, welches entsprechend Rails als Vorbild hat (siehe Kapitel 4.3.5) und die Entwicklungsprinzipien Convention over Configuration und DRY verfolgt.

Basierend auf *Scala*, einer funktionalen und objektorientierten Programmiersprache, wird das Web Application Framework *Lift*[49] entwickelt. Es orientiert sich ebenfalls an den Konzepten von Rails und wurde im Februar 2009 als Version 1.0 veröffentlicht.

Sowohl Groovy als auch Scala erlauben die Integration von Java-Programmbibliotheken. Damit werden die Vorteile der modernen Programmiersprachen mit dem reichhaltigen Angebot an Programmbibliotheken für die Java-Plattfom kombiniert.

4.3.3 PHP

Die Scriptsprache *PHP* ist für die Erstellung von dynamischen Webseiten besonders gut geeignet, weil sich der Quellcode direkt in HTML einbetten lässt und ein großer Funktionsumfang zur Verfügung steht. In Kombination mit Linux als Betriebssystem, Apache als Webserver und MySQL als Datenbank hat PHP eine weite Verbreitung gefunden. Entsprechend vielfältig ist das Angebot an Web Application Frameworks.

Zu den populären PHP-Frameworks gehören beispielsweise *CakePHP*[50], *Kohana*[51] und *Symfony*[52], die alle das MVC-Architekturmuster anwenden, einen vergleichbaren Funktionsumfang haben und größtenteils den beschriebenen Entwicklungsprinzipien folgen. Erwähnenswert ist ein Plugin[53] für Symfony, welches die Integration von *MapFish*, einem WebGIS-Framework, erlaubt. Abgesehen davon sind Erweiterungen zur Verarbeitung von räumlichen Daten nicht vorhanden.

Ein Projekt, das in der Grauzone zwischen Web Application Framework und *Content Management System* (CMS) zu finden ist und über eine Erweiterung für Geodaten verfügt, ist *Drupal*[54]. Mit Hilfe des Moduls *Geo*[55] können Geodaten auf Basis von PostgreSQL und MySQL nach der OGC-Spezifikation Simple feature access verarbeitet werden.

46 http://code.google.com/webtoolkit [Abruf: 2.9.2009]
47 http://www.zkoss.org/ [Abruf: 2.9.2009]
48 http://www.grails.org/ [Abruf: 2.9.2009]
49 http://liftweb.net/ [Abruf: 2.9.2009]
50 http://www.cakephp.org/ [Abruf: 2.9.2009]
51 http://kohanaphp.com/ [Abruf: 2.9.2009]
52 http://www.symfony-project.org/ [Abruf: 2.9.2009]
53 http://www.symfony-project.org/plugins/sfMapFishPlugin [Abruf: 2.9.2009]
54 http://drupal.org/ [Abruf: 2.9.2009]
55 http://drupal.org/project/geo [Abruf: 2.9.2009]

4.3.4 Python

Eine ebenfalls weit verbreitete Plattform für die Entwicklung von Webanwendungen ist die Scriptsprache Python. Beispiele für beliebte Python-Frameworks sind *Django*[56], *Pylons*[57] und *TurboGears*[58]; alles mehr oder weniger junge Projekte, die einen agilen Entwicklungsstil erlauben und einen vergleichbaren Funktionsumfang haben.

Erwähnenswert ist die Erweiterung GeoDjango[59], die Django zu einem *geographic web framework* ausbauen soll. Auf Basis der Programmbibliotheken GEOS und GDAL können komplexe Probleme bewältigt werden. Im gleichen Anwendungsbereich ist das Projekt *MapFish*[60], eine vollständige Client-Server Entwicklungsumgebung, angesiedelt. Während der Client auf GeoExt aufbaut, können die Komponenten auf dem Server nicht nur in Python, sondern auch in Java, Ruby und PHP entwickelt werden.

4.3.5 Sonstige

Weitere Frameworks für Webanwendungen sind unter anderem in den Programmiersprachen Perl und C# verfügbar, für die es aber offensichtlich keine dezidierten Erweiterungen zur Unterstützung von räumlichen Daten gibt.

Eine Ausnahme stellt das Framework *Rails*[61] dar, welches in der Programmiersprache Ruby entwickelt wird und einen bedeutenden Einfluss auf viele neue Projekte hatte. Außerdem ist mit *GeoRuby*[62] ein Plugin vorhanden, welches die Simple Feature Access Spezifikation unterstützt und Rails um die Verarbeitung von Geodaten auf Basis von MySQL oder PostgreSQL erweitert.

4.3.6 Schlussfolgerung

Wie unschwer zu bemerken ist, sind die Auswahlmöglichkeiten im Bereich der Software für Webanwendungen zahlreich und eine objektive Entscheidung unrealistisch. Aus der Masse stachen insbesondere die Frameworks Django, MapFish, Grails und Rails hervor, die nicht nur eine schnelle und flexible Entwicklung von Webanwendungen erlauben, sondern auch über Erweiterungen für räumliche Datenstrukturen und Funktionen verfügen.

Letztendlich fiel die Entscheidung auf Grails, ein Framework, das auf der Java-Plattform aufbaut und damit die Verwendung von zahlreichen Programmbibliotheken erlaubt. Ebenso könnte das Vorhaben aber auch mit Hilfe von einem der anderen drei Projekte realisiert werden.

56 http://www.djangoproject.com/ [Abruf: 2.9.2009]
57 http://pylonshq.com/ [Abruf: 2.9.2009]
58 http://www.turbogears.org/ [Abruf: 2.9.2009]
59 http://geodjango.org/ [Abruf: 2.9.2009]
60 http://trac.mapfish.org/trac/mapfish/wiki [Abruf: 2.9.2009]
61 http://rubyonrails.org/ [Abruf: 2.9.2009]
62 http://rubyforge.org/projects/georuby/ [Abruf: 2.9.2009]

4.4 Software zur Geodatenvisualisierung

4.4.1 Einleitung

Bisher wurde ausschließlich Software auf Seite des Servers berücksichtigt. Einen unerlässlichen Bestandteil von Webanwendungen stellt jedoch die graphische Benutzeroberfläche dar, die erst eine Interaktion zwischen Benutzer und Anwendung ermöglicht. Anwendungen auf Basis von Geodaten stellen besonders hohe Anforderungen an die Benutzeroberfläche, schließlich sollen die Geodaten auf irgendeine Art kartographisch visualisiert werden. In der Regel wird für diesen Zweck eine Kombination aus HTML und JavaScript eingesetzt; für interaktive und grafikintensive Anwendungen kommen teilweise auch Flash, Canvas oder SVG zum Einsatz. In der einen oder anderen Form setzen aber alle Projekte entweder JavaScript oder Flash für dynamische Inhalte ein; entsprechend wurde die weitere Bestandsaufnahme in die Kapitel 4.4.2 und 4.4.3 unterteilt.

Von besonderer Bedeutung für Anwendungen auf Basis von Geodatenbanken sind Möglichkeiten zur Visualisierung von Vektordaten. Die Basiskarte wird in der Regel als Rasterbild umgesetzt, daher spielen auch Rasterdaten eine Rolle für die kartographische Darstellung.

4.4.2 JavaScript

Die Scriptsprache JavaScript wird von allen gängigen Internetbrowsern unterstützt und wird daher standardmäßig für dynamische oder interaktive Inhalte auf Internetseiten eingesetzt. Die Darstellung von Geodaten kann allerdings nicht einzig mit JavaScript gelöst werden, sondern benötigt eine effiziente Visualisierungstechnik. Insgesamt stehen - abhängig vom verwendeten Browser - drei unterschiedliche Techniken für diesen Zweck zur Verfügung.

Die *Vector Markup Language* (VML), eine zur Beschreibung zweidimensionaler Vektorgrafiken geeignete Auszeichnungssprache, die 1998 von Microsoft zur Standardisierung an das World Wide Web Consortium (W3C) übergeben wurde (W3C, 1998), wird einzig und allein vom Internet Explorer unterstützt. Erst eine überarbeitete und mit der *Precision Graphics Markup Language* (PGML) vereinte Version wurde vom W3C unter dem Namen *Scalable Vector Graphics* (SVG) veröffentlicht und wird inzwischen von allen gängigen Browsern, abgesehen vom Internet Explorer, weitgehend unterstützt (W3C, 2003).

Ein recht neuer, aber bereits ebenso gut unterstützter Ansatz wie SVG, ist das Canvas-Element aus der HTML5-Spezifikation. Mit dieser Technik ist es möglich, alleine auf Basis von HTML5 und JavaScript dynamische Vektografiken zu generieren.

Eine weit verbreitete Software zur Visualisierung von Geodaten ist *OpenLayers*[63], welche alle drei erwähnten Techniken für diese Aufgabe einsetzt. Der Funktionsumfang von OpenLayers reicht von Editionsmöglichkeiten der Vektodaten über Schnittstellen zu WMS, WFS, Google Maps und anderen Kartenprovidern bis zu Topologie- sowie Aggregationsfunktionen und wird meines Wissens von keiner anderen JavaScript-Bibliothek erreicht. Die Software wird unter BSD-Lizenz entwickelt und wurde am 23. Juni 2009 in Version 2.8 veröffentlicht.

63 http://openlayers.org/ [Abruf: 2.9.2009]

Interessant für Webanwendungen mit komplexer Benutzeroberfläche ist das Projekt *GeoExt*[64], welches OpenLayers in das JavaScript-Framework *ExtJS* integriert und auf diese Weise den Anforderungen von umfangreichen WegGIS-Anwendungen gerecht werden sollte. Mit Version 0.5, die am 2. Juli 2009 unter BSD-Lizenz veröffentlich wurde, befindet sich die Software allerdings noch im Entwicklungsstadium.

Der Versuch eine einheitliche Schnittstelle für die verschiedenen Kartenprovider im Internet anzubieten, wird von *Mapstraction*[65] verfolgt. Insgesamt elf unterschiedliche Kartenquellen können verwendet werden und obwohl auch Möglichkeiten zur Visualisierung von eigenen Vektordaten zur Verfügung stehen, eignet sich die JavaScript-Bibliothek weniger für umfangreiche WebGIS-Anwendungen als beispielsweise OpenLayers.

Ein neuer Ansatz, der einzig auf das Canvas-Element zur Visualisierung aufbaut, ist *Cartagen*[66]. Die Geodaten müssen im JSON-Format zur Verfügung stehen und können mit *Geo Style Sheets* (GSS), einer mit *Cascading Style Sheets* (CSS) vergleichbaren Formatierungssprache, graphisch aufbereitet werden. Obwohl der Ansatz sehr vielversprechend ist, funktioniert die Darstellung von umfangreichen Datensätzen noch nicht ganz reibungslos und die fehlende Unterstützung des Internet Explorers kann für manche Vorhaben das Auschlusskriterium bedeuten.

4.4.3 Flash

Eine beliebte Alternative zu Ansätzen, die auf JavaScript aufbauen, ist die Flash-Technologie von *Adobe*. Die Technik ist für den Einsatz in Multimedia-Anwendungen optimiert und erzielt durch den Einsatz von hochwertigen, aber lizenzgeschützten Video- und Audio-Codecs teilweise bessere Ergebnisse, als mit anderen Techniken möglich ist. Der fest definierte Funktionsumfang und die Layouttreue über alle unterstützten Plattformen hinweg sind weitere Vorteile. Nicht zuletzt bietet Adobe außerdem eine professionelle, integrierte Entwicklungsumgebung an.

Im Gegensatz lassen sich allerdings auch nicht zu vernachlässigende Nachteile identifizieren. Für die Anzeige von Flash-Inhalten ist ein Plugin erforderlich, das für die Betriebssystem Windows, Mac OS X, Linux und Solaris verfügbar ist. Außerdem wird durch die geschlossene, proprietäre Technik der freie Informationsaustausch erschwert und die Anschaffung der erwähnten Entwicklungsumgebung ist mit Kosten verbunden.

Als Beispiele für frei verfügbare Software zur Visualisierung von Geodaten seien die Projekte *Modest Maps*[67] und *OpenScales*[68] genannt. Während Modest Maps nur einen minimalen Funktionsumfang zur Darstellung von Geodaten bereit stellt, wird von OpenScales ein ähnlicher Funktionsumfang wie OpenLayers angestrebt. Damit eignet sich Modest Maps eher für einfache Kartenanwendungen oder als Grundlage für eine Eigenentwicklung. Mit Editionsfunktionen und zahlreichen Schnittstellen zu Kartenprovidern ist OpenLayers für den Einsatz in WebGIS-Anwendungen optimiert.

64 http://www.geoext.org/ [Abruf: 2.9.2009]
65 http://mapstraction.com/ [Abruf: 2.9.2009]
66 http://cartagen.org/ [Abruf: 2.9.2009]
67 http://modestmaps.com/ [Abruf: 2.9.2009]
68 http://openscales.org/ [Abruf: 2.9.2009]

4.4.4 Schlussfolgerung

Als Standardlösung für die Visualisierung von umfangreichen Geodaten auf Internetseiten kann OpenLayers angesehen werden. Wenn auf Flash verzichtet werden soll, stehen zudem keine Alternativen mit vergleichbarem Funktionsumfang zur Verfügung. Für die Entwicklung von WebGIS-Anwendungen eignet sich OpenLayers besonders gut, da JavaScript-Bibliotheken generell und OpenLayers im Speziellen leicht erweitert und angepasst werden können.

Ein mit OpenLayers verbleichbares Projekt ist für die Flash-Plattform verfügbar. Allerdings haben die genannten Nachteile und der höhere Einarbeitungsaufwand zu der Entscheidung geführt OpenLayers für die weitere Entwicklung einzusetzen.

5 Realisierung

Die Umsetzung der Anforderungen aus Kapitel 3 in eine Anwendung wird im folgenden Kapitel erläutert. Dabei wird nicht der gesamte Quellcode dokumentiert, sondern vielmehr der prinzipielle Weg beschrieben, der zur Realisierung der Anwendung geführt hat. Die Beschreibung folgt im Wesentlichen der Daten-, Steuerungs- und Präsentationsebene der MVC-Architektur. Zunächst wird zur Förderung des Gesamtverständnisses in **Kapitel 5.1** die Gesamtarchitektur der Anwendung geschildert. Abschließend werden in **Kapitel 5.5** die zur Realisierung eingesetzten Werkzeuge erwähnt.

5.1 Gesamtarchitektur

Im Folgenden soll ein Überblick über das gesamte System und das Zusammenspiel der einzelnen Komponenten gegeben werden. Als Informationssystem besteht die Anwendung aus einer mehrschichtigen Architektur, die von der Datenebene über die Steuerung bis zur Benutzeroberfläche reicht. Jede Ebene umfasst eine Vielzahl von Komponenten die - teilweise über Ebenengrenzen hinweg - miteinander kommunzieren und aufeinander aufbauen. In Abbildung 5.1 werden wichtige Komponenten als farbige Rechtecke und die drei Architekturebenen Model, Steuerung sowie Präsentation als Rahmen visualisiert. Die weitere Beschreibung folgt der Abbildung von der Datenebene bis zur Benutzeroberfläche.

Die Basis der gesamten Anwendung bildet ein relationales Geodatenbanksystem. Obwohl die Anwendung prinzipiell unabhängig von einem spezifischen Datenbanksystem sein soll, wird für die Entwicklung, wie in Kapitel 4.1.5 erläutert, das Geodatenbanksystem PostgreSQL in Kombination mit PostGIS eingesetzt. Der gesamte Datenbestand, bestehend aus Sachdaten, Geometriedaten, Fotos und Dokumenten, wird in der Datenbank zusammengeführt, die damit den Kern des Informationssystems bildet.

Die eigentliche Anwendung baut auf der Java-Plattform auf und bedient sich für die Verbindung zur Datenbank der Java-Datenbankschnittstelle *Java Database Connectivity* (JDBC). Das objektorientierte Datenmodell der Anwendung wird von der Java-Programmbibliothek Hibernate in ein relationales Datenbankmodell abgebildet; man spricht auch von *object-relational mapping* (ORM). Außerdem stellt Hibernate mit der *CritireaAPI* eine umfangreiche Schnittstelle für die Datenbankabfrage nach benutzerdefinierten Kriterien zur Verfügung. Die Funktionalitäten von PostGIS werden auf Seite der Anwendung von Hibernate Spatial bereitgestellt. Die gesamte Kommunikation zwischen Datenbank und Hibernate läuft über die JDBC-Schnittstelle.

Das objektorientierte Datenmodell der Anwendung wird im Rahmen des Grails-Frameworks mit Hilfe so genannter Domain Klassen in der Programmiersprache Groovy modelliert. *Grails Object Relational Mapping* (GORM), eine Komponente des Grails-Frameworks, basiert auf Hibernate und interpretiert das erstellte Datenmodell. Im Sinne des MVC-Architekturmusters kann dieser Bereich dem Model zugeordnet werden.

Die Steuerung wird ebenfalls in Groovy implementiert und zwar innerhalb so genannter Kontroller Klassen. In der Regel entspricht eine Kontroller Klasse einer Domain Klasse. Funktionalitäten, die über mehrere Kontroller hinweg zur Verfügung stehen müssen, können mit Hilfe von Services realisiert werden. Auf diese Weise wird die mehrfache Implemenierung der gleichen Funktionaliät vermieden.

Die Geometriedaten werden sowohl auf der Daten- als auch auf der Steuerungsebene von der Java-Programmbibliothek Java Topology Suite (JTS) verwaltet.

Die Benutzeroberfläche und damit die Präsentations-Schicht der MVC-Architektur, wird mit Hilfe von Groovy Server Pages (GSP) realisiert, die innerhalb von Grails als *Views* bezeichnet werden. Für die seitenübergreifende Gestaltung stehen *Layouts* und für wiederholt vorkommende Darstellungselemente *Taglibs* sowie *Templates* zur Verfügung. Die einzelnen Elemente werden von der

Java-Programmbibliothek *SiteMesh* zusammengeführt. Das Ergebnis der GSP-Dateien sind in der Regel HTML-Dateien, es lassen sich allerdings auch andere textcodierte Formate wie beispielsweise XML oder JSON ausgeben.

Im Browser kommt zur Darstellung der Geodaten die JavaScript-Bibliothek *OpenLayers* zum Einsatz. Weitere interaktive Elemente werden mit Hilfe von *jQuery*, einem JavaScript-Framework, das durch Plugins erweitert werden kann, realisiert.

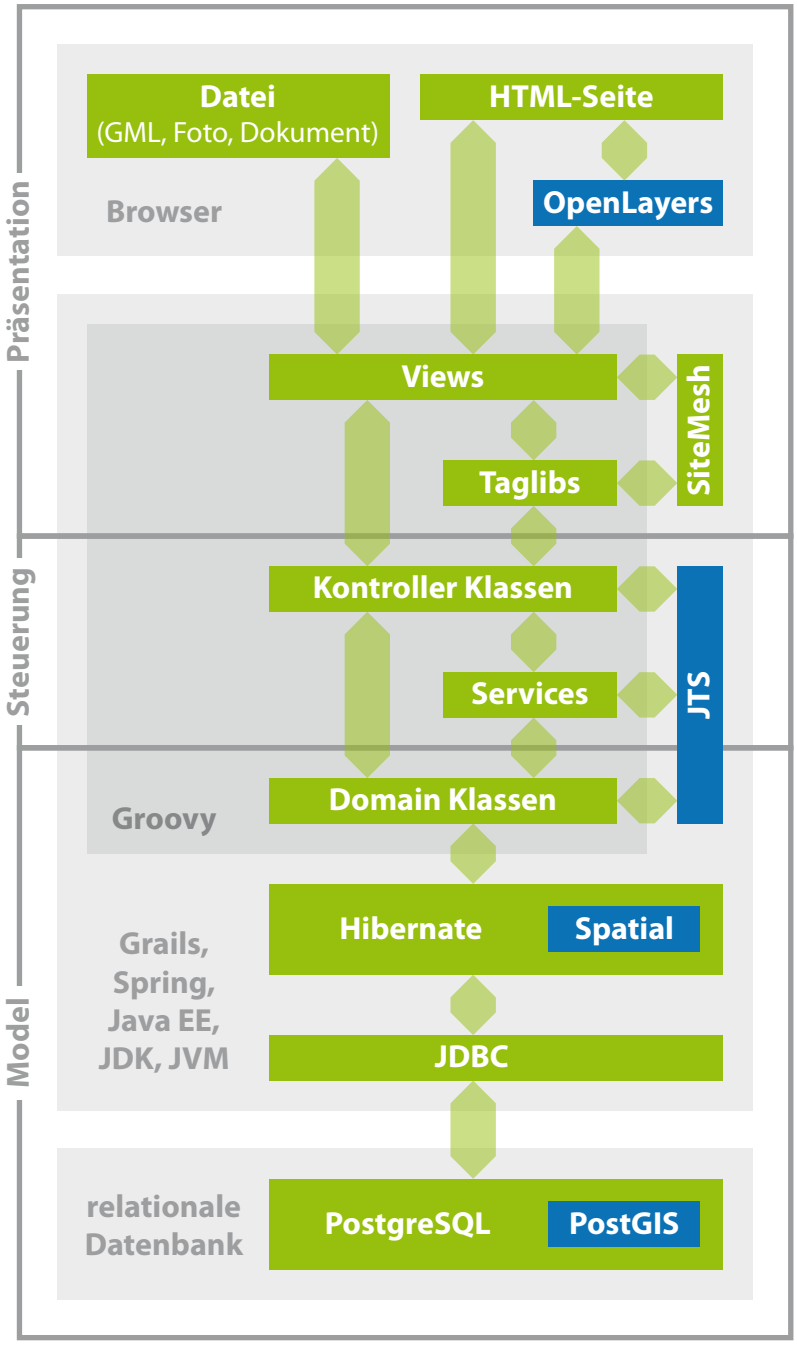


Abbildung 5.1: Gesamtarchitektur des Naturereignisinformationssystems

5.2 Datenebene

5.2.1 Einleitung

In diesem Kapitel soll die Datenebene der Webanwendung beschrieben werden. Im Vordergrund steht eine Beschreibung wie die Klassen, die Attribute im Hinblick auf ihre Datentypen und Randbedingungen, sowie die Beziehungen zu anderen Klassen implementiert wurden. Außerdem soll das Zusammenspiel von Datenbank und Webanwendung beleuchtet werden. Zunächst werden aber die generellen Möglichkeiten zur Datenmodellierung mit Hilfe von Grails und die Konfiguration der Datenbankverbindung beschrieben.

Gleichwohl es mit dem Grails-Framework möglich ist, auf ein existierendes Datenbankschema aufzubauen, wird bei neuen Anwendungen, die nicht auf eine bestehende Datenbank zugreifen müssen, mehrheitlich eine andere Herangehensweise verfolgt. Vergleichbar mit einem Top-Down-Ansatz werden die Klassen innerhalb der Serveranwendung modelliert und von dem Framework in ein automatisch generiertes Datenbankschema konvertiert. Das Prinzip *Convention over Configuration* sorgt dabei für ein sinnvolles Datenbankschema, ermöglicht aber trotzdem, das Ergebnis durch Änderung der Standardeinstellungen an die eigenen Wünsche anzupassen. Für die Verwendung eines bestehenden Datenbankschemas sind gleichsam Änderungen an den Standardeinstellungen notwendig, daher führt der Einsatz eines neu erstellten Schemas, welches nicht den Konventionen von Grails folgt, zu einem unnötigen Mehraufwand.

Obwohl für diese Arbeit bereits ein Datenbankschema von Jonas Büchel vorlag, war dieses als Oracle-Datenbank realisiert und wurde auf Grund von aufwändigen Anpassungen und wegen der Änderungen, die während der Anforderungsanalyse entstanden sind, nicht verwendet. Stattdessen wurde der Top-Down-Ansatz und damit ein automatisch generiertes Datenbankschema gewählt, welches den Konventionen des Grails-Framework entspricht. Nichtsdestotrotz sind beide Schemata, abgesehen von den in der Anforderungsanalyse dokumentierten Änderungen, weitgehend vergleichbar.

Das Problem von nebenläufigen Änderungen in der Datenbank, d.h. zwei Benutzer bearbeiten einen Datensatz zur gleichen Zeit, wird von Grails in der Standardkonfiguration mit Hilfe der Methode *Optimistic Locking* gelöst. Dazu wird in jeder Tabelle eine Spalte namens `version` hinzugefügt, die eine Versionsnummer für jeden Datensatz enthält. Wenn beim Aktualisieren eines Datensatzes die Versionsnummer in der Datenbank und im bearbeiteten Datensatz nicht übereinstimmen, wird der Aktualisierungsvorgang abgebrochen. Der Vorteil ist, dass der Datensatz auch während einer Aktualisierung lesbar bleibt und die Leistung der Datenbank nicht stark beansprucht wird. Allerdings bleibt es Aufgabe der Anwendung, die Ausnahme zu behandeln. Alternativ steht auch *Pessimistic Locking*, eine Methode, die den Datensatz während einer Aktualisierung gegen andere Zugriffe sperrt, zur Verfügung.

Grundvoraussetzung für die Verwendung einer Datenbank ist eine Datenbankverbindung. Wie für Anwendungen üblich, die auf der Java-Plattform aufbauen, kommt zu diesem Zweck die Datenbankschnittstelle *Java Database Connectivity* (JDBC) zur Anwendung. Vergleichbar mit ODBC für

Windows, stellt JDBC eine einheitliche Schnittstelle für unterschiedliche relationale Datenbanken zur Verfügung. Zentrale Aufgabenbestandteile von JDBC sind Aufbau und Verwaltung von Datenbankverbindungen sowie Weiterleitung von SQL-Abfragen und Bereitstellungen der Ergebnisse. Spezielle Datenbanktreiber, die in der Regel von den Datenbankherstellern zur Verfügung gestellt werden, übersetzen die Befehle in datenbankspezifische Äquivalente.

Die Konfiguration der Datenbankverbindung wird in der Datei `DataSource.groovy` im Ordner `grails-app/conf/` vorgenommen. Die Parameter `url`, `username`, `password` und `driverClassName` geben die Adresse, den Benutzer, das Passwort und den Treiber für die JDBC-Verbindung an (siehe Listing 5.1; Zeile 2-5). Die Variable `dialect` definiert, in welcher Datenbanksprache mit der Datenbank kommuniziert werden soll, und mit der Variable `dbCreate` wird festgelegt, ob die Datenbank beim Initialisieren der Anwendung gelöscht und neu erstellt werden soll (`create-drop`), nur erstellt werden soll, wenn die Datenbank noch nicht existiert (`create`) oder ausschließlich aktualisiert werden soll (`update`) (siehe Listing 5.1; Zeile 6-7). Des Weiteren stehen die Parameter `pooled`, zur Aktivierung mehrere Verbindungen zu einer Datenbank und `logSql` zur Aufzeichnung von SQL-Befehle bereit (siehe Listing 5.1; Zeile 8-9). Zusätzlich können die Parameter, abhängig davon, ob die Anwendung in einer Entwicklungs-, Test- oder Produktionsumgebung läuft, definiert werden (siehe Listing 5.1; Zeile 12-22).

```
1  dataSource {
2       url = 'jdbc:postgresql://localhost:5432/name'
3       username = 'benutzer'
4       password = 'passwort'
5       driverClassName = 'org.postgis.DriverWrapper'
6       dialect = org.hibernatespatial.postgis.PostgisDialect
7       dbCreate = update
8       pooled = false
9       logSql = false
10 }
11
12 environments {
13      development {
14           dataSource {...}
15      }
16      test {
17           dataSource {...}
18      }
19      production {
20           dataSource {...}
21      }
22 }
```

Listing 5.1: Konfiguration der Datenbankverbindung

Die weitere Beschreibung der Datenebene wird unterteilt nach Sachdaten, Geometriedaten und Binärdaten von Fotos sowie Dokumenten.

5.2.2 Sachdaten

Unter dem Begriff Sachdaten werden alle textkodierten Informationen ohne Raumbezug zusammengefasst und damit schließt der Begriff Geodaten sowie Binärdaten aus. Obwohl diese Gruppe an Daten den größten Anteil ausmacht, ist sie gleichzeitig auch am einfachsten zu handhaben, weil sie dem Standardproblem einer Webanwendung entspricht. Die Definition von Randbedingungen und Beziehungen ist auf Grund der komplexen Datenstruktur, die in Kapitel 3 analysiert wurde, jedoch entsprechend aufwendig.

Für die Speicherung der Daten stehen die üblichen Datentypen der Java-Plattform wie `String`, `Date` und `int` zur Verfügung (siehe Listing 5.2; Zeilen 3-11). Diese können mit Hilfe von Randbedingungen, so genannten `constraints`, genauer spezifiziert werden. Abhängig vom Datentyp können folgende Bedingungen definiert werden:

`blank:` legt fest, ob ein Attribut vom Datentyp String leer sein darf.
`email:` stellt sicher, dass ein String einer gültigen Email-Adresse entspricht.
`inList:` der Wert eines Attributs muss einem in einer Liste aufgeführten Wert entsprechen.
`matches:` stellt sicher, dass der Wert einem benutzerdefinierten regulären Ausdruck entspricht.
`maxSize:` legt die maximale Länge eines Attributs fest.
`minSize:` legt die minimale Länge eines Attributs fest.
`nullable:` gibt an, ob ein Attribut null sein darf.
`range:` legt einen numerischen Wertebereich fest, in dem ein Attribut liegen muss.
`scale:` bestimmt die Genauigkeit von Attributen vom Datentyp Gleitkommazahl.
`size:` bestimmt den Wertebereich eines Attributs und entsprecht einer Kombination aus `maxSize` und `minSize`.
`unique:` gibt an, ob Werte nur einmalig vorkommen dürfen.
`url:` stellt sicher, dass ein String einer gültigen URL entspricht.
`validator:` ermöglicht die Definition von benutzerdefinierten Einschränkungen.

Die genannten Randbedingungen wirken sich in zweifacher Hinsicht auf die Anwendung aus. Zum einen kann mit Hilfe bestimmter Randbedingungen die Generierung des Datenbankschemas angepasst werden, sodass schon auf der Datenbankebene eine Datenvalidierung stattfindet. Für Attribute vom Datentyp `String` wirken sich `inList`, `maxSize` und `size` auf die maximale Länge der entsprechenden Tabellenspalte aus. Die Randbedingungen `max`, `min`, `range` und `scale` beeinflussen die Generierung des Datenbankschemas für numerische Datentypen, indem die Genauigkeit (`precision`) der Tabellenspalte festgelegt wird.

Die zweite Auswirkung der Randbedingungen findet sich auf der funktionalen Ebene der Anwendung. Sobald ein Objekt erstellt oder verändert wird, kann mit der Methode `validate()` geprüft werden, ob die festgelegten Bedingungen erfüllt werden. In Kombination mit einer Datenvalidierung auf Ebene der Benutzeroberfläche ist auf diese Weise eine robuste Datenhaltung möglich, die ungültige Werte in der Datenbank weitestgehend ausschließt.

```
1  class Person {
2
3       String firstname
4       String lastname
5       String password
6       String email
7       int role
8       Person createdByPerson
9       Person updatedByPerson
10      Date dateCreated
11      Date lastUpdated
12
13      static constraints = {
14           password(size:3..30)
15           firstname(nullable: true)
16           lastname(nullable: true)
17           email(email:true, unique:true)
18           role(inList:[1, 2, 3, 4], size: 1)
19      }
20
21 ...
```

Listing 5.2: Ausschnitt aus der Domain Klasse `Person`

Ein weiterer, ebenso wichtiger Gesichtspunkt der Datenhaltung von Sachdaten ist die Modellierung von Beziehungen zwischen den Klassen. Ähnlich wie die Randbedingungen wirken sich diese sowohl auf das zu generierende Datenbankschema als auch auf die funktionale Ebene der Anwendung aus und bestimmen im entscheidenden Maße die Datenhandhabung innerhalb der Anwendung. Unterschieden wird zwischen der Kardinalität, der Richtung und der Kaskadierung der Beziehung. Das wichtigste Unterscheidungsmerkmal ist die Kardinalität, welche die Komplexität der Beziehung bestimmt und als *1:1* (one-to-one), *1:N* (one-to-many) oder *N:M* (many-to-many) Beziehung bezeichnet wird. Die Richtung einer Beziehung kann entweder *unidirektional* oder *bidirektional* sein und legt fest, von welcher Seite die Beziehung sichtbar ist. Mit Kaskadierung sollen in diesem Zusammenhang die Auswirkungen von Änderungen auf verwandte Objekte beschrieben werden. Als Auswirkungen kommen Erstellung, Aktualisierung und Entfernung in Frage.

Die einfachste Form einer 1:1-Beziehung wird erreicht, indem eine Klasse A als Attribut zu einer anderen Klasse B hinzugefügt wird (siehe Listing 5.2; Zeilen 3-5). Allerdings wird eine bidirektionale Beziehung erst aufgebaut, wenn auch Klasse B als Attribut zu Klasse A hinzugefügt wird. Eine andere Möglichkeit zur Implementierung einer solchen Beziehung wird durch die Eigenschaft `belongsTo` erreicht (siehe Listing 5.2; Zeile 11). Der Vorteil einer Beziehung, die mit `belongsTo` definiert wird, ist die Kaskadierung von Änderungen auf entsprechende Datensätze. Beispielsweise wird automatisch der verknüpfte Punktdatensatz aus der Tabelle entfernt, wenn ein Ereignis gelöscht wird. Entsprechend wird auch die Erstellung und Aktualisierung von Punktdaten eines Ereignisses selbstständig gehandhabt.

Eine 1:N-Beziehung wird mit der Eigenschaft `hasMany` erreicht, die für sich allein gestellt eine unidirektionale Beziehung herstellt und Erstellung sowie Aktualisierung automatisch kaskadiert (siehe Listing 5.2; Zeile 13). Erst durch den Einsatz von `belongsTo` werden verwandte Objekte auch automatisch gelöscht.

```
1  class Event {
2
3      GeoPoint point
4      Basedata basedata
5      Eventdata eventdata
6
7      static constraints = {
8          eventdata(nullable: true)
9      }
10
11     static belongsTo = [ Photo, Document ]
12
13     static hasMany = [photos: Photo, documents: Document]
14
15     static mapping = {
16         point lazy:false
17         basedata lazy:false
18         eventdata lazy:false
19     }
20 }
```

Listing 5.3: Domain Klasse `Event`

Am komplexesten ist die Realisierung und Handhabung einer N:M-Beziehung. Beide Klassen müssen jeweils mit der Eigenschaft `hasMany` verknüpft werden (siehe Listing 5.3; Zeile 13). Außerdem muss mit `belongsTo` eine verantwortliche Klasse festgelegt werden, von der aus die verknüpften Klassen verwaltet werden (siehe Listing 5.3; Zeile 11). Mit Hilfe der Methoden `addTo[Class]()` und `removeFrom[Class]()` - `[Class]` als Platzhalter für den entsprechenden Klassennamen - können nun Objekte zu Objekten der Klasse mit der Eigenschaft `belongsTo` hinzugefügt und entfernt werden.

Wenn für die Eigenschaft `hasMany` statt einer Klasse ein Datentyp angegeben wird, können auf einfache Art und Weise Listen zu einer Klasse hinzugefügt werden, für die eine entsprechende Tabelle in der Datenbank erstellt wird. Dadurch wird verhindert, dass für mehrwertige Attribute eine eigene Klasse erstellt werden muss.

Mit der mapping Eigenschaft steht eine weitere Möglichkeit zur Verfügung, Einfluss auf das Datenmodell zu nehmen. Unter anderem kann festgelegt werden, wie die Objekte und Attribute auf die Datenbank abgebildet werden sollen. Außerdem kann definiert werden, ob zusammenhängende Daten komplett (*eager fetching*), so wie in Zeile 16-18 von Listing 5.3, oder nach Bedarf aus der Datenbank geladen werden sollen.

5.2.3 Geometriedaten

Im weiteren Sinne haben alle Naturereignisdaten einen Raumbezug und könnten folglich den Geodaten zugeordnet werden. In diesem Kapitel liegt der Fokus jedoch auf den eigentlichen Geometriedaten, die eine geographische Lagebestimmung der Naturereignisdaten erlauben.

Auf Datenbankebene werden die Geometriedaten entsprechend der *Simple Features for SQL* (SF SQL) OGC-Spezifikation in einer um PostGIS erweiterten PostgreSQL-Datenbank gespeichert. Die

Spezifikation und damit das Geometriemodell wird von der Programmbibliothek GEOS implementiert, die als Grundlage für PostGIS dient. Die Tabellenspalten, welche die Geometriedaten enthalten, sind vom Datentyp `geometry`, der unhabhängig vom Geometrietyp ist und sowohl Punkt, Linie und Fläche sowie die entsprechenden Multi-Varianten speichern kann. Weitere Metadaten wie Koordinatensystem, Dimension und Geometrietyp werden in der `geometry_columm` Tabelle gespeichert, die zwar keinen direkten Einfluss auf die Webanwendung haben, aber beispielsweise für einen WFS auf Basis der Geodaten von Bedeutung sind.

Die Datenebene der Webanwendung verlangt nach einer vergleichbaren Repräsentation der Geometriedaten. Für diese Aufgabe steht auf der Java-Plattform mit der Java Topology Suite (JTS) Programmbibliothek das Vorbild von GEOS zur Verfügung, welche sich ohne weitere Einschränkungen für diesen Zweck einsetzen lässt. Auf diese Weise stehen auch auf Anwendungsebene das Geometriemodell und die Topologiefunktionen der SF SQL Spezifikation bereit.

```
1  import com.vividsolutions.jts.geom.Point
2
3  class GeoPoint {
4
5       int method
6       int quality
7       int auxiliaries
8       Point geometry
9
10      static constraints = {
11           method(inList:[1, 2, 3], size: 1)
12           auxiliaries(inList:[1, 2, 3, 4, 5, 6, 7, 8, 9], size: 1)
13           quality(inList:[1, 2, 3], size: 1)
14     }
15
16      static belongsTo = [Event, Basedata, Photo]
17
18      static mapping = {
19           columns {
20           geometry type: org.hibernatespatial.GeometryUserType
21           }
22      }
23 }
```

Listing 5.4: Domain Klasse `GeoPoint` als Beispiel für Geometriedaten

Die Kommunikation zwischen dem Geometriemodell der Datenbankebene und der Anwendungsebene wird von der Programmbibliothek Hibernate Spatial übernommen. Als Erweiterung für Hibernate läuft die Kommunikation weitgehend transparent im Hintergrund; lediglich zwei Konfigurationsparameter werden vorausgesetzt (siehe Listing 5.1; Zeile 5-6). Im Prinzip sollte nach Anpassung der Konfiguration auch die Verwendung einer Oracle oder MySQL Datenbank möglich sein, allerdings wurde die Anwendung bisher nur auf Basis einer PostgreSQL Datenbank entwickelt und getestet. Zumindestens die Treiber für alle drei Geodatenbanksysteme sind vorhanden.

Innerhalb der Anwendung werden die Geometriedaten mit Hilfe der von JTS bereitgestellten Klassen abgebildet. Konkret kommen die Klassen `Point`, `LineString` und `Polygon` zum Einsatz, wie in Listing 5.4, Zeile 8 für die Klasse `GeoPoint` veranschaulicht wird. In den Standardein-

stellungen kennt Grails nur die üblichen Datentypen der Java-Plattform, deshalb muss Grails noch darüber in Kenntnis gesetzt werden, wie der neue Datentyp `Point` in der Datenbank gespeichert werden soll. Dies geschieht mit Hilfe der Eigenschaft `mapping` und der Hibernate Spatial Klasse `GeometryUserType` (siehe Listing 5.4; Zeile 20).

Das Koordinatensystem der Geometriedaten ist auf Datenbank- und Anwendungsebene identisch, es findet also keine Koordinatentransformation beim Speichern oder Lesen der Geometriedaten statt. Für den Schweizerischen Nationalpark und den Wildnispark Zürich wurde das zur Zeit in der Einführung befindliche neue Schweizerische Landeskoordinatensystem CH1903+ LV95 ausgewählt. Prinzipiell ist die Anwendung jedoch unabhängig von einem bestimmten Koordinatensystem. Einzig der Import von GPS-Koordinaten aus georeferenzierten Fotos wurde am Beispiel des Schweizerischen Landeskoordinatensystems implementiert (siehe Kapitel 5.3.3).

Als Austauschformat der Geometriedaten zwischen Browser und Server kommt WKT, GeoJSON und GML zum Einsatz. Die Handhabung von Geometriedaten wird in Kapitel 5.3.3 beschrieben, während der GML Import und Export in den Kapiteln 5.3.5 und 5.3.6 behandelt wird.

5.2.4 Fotos und Dokumente

Neben den Geometriedaten nehmen Fotos und Dokumente in Form von Binärdaten eine besondere Stellung aus Sicht der Datenhaltung ein. Die Frage, ob umfangreiche Binärdaten wie Fotos und Dokumente besser in einer relationalen Datenbank oder im Dateisystem gespeichert werden sollten, scheint bis heute ungeklärt. Die gebräuchlichen Datenbanksysteme stellen für diesen Zweck zumindestens den Datentyp *Binary Large Object* (BLOB) zur Verfügung.

In der Anforderungsanalyse wurde bereits deutlich, dass der Bedarf nach Skalierbarkeit der Anwendung zunächst sehr niedrig sein wird; daher wurde einer einfachen Administration und Implementierung von binären Daten ein größerer Wert beigemessen. Das Ergebnis ist eine Lösung, die eine Speicherung der binären Daten innerhalb der Datenbank als BLOB vorsieht. Auf diese Weise braucht für eine Datensicherung nur die Datenbank berücksichtigt werden und der Zugriff auf den Datenbestand kann vollständig über die Datenbankschnittstelle erfolgen.

Auf Datenebene der Webanwendung werden die binären Informationen in einem Array vom Datentyp `byte` gespeichert (siehe Listing 5.5; Zeile 3). In der Datenbank werden die Informationen in einem binären String vom Datentyp `bytea` abgelegt.

```
1  class Photo {
2
3        byte[] data
4        int direction
5        String exposureTime
6        ...
```

Listing 5.5: Ausschnitt aus der Domain Klasse `Photo`

5.3 Steuerungsebene

5.3.1 Verwaltung von Datensätzen

Die Verwaltung von Datensätzen umfasst die Grundfunktionalitäten Erstellen, Anzeigen, Aktualisieren sowie Löschen und bildet daher die Basis eines jeden Informationssystems. Nach einer allgemeinen Beschreibung der Implementierung dieser vier Funktionen folgen einige Anmerkungen zu datensatzspezifischen Anpassungen.

Alle vier Verwaltungsaufgaben werden als Methoden in einer Kontroller Klasse zusammengefasst und beziehen sich in der Regel auf eine Domain Klasse der Datenebene. Die Kontroller Klassen sind jedoch nicht auf eine Klasse der Datenebene beschränkt, sondern können nach Bedarf weitere Klassen einbeziehen. Die folgende Beschreibung bezieht sich auf die fiktive Domain Klasse Datensatz, berücksichtigt mit textkodierten Informationen nur den Bereich der Sachdaten und ist möglichst allgemein gehalten.

Die Erstellung eines Datensatzes beginnt mit der Instanziierung eines Objekt `datensatz` der Klasse `Datensatz`. Als Parameter werden die Eigenschaften des neuen Datensatzes in der Variable `params` übergeben (siehe Listing 5.6; Zeile 2). Bevor der Datensatz gespeichert wird, kann er mit Hilfe der Methode `validate()` auf die in der Domain Klasse definierten Randbedingungen und Datentypen überprüft werden (siehe Zeile 3). Anschließend wird sichergestellt, dass der Datensatz keine Fehler enthält und es wird versucht, den Datensatz zu speichern (siehe Zeile 4). Wenn eine der Bedingungen nicht erfolgreich abläuft, werden die Datensatzparameter zusammen mit dem Erstellungsformular in Zeile 9 zurückgegeben. Andernfalls wird in die Variable `flash.message` eine Nachricht zur erfolgreichen Erstellung des Datensatzes geschrieben und der entsprechende Datensatz angezeigt.

Mit dem Objekt `flash` können Variablen dem nächsten Seitenaufruf übergeben werden. Anschließend wird das Objekt geleert. Für Aufgaben, die eine längere Lebenszeit der Variablen benötigen, steht das Objekt `session` zur Verfügung, welches erst nach der Abmeldung eines Benutzers geleert wird.

```
1  def erstellen = {
2        def datensatz = new Datensatz(params)
3        datensatz.validate()
4        if(!datensatz.hasErrors() && datensatz.save()) {
5             flash.message = "Datensatz ${datensatz.id} erstellt"
6             redirect(action:show,id:datensatz.id)
7        }
8        else {
9             render(view:"create",model:["datensatz":datensatz])
10       }
11 }
```

Listing 5.6: Methode zum Erstellen eines neuen Datensatzes

Am einfachsten ist, wie nicht anders zu erwarten, die Implementierung der Funktion Anzeigen. Wenn die ID eines Datensatzes zur Verfügung steht, kann mit Hilfe der Methode `get()` der entsprechenden Domain Klasse ein Datensatz aus der Datenbank geladen werden (siehe Listing 5.7; Zeile 2). Die Möglichkeit einer ungültigen ID wird anschließend behandelt, indem eine entsprechende Nachricht gesetzt und zurück zur Startseite umgeleitet wird (siehe Zeile 3-6). Wenn der Datensatz erfolgreich aus der Datenbank geladen werden konnte, wird er als Variable `datensatz` der Präsentationsebene übergeben (siehe Zeile 8).

```
1  def anzeigen = {
2        def datensatz = Datensaz.get( params.id )
3        if(!datensaz) {
4              flash.message = "Datensatz ${params.id} nicht gefunden"
5              redirect(uri:"/")
6        }
7        else {
8              return ["datensatz":datensatz]
9        }
10 }
```

Listing 5.7: Methode zum Anzeigen eines Datensatzes

Die Methoden für das Aktualisieren und Löschen von Datensätzen beginnen genauso wie die Anzeige-Methode, indem ein Datensatz mit Hilfe einer ID aus der Datenbank geladen und geprüft wird, ob der Datensatz vorhanden ist.

```
1  def loeschen = {
2        def datensaz = Datensaz.get( params.id )
3        if(datensaz) {
4              try {
5                    datensaz.delete(flush:true)
6                    flash.message = „Datensaz ${params.id} gelöscht“
7                    redirect(uri:"/")
8              }
9              catch(...DataIntegrityViolationException e) {
10                   flash.message = "Datensaz konnte nicht gelöscht werden"
11                   redirect(action:show,id:params.id)
12             }
13       }
14       else {
15             flash.message = "Datensatz ${params.id} nicht gefunden"
16             redirect(uri:"/")
17       }
18 }
```

Listing 5.8: Methode zum Löschen eines Datensatzes

Soll der Datensatz gelöscht werden, wird innerhalb eines `try` Block die Methode `delete()` aufgerufen; der Parameter `flush:true` weist Hibernate an, die Datenbankoperation sofort auszuführen, nicht erst beim nächsten anstehenden Datenbankzugriff (siehe Listing 5.8; Zeile 4-8). Wenn die Löschung erfolgreich verlief, wird eine entsprechende Nachricht übergeben und auf die Startseite umgeleitet. Ansonsten wird eine Fehlermeldung und der Datensatz selber angezeigt.

Die Aktualisierung eines Datensatz wird durch die Möglichkeit einer zwischenzeitlichen Aktualisierung durch einen anderen Benutzer zur komplexesten Verwaltungsfunktion. Wie in Kapitel 5.2.1 beschrieben, setzt Grails für diesen Zweck in der Standartkonfiguration optimistic locking ein. Wenn ein zu speichernder Parameterdatensatz die Variable version enthält und diese niedriger ist als die aktuelle Version des zu aktualisierenden Datensatzes, wird eine Fehlermeldung mit dem Aktualisierungsformular zurückgegeben und die Operation abgebrochen (Listing 5.9; Zeile 4-14). Ansonsten entspricht der weitere Ablauf der Erstellung eines Datensatzes.

```
1  def aktualisieren = {
2       def datensatz = Datensatz.get( params.id )
3       if(datensatz) {
4            if(params.version) {
5                 def version = params.version.toLong()
6                 if(datensatz.version > version) {
7                      datensatz.errors.rejectValue("version",
8                           "datensatz.optimistic.locking.failure",
9                           "Ein anderer Benutzer hat den Datensatz
10                          in der Zwischenzeit aktualisiert.“)
11                     render(view:"edit",model:["datensatz":datensatz])
12                     return
13                }
14           }
15           datensatz.properties = params
16           datensatz.validate()
17           if(!datensatz.hasErrors() && datensatz.save()) {
18                flash.message = "Datensatz ${params.id} aktualisiert"
19                redirect(action:show,id:datensatz.id)
20           }
21           else {
22                render(view:"edit",model:["datensatz":datensatz])
23           }
24      }
25      else {
26           flash.message = "Datensatz ${params.id} nicht gefunden"
27           redirect(uri:"/")
28      }
29 }
```

Listing 5.9: Methode zum Aktualisieren eines Datensatzes

Um den spezifischen Anforderungen der unterschiedlichen Domain Klassen gerecht zu werden, waren teilweise erhebliche Anpassungen an den allgemein beschriebenen Verwaltungsfunktionen notwendig. Im Folgenden sollen einige zentrale Modifikationen herausgegriffen und kurz erläutert werden.

Alle Datensätze mit Raumbezug, also Grunddaten, Fotos und Datensätze der dritten Erfassungsstufe, müssen um eine Handhabung der Geodaten erweitert werden. Im Vordergrund der Modifikation steht die Konvertierung zwischen dem internen Geometriemodell und dem Ausgabeformat bzw. Eingabeformat. Außerdem ist in Bezug auf georeferenzierte Fotos eine Koordinatentransformation notwendig. Diese Funktionalitäten wurden als Service ausgelagert und werden in Kapitel 5.3.3 detailliert beschrieben.

Eine weitere Ausnahme stellt die Erfassung von Fotos dar. Während die Binärdaten problemlos von dem Grails-Framework verarbeitet werden, erfordert die automatische Erfassung von Metadaten eine umfangreiche Erweiterung der Erfassungsmethode, die ebenfalls als Service ausgelagert wurde und in Kapitel 5.3.2 beschrieben wird.

Erwähnenswert ist noch der Umgang mit der Positionsinformation von Ereignissen. Neben dem Aufnahmeort, der zusammen mit den Grunddaten erstellt und aktualisiert wird, besitzt jedes Ereignis einen weiteren Punktdatensatz. Dieser wird zur Positionierung von Ereignissen auf der Übersichtskarte verwendet und entspricht dem Aufnahmeort der Grunddaten, wenn keine Daten der dritten Erfassungsstufe vorhanden sind. Sobald Geometriedaten der dritten Stufe hinzugefügt werden, wird daraus der Mittelpunkt berechnet und von nun an als Positionierungsdatensatz verwendet (siehe Linsting 5.10). Diese Anpassung betrifft also nicht nur die Klasse der Grunddaten, sondern auch die Klassen der dritten Erfassungsstufe.

```
1 CentroidPoint centroid = new CentroidPoint()
2 geo.lines.each {
3       centroid.add(it.geometry.getCentroid())
4 }
5 geo.polygons.each {
6       centroid.add(it.geometry.getCentroid())
7 }
```

Listing 5.10: Berechnung der neuen Ereignisposition

Eine weitere Aufgabe der Anwendung, die als Service realisiert wurde und in Kapitel 5.3.4 erläutert wird, sind die Abfragemöglichkeiten nach Naturereignissen. Darüber hinaus waren für administrative Funktionen Modifikationen an dem entsprechenden Kontroller notwendig, die allerdings nicht im Mittelpunkt dieser Arbeit stehen und daher nicht weiter beschrieben werden.

5.3.2 Handhabung von Fotos

Die Verwaltung von Fotos folgt weitestgehend der allgemeinen Beschreibung aus Kapitel 5.3.1 und verfügt dementsprechend über eine vergleichbare Kontroller-Klasse. Alleinige Ausnahme ist die Erfassung von Fotos, die um einen automatischen Import von Metadaten ergänzt wurde. Anstelle einer neuen Instanz wird zur Erstellung eines neuen Fotodatensatzes die Methode `createPhoto()` der Service Klasse `PhotoService` mit den entsprechenden Parametern aufgerufen.

Die Metadaten eines Fotodatensatzes werden aus den EXIF-Informationen der JPEG-Dateien gelesen. Für diesen Zweck wird die Java-Programmbibliothek *metadata extractor*[1] eingesetzt, die genau für diesen Einsatzbereich entwickelt wurde.

Zunächst wird der EXIF-Teil aus einer JPEG-Datei extrahiert (siehe Listing 5.11; Zeile 1-3). Wenn der EXIF-Teil vorhanden ist, werden die Informationen ausgelesen und in Form eines Directory, welches einen leichten Zugriff auf die Informationen ermöglicht, aufbereitet (siehe Zeile 5-7).

Nun stehen die Informationen mit den entsprechenden EXIF-Bezeichnungen als Schlüsselwort, zur Verfügung (EXIF, 2002). Im konkreten Fall wird der Aufnahmezeitpunkt (`TAG_DATETIME_ORIGINAL`), die Belichtungszeit (`TAG_EXPOSURE_TIME`), die Blende (`TAG_FNUMBER`), die Brennweite (`TAG_FOCAL_LENGTH`), die ISO-Zahl (`TAG_ISO_EQUIVALENT`), der Herstellername (`TAG_MAKE`) und die Kameramodellbezeichnung (`TAG_MODEL`) ausgelesen.

1 http://www.drewnoakes.com/code/exif/ [Abruf: 15.9.2009]

Die Qualität der EXIF-Daten ist von Kamera zu Kamera leider sehr unterschiedlich. Daher muss für jede Information geprüft werden, ob sie überhaupt vorhanden ist, und als Datentyp kommt nur ein String in Frage, weil teilweise die Einheit eines Wertes - beispielsweise mm für Brennweite - enthalten ist (siehe Zeile 11-30).

Wenn GPS-Koordinaten in den EXIF-Daten enthalten sind, werden diese, wie in Kapitel 5.3.3 erläutert, importiert; andernfalls wird der vom Benutzer angegebene Aufnahmeort verwendet (siehe Zeile 32-37). Gesetzt den Fall, dass ein Foto keine EXIF-Daten enthält, wird für die Metadaten der Wert *unbekannt* eingetragen.

```
1  Metadata metadata = new Metadata();
2  JpegSegmentReader reader = new JpegSegmentReader(photo.data);
3  byte[] exifSegment = reader.readSegment(JpegSegmentReader.SEGMENT_APP1);
4
5  if ( exifSegment != null ) {
6       new ExifReader(exifSegment).extract(metadata);
7       Directory exifDirectory = metadata.getDirectory(ExifDirectory.class);
8
9       SimpleDateFormat sdf = new SimpleDateFormat("yyyy:MM:dd HH:mm:ss");
10
11      try {
12           params["dateRecorded"] = sdf.parse(
13                exifDirectory.getString(ExifDirectory.TAG_DATETIME_ORIGINAL))
14      } catch ( Exception e )  {
15           params["dateRecorded"] = new Date()
16      }
17
18      String s = exifDirectory.getString(ExifDirectory.TAG_EXPOSURE_TIME);
19      if ( s != null ) params["exposureTime"] = s
20      else             params["exposureTime"] = unknown
21
22      s = exifDirectory.getString(ExifDirectory.TAG_FOCAL_LENGTH);
23      if ( s != null ) params["focalLength"] = s
24      else             params["focalLength"] = unknown
25
26      ...
27
28      s = exifDirectory.getString(ExifDirectory.TAG_MODEL);
29      if ( s != null ) params["model"] = s
30      else             params["model"] = unknown
31
32      if (params.point.exif == "true") {
33           ...
34      } else {
35           def geoPoint = geoService.createGeoPoint(params["point"])
36           params.pointRecorded = geoPoint
37      }
38 } else {
39      ...
40 }
```

Listing 5.11: Import von EXIF-Daten

5.3.3 Handhabung von Geometriedaten

Wie bereits angedeutet, ist für Geometriedaten eine gesonderte Handhabung notwendig, die eine Koordinatentransformation und Formatkonvertierung erlaubt.

Die Koordinatentransformation beschränkt sich einzig auf die GPS-Koordinaten, die in den EXIF-Daten eines Fotos gespeichert sein können; die übrigen Geometriedaten bleiben erleichternswerterweise von der Datenbankebene bis zur Präsentationsebene im gleichen Koordinatensystem. Darüber hinaus stellt die Benutzeroberfläche verschiedene Koordinatensysteme für die Anzeige von Punktkoordinaten zur Verfügung.

Zusätzlich muss eine Ein- und Ausgabe der Geometriedaten in unterschiedlichen Formaten möglich sein. Als Eingabeformate kommen WKT, GML und GPS-Koordinaten aus den EXIF-Daten in Frage, während die Ausgabe nur in den Formaten WKT und GML realisiert werden muss.

Für das WKT-Format, welches innerhalb der Benutzeroberfläche eingesetzt wird, stellt JTS die Klassen `WKTReader` und `WKTWriter` zur Verfügung (siehe Listing 5.12; Zeile 1-2). Während mit der Reader-Klasse eine einfache Möglichkeit zum Einlesen von Geometriedaten im WKT-Format in das JTS Geometriemodell bereit steht (siehe Zeile 3-5), können mit der Writer-Klasse die Informationen aus dem JTS-Geometriemodell als WKT ausgegeben werden (siehe Zeile 6). Für alle drei Geometrietypen wurde für diese beiden Aufgaben jeweils eine Methode implementiert und in einer Service Klasse zusammengefasst.

```
1  WKTReader reader = new WKTReader()
2  WKTWriter writer = new WKTWriter()
3  Point geometry = (Point)reader.read(params["geometry"])
4  LineString geometry = (LineString)reader.read(params["geometry"])
5  Polygon geometry = (Polygon)reader.read(params["geometry"])
6  def wktGeometry = writer.write(geometry)
```

Listing 5.12: Formatkonvertierung von Geometriedaten zwischen WKT und JTS

Die GPS-Koordinaten eines georeferenzierten Fotos werden, wie in Kapitel 5.3.2 erläutert, aus den EXIF-Daten gelesen (siehe Listing 5.13; Zeile 1-3). Anschließend müssen die Koordinaten vom WGS84 Referenzsystem in das Schweizerische Landeskoordinatensystem CH1903+ LV95 konvertiert werden. Die zugrundeliegende Berechnungsformel wurde, wie in swisstopo (2008) beschrieben, implementiert (siehe Zeile 4-11). Die Genauigkeit der Transformation spielt sich im Bereich der Empfangsgenauigkeit von GPS-Handgeräten ab und ist damit für diesen Zweck ausriechend. Im weiteren Verlauf können die konvertierten GPS-Koordinaten als WKT behandelt werden.

Neben den beschriebenen Eingabeformaten ist der Import von GML-Datein möglich, der in Kapitel 5.3.5 ausführlich beschrieben wird. Von zentraler Bedeutung für die Awendung ist außerdem eine Export-Möglichkeit, damit die Daten beispielsweise in einem GIS genutzt werden könne. Für diese Aufgabe, deren Realisierung in Kapitel 5.3.6 geschildert wird, kommt ebenfalls das GML-Format zur Anwendung.

```
Directory gpsDirectory = metadata.getDirectory(GpsDirectory.class);
Rational[] lon = gpsDirectory.getRationalArray(GpsDirectory.TAG_GPS_LONGITUDE);
Rational[] lat = gpsDirectory.getRationalArray(GpsDirectory.TAG_GPS_LATITUDE);
def p = ((getDecimalDegree(lat)*3600)-169028.66)/10000;
def h = ((getDecimalDegree(lon)*3600)-26782.5)/10000;
def y = 600072.37 + 211455.93 * h - 10938.51 * h * p - 0.36 * h
     * Math.pow(p,2) - 44.54 * Math.pow(h,3);
def x = 200147.07 + 308807.95  * p + 3745.25 * Math.pow(h,2) + 76.63
     * Math.pow(p,2) - 194.56 * Math.pow(h,2) * p + 119.79 * Math.pow(p,3);
def east = x + 1000000;
def north = y + 2000000;

def getDecimalDegree (Rational[] data) {
     double result
     double deg = data[0].doubleValue();
     double min = data[1].doubleValue();
     double minFrac = min - Math.floor(min);
     double sec = data[2].doubleValue();
     if ( minFrac > 0 ) {
          min = Math.floor(min);
          sec += minFrac * 60;
     }
     return result = deg+(((min*60)+(sec))/3600);
}
```

Listing 5.13: Koordinatentransformation von GPS-Koodinaten in CH1903+ LV95

5.3.4 Filter- und Suchfunktionen

Eine zentrale Voraussetzung, die von einem produktiv einsetzbaren Informationssystem erfüllt werden muss, ist die Möglichkeit bestimmte Datensätze wiederzufinden. Für das Naturereignisinformationssystem wurde für diese Aufgabe eine Kombination aus Filter- und Suchfunktion realisiert. Während mit Filtern die Ereignisse nach ausgewählten Kriterien eingrenzt werden können, kann mit Hilfe der Suchfunktion nach benutzerdefinierten Zeichenketten gesucht werden. Eine Kombination aus mehreren Filterkriterien und Suchwörtern erlaubt auf diese Weise auch in großen Datenmengen die Identifizierung von einzelnen Datensätzen.

Folgende Filtertypen stehen neben der Textsuche, die sich auf die Felder Ortsbezeichnung und Beschreibung aus den Grunddaten sowie den Hinweis zur Erfassungsmethode aus den Ereignisdaten bezieht, zur Auswahl:

Ereignistyp: zur Auswahl stehen alle 22 Ereignistypen, die in der zweiten Erfassungsstufe angegeben werden können.
Ereigniszeitraum: bezieht sich auf die Angabe zum Zeitraum des Ereignisses, die in der zweiten Erfassungsstufe aufgenommen wird. Zur Auswahl steht ein benutzerdefinierter Zeitraum.

Entdeckungszeitpunkt: der Entdeckungszeitpunkt, der in der ersten Erfassungsstufe aufgenommen wird, kann ebenfalls nach einem Zeitraum gefiltert werden.
Ereignisgröße: zur Auswahl stehen die fünf Ereignisgrößen, die in der zweiten Stufe erfasst werden.
Benutzer: bezieht sich sowohl auf Grunddaten als auch auf Ereignisdaten und berücksichtigt die Erstellung und die letzte Aktualisierung. Zur Auswahl stehen alle registrierten Benutzer des Systems.

Darüber hinaus dient die Kartenansicht als einfache, räumliche Filtermöglichkeit auf Basis einer *bounding box*. Sobald der Benutzer die Kartenansicht verändert, wird der neue Kartenrahmen der Filterfunktion als Kriterium übergeben.

Obwohl die Filter- und Suchfunktion bisher nur von einem Kontroller verwendet wird, wurde sie aus Gründen der Flexibilität und Übersichtlichkeit in einen Service ausgelagert. Die gesamte Funktionalität basiert auf der *CriteriaAPI* von Hibernate. Als Parameter werden der Klasse `SearchService` die Filtertypen sowie Filterkriterien übergeben und als Liste für jeden Filtertyp in einer `HashMap` gespeichert.

Für jede Domain Klasse mit Feldern, nach denen gefiltert werden soll, muss eine Instanz der Klasse `Criteria` erstellt werden, die alle von der Criteria-Instanz der Domain Klasse Event abstammen (siehe Listing 5.14; Zeile 1-4). Die Sortierung der Ergebnisliste wird nach der Ereignis ID und damit dem Erstellungszeitpunkt vorgenommen (siehe Zeile 6-7).

```
1  def session = sessionFactory.getCurrentSession()
2  Criteria eventCriteria = session.createCriteria(Event.class)
3  Criteria eventdataCriteria = eventCriteria.createAlias("eventdata","e",4)
4  Criteria basedataCriteria = eventCriteria.createAlias("basedata","b",4)
5
6  Order eventOrder = new Order("id", false)
7  eventCriteria.addOrder(eventOrder)
```

Listing 5.14: Instanziierung der Criteria Klassen und Sortierung der Ergebnisliste

Die räumliche Einschränkung auf das sichtbare Kartenfeld wird mit der Methode `within()` der Klasse `SpatialRestrictions` realisiert und auf den Punktdatensatz eines Ereignisses angewendet (siehe Listing 5.15; Zeile 3-4). Im Voraus muss der im WKT-Format übergebene Kartenrahmen in eine JTS Geometrie vom Typ Polygon konvertiert werden (siehe Zeile 1-2).

```
1  WKTReader fromText = new WKTReader();
2  Polygon bb = fromText.read(params.bb)
3  Criteria pointCriteria = eventCriteria.createCriteria("point")
4  pointCriteria.add(SpatialRestrictions.within("geometry", null, bb))
```

Listing 5.15: Räumlicher Filter

Die Filterung nach dem Kriterium Ereignistyp wendet, wie in Listing 5.16 dargestellt, die Methode `eq()` der Klasse `Restrictions` an und grenzt damit die Ergebnisliste auf den angegebenen Wert ein. Wenn mehrere Ereignistypen angegeben werden, kommt die Methode `or()` zur Anwendung, die zwei Filter über eine Oder-Bedingung verknüpft. Das Kriterium Ereignisgröße wird auf gleiche Art und Weise realisiert.

```
1  def restriction = Restrictions.eq("e.type",Integer.parseInt(criterions.type[0]))
2  for (def i = 1; i < criterions.type.size(); i++) {
3       restriction = Restrictions.or(restriction,
4            Restrictions.eq("e.type",Integer.parseInt(criterions.type[i])))
5  }
6  eventdataCriteria.add(restriction)
```

Listing 5.16: Filterung nach Ereignistyp

Für die Kriterien Ereigniszeitraum und Entdeckungszeitpunkt muss zunächst der übergebene Zeitraum in zwei Felder aufgeteilt und als Datum interpretiert werden (siehe Listing 5.17; Zeile 1-4 und 12-14). Während für den Entdeckungszeitpunkt nur eine `between()` Bedingung notwendig ist, müssen für den Ereigniszeitraum zwei `between()` Bedingungen für das Anfangsdatum und das Enddatum mit einer Oder-Bedingung verknüpft werden (siehe Zeile 6-9 und 15-21).

```
1  SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
2  def dates = criterions.date[0].split(",")
3  def dateStart = dateFormat.parse(dates[0])
4  def dateEnd = dateFormat.parse(dates[1])
5
6  def restriction = Restrictions.or(
7       Restrictions.between("e.dateStart", dateStart, dateEnd+1),
8       Restrictions.between("e.dateEnd", dateStart, dateEnd+1)
9  )
10
11 for (def i = 1; i < criterions.date.size(); i++) {
12      dates = criterions.date[i].split(",")
13      dateStart = dateFormat.parse(dates[0])
14      dateEnd = dateFormat.parse(dates[1])
15      restriction = Restrictions.or(
16           restriction,
17           Restrictions.or(
18                Restrictions.between("e.dateStart", dateStart, dateEnd+1),
19                Restrictions.between("e.dateEnd", dateStart, dateEnd+1)
20           )
21      )
22 }
23 eventdataCriteria.add(restriction)
```

Listing 5.17: Filterung nach Ereigniszeitraum

Die Filterung nach Benutzern und die Textsuche wird nach dem gleichen Muster realisiert. Während der Filtertyp Benutzer jeweils auf die Grunddaten und Ereignisdaten mit einer `eq()` Bedingung für die Erstellung und einer weiteren für die letzte Aktualisierung, die mit `or()` Bedingungen verknüpft werden, angewendet wird, kommt für den Filtertyp Text die Bedingung `ilike()` zur Anwendung, die unabhängig von der Groß- und Kleinschreibung die Felder Beschreibung, Ortsbezeichnung sowie Hinweise zur Erfassungsmethode nach einer Zeichenkette durchsucht.

Die Ergebnisliste kann entweder im JSON-Format für die Verarbeitung im Browser oder als GML-Datei für den Datenexport ausgegeben werden.

5.3.5 GML Import

Die Erfassung von Geodaten wird in der dritten Erfassungsstufe mit unterschiedlichen Geräten und Methoden erfolgen. Daher ist neben der manuellen Erfassung innerhalb des Browsers, die nur eine beschränkte Genauigkeit und Komplexität erlaubt, eine Möglichkeit zum Import von Geodaten notwendig.

Während der Entwicklung wurden unterschiedliche Ansätze und Datenformate für diese Aufgabe in Erwägung gezogen. Eine Möglichkeit, die von Martin Tomko, Mitarbeiter an der Universität Zürich, vorgeschlagen wurde, ist der Einsatz von OGR2OGR[2], einer Open Source Bibliothek zur Konvertierung von georeferenzierten Vektordaten. Obwohl auf diese Weise eine sehr hohe Flexibilität und zahlreiche Eingabeformate möglich wären, wurde der Vorschlag auf Grund des hohen Einrichtungs- und Integrationsaufwandes der Bibliothek nicht umgesetzt.

Eine weitere Möglichkeit bieten Java-Bibliotheken wie GeoTools[3] und deegree[4], die als Eingabeformate Shapefiles und teilweise auch GML unterstützen. Letztendlich wurde als Eingabeformat das Simple Feature Profil von GML festgelegt (OGC, 2006a), welches - wie sich im Laufe der Entwicklung herausstellte - auch mit den Hilfsmitteln, die Groovy und Grails zur Verfügung stellen, eingelesen werden kann.

Als XML-Dialekt kann GML mit Hilfe der Klasse *XmlSlurper*, die von Groovy bereit gestellt wird, auf einfache Art und Weise importiert werden. Zunächst wird die Datei eingelesen und die verwendeten Namensräume werden angegeben (siehe Listing 5.18; Zeile 1-3). Anschließend wird für jedes GML-Element *featureMember* ein Objekt von der Klasse des entsprechenden Geometrietyps erstellt, welches die Geometriedaten und Angaben zur Aufnahmemethode, Hilfsmitteln und Qualität enthält. Dieser Ablauf wird am Beispiel eines Punktdatensatzes in Listing 5.18 veranschaulicht. Für das GML-Eingabeformat wurde ein XML-Schema nach dem Level 0 Simple Feature Profil entwickelt, welches auf der beiliegenden CD unter `/gml/gneis.xsd` zusammen mit einer entsprechenden GML-Datei zu finden ist.

```
1  def gneis = new XmlSlurper().parse(file.getInputStream())
2  gneis.declareNamespace(gml: 'http://www.opengis.net/gml',
3       gneis: 'http://neis.geo.uzh.ch/gneis')
4
5  gneis.'gml:featureMember'.each{
6       geoParams['method'] = it.'*'[0].'gneis:method'.toString()
7       geoParams['auxiliaries'] = it.'*'[0].'gneis:auxiliaries'.toString()
8       geoParams['quality'] = it.'*'[0].'gneis:quality'.toString()
9
10      if(it.'*'[0].name() == 'GeoPoint') {
11           def lonlat = it.'*'[0].'gml:pointProperty'.toString().split([' '])
12           geoParams['geometry'] = (Point)reader
13                .read('POINT ('+lonlat[0]+' '+lonlat[1]+')')
14           points.add(new GeoPoint(geoParams))
15      }
16      ...
17 }
```

Listing 5.18: GML Import

2 http://www.gdal.org/ogr2ogr.html [Abruf: 15.9.2009]

3 http://geotools.org/ [Abruf: 15.9.2009]

4 http://deegree.org/ [Abruf: 15.9.2009]

5.3.6 GML Export

Das Naturereignisinformationssystem, wie es bisher realisiert ist, bietet nur einfache Anzeige- und Abfragemöglichkeiten. Eine sinnvolle und nachhaltige Nutzung der erfassten Naturereignisdaten ist erst möglich, wenn die Daten auch in anderen Anwendungen, beispielsweise in Geoinformationssystemen, zur Verfügung stehen. Ein interessanter Einsatzbereich der Daten sind zum Beispiel räumliche Analysen, wie die Berechnung des Windwurfpotentials. All diese Einsatzmöglichkeiten innerhalb der Anwendung zu implementieren, ist kaum vorstellbar, daher wird der Ansatz einer Exportfunktion verfolgt.

Es wurden wiederum, wie schon für den Import, unterschiedliche Ansätze in Betracht gezogen. Nachdem GML als Importformat festgelegt war, wurde schnell deutlich, dass sich GML gleichermaßen für den Export eignet. Als offener und etablierter Standard wird es von zahlreichen Anwendungen, insbesondere im GIS-Bereich, unterstützt und lässt sich mit Hilfe von Grails ebenso einfach exportieren wie importieren.

Zunächst wurde eine Methode entwickelt, die, basierend auf Level 1 des Simple Feature Profil, alle zu exportierenden Daten in einer GML-Datei zusammenfasst. Es stellte sich jedoch heraus, dass diese Datei nur mit erheblichem Aufwand von anderen Programmen wie ArcGIS gelesen werden kann. Daher sollte auch für den Export ein GML-Format auf Level 0 Basis, welches allerdings keine 1:N-Beziehungen in einer Datei unterstützt, zur Anwendung kommen.

Das Problem der 1:N-Beziehung lässt sich auf zwei verschiedenen Wegen lösen. Entweder werden die Geometriedaten der dritten Erfassungsstufe in einer Datei zusammengefasst und über die Ereignis ID mit den allgemeinen Ereignisdaten und den Sachdaten der dritten Erfassungsstufe, die jeweils - nach Ereignistyp getrennt - in separaten Dateien exportiert werden, verknüpft. Oder aber die Sachdaten der dritten Erfassungsstufe werden mit den entsprechenden Geometriedaten in einer Datei zusammengefasst und müssten nur noch mit den allgemeinen Ereignisdaten, die in einer weiteren Datei exportiert werden, in Beziehung gesetzt werden. Der zuletzt genannte Ansatz hat den Nachteil, das für Ereignisse mit mehreren Geometrieobjekten - die vermutlich nur selten vorkommen - die Sachdaten entsprechend oft wiederholt werden müssen. Andererseits sind die Daten aus Anwendersicht wesentlich einfacher zur handhaben. Daher wurde letztendlich dieser Weg gewählt, bisher aber nur für die erste und zweite Erfassungsstufe implementiert. Die Daten der dritten Stufe lassen sich jedoch nach dem gleichen Schema exportieren. Wiederum wurde ein mit dem GML-Eingabeformat vergleichbares XML-Schema entwickelt, welches auf der beiliegenden CD unter `/gml/neis.xsd` zusammen mit einer GML-Datei als Beispiel zu finden ist.

Die Auswahl der zu exportierenden Daten wird mit Hilfe der in Kapitel 5.3.4 beschriebenen Abfragemöglichkeiten bewerkstelligt. Dadurch kann der Benutzer sehr genau festlegen, welche Daten in der GML-Datei enthalten sein sollen.

Mittels der Methode `setHeader()` wird festgelegt, dass die Datei nicht im Browser angezeigt wird, sondern als events.gml zum Herunterladen angeboten wird (siehe Listing 5.19; Zeile 1). Anschließend wird der Inhaltstyp mit `XML` angegeben und die Ergebnisliste `events` an die GSP-Datei `gml` zur Formatierung überreicht. Die eigentliche Generierung der GML-Datei nach dem XML-Schema findet damit auf der Präsentationsebene statt.

```
1 response.setHeader('Content-disposition', 'attachment;filename=events.gml')
2 render(contentType:'text/xml', view:'gml',
3       model:[events:searchService.listEvents(params)])
```

Listing 5.19: Ausgabe als GML Datei

5.4 Präsentationsebene

5.4.1 Navigationsstruktur

Die Navigationsstruktur wurde mit dem Ziel gestaltet, schnell und intuitiv bedienbar zu sein. Während des Entwurfs wurde zum Erreichen dieses Ziels ständig zwischen einer möglichst geringen Anzahl an Navigationselementen und einer möglichst flachen Navigationshierarchie abgewogen. In Abbildung 5.2 werden wichtige Elemente und die Hierarchie der Navigation visualisiert.

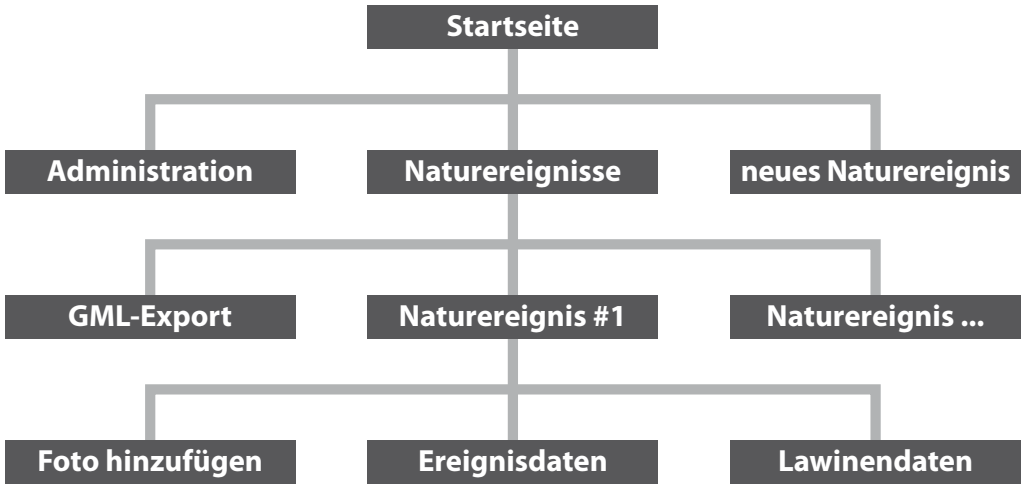


Abbildung 5.2: Navigationsstruktur des Naturereignisinformationssystems

Als zentrale Navigationselemente, die von der Startseite und auch von allen anderen Seiten erreichbar sein sollten, wurden die Administrationsseite, die Übersichtsseite aller vorhandenen Naturereignisse und das Erfassungsformular der Grunddaten zur Erstellung eines neuen Naturereignisses identifiziert.

Auf der zweiten Hierarchieebene befinden sich auf der Administrationsseite bisher die Links zur Karten- und Benutzerverwaltung (in der Abbildung nicht visualisiert). Die Übersichtsseite aller vorhandenen Naturereignisse enthält einen Link zum GML-Export sowie Navigationselemente zu den Datenblättern der eigentlichen Naturereignisse.

Die dritte Ebene der Navigationshierarchie befindet sich auf der Ansichtseite von Naturereignisdaten und stellt Elemente zum Erstellen, Bearbeiten und Löschen von Daten der zweiten und dritten Erfassungsstufe sowie von Fotos zur Verfügung.

Die Navigation wird außerdem dadurch erleichtert, dass den Benutzern nur ihrer Qualifikation entsprechende Navigationselemente präsentiert werden. Beispielsweise ist der Link zur Administrationsseite nur für Benutzer mit der Qualifikation „Administrator“ sichtbar. Dieser Ansatz wird auch für Navigationselemente der zweiten und dritten Erfassungsstufe verfolgt.

5.4.2 Seitenstruktur

Bevor die graphische Formatierung der Webanwendung im nachfolgenden Kapitel beschrieben wird, soll die allgemeine Struktur der Seite beschrieben werden. Charakteristisch für den Entwurf der Struktur war der Zwiespalt zwischen einer klassischen Webseite und einer typischen Desktop-Software, welche die Extrempositionen darstellen, zwischen denen sich die zu entwickelnde Anwendung positionieren muss.

Während der Entwicklung wurde mit unterschiedlichen Anordnungen der Seitenelemente experimentiert. Letztendlich ist ein Seitenaufbau entstanden, der - wie in Abbildung 5.3 zu sehen ist - von einer links angeordnenten Navigationsspalte geprägt wird. Die Navigation und das darüber liegende Logo haben eine feste Breite und sind fixiert, d.h. sie bewegen sich beim Scrollen nicht.

Der Hauptbereich wird in aufeinanderliegende Blöcke aufgeteilt, die jeweils eine Titelleiste und einen Inhaltsbereich aufweisen. Neben dem Titel kann die Titelleiste außerdem einen Navigationsbereich sowie den Zeitpunkt und den Benutzer der Erstellung und der letzten Aktualisierung des Blockinhaltes enthalten.

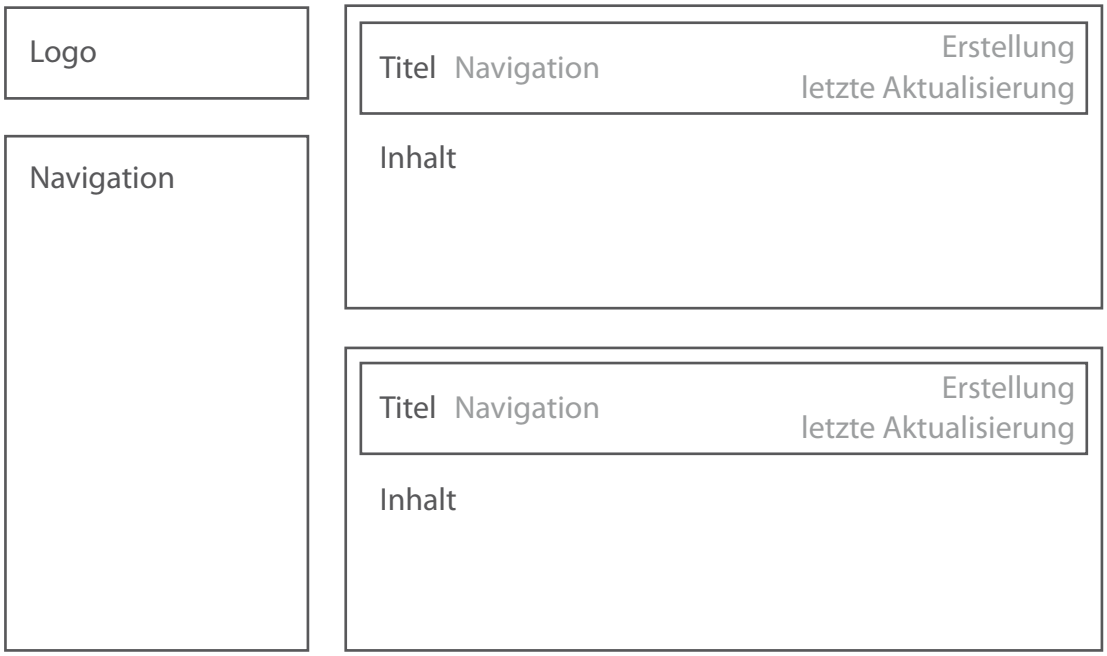


Abbildung 5.3: Allgemeine Seitenstruktur

Innerhalb des Grails-Frameworks entspricht jede Seite einer GSP-Datei, die weitesgehend aus HTML-Code besteht. Wiederkehrende Elemente im HTML-Code können mit Hilfe von XML-Tags, die Grails zur Verfügung stellt, realisiert werden. Außerdem besteht die Möglichkeit, anwendungsspezifische Tags zu definieren.

Wenn es sich um größere Bereich einer Seite handelt, die sich an mehreren Stellen der Anwendung wiederholen, können diese in so genannte Templates ausgelagert werden. Diese Möglichkeit wurde intensiv für die Handhabung der einzelnen Blöcke genutzt. Dadurch werden Abweichungen innerhalb der gleichen Blöcke verhindert und die Wartung sowie die Anpassung wesentlich vereinfacht.

Für die Domain Klassen der Grunddaten, Ereignisdaten, Lawinendaten, Fotos und Benutzer wurde jeweils ein Template mit den Formularfeldern und eines für die Anzeige erstellt. Zusätzlich wur-

den Templates für die Erstellung von Punktdatensätzen, für die Erfassung von Geometriedaten in der dritten Erfassungsstufe und für die Kartendarstellung eingerichtet.

Das seitenübergreifende Layout kann in mehreren GSP-Datein im Ordner /grails-app/views/layouts festgelegt und nach Bedarf in eine Seite eingebunden werden. Insgesamt kommt die Anwendung mit zwei Layouts, eines für die Anmeldung und das andere für die restlichen Seiten, aus. Die Zusammenführung der Elemente, aus denen sich eine Seite aufbaut, wird von der Java-Programmbibliothek SiteMesh übernommen, die in das Grails-Framework integriert ist.

Eine Ausnahme in Bezug auf die Seitenstruktur stellt die Übersichtsseite der Ereignisse dar. Diese verfügt neben der Navigationsspalte über eine weitere Spalte zur rechten Seite mit der Ereignisliste als Inhalt. Weil die Liste sehr lang und wegen der begrenzten Ressourcen nicht immer komplett geladen werden kann, müssen die Ereignisse dem Bedarf entsprechend nachgeladen werden. Der übliche Ansatz für lange Ergebnislisten im Internet, der dieses Problem mit mehreren nacheinander zu ladenden Seiten löst, ist für die Übersichtsseite nicht praktikabel, weil die Kartenansicht nicht neu geladen werden soll. Daher wird mit Hilfe des jQuery-Plugins *Endless Scroll*[5] eine Technik eingesetzt, die neue Listenelemente dynamisch nachlädt sobald der Benutzer das Ende der Liste erreicht hat und im Internet als *pageless* oder auch *endless scrolling*[6] bezeichnet wird.

5.4.3 Seitenformatierung

Die graphische Formatierung der Anwendung verfolgt einen möglichst schlichten und zweckmäßigen Ansatz. Wie in Abbildung 5.4 zu sehen ist, sind rechteckige Elemente mit breiten Rahmen und helle Grautöne als Grundfarbe charakteristisch für das Design. Das Farbschema wird durch ein helles Grün (Hex-Code `#99CC33`), welches als Hintergrundfarbe für die Titelleiste eines Blocks eingesetzt wird, und Blau (Hex-Code `#006699`), welches als Farbe für Links zum Einsatz kommt, ergänzt.

Abbildung 5.4: Screenshot des Formulars der ersten Erfassungsstufe

5 http://www.beyondcoding.com/2009/01/15/release-jquery-plugin-endless-scroll/ [Abruf: 15.9.2009]

6 http://uipatternfactory.com/p=endless-scrolling/ [Abruf: 15.9.2009]

Einzelne Elemente, die sich oft wiederholen, wurden in Form von platzsparenden Piktogrammen realisiert, die möglichst intuitiv erfassbar sein sollen. Die Gestaltung der Icons orientiert sich an der *Pictorial Communication Language*[7] (PICOL), ein Projekt, welches im Rahmen der Diplomarbeit von Melih Bilgil entstanden ist und ein reduziertes Zeichensystem zur elektronischen Kommunikation zum Ziel hat.

Zum einen wird die Qualifikation von Benutzern, wie in Abbildung 5.5 dargestellt als einfaches Icon visualisiert. Darüber hinaus werden Navigationselemente für Aktionen wie Anzeigen, Bearbeiten, Hinzufügen und Löschen als Icons dargestellt (siehe Abbildung 5.5).

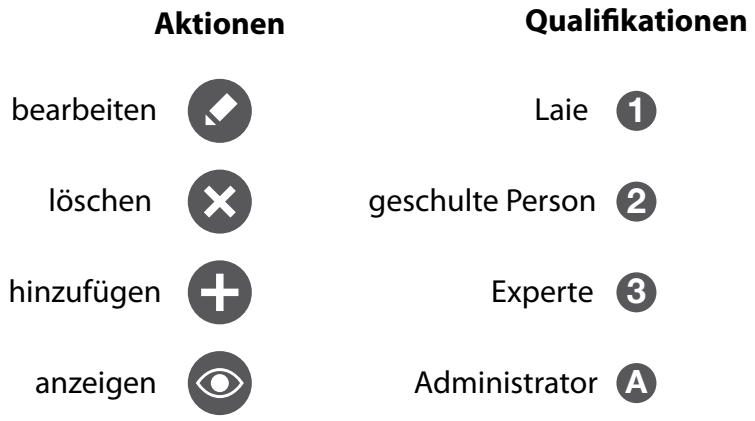


Abbildung 5.5: Icons für die Webanwendung

5.4.4 Formulare

Eine zentrale Aufgabe der Anwendung stellt die Erfassung von Daten dar, die mit Hilfe von typischen Formularen erfüllt wird. Realisiert werden die Formulare mit den üblichen HTML-Elementen wie `<label>` für die Feldbeschriftung, `<select>` für Auswahllisten, `<textarea>` für mehrzeilige Texteingabe und `<input>` für die einzeilige Texteingabe. Die Beschriftung befindet sich immer zur linken Seite eines Formularfeldes.

Unter idealen Umständen würde ein Benutzer alle Daten korrekt eingeben und die Aufgabe erfolgreich abschließen. Diese Idealvorstellung ist jedoch unrealistisch, da dem Benutzer - ob aus Unachtsamkeit oder Unwissenheit - immer wieder Fehler unterlaufen werden. An dieser Stelle kommt die Datenvalidierung ins Spiel, die schon in Kapiterl 5.3.1 unter dem Aspekt der serverseitigen Validierung von Eingabedaten und in Kapitel 5.2.2 in Bezug auf die Definition von Rahmenbedingungen angesprochen wurde. Auch auf Ebene der Benutzeroberfläche findet eine Datenvalidierung statt, die mit Hilfe des jQuery-Plugins *Validation*[8] implementiert wurde. Als Einschränkungen kommen Pflicht-, Nummern- und Email-Eingabefelder zum Einsatz, auf die der Benutzer durch rote Fehlermeldungen hingewiesen wird (siehe Abbildung 5.4).

Obwohl mittelfristig eine mobile Version der Anwendung für die Erfassung im Feld angestrebt wird, wurden für die Zwischenzeit Papierformulare entworfen, die den HTML-Formularen möglichst ähnlich sind (siehe Anhang). Dadurch soll die Erfassung im Feld auch kurzfristig ermöglicht und die Übertragung der Daten in das Naturereignisinformationssystem vereinfacht werden.

7 http://picol.org/ [Abruf: 15.9.2009]

8 http://bassistance.de/jquery-plugins/jquery-plugin-validation/ [Abruf: 15.9.2009]

5.4.5 Geodatenvisualisierung

Die Visualisierung von Geodaten gehört zu den Aufgaben, die im Mittelpunkt der Anwendung stehen. Mit Hilfe der JavaScript-Bibliothek OpenLayers konnte diese Aufgabe ohne Probleme realisiert werden.

Voraussetzung für eine praxistaugliche Geodatenvisualisierung ist eine Basiskarte, die erst eine räumliche Einordnung der dargestellten Daten erlaubt. Von OpenLayers werden zahlreiche Möglichkeiten zur Verfügung gestellt, unterschiedliche Datenquellen als Kartenebene einzubinden. Als praktikabel werden Datenquellen nach der TMS und WMS Spezifikation eingeschätzt, die sich zusammen mit anderen Karteneinstellungen im Administrationsbereich konfigurieren lassen.

Die Basiskarte soll letztendlich eine topographische Karte, eine Geländedarstellung und ein Luftbild zur Auswahl stehen. Auf Grund des großen Datenumfangs wurde ein Luftbild bisher nicht berücksichtigt, dieses sollte vielmehr direkt von der Datenquelle als WMS-Kartenebene zur Verfügung gestellt werden und kann dann ohne Probleme in die Anwendung integriert werden. Die topographische Karte und die Geländedarstellung wurden jeweils am Beispiel des Schweizerischen Nationalpark als TMS-Kartenebene realisiert und sind auf der beiliegenden CD im Ordner `/neis/web-app/map/` zu finden.

Die hervorragende topographische Karte der Schweiz, vom Bundesamt für Landestopographie swisstopo stand in den Maßstäben 1 : 25 000 und 1 : 100 000 als Datenquelle in einer ArcSDE-Datenbank zur Verfügung. Die Daten wurden mit Hilfe von ArcGIS als GeoTIFF exportiert und anschließend mit dem Kommandozeilen-Werkzeug GDAL, wie in Listing 5.20 beschrieben, aufbereitet.

```
1 gdal_translate -expand rgb karte.tif karte-rgb.tif
2 gdalwarp -s_srs 21781.prf -t_srs 2056.prf karte-rgb.tif karte-rgb-lv95.tif
3 gdal2tiles.py karte.tif
```

Listing 5.20: Aufbereitung der swisstopo-Karten mit GDAL

Zunächst mussten die Farben, welche als indizierte Farben in einem Band gespeichert waren, in RGB-Farben übersetzt werden (siehe Listing 5.20; Zeile 1). Als nächstes wurde eine Koordinatentransformation in das CH1903+ LV95 Koordinatensystem vorgenommen (siehe Zeile 2). Abschließend wurden mit Hilfe des Python-Scripts gdal2tiles.py die Kartenkacheln nach der TMS-Spezifikation erstellt. Die Zoomstufen der 1 : 25 000 und 1 : 100 000 Karte wurden in einer Karte kombiniert, sodass beim Heranzoomen automatisch das großmaßstäbige Kartenmaterial angezeigt wird.

Die Geländedarstellung wurde auf Basis von Daten aus der ArcSDE-Datenbank des Schweizerischen Nationalparks mit Hilfe von Mapnik[9] erstellt. Alle Daten mussten einer Koordinatentransformation in das CH1903+ LV95 Koordinatensystem unterzogen werden.

Als Kartenhintergrund wurde mit dem Kommandozeilen-Programm *hillshade* aus den *demtools*[10] eine Geländeschummerung, basierend auf einem digitalen Geländemodell (DGM) mit 4 Meter Auflösung, generiert. Außerdem hat das DGM zur Berechnung von Höhenlinien mit Hilfe des Programms *gdal_contour* gedient. Die Höhenlinien mit einer Äquidistanz von 20 Metern, die nach der Berechnung als Shapefiles vorlagen, wurden in eine PostGIS-Datenbank importiert.

9 http://mapnik.org/ [Abruf: 15.9.2009]

10 http://perrygeo.googlecode.com/svn/trunk/demtools/ [Abruf: 15.9.2009]

```
<Style name="raster">
     <Rule>
          <RasterSymbolizer>
               <CssParameter name="opacity">0.4</CssParameter>
          </RasterSymbolizer>
     </Rule>
</Style>

<Style name="contour-20m">
     <Rule>
          <LineSymbolizer>
               <CssParameter name="stroke">#825534</CssParameter>
               <CssParameter name="stroke-width">0.4</CssParameter>
               <CssParameter name="stroke-opacity">0.8</CssParameter>
          </LineSymbolizer>
     </Rule>
</Style>

<Layer name="hillshade" status="on">
     <StyleName>hillshade</StyleName>
     <Datasource>
          <Parameter name="type">gdal</Parameter>
          <Parameter name="file">hillshade.tif</Parameter>
          <Parameter name="format">tiff</Parameter>
     </Datasource>
</Layer>

<Layer name="tunnel" status="on">
     <StyleName>tunnel</StyleName>
     <Datasource>
          <Parameter name="type">shape</Parameter>
          <Parameter name="file">tunnel</Parameter>
     </Datasource>
</Layer>

<Layer name="20m" status="on">
     <StyleName>contour-20m</StyleName>
     <Datasource>
          <Parameter name="type">postgis</Parameter>
          <Parameter name="host">localhost</Parameter>
          <Parameter name="dbname">...</Parameter>
          <Parameter name="user">...</Parameter>
          <Parameter name="password">...</Parameter>
          <Parameter name="table">(select the_geom,height from contours
               WHERE height::integer % 10 = 0) as "contour-20"</Parameter>
          <Parameter name="estimate_extent">false</Parameter>
          <Parameter name="extent">2790000.000, 1158000.000,
               2830000.000, 1198000.000</Parameter>
     </Datasource>
</Layer>
```

Listing 5.21: Beispiel eines Mapnik Stylesheets

Darüber hinaus wurden weitere Geodaten über das Gewässernetz, die Infrastruktur, die Wald- und Gletscherfläche, die größtenteils auf dem Vector25-Datensatz der swisstopo basieren, aus der ArcSDE-Datenbank als Shapefiles exportiert. Damit stand eine umfangreiche Datengrundlage zur Generierung der Geländedarstellung zur Verfügung.

Für jede Zoomstufe wurde nun ein XML-Stylesheet entwickelt, welches genau festlegt, welche Inhalte wie dargestellt werden sollen (siehe Listing 5.22). Die Formatierungsmöglichkeiten sind mit CSS vergleichbar (siehe Zeile 1-17) und als Datenquellen können GeoTIFFs (siehe Zeile 21-25), Shapefiles (siehe Zeile 30-33) und PostGIS-Datenbanken (siehe Zeile 38-49) angegeben werden. Die XML-Stylesheets für die sechs Zoomstufen sind auf der beiliegenden CD im Ordner `/mapnik/` zu finden.

In einem Python-Script werden anschließend die Kartenparameter, wie beispielsweise der Kartenausschnitt, das Koordinatensystem, das XML-Stylesheet und die Auflösung festgelegt (siehe Listing 5.21). Sobald das Script ausgeführt wird, generiert Mapnik die entsprechende Karte, die anschließend mit gdal2tiles.py in Kacheln aufgeteilt wird. Die Kacheln der verschiedenen Zoomstufen werden nun in einer Karte nach der TMS Spezifikation kombiniert.

```
1  #!/usr/bin/env python
2  import mapnik
3  epsg2056 = '...'
4  bottomleft = mapnik.Coord(2790000.000, 1158000.000)
5  topright = mapnik.Coord(2830000.000, 1198000.000)
6  bbox = mapnik.Envelope(bottomleft, topright)
7  mapfile = "settings.xml"
8  map = mapnik.Map(16000, 16000, epsg2056)
9  mapnik.load_map(map, mapfile)
10 map.zoom_to_box(bbox)
11 mapnik.render_to_file(map, 'terrain.png', 'png')
```

Listing 5.22: Python-Script zur Kartenerstellung mit Mapnik

Die Geländekarte wurde mit dem Ziel erstellt, eine möglichst plastische und detaillierte Darstellung der Geländeoberfläche zu erreichen, ohne vom eigentlichen Karteninhalt abzulenken. Dazu wurde, abgesehen von Höhenlinienzahlen, auf eine Kartenbeschriftung verzichtet und die Farben dezent gehalten. Zur Orientierung wurden Elemente der Infrastruktur wie Straßen, Gebäude und Wege aufgenommen. Zusätzlich wurde im Hinblick auf Ereignisse, welche die Vegetation betreffen, die Waldbedeckung visualisiert.

Das Ergebnis der Geländedarstellung ist für den Bildschirm optimiert, daher wird für eine genauere Betrachtung der Karte auf die beiliegende CD verwiesen. Die Karte kann in den verschiedenen Zoomstufen in einem Browser betrachtet werden und findet sich auf der beiliegenden CD im Ordner `/neis/web-app/map/`.

Für die Darstellung der Ereignisse auf der Übersichtskarte wurde für jeden Ereignistyp ein Piktogramm entworfen, welches farblich einer der fünf Ereigniskategorien zugeordnet wird (siehe Abbildung 5.6). Die Piktogramme sollen von einem Betrachter, der die verschiedenen Ereignistypen kennt, möglichst ohne Legende zugeordnet werden können. Aus diesem Grund wurde versucht charakteristische Elemente eines Ereignisses herauszugreifen und schematisch darzustellen.

Abbildung 5.6: Piktogramme der Ereignistypen und Aufnahmeorte

5.5 Werkzeuge

Die Realisierung der Anwendung wurde größtenteils mit Hilfe der integrierten Entwicklungsumgebung *NetBeans 6.7*[11], die mit einem Plugin auch die Entwicklung von Grails-Anwendungen unterstützt, bewerkstelligt. Der Quellcode wird mit *Subversion*[12], einer Software zur Versionsverwaltung von Dateien und Verzeichnissen, verwaltet, und für die Ausführung der erstellten Grails-Anwendung kam *Apache Tomcat 6*[13] zum Einsatz. Die Administration der PostGIS-Datenbank und die Fehlersuche wurde durch die Verwendung von *pgAdmin*[14] wesentlich erleichtert.

Die Herstellung der Geländedarstellung wurde mit Werkzeugen der Programmbibliothek *GDAL*[15] und der darin enthaltenen Programmbibliothek *OGR2OGR* sowie mit *Mapnik*[16] durchgeführt. Außerdem kamen für diese Aufgabe die *demtools*[17] zur Anwendung.

11 http://www.netbeans.org/ [Abruf: 15.9.2009]
12 http://subversion.tigris.org/ [Abruf: 15.9.2009]
13 http://tomcat.apache.org/ [Abruf: 15.9.2009]
14 http://www.pgadmin.org/ [Abruf: 15.9.2009]
15 http://www.gdal.org/ [Abruf: 15.9.2009]
16 http://mapnik.org/ [Abruf: 15.9.2009]
17 http://perrygeo.googlecode.com/svn/trunk/demtools/ [Abruf: 15.9.2009]

6 Resultate und Beurteilung

Die im vorherigen Kapitel beschriebene Realisierung ist nur die eine Seite der Medaille einer produktiv einsetzbaren Anwendung. Ebenso wichtig ist eine benutzerfreundliche Steuerung der implementierten Funktionen, die andernfalls wohl kaum zum Einsatz kommen werden.
Im folgenden Kapitel sollen die Resultate der erstellten Anwendung beurteilt werden. Zunächst dienen in **Kapitel 6.1** die im Pflichtenheft festgelegten Kriterien als Grundlage für die Beurteilung der Zielerreichung. Anschließend wird die Anwendung im Rahmen einer Evaluation einem Benutzertest unterzogen, der in **Kapitel 6.2** beschrieben und in **Kapitel 6.3** beurteilt wird.

6.1 Beurteilung der Zielerreichung

Grundlage der Beurteilung bildet im Verlauf dieses Kapitels das Pflichtenheft, welches im Zuge der Anforderungsanalyse entstanden ist (siehe Kapitel 3.2). Ein wichtiger Gesichtspunkt zur Beurteilung der Ergebnisse ist, inwieweit die als Zielbestimmung festgehaltenen Kriterien umgesetzt wurden.

Kriterien, die für eine einsatzfähige Anwendung erfüllt sein müssen, wurden als Muss-Kriterien formuliert. Dazu gehören die Implementierung des Datenbankschemas aus der Masterarbeit von Jonas Büchel (2009), eine Abfrage- und Exportmöglichkeit sowie die Verwaltung und dreistufige Erfassung von Naturereignisdaten.

Im Laufe der Anforderungsanalyse sind einige dokumentierte Änderungen an dem Datenbankschema von Jonas Büchel entstanden. Positiver Nebeneffekt der Anpassungen ist eine deutlich reduzierte Tabellenanzahl, in erster Linie erreicht durch die allgemeinen Entitätstypen Punkt, Linie und Fläche. Abgesehen von diesen Änderungen wurde das Datenbankschema für die erste und zweite Erfassungsstufe sowie den Ereignistyp Lawine erfolgreich umgesetzt. Weitere Entitätstypen der dritten Erfassungsstufe wurden bisher nicht berücksichtigt, weil diese alle einem Schema folgen, welches am Beispiel Lawinendaten aufgezeigt wurde. Während der Entitätstyp Betroffenes Objekt zumindestens auf Datenebene als Domain Klasse realisiert wurde, fehlen die Entitätstypen Park und Parkgrenze noch. Insgesamt ist die Implementierung des Datenbankschemas zwar positiv verlaufen, wurde angesichts des eingeschränkten Zeitrahmens allerdings nicht vollständig umgesetzt.

Die Abfragemöglichkeit wurde mit Hilfe von Filtern erfolgreich implementiert und ist einsatzbereit. Mit den Kriterien Ereignistyp, Ereigniszeitpunkt, und Entdeckungszeitpunkt sowie der Kartenrahmen als räumliches Kriterium wurden alle geforderten Abfragemöglichkeiten realisiert. Eine Exportfunktion steht ebenso zur Verfügung, die sich allerdings auf die Daten der ersten und zweiten Erfassungsstufe beschränkt.

Die dreistufige Erfassung von Naturereignisdaten wurde komplett realisiert und funktioniert problemlos. Zusätzlich wurde als vierte Stufe die Administration hinzugefügt. Die Verwaltung, also Erstellen, Anzeigen, Bearbeiten und Löschen von Datensätzen, speziell von Naturereignisdatensätzen, ist zu einem großen Teil erfolgreich realisiert. Die dritte Erfassungsstufe ist bisher nur am Beispiel von Lawinendaten implementiert und die Funktion zum Bearbeiten von Fotos sowie von Geometriedaten der dritten Erfassungsstufe ist noch unvollständig. Alles in allem ist die dreistufige Erfassung und Verwaltung von Naturereignisdaten, sobald die restlichen Ereignistypen der dritten Erfassungsstufe verfügbar sind, einsatzbereit.

Optionale Funktionen mit geringerer Priorität wurden als Kann-Kriterien im Pflichtenheft festgehalten. Dazu gehören eine Schnittstelle zu einer mobilen Anwendung, weitere Abfragemöglichkeiten, eine Benutzer- und Rechteverwaltung sowie Backup- und Importfunktionalitäten.

Eine Schnittstelle zu einer mobilen Anwendung wurde im Rahmen dieser Arbeit nicht realisiert, sondern nur auf konzeptioneller Ebene berücksichtigt. Die Abfragemöglichkeiten wurden um die Kriterien Ereignisgröße und Benutzer sowie um eine Volltextsuche ergänzt. Damit wurden alle von Ronald Schmidt (WPZ) genannten Kriterien berücksichtigt. Die Benutzer- und Rechteverwaltung wurde in einem eigenen Administrationsbereich implementiert und ist einsatzbereit. Abgesehen von

der Möglichkeit, Geometriedaten als GML-Datei zu importieren, stehen keine Backup- oder Importfunktionen zur Verfügung. Mit Software zur Datenbankadministration, wie pgAdmin III[1], lässt sich diese Aufgabe ohnehin zuverlässiger ausführen.

Wenn man von der Beschränkung auf den Ereignistyp Lawine absieht, wurden die Muss-Kriterien weitestgehend erfüllt, während die Kann-Kriterien etwa zur Hälfte realisiert wurden. Im Nachhinein hat sich die Unterteilung in Muss- und Kann-Kriterien, wie sie von Balzert (2005) vorgeschlagen wird, als zu ungenau erwiesen. Eine differenziertere Unterteilung wie beispielsweise die MoSCoW-Methode[2], welche die Kriterien den Prioritäten must, should, could und won't zuordnet, scheint für diese Aufgabe besser geeignet. Außerdem wurden die Kriterien selber teilweise zu Allgemein beschrieben und lassen sich daher im Nachhinein nur schwer als erfüllt oder nicht erfüllt einstufen. Gleichwohl kann mit dem Ergebnis einer am Beispiel des Ereignistyps Lawine einsatzfähigen Anwendung insgesamt eine positive und gleichzeitig lehrreiche Bilanz gezogen werden.

6.2 Evaluation

Obwohl es zahlreiche Normen mit Anforderungen an Software-Produkte wie beispielsweise die DIN 9241 gibt, stehen kaum einheitliche Methoden zur Überprüfung derselben zur Verfügung. Gleichwohl haben sich im Laufe der Zeit einige Testarten entwickelt, die von Dahm (2006) in folgende vier Kategorien eingeteilt werden:

Expertentests: Die Software wird von einem oder mehreren Experten aufgrund anwendbarer Normen und Erfahrungswerten beurteilt. Der Vorteil dieser Tests sind aussagekräftige Ergebnisse zur Gebrauchstauglichkeit, während als Nachteil die schlechte Bewertbarkeit und Reproduzierbarkeit von Erfahrung angeführt wird.
Benutzertests: Die Software wird von Personen, die vergleichbar mit späteren Benutzern sind, anhand von möglichst vollständigen Arbeitsschritten in einer realitätsnahen Arbeitsumgebung getestet. Während die entsprechenden Testpersonen gut einschätzen können, ob die Anwendung den echten Arbeitsaufgaben gerecht wird, unterliegen die Personen verständlicherweise keiner Norm; die Ergebnisse sind daher nicht reproduzierbar.
Quantitative Tests: Ziel sind messbare Ergebnisse wie Dauer der Bearbeitung, Anzahl der Klicks oder Anzahl der Fehler in einer Interaktion. Mit Hilfe von Fragebögen lässt sich außerdem die Zufriedenheit und Befindlichkeit von Testpersonen quantitativ erfassen.
Qualitative Tests: Dieses Verfahren eignet sich zur Beschreibung des allgemeinen Eindrucks von einer Software und ergeben in der Regel sprachlich ausgedrückte Ergebnisse.

Die vorliegende Evaluation wurde als Benutzertest mit zwei Testpersonen am 14. September 2009 im Sihlwald durchgeführt. Obwohl die Testgruppe sehr klein ist und daher kein repräsentatives Ergebnis erwartet werden kann, wird die Gebrauchstauglichkeit der Anwendung nichtsdestotrotz auf die Probe gestellt. Als Testpersonen haben sich dankenswerterweise zwei Mitarbeiter vom Wildnispark Zürich zur Verfügung gestellt, die beide als potenzielle Benutzer der Anwendung in Frage kommen und daher einen realitätsnahen Test erlauben. Der Benutzertest wurde im Gemeinschaftsbüro

1 http://www.pgadmin.org/ [Abruf: 15.9.2009]
2 http://www.coleyconsulting.co.uk/moscow.htm [Abruf: 15.9.2009]

der beiden Mitarbeiter auf meinem Laptop durchgeführt, weil auf den Computern im Büro bisher nur der Internet Explorer zur Verfügung steht, der von dem Naturereignisinformationssystem noch nicht unterstützt wird.

Zu Beginn wurde das System anhand einer Naturereigniserfassung kurz vorgestellt. Beiden Testpersonen wurde anschließend jeweils ein fiktiver Naturereignisdatensatz als ausgefülltes Papierformular vorgelegt, der alle Erfassungsstufen betraf und durch einen Fotodatensatz ergänzt wurde. Für eines der beiden Ereignisse lag ein georeferenziertes Foto und eine GML-Datei mit den Geometriedaten der Lawine bereit, während die Geodaten für das andere Ereignis manuell in der Karte erfasst werden sollten. Insgesamt wurde auf diese Weise mit jeder Testperson der gesamte Ablauf einer Naturereigniserfassung durchgespielt. Dadurch, dass sich die Personen während der Erfassung abgewechselt haben, musste auch die Abfragemöglichkeit zur Anwendung kommen, die durch 50 im voraus automatisch hinzugefügte Ereignisse nicht trivial war.

Abgeschlossen wurde die Evaluation mit dem ISONORM-Fragebogen, der 1993 von Jochen Prümper und Michael Anft entwickelt wurde und sich an der DIN 9241 orientiert. Zu den sieben Gestaltungsgrundsätzen Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Fehlertoleranz, Lernunterstützung, Erwartungskonformität, Steuerbarkeit und Individualisierbarkeit werden jeweils fünf Fragen gestellt, die auf einer siebenstufigen Skala von sehr schlecht (---) bis sehr gut (+++) bewertet werden sollen.

6.3 Resultate der Evaluation

Obwohl die Software das erste Mal und nur sehr knapp vorgestellt wurde, haben sich beide Testpersonen nach einer kurzen Eingewöhnungsphase gut zurecht gefunden. Die Orientierung auf der Karte, die nur für den Schweizerischen Nationalpark erstellt wurde und daher eine unbekannte Umgebung darstellte, war dementsprechend etwas schwieriger, aber verlief insgesamt reibungslos. Dank der weiten Verbreitung von Google Maps und ähnlichen Webangeboten scheint die Bedienung von Kartenanwendungen im Internet weitgehend bekannt zu sein.

Die Erfassung der Grunddaten wurde in beiden Fällen ohne Probleme gelöst. Kleine Verbesserungen sind mit der Beschriftung der Koordinatensystemauswahl und einer einheitlichen Reihenfolge der Auswahlfelder auf dem Papierformular und dem Webformular möglich. Mit dem Kommentar „da muss man ja gar nicht mehr denken“ wurde allerdings auch deutlich, dass die Bedienung insgesamt sehr intuitiv ist.

Die Formulare bzw. die Eingabemasken der zweiten und dritten Erfassungsstufe wurden im ersten Moment als unübersichtlich und überladen wahrgenommen. Während der Erfassung wurde jedoch die Struktur deutlich und es traten keine entscheidenden Probleme auf. Die kurzen Erklärungen, welche am Beispiel der Ereignisdaten umgesetzt wurden, scheinen vor allem für neue Benutzer hilfreich zu sein und sollten auch für die übrigen Formulare implementiert werden. Für erfahrene Benutzer sollte es gleichzeitig eine Möglichkeit geben, die Erklärungen auszublenden.

Aufgefallen ist die recht zeitaufwendige Erfassung von Zeitpunkten, die zwar intuitiv und fehlerfrei bewältigt wurde, aber trotzdem langwierig ist. Eine Lösung wäre der Einsatz so genannter Widgets, die als Komponente in die graphische Oberfläche eingebunden werden und möglicherweise die Zahl der notwendigen Klicks reduzieren können. Außerdem könnte der Aufnahmezeitpunkt der Grunddaten für die weiteren Erfassungsstufen übernommen werden und müsste nur bei einer Abweichung geändert werden.

Des Weiteren wurde eine ausführlichere Hilfe gewünscht, die bisher nur für die Ereignisdaten realisiert wurde. Gerade neue Benutzer werden ohne die Hilfe eines erfahrenen Benutzers wohl Probleme bei der Benutzung haben. Zu überlegen ist, ob eine Schulung oder eine ausführliche Hilfe sinnvoller ist.

Außerdem wurde die Möglichkeit zur Unterbrechung der Erfassung als Verbesserungsvorschlag genannt. Die Pflichtfelder verlangen bisher eine fast vollständige Erfassung für ein Formular.

Die Fragen des ISONORM-Fragebogens wurden insgesamt positiv bewertet. Die Beurteilung der Fragen wurde nach den sieben Gestaltungsgrundsätzen zusammengefasst und ergibt für zwei befragte Personen jeweils zehn Bewertungen zwischen sehr schlecht (---) und sehr gut (+++), die in Tabelle 6.1 aufgeführt sind. Die Fragen zur Fehlertoleranz und Individualisierbarkeit wurden aufgrund mangelnder Erfahrung der Testpersonen nicht beantwortet.

	---	--	-	-/+	+	++	+++
Aufgabenangemessenheit	-	-	-	-	3	3	4
Selbstbeschreibungsfähigkeit	-	-	-	-	-	3	7
Steuerbarkeit	-	-	-	-	-	2	8
Erwartungskonformität	-	-	-	-	-	2	8
Lernförderlichkeit	-	-	-	-	-	4	6

Tabelle 6.1: Ergebnisse des ISONORM-Fragebogens

Zusammenfassend kann die kleine Evaluation durchaus als erfolgreich bezeichnet werden. Obwohl einige Verbesserungsmöglichkeiten identifiziert wurden, traten keine schwerwiegenden Probleme auf. Auch die Bewertung des ISONORM-Fragebogens hinterlässt einen positiven Eindruck, der sich ebenso im Gespräch mit den Testpersonen gezeigt hat. Eine ausführliche Beschreibung der Erweiterungs- und Verbesserungsansätzen folgt in Kapitel 7.2.

7 Schlussfolgerung und Ausblick

In **Kapitel 7.1** wird zunächst ein Fazit gezogen und die bearbeiteten Themen aus einer subjektiven Sicht zusammengefasst. Abschließend werden in **Kapitel 7.2** Erweiterungs- und Verbesserungsmöglichkeiten aufgezeigt.

7.1 Schlussfolgerung

Abschließend soll in Bezug auf die im ersten Kapitel genannten Fragestellungen ein Fazit gezogen werden. Im Laufe dieser Arbeit wurden zahlreiche Aspekte der Entwicklung einer webbasierten Anwendung zur Verarbeitung von räumlichen Daten behandelt. Angefangen hat die Arbeit mit einer objektorientierten Analyse und einem entsprechenden Entwurf, die beide unabhängig von der realisierten Anwendung sind und damit auch für zukünftige Entwicklungen interessant sein können.

Aufbauend auf der Analyse wurde ein objektorientiertes Datenmodell erstellt, welches auf Hibernate basiert und daher auch in weiteren Projekten eingesetzt werden kann. Zudem wurde auf diese Weise nicht nur eine hohe Flexibilität hinsichtlich der Anwendungsebene erreicht, sondern auch auf Datenbankebene sollten nach wenigen Anpassungen andere Datenbanksysteme, wie beispielsweise Oracle, eingesetzt werden können.

Die Entscheidung Grails als Framework für die Webanwendung zu verwenden, hat sich als positiv herausgestellt. Auch wenn zunächst eine gewisse Einarbeitungszeit notwendig war, konnten recht schnell funktionsfähige Ergebnisse erzielt werden, die anschließend flexibel angepasst werden konnten. Zudem kann der große Funktionsumfang von Grails durch zahlreiche Plugins erweitert werden und mit der Möglichkeit beliebige Java-Programmbibliotheken einzubinden, können viele Probleme gelöst werden. Der Funktionsumfang und die Flexibilität von Grails wird durch den Einsatz von bewährten Programmbibliotheken ermöglicht, die hingegen eine Anwendung mit recht hohen Ressourcenanforderungen zur Folge haben.

Ein weiterer interessanter Gesichtspunkt der Arbeit war die Entwicklung einer Import- und Exportfunktion auf Basis von GML. Für diese Aufgabe wurden zwei XML-Schema entworfen, die eine Validierung der GML-Dateien erlauben. Wie sich im Laufe der Realisierung herausgestellt hat, eignet sich das GML Simple Feature Profile gut für den Austausch von räumlichen Daten, da es nicht nur von Geoinformationssystemen wie ArcGIS gelesen und geschrieben werden kann, sondern auch mit den Hilfsmitteln von Grails verarbeitet werden kann.

Eine willkommene Abwechslung zum Programmieren hat die Gestaltung der grafischen Benutzeroberfläche geboten. Verschiedene Designelemente und Farbkombinationen wurden ausprobiert, bis eine befriedigendes Ergebnis erzielt wurde. Darüber hinaus wurde eine Reihe von Icons für eine übersichtlichere Bedienung der Anwendung entworfen.

Aus kartographischer Sicht war die Herstellung einer Karte, die eine möglichst ausführliche Darstellung des Geländes erzielt und für den Einsatz innerhalb einer Webanwendung optimiert ist, eine Herausforderung. Verbessert wurde die kartographische Darstellung der räumlichen Ereignisdaten durch die Gestaltung von Piktogrammen für die einzelnen Ereignistypen.

Insgesamt hat sich die eingesetzte Open Source Software durchweg bewährt und kann ohne Einschränkungen weiterempfohlen werden. Von Nachteil ist ein teilweise hoher Einarbeitungsaufwand, der in manchen Fällen durch mangelnde Dokumentation erschwert wird.

Das Ergebnis der Arbeit wurde im Rahmen einer kleinen Evaluation positiv bewertet und bietet damit eine brauchbare Möglichkeit zur Dokumentation von Naturereignisssen in Schutzgebieten. Bevor die Anwendung produktiv eingesetzt werden kann, sind allerdings noch Anpassungen und Erweiterungen notwendig, die im folgenden Kapitel zusammengefasst werden.

7.2 Ausblick

Obwohl die Anwendung ein funktionsfähiges Stadium erreicht und sich in einem Benutzertest bewährt hat, ist sie für den produktiven Einsatz noch nicht geeignet. Zum einen wurde die Anwendung bisher noch nicht ausführlich getestet und wird daher kaum fehlerfrei sein. Entscheidender ist allerdings, dass bislang nur der Ereignistyp Lawine unterstützt wird und der mögliche Einsatzbereich damit sehr eingeschränkt ist. Darüber hinaus sind weitere Anpassungen und Erweiterungen möglich, die im Folgenden zusammengefasst werden.

Eine notwendige Erweiterung betrifft den GML Export, der im jetzigen Zustand nur Daten der ersten und zweiten Erfassungsstufe exportiert. Die Daten der dritten Erfassungsstufe und die Metadaten von Fotos ließen sich im gleichen XML-Schema in jeweils einer seperaten GML-Datei ausgeben, die im Idealfall zusammen mit den allgemeinen Ereignisdaten als ZIP-Datei zum Herunterladen angeboten wird.

Das Hinzufügen von Binärdaten zu einem Naturereignisdatensatz wurde bislang nur am Beispiel von Fotos implementiert. Die Möglichkeit auch Dokumente hinzufügen zu können, muss noch realisiert werden. Überlegenswert wäre zudem die Möglichkeit Ereignisse mit Weblinks (URLs) ergänzen zu können. Zu überdenken ist, ob die N:M-Beziehung zwischen Fotos bzw. Dokumenten und Ereignissen sinnvoll ist und beibehalten werden soll, oder ob eine 1:N-Beziehung ausreichend ist. Eine 1:N-Beziehung würde die Benutzeroberfläche wesentlich vereinfachen, Probleme beim GML Export der Daten vermeiden und in den allermeisten Fällen ausreichend sein.

Im Allgemeinen setzt die Anwendung in ihrer jetzigen Form ihren Schwerpunkt auf die Erfassung von Naturereignisdaten. Sobald eine umfangreiche Datengrundlage geschaffen wurde, werden weitere Nutzungsmöglichkeiten und Schnittstellen von Interesse sein. Ein Beispiel für einen möglicherweise nützlichen Netzwerkdienst ist WFS, der sich entweder auf Basis der PostGIS-Datenbank mit einer Software wie *GeoServer*[1] realisieren ließe oder mit Hilfe einer Java-Programmbibliothek direkt in die Anwendung integrieren lässt.

Eine Anwendung, die auf einem mobilen Endgerät eingesetzt wird, könnte auf zwei Wegen in das Naturereignisinformationssystem eingebunden werden, die allerdings beide noch nicht implementiert sind. Zum einen könnte der GML Import von der Beschränkung auf Geometriedaten befreit werden und, ähnlich wie der GML Export, den gesamten Umfang von Naturereignisdaten berücksichtigen. Ein anderer Ansatz wäre die Realisierung einer REST-Schnittstelle, die bisher ansatzweise für den Export und die Übersichtsseite implementiert ist. Sollte der zuletzt genannte Weg gewählt werden, besteht außerdem die Möglichkeit, dass auch die Webanwendung auf der REST-Schnittstelle aufbaut.

Hinsichtlich der Leistungsfähigkeit, die bislang weitgehend unbeachtet blieb, bestehen ebenfalls Verbesserungsmöglichkeiten, die wohl aber erst interessant werden, wenn die Anwendung von vielen Benutzern beansprucht wird. Einen Leistungsschub bringt vermutlich ein Zwischenspeicher für die Fotos, damit diese nicht für jede Anfrage aus der Datenbank gelesen und gerendert werden müssen. Außerdem könnte der GML Export vollständig auf Steuerungsebene implementiert werden und auf diese Weise wesentlich schneller arbeiten. Zudem könnte der JavaScript- und CSS-Code, der bislang auf zahlreiche Dateien verteilt ist, zur Verringerung der HTTP-Anfragen in zwei Dateien zusammengefasst und komprimiert werden.

1 http://geoserver.org/ [Abruf: 15.9.2009]

Literaturverzeichnis

Ahnert, Frank (2003):
Einführung in die Geomorphologie. Stuttgart : Verlag Eugen Ulmer, 2003

BAFU (2008):
StorMe. URL `http://www.bafu.admin.ch/naturgefahren/01922/01926/01927/` (20.9.2009)

Balzert, Heide (2004):
Lehrbuch der Objektmodellierung: Analyse und Entwurf mit der UML 2. Heidelberg : Spektrum Akademischer Verlag, 2004

Badman, Tim ; Bomhard, Bastian (2008):
World Heritage and Protected Areas 2008 Edition. World Commission on Protected Areas.
URL `http://cmsdata.iucn.org/downloads/world_heritage_and_protected_areas_2008.pdf` (20.9.2009)

Baumhauer, Roland (2006):
Geomorphologie. Darmstadt : Wissenschaftliche Buchgesellschaft, 2006

Becht, Michael ; Copien, Claudia ; Frank, Christian (2006):
Abschlussbericht zum Projekt HANG. URL `http://www.lfu.bayern.de/wasser/fachinformationen/gefahren_im_alpenraum/doc/hang_lang.pdf` (20.9.2009)

Beck, Kent ; Cockburn, Alistair ; Cunningham, Ward ; Fowler, Martin [u.a.] (2001):
Manifesto for Agile Software Development. URL `http://agilemanifesto.org/` (20.9.2009)

Boston GIS (2009):
Cross Compare SQL Server 2008 Spatial, PostgreSQL/PostGIS 1.3-1.4, MySQL 5-6.
URL `http://www.bostongis.com/PrinterFriendly.aspx?content_name=sqlserver2008_postgis_mysql_compare` (20.9.2009)

Brinkhoff, Thomas (2008):
Geodatenbanksysteme in Theorie und Praxis: Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial. Heidelberg : Herbert Wichmann Verlag, 2008

Büchel, Jonas (2009):
Naturereignisdokumentation: Konzept zur Dokumentation und Datenbanksystem für Naturereignisse in Schutzgebieten. Masterarbeit, Geographisches Institut Universität Zürich, Mai 2009

Butler, Howard ; Daly, Martin ; Gillies, Sean ; Schaub, Tim ; Schmidt, Christopher [u.a.] (2008):
The GeoJSON Format Specification. URL `http://geojson.org/geojson-spec.html` (20.9.2009)

BUWAL (1997):
Berücksichtigung der Massenbewegungsgefahren bei raumwirksamen Tätigkeiten. Bundesamt für Umwelt, Wald und Landschaft. URL `http://www.bafu.admin.ch/publikationen/publikation/00783` (20.9.2009)

BUWAL (1998):
Begriffsdefinitionen zu den Themen Geomorphologie, Naturgefahren, Forstwesen, Sicherheit und Risiko. Bundesamt für Umwelt, Wald und Landschaft, Eidg. Forstdirektion. Bern, 1998

Dahm, Markus (2006):
Grundlagen der Mensch-Computer-Interaktion. München ; Boston [u.a.] : Pearson Studium, 2006

DIS-ALP (2007):
Disaster Information System of Alpine Regions. URL `http://www.alpinespace.org/uploads/media/DIS-ALP_Final_Report.pdf` (20.9.2009)

Elmasri, Ramez ; Navathe, Shamkant (2009):
Grundlagen von Datenbanksystemen. München ; Boston [u.a.] : Pearson Studium, 2009

EXIF (2002):
Exchangeable image file format for digital still cameras: Exif Version 2.2. Standard of Japan Electronics and Information Technology Industries Association, 2002

Fielding, Roy (2000):
Architectural Styles and the Design of Network-based Software Architectures. Disertation, University of Califormia. URL `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm` (20.9.2009)

Fitzke, Jens (1999):
GIS im Internet. URL `http://www.giub.uni-bonn.de/gistutor/internet/inetgis/welcome.htm` (20.9.2009)

FORSTnet (2009):
Ereignisdokumentation Alpine Naturgefahren. URL: `http://www.forstnet.at/article/articleview/69039/1/25423/` (20.9.2009)

Gamma, Errich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John (1995):
Design Patterns: Elements of Reusable Object-Oriented Software. Boston ; San Francisco [u.a.] : Addison-Wesley, 1995

Gumm, Heinz-Peter ; Sommer, Manfred (2004):
Einführung in die Informatik. München ; Wien : Oldenbourg Wissenschaftsverlag, 2004

Haerderer, Reuter (1983):
Principles of Transaction-Oriented Database Recovery. Computing Surveys, Vol. 16, No. 4, 1983. URL http://www.minet.uni-jena.de/dbis/lehre/ss2004/dbs2/HaerderReuter83.pdf (20.9.2009)

Hardy, Will (2007):
Comparing Web Development Platforms Through the Eyes of Professional Developers. URL http://www.inf.fu-berlin.de/inst/ag-se/pubs/platforms-surveyTR-2007.pdf (20.9.2009)

Hübl, Johannes ; Kienholz, Hans ; Loipersberger, Anton (2006a):
DOMODIS – Dokumentation alpiner Naturereignisse. Internationale Forschungsgesellschaft INTERPRAEVENT, 2006. URL http://www.baunat.boku.ac.at/fileadmin/_/H87/H871/Downloads/060110-1A__DOMODIS_Inhalt_.pdf (20.9.2009)

Hübl, Johannes ; Habersack, Helmut ; Kienholz, Hans [u.a.] (2006b):
Disaster Information System of Alpine Regions (DIS-ALP): Methodik Teil 1, Appendix 2: Definitions. IAN Report 101, Institut für Alpine Naturgefahren, Universität für Bodenkultur-Wien, 2006

Ingres (2009):
IngresGeospatial. URL: http://community.ingres.com/wiki/IngresGeospatial (20.9.2009)

Jackson, Michael (1995):
Software Requirements & Specifications: a lexicon of practice, principles and prejudices. Boston ; San Francisco [u.a.] : Addison-Wesley, 1995

Krasner, Glenn E. ; Pope, Stephen T. (1988):
A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Object Oriented Program, Vol. 1, No. 3, 1988, SIGS Publications

Leser, Hartmut [Hrsg.] (2001):
Wörterbuch Allgemeine Geographie. München : westermann Deutscher Taschenbuch Verlag, 2001

Nationalparkgesetz (1980):
Bundesgesetz über den Schweizerischen Nationalpark im Kanton Graubünden (Nationalparkgesetz). Die Bundesversammlung der Schweizerischen Eidgenossenschaft. URL http://www.nationalpark.ch/download/dwn/D_Nationalparkgesetz.doc.pdf (20.9.2009)

Oestereich, Bernd (2009):
Analyse und Design mit UML 2.1. München ; Wien : Oldenbourg Wissenschaftsverlag, 2009

OGC (2004):
OpenGIS Geography Markup Language Implementation Specification. Open Geospatial Consortium. URL http://portal.opengeospatial.org/files/?artifact_id=4700 (20.9.2009)

OGC (2006a):
Geography Markup Language simple features profile. Open Geospatial Consortium.
URL http://portal.opengeospatial.org/files/?artifact_id=15201 (20.9.2009)

OGC (2006b):
OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option. Open Geospatial Consortium. URL http://portal.opengeospatial.org/files/?artifact_id=18242 (20.9.2009)

OSGeo (2008):
Tile Map Service Specification. Open Source Geospatial Foundation. URL http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification (20.9.2009)

Partsch, Helmut (1998):
Requirements-Engineering systematisch: Modellbildung für softwaregestützte Systeme. Berlin ; Heidelberg [u.a.] : Springer, 1998

PLANALP (2006):
Dokumentation von Naturereignissen: Feldanleitung. Plattform Naturgefahren der Alpenkonvention.
URL http://www.alpinespace.org/uploads/media/DIS-ALP_Dokumentation_von_Naturereignissen_Feldanleitung.pdf (20. September 2009)

PostgreSQL (2009):
History. PostgreSQL. URL: http://www.postgresql.org/about/history (20.9.2009)

Shekar, Shashi ; Xiong, Hui [Hrsg.] (2008):
Encyclopedia of GIS. Berlin ; Heidelberg ; New York : Springer. 2008

SNP (2009a):
Ziele. Schweizerischer Nationalpark.
URL http://www.nationalpark.ch/deutsch/A_1_2.php (20.9.2009)

SNP (2009b):
Fakten Nationalpark. Schweizerischer Nationalpark.
URL http://www.nationalpark.ch/deutsch/A_1_3.php (20.9.2009)

swisstopo (2008):
Formeln und Konstanten für die Berechnung der Schweizerischen schiefachsigen Zylinderprojektion und der Transformation zwischen Koordinatensystemen. Bundesamt für Landestopografie swisstopo, 2008

TIOBE (2009):
TIOBE Programming Community Index for September 2009. TIOBE Software.
URL http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (20.9.2009)

Wikipedia (2009):
Comparison of web application frameworks. Wikipedia, the free encyclopedia.
URL http://en.wikipedia.org/w/index.php?title=Comparison_of_web_application_frameworks&oldid=300785258 (20.9.2009)

WPZ (2009):
Geschichte. Wildnispark Zürich. URL http://www.wildnispark.ch/zuerichs-wildnis/geschichte/detail/1/ (20.9.2009)

WSL (2006):
Definitionen „Murgang". Eidg. Forschungsanstalt für Wald, Schnee und Landschaft.
URL http://www.wsl.ch/dienstleistungen/murgang/def_mg.pdf (20.9.2009)

W3C (1998):
Vector Markup Language (VML). World Wide Web Consortium.
URL http://www.w3.org/TR/1998/NOTE-VML-19980513 (20.9.2009)

W3C (2003):
Scalable Vector Graphics (SVG) 1.1 Specification. World Wide Web Consortium.
URL http://www.w3.org/TR/SVG11/ (20.9.2009)

Anhang

Grunddaten

Erste Erfassungsstufe

Ereignisnummer _____ **Vorname** _____

Datenbank ID _____ **Nachname** _____

Beschreibung _____

_____

Ortsbezeichnung _____

Entdeckungszeitpunkt _____ **Aufnahmezeitpunkt** _____

z.B. 22.9.2009 8:12 z.B. 22.9.2009 8:12

Erfassungsmethode

- Felderfassung ☐
- Büroerfassung ☐

Qualität

- hoch ☐
- mittel ☐
- niedrig ☐

Aufnahmeort

Hilfsmittel

- Orthofoto ☐
- Meldung ☐
- Karte ☐
- GPS ☐
- DGPS ☐
- Laserdistanzmessung ☐
- Fernerkundungsmethoden ☐

Foto

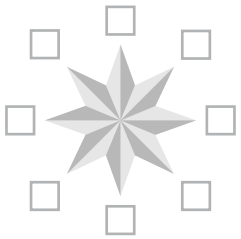
Ereignisnummer _____ **Vorname** _____

Datenbank ID _____ **Nachname** _____

Aufnahmezeitpunkt _____

z.B. 22.9.2009 8:12

Aufnahmerichtung _____ ° oder



Erfassungsmethode

Felderfassung ☐

Büroerfassung ☐

Qualität

hoch ☐

mittel ☐

niedrig ☐

Aufnahmeort

Hilfsmittel

Orthofoto ☐

Meldung ☐

Karte ☐

GPS ☐

DGPS ☐

Laserdistanzmessung ☐

Fernerkundungsmethoden ☐

Ereignisdaten

Zweite Erfassungsstufe

Ereignisnummer _____ **Vorname** _____

Datenbank ID _____ **Nachname** _____

Aufnahmezeitpunkt _____

Ereignisgröße

- 1m oder 10m² ☐
- 10m oder 100m² ☐
- 100m oder 10.000m² ☐
- 1.000m oder 1 Mio. m² ☐
- 10.000m oder 100 Mio. m² ☐

Mit Hilfe der Ereignisgröße soll die vom Ereignis betroffene Fläche angegeben werden. Ohne großen Aufwand soll das Ereignis einer Größenordnung zugewiesen werden.

MAXO-Code

Der MAXO-Code gibt die qualitative Genauigkeit an. Folgende Stufen stehen zur Verfügung

M Messung
A Annahme
X Wert unklar
O Wert nicht bestimmbar

Hinweis zur Erfassungsmethode der Raumdaten

Je nach Größe und Lage eine Ereignisses muss eine andere Methode zur Erfassung der räumlichen Ausdehnung gewählt werden. In diesem Textfeld sollen dem Erfasser der 3. Stufe Angaben zum Ereignis und der Erfassung der Raumdaten gemacht werden.

_____

_____

Zeitraum des Ereignisses

Falls das Ereignis direkt bobachtet wurde, kann ein genauer Zeitpunkt angegeben werden. Ansonsten muss der Zeitraum durch die Angabe einer Zeitspanne festgehalten werden.

MAXO-Code _____ **von** _____ **bis** _____

z.B. 22.9.2009 8:12 z.B. 22.9.2009 20:12

Wiederkehrendes Ereignis

Ein wiederkehrendes Ereignis ist als "gleichartiges Ereignis an der selben Stelle" zu verstehen. Die Wiederkehrdauer soll aufgrund des Erfahrungswissens des Erfassers angegeben werden.

Wiederkehrdauer

- keine ☐
- täglich ☐
- wöchentlich ☐
- monatlich ☐
- jährlich ☐
- alle 30 Jahre ☐
- alle 100 Jahre ☐

MAXO-Code _____

Präzisierung Ereignistyp

Mit Hilfe der gewonnenen Informationen soll das Ereignis einem Ereignistyp zugeordnet werden.

Massebewegung

- Rutschung ☐
- Hangmure ☐
- Kriechphänomen ☐
- Absenkung / Einsturz ☐
- Sturzbewegung ☐
- Lawine ☐

Wasserereignisse

- Überschwemmung ☐
- Murgang ☐
- Seitenerosion ☐
- Quelle ☐

MAXO-Code _____

Vegetationsschädigung

- Brand ☐
- Dürre ☐
- Frost ☐
- Schneebruch ☐
- Windwurf ☐
- Blitzschlag ☐

Andere Ereignisse

- Fallwild ☐
- Anthropogene Veränderung ☐
- Lebensraumbeeinträchtigung ☐

Forstschädlinge

- Insekten ☐
- Nagetiere ☐
- Wild ☐

Lawinendaten

Dritte Erfassungsstufe

Ereignisnummer _____ **Vorname** _____

Datenbank ID _____ **Nachname** _____

Aufnahmezeitpunkt _____

Form des Anrisses punktförmig ☐ linienförmig ☐

Spuren der Lawine _____

_____

Ursache **MAXO-Code** _____

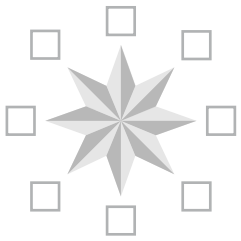
Akkumulation (Wind) ☐ Schneemetamorphose ☐

Akkumulation (Schneefall) ☐ Sprengung ☐

Niederschlag (Regen) ☐ Zusatzbelastung (Personen) ☐

Exposition _____ ° oder

MAXO-Code _____



Tiefe der Gleitfläche _____ **m** **MAXO-Code** _____

Ablagerung

Volumen _____ **m³** **MAXO-Code** _____

maximale Breite _____ **m** **MAXO-Code** _____

maximale Mächtigkeit _____ **m** **MAXO-Code** _____

Fremdmaterialien _____

_____

Lawinentyp **MAXO-Code** _____

Schneebrettlawine ☐ Staublawine ☐

Lockerschneelawine ☐ Oberlawine ☐

Fliesslawine ☐ Grundlawine ☐

Geodaten

Erfassungsmethode

Felderfassung ☐

Büroerfassung ☐

Qualität

hoch ☐

mittel ☐

niedrig ☐

Hilfsmittel

Orthofoto ☐

Meldung ☐

Karte ☐

GPS ☐

DGPS ☐

Laserdistanzmessung ☐

Fernerkundungsmethoden ☐