Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik
Lehrstuhl für Systemarchitektur
Unter den Linden 6
D-10099 Berlin

Gutachter:
Prof. Dr. rer. nat. Jens-Peter Redlich
Prof. Dr. Miroslaw Malek

Diplomarbeit

Konzeption und Implementierung einer Community-Plattform für Surfer

Roman Scherer

scherer@informatik.hu-berlin.de

7. Dezember 2009

Zusammenfassung

Surfen oder Wellenreiten ist eine Wassersportart, die besonders stark vom Wetter, den Gezeiten und den Eigenschaften der zu surfenden Wellen abhängig ist. Im Rahmen dieser Diplomarbeit wurde eine *Community-Plattform* für das Internet entwickelt, auf der Surfer für sie wichtige Informationen finden und austauschen können. Der Fokus dieser Arbeit liegt auf der Integration von Wetter- und Wellenvorhersagen, die einen integralen Bestandteil der Plattform bilden. Außerdem werden einige Technologien und Methoden vorgestellt, die sich während der Konzeption und Implementierung besonders bewährt haben und die bei der Entwicklung von Web Applikationen berücksichtigt werden sollten. Abschließend werden einige Vorschläge gemacht, wie die vorhandenen Wetter- und Wellenvorhersagen verbessert werden könnten.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

1.1 Surfen - Eine Wassersportart

Mit Surfen oder Wellenreiten bezeichnet man eine Wassersportart, bei der versucht wird, eine brechende Welle mit einem Surfbrett im Stehen entlang zu fahren. Anders als beim Windsurfen wird hier nicht die Kraft des Windes, sondern die Kraft der brechenden Welle benutzt, um die zum Aufstehen und Fahren benötigte Geschwindigkeit zu erreichen. Zum Surfen geeignete Wellen fangen idealerweise an einem Punkt an zu brechen und fallen dann kontinuierlich in eine oder beide Richtungen in sich zusammen. Der Surfer versucht die Welle kurz vor dem brechenden Punkt zunächst anzupaddeln,

dann so schnell wie möglich aufzustehen, um anschließend auf der Welle so lange wie möglich in die brechende Richtung zu fahren.

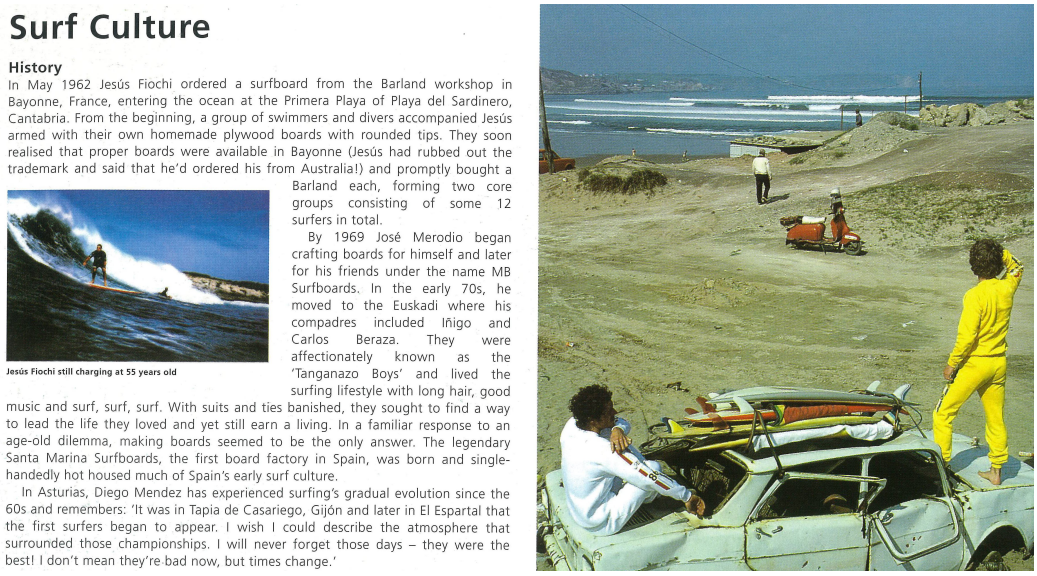
Surfbare Wellen sind allerdings nicht immer dort anzutreffen, wo es ein Meer oder einen Strand gibt. Vielmehr sind für Surfer interessante Wellen an den Orten zu finden, im folgenden *Spots* genannt, deren geografische Lage das Eintreffen von Wellen begünstigt, die weit entfernt entstanden und viele Kilometer weit gereist sind. Diese Art von Wellen wird *Swell* oder auch Dünung genannt und ist eine der Grundvoraussetzung für gute Surfbedingungen. Die Beschaffenheit des Untergrunds, über dem Wellen anfangen zu brechen, ist ein weiterer wichtiger Faktor, der die Qualität der zu surfenden Wellen beeinflusst. An den bei Surfern sehr beliebten *Pointbreaks* brechen die Wellen immer an der gleichen Stelle. An diesen Spots treffen die Wellen meist auf ein Riff oder einen aus Steinen bzw. Felsen bestehenden Untergrund, der sie abbremst und zum Brechen bringt. Dies sind die beständigsten, am besten einschätzbaren, aber auch gefährlichsten Spots. Besteht der Untergrund aus Sand, sind meist sich durch Gezeiten, Strömungen und Stürme ständig verändernde Sandbänke für das Brechen der Wellen verantwortlich. Diese Spots sind weniger beständig und verändern sich durch die Meeresströmung im Laufe der Zeit sehr viel schneller als die weitaus beständigeren *Pointbreaks*.

Weitere wichtige Faktoren, die sich auf die Eigenschaften von surfbaren Wellen auswirken, sind die Gezeiten, die Richtung aus der die Wellen kommen sowie die Wind- und Wetterverhältnisse in den jeweiligen Jahreszeiten. Insbesondere die Windstärke und die Windrichtung sind hier von großem Interesse. Potentiell surfbare Wellen können durch starken Wind aus der falschen Richtung sehr schnell zunichte gemacht werden. Die Ausrichtung eines Spots spielt dabei ebenfalls eine Rolle, da je nach Windrichtung einige der Spots windgeschützter sind als andere.

Da surfbare Wellen von vielen Faktoren beeinflusst werden, beschäftigen sich die meisten Surfer vor und während ihrer Reisen insbesondere mit den aktuellen Wetter- und Wellenvorhersagen und versuchen herauszufinden, welche Spots bei welchen Verhältnissen am besten brechen. Der typische Surfer liegt deshalb nicht nur am Strand herum und wartet dort auf gute Wellen, sondern ist oft auf abgelegenen Straßen und Trampelpfaden entlang der Küste unterwegs, in der Hoffnung, einen abgelegenen Spot mit den perfekten Wellen zu finden.

1.2 The Stormrider Guides

Die sogenannten *Stormrider Guides* des *Low Pressure* [1] Verlags sind seit langem die populärsten Reiseführer in der Surfszene. Sie erfreuen sich dank der vielen hilfreichen Informationen und Tipps rund ums Surfen einer sehr großen Beliebtheit. Insbesondere die detaillierten Informationen über die Eigenschaften der Wellen an den Spots sind dabei von großem Nutzen. Die nach Kontinenten, Ländern und Regionen gegliederten Bücher enthalten Reiseinformationen über Land und Leute, die Kultur des Landes, das dort herrschende Klima sowie Kartenausschnitte mit Beschreibungen zu den Surfbedingungen an den Spots. Die Bücher sind mit qualitativ hochwertigen und teilweise spektakulären Bildern, sowohl aus heutigen als auch aus vergangenen Zeiten illustriert.



Surf Culture

History

In May 1962 Jesús Fiochi ordered a surfboard from the Barland workshop in Bayonne, France, entering the ocean at the Primera Playa of Playa del Sardinero, Cantabria. From the beginning, a group of swimmers and divers accompanied Jesús armed with their own homemade plywood boards with rounded tips. They soon realised that proper boards were available in Bayonne (Jesús had rubbed out the trademark and said that he'd ordered his from Australia!) and promptly bought a Barland each, forming two core groups consisting of some 12 surfers in total.

Jesús Fiochi still charging at 55 years old

By 1969 José Merodio began crafting boards for himself and later for his friends under the name MB Surfboards. In the early 70s, he moved to the Euskadi where his compadres included Iñigo and Carlos Beraza. They were affectionately known as the 'Tanganazo Boys' and lived the surfing lifestyle with long hair, good music and surf, surf, surf. With suits and ties banished, they sought to find a way to lead the life they loved and yet still earn a living. In a familiar response to an age-old dilemma, making boards seemed to be the only answer. The legendary Santa Marina Surfboards, the first board factory in Spain, was born and single-handedly hot housed much of Spain's early surf culture.

In Asturias, Diego Mendez has experienced surfing's gradual evolution since the 60s and remembers: 'It was in Tapia de Casariego, Gijón and later in El Espartal that the first surfers began to appear. I wish I could describe the atmosphere that surrounded those championships. I will never forget those days – they were the best! I don't mean they're bad now, but times change.'

Abbildung 1.1: Ausschnitt zur Geschichte der Surf Kultur in Spanien

In den Reiseführern ist jedem der darin beschriebenen Länder ein eigenes Kapitel gewidmet, in denen zunächst auf die Geschichte und die Entwicklung des Surfens in dem jeweiligen Land eingegangen wird. In Abbildung 1.1 ist z.B. ein Ausschnitt aus dem *Stormrider Guide Europe* zu sehen, in dem die Anfänge des Surfens in Spanien beschrieben werden. Anschließend werden die am Meer liegenden Regionen eines Landes vorgestellt und allgemeine Aussagen über die dort herrschenden Surfbedingungen getroffen. Dabei werden unter anderem die Wasser- und Lufttemperaturen zu den verschiedenen

[1] `http://www.lowpressure.co.uk`

Jahreszeiten angegeben, vor Umwelt- und Wasserverschmutzung in stark besiedelten Gebieten gewarnt und die besten Jahreszeiten zur Planung einer Reise vorgeschlagen.

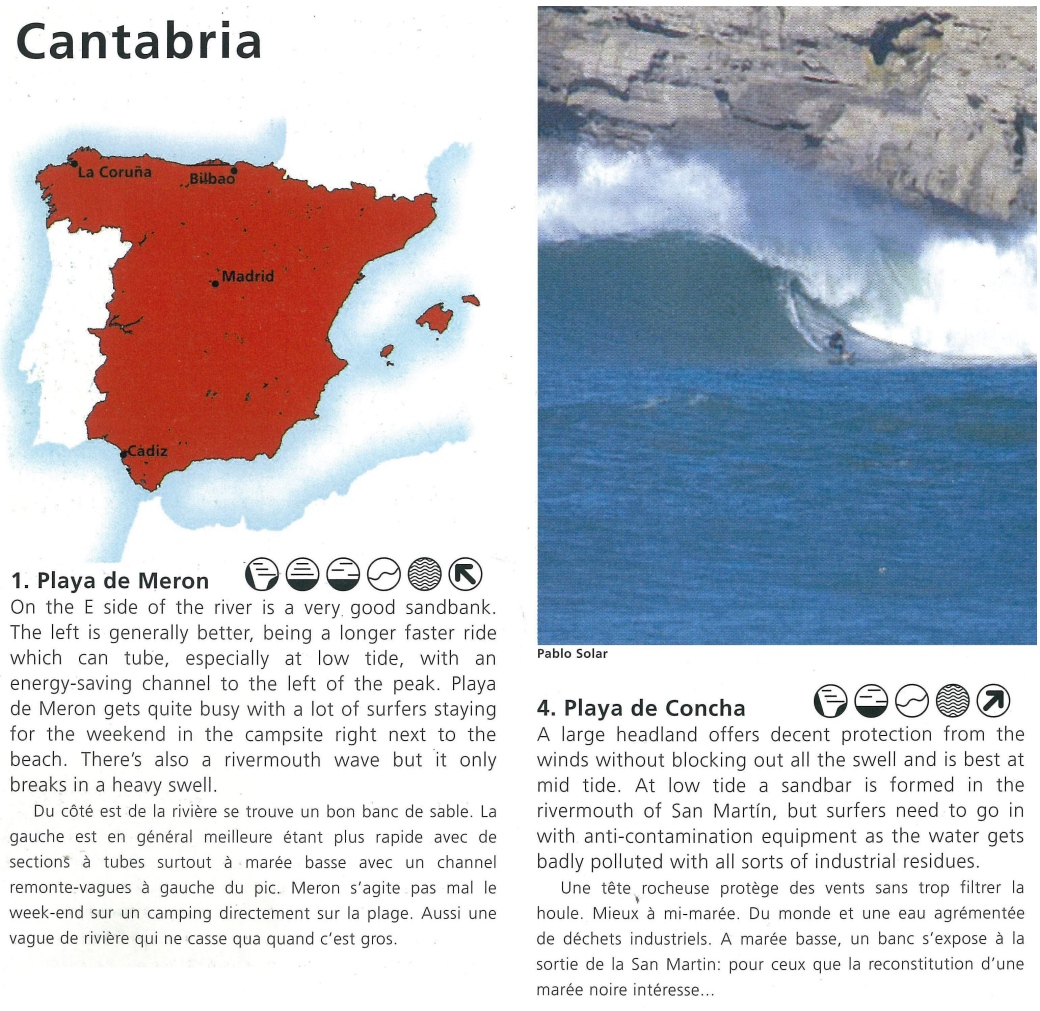


Abbildung 1.2: Spotbeschreibungen der Region Cantabria, Spanien

Ein weiterer Teil eines jeden Kapitels beschäftigt sich mit Informationen rund um das Reisen. Hier sind sowohl Informationen zur Einreise in das jeweilige Land, zu den Flughäfen, dem Bus- und Zugnetz zu finden als auch Telefonnummern von Tourismusbüros, Krankenhäusern und anderen Einrichtungen. Weiterhin werden Preise für Benzin, Mietwagen, Unterkunft und Essen angegeben, die zwar nicht immer auf dem neusten Stand sind, sich aber insbesondere in fremden Ländern als hilfreich erweisen und als Richtlinie zu verstehen sind. Der Großteil eines Kapitels in den *Stormrider Guides*

widmet sich dann den einzelnen Spots und den dort herrschenden Surfbedingungen. Beispielsweise wird beschrieben zu welcher Gezeit bzw. Tide die Wellen an einem Spot am besten brechen, wie stark die Meeresströmung ist, ob die Brandung an einem Sandstrand oder auf einem flachen Riff ist oder ob irgendwelche Gefahren zu beachten sind. Jeder Beschreibung sind zusätzlich Piktogramme zugeordnet, welche die Surfbedingungen an einem Spot widerspiegeln und einen schnellen Überblick ermöglichen sollen. In Abbildung 1.2 sind z.B. zwei Spotbeschreibungen und die Piktogramme aus der Region Cantabria in Spanien zu sehen. Um die beschriebenen Spots zu finden, werden diese in den Reiseführern auf Kartenauschnitten eingezeichnet.

Abbildung 1.3: *"Don't even think about riding big Mavericks"* - Kommentar des *World Stormrider Guide* zu *Mavericks*, einem bekannten *Big Wave Spot* in Nordkalifornien, USA.

1.3 Swell - Die Entstehung von Wellen

Die besten Surfspots auf der Welt sind meist in den Ländern zu finden, in denen regelmäßig *Swell* eintrifft. Mit *Swell* oder Dünung werden Seewellen bezeichnet, deren Entstehungsgebiet weit von dem Ort entfernt ist, an

dem sie später eintreffen und brechen. Swell wird von den unterschiedlichsten Wetterphänomenen erzeugt, zu denen z.B. Wirbelstürme, Passatwinde, Monsune und Tiefdruckgebiete gehören [BS98, S.15]. Der in Europa eintreffende Swell entsteht dabei hauptsächlich durch die Luftzirkulation in Tiefdruckgebieten über dem Atlantik. Bei ruhigen Wetterverhältnissen kann man einen größeren Swell sehr gut erkennen, da die eintreffenden Wellen meist in gleichem Abstand linienförmig angeordnet sind. Diese linienförmige Anordnung ist z.B. besonders gut in Abbildung 1.4 zu erkennen.

Abbildung 1.4: Linienförmig eintreffender Swell (Quelle: *Flickr*)

Am Ursprungsort entstehen Wellen dadurch, dass die Wasseroberfläche durch die über ihr strömenden Winde in Bewegung gesetzt wird. Die Gipfel und Täler der Wellen werden durch die Zirkulation des Windes über der Oberfläche immer höher und tiefer, bis sie ein Limit erreichen und in sich zusammenbrechen. Die Höhe der Wellen ist dabei von der Stärke, der Dauer und der Strecke, über die der Wind strömt, abhängig. Die Wellen breiten sich vom Entstehungsort kreisförmig auf ihre Umgebung im Meer aus. Die Geschwindigkeit der Ausbreitung hängt dabei von dem Abstand zwischen zwei Wellengipfeln ab, der sogenannten Wellenlänge. Das Wellenchaos am Entstehungsort mit vielen unterschiedlichen Wellenlängen beginnt sich mit der Ausbreitung langsam zu legen. Die schnelleren Wellen, mit weiter auseinander liegenden Gipfeln, beginnen, die langsameren Wellen zu überholen und fangen an, sich linienförmig zu ordnen. Die vorderen Wellen werden da-

bei zu den kräftiger und linienförmiger angeordneten, die hinteren Wellen zu den schwächeren und chaotischeren. Je weiter die Strecke, die ein Swell hinter sich gelegt hat, und je linienförmiger er geordnet ist, desto größer ist die Geschwindigkeit und die Kraft der Wellen an dem Ort, an dem sie brechen. Das Eintreffen von Swell ist eine der Grundvoraussetzungen für gute und surfbare Wellen. Deshalb gibt es z.B. in Deutschland keine guten Wellen, da der potentiell eintreffende Swell meist durch England abgeschirmt wird.

1.4 Ziel der Diplomarbeit

Ziel der Diplomarbeit ist die Entwicklung einer *Community Plattform* für Surfer, welche den Grundgedanken der *Stormrider Guides* aufgreift und mit den neuen Möglichkeiten des Internets und des *Web 2.0* verknüpft. Grundlage der Plattform sollen Informationen zu den Surfbedingungen an den verschiedenen Spots sein, die gemeinschaftlich durch die Mitglieder der Community erstellt und bearbeitet werden können. Diese Spotbeschreibungen sollen mit aktuellen Wetter- und Wellenvorhersagen verknüpft und den Nutzern der Plattform zur Verfügung gestellt werden. Durch die Integration externer Dienste wie *Google Maps*, *Flickr* und *YouTube* sollen die bisherigen Informationen aufgewertet werden. Ziel ist es, eine Plattform zu entwickeln, auf der für Surfer wichtige Informationen angeboten werden.

Einige der dabei aufgetretenen Probleme und Lösungen werden in dieser Arbeit vorgestellt und diskutiert. Im Kapitel *Anforderungen an die Web-Applikation* wird auf die Funktionalität der Anwendung und auf einige der zugrunde liegende Konzepte und Dienste eingegangen. Die zur Umsetzung verwendeten Technologien und einige nennenswerte Methoden zur Entwicklung von Web-Applikationen werden anschließend im Kapitel *Implementierung der Web-Applikation* vorgestellt. Der Hauptteil beschäftigt sich mit der Integration der Wetter- und Wellenvorhersagen, die im Kapitel *Aufbau und Analyse der ETL Prozesse* näher beschrieben werden. Im Kapitel *Ausblick* werden schließlich einige Vorschläge gemacht, mit denen die Wetter- und Wellenvorhersagen verbessert werden könnten, indem die lokalen Gegebenheiten eines Spots indirekt mit einbezogen werden. Einer dieser experimentellen Vorschläge bedient sich dabei Verfahren aus dem Bereich des Data Mining, konnte hier aber wegen fehlenden Trainingsdaten nicht weiter verfolgt werden.

KAPITEL 2

Anforderungen an die Web-Applikation

2.1 Spotbeschreibungen

Eine der bekanntesten Wellen in Europa ist im spanischen Baskenland an einer Flussmündung östlich des Ortes *Mundaka* zu finden. Im *Stormrider Guide Europe* [BS98, S.180] werden die Surfbedingungen an diesem Spot wie folgt beschrieben.

Some of the longest, hollowest lefts in Europe break over sandbanks at the mouth of the river Gernike. It's best at low tide, when the rivers current (a useful conveyor belt) is less intense, and holds swell up to 4m (12ft). On the other side of the rivermouth are the beaches Laida and Laga where you can find waves at small swells. The river and its estuary are a Worldwide Fund for Nature reserve, but even so, the water is not as clean as could expected.

Zusätzlich werden den Beschreibungen Piktogramme aus verschiedenen Kategorien zugeordnet, welche die Gegebenheiten vor Ort durch eine vereinfachte grafische Darstellung widerspiegeln. Einige dieser Kategorien sind die bevorzugte Windrichtung, die optimale Gezeit, die Art der brechenden Welle, Beschaffenheit des Bodens sowie mögliche Gefahren. Durch die Piktogramme ist mit einem kurzen Blick schnell ersichtlich, welche Bedingungen an einem Spot herrschen. In Abbildung 2.1 sind die Piktogramme zu sehen, die für die obige Beschreibung in Mundaka verwendet wurden. Sie sollen vermitteln, dass dieser Spot sehr bekannt ist und an guten Tagen mit vielen Surfern zu rechnen ist. Die Welle bricht mit viel Kraft von links nach rechts (*Left-hander*, immer vom Strand aus gesehen) über einer Sandbank, wobei vereinzelt Surfbretter zu Bruch gehen können. Sie ist nicht bei Flut (*High*

Abbildung 2.1: Piktogramme zu den Surfbedingungen in Mundaka

Tide) surfbar, und ablandiger Wind (*Offshore*) aus dem Süden trägt dazu bei, dass die Wellen geglättet werden, später brechen und hohler werden.

Das Konzept der Stormrider Guides soll die Grundlage der Web Applikation bilden und mit den Möglichkeiten des Internets verknüpft werden. Die Beschreibungen zu den Spots und deren Gegebenheiten sollen gemeinschaflich durch die Mitglieder der Surf Community verfasst werden, und als sogenannter *User Generated Content* verwaltet werden. Die gemeinsam erstellten Beschreibungen sollen eine objektive Sicht auf die Gegebenheiten und Surfbedingungen an den jeweiligen Spots bieten. Für persönliche Ansichten und Diskussionen werden zusätzlich Kommentarfunktionen zur Verfügung gestellt. Um Spam und anderer mutwilliger Zerstörung oder Verunreinigung des Contents vorzubeugen, ist für bestimmte Funktionen eine Registrierung der Nutzer erforderlich. Für alle Informationen, die gemeinschaftlich von Nutzern der Web-Applikation verändert werden können, soll eine Historie verwaltet werden. Dadurch soll sichergestellt werden, dass bei einer eventuellen Verunreinigung des Contents auf eine frühere Version der Information zurückgegriffen und der ursprüngliche Zustand wieder hergestellt werden kann.

2.2 Kartenmaterial

Um das Auffinden von Spots zu vereinfachen, sollen diese auf einer Karte dargestellt werden. Webservice-Dienste wie *Google Maps*, *Yahoo! Maps* oder Microsoft's *Bing Maps* [1] bieten die Möglichkeit, interaktive Karten per Javascript oder Flash in eine Webseite einzubetten. Diese Dienste bieten nicht nur die typischen Land- bzw. Straßenkarten an, sondern stellen auch Satelitenbilder und teilweise dreidimensionale Ansichten für bestimmte Gebiete zur Verfügung. In Abbildung 2.2 ist eine Karten- und eine Satellitenansicht von dem Surfspot in *Mundaka* zu sehen. Insbesondere die Satellitenansicht ist beim Auffinden von neuen oder geheimen Surfspots recht nützlich, da man darauf sehr gut brechende Wellen erkennen und sich einen Überblick auf das Gelände verschaffen kann.

[1] seit Juni 2009, früher: Microsoft's Virtual Earth

(a) Karten Ansicht (b) Satelliten Ansicht

Abbildung 2.2: *Google Maps* Kartenmaterial für Mundaka, Spanien

2.3 Wetter- und Wellendaten

Wie schon in der Einleitung erwähnt, ist das Vorhandensein von Swell eine der Grundvoraussetzungen zum Surfen. Viele Surfer nutzen deshalb regelmäßig Dienste im Internet, um sich über die Wetter- und Wellenverhältnisse in den nächsten Tagen zu informieren. Dabei ist hauptsächlich die Wellenhöhe, die Wellenperiode sowie die Stärke und die Richtung des Windes von Interesse. Die Spotbeschreibungen sollen deshalb mit aktuellen Wetter- und Wellenvorhersagen verknüpft werden, um den Benutzern der Web-Applikation einen Ausblick auf die Surfbedingungen in den nächsten Tagen bieten zu können. Die *National Oceanic and Atmospheric Administration (NOAA)* ist die Wetter- und Ozeanografiebehörde der Vereinigten Staaten. Sie besteht aus 5 größeren Organisationen, zu denen unter anderen auch der *National Weather Service (NWS)* und der *National Ocean Service (NOS)* gehören, welche die benötigten Wetter- und Wellendaten zur Verfügung stellen. Viele der im Internet verfügbaren Dienste, die Wetter- oder Wellenvorhersagen anbieten, beziehen ihre Daten ebenfalls von diesen Organisationen. Diese Vorhersagen sind zwar insbesondere an Küstengegenden nicht sehr genau, tragen aber einen wichtigen Teil dazu bei, die Surfbedingungen in einer bestimmten Region oder an einem Spot grob einzuschätzen zu können.

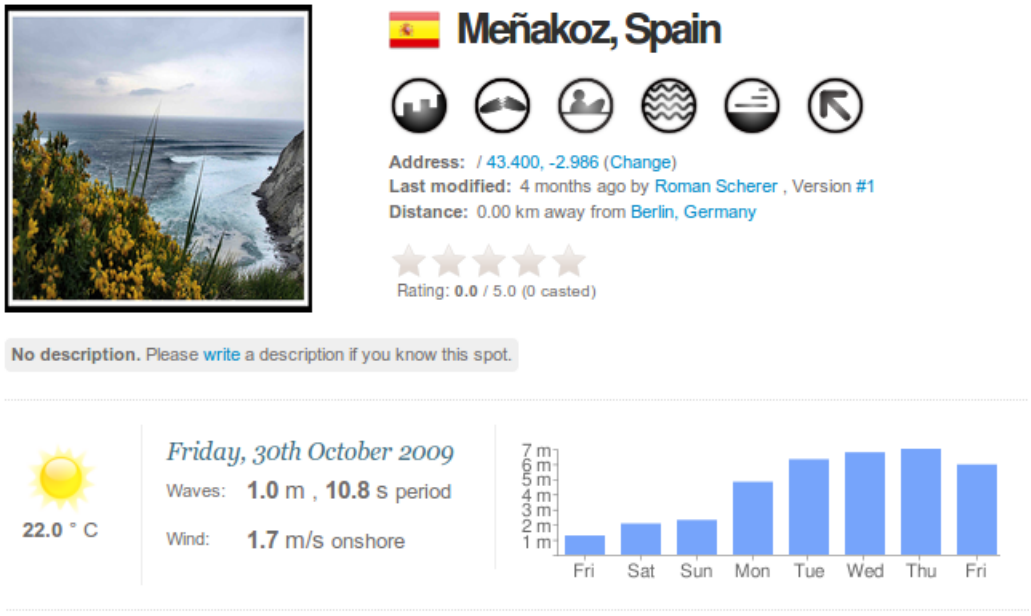


Abbildung 2.3: Wetter- und Wellenvorhersage für Meñakoz, Spanien

2.4 Bilder & Videos

Um Surfern einen visuellen Eindruck der Spots bieten zu können, sollen diese mit Bildern und Videos verknüpft werden. Diese in der Surf Community gerne gesehenen Bilder und Videos sind ideal um das bisher aus Beschreibungen, Wetter- und Wellenvorhersagen bestehende Informationsangebot aufzuwerten. Zum einen sollen externe Bilder und Videos durch die Nutzung von *Web Services* in die Applikation integriert werden, und zum anderen den Nutzern die Möglichkeit gegeben werden, ihre eigenen Bilder und Videos auf der Plattform zu publizieren.

Integration externer Bilder und Videos

Auf Internetseiten wie *Flickr* und *YouTube* sind von vielen bekannten Spots Bilder und Videos zu finden. Eine Suchanfrage nach einem bekannten Spot mit den Stichwörtern *Mavericks* und *Surf* ergab im Juni 2009 bei *Flickr* 2319 und bei *YouTube* 548 Ergebnisse. Die Integration diese externen Informationen ist insbesondere im Anfangsstadium der Web-Applikation nützlich, da in dieser Phase noch wenig Content durch die Nutzer generiert wurde. Durch Einbindung externer Bilder und Videos kann somit die Qualität des gesamten Content verbessert werden. Sowohl *Flickr* als auch *YouTube* bieten Web

Service Schnittstellen an, mit denen es möglich ist deren Bilder und Videos in eigene Anwendungen zu integrieren.

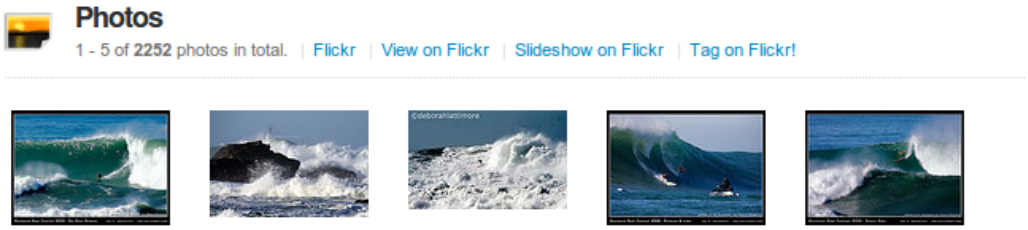


Abbildung 2.4: Integration von Bildern durch den *Flickr* Webservice

Publikation eigener Bilder und Videos

Zudem soll den Nutzern die Möglichkeit gegeben werden, ihre eigenen Bilder und Videos auf die Community-Plattform zu laden und diese dort zu veröffentlichen. Im Moment wird genau diese Funktionalität mit großem Erfolg zwar auch von *Flickr* und *YouTube* angeboten, aber mit Blick auf zukünftige Erweiterungen ist es sinnvoll, die volle Kontrolle über die auf der Plattform angezeigten Bilder und Videos zu haben. Der Einbau eines Abstimmungssystems und ähnlicher Funktionen kann mit externen Abhängigkeiten schnell problematisch werden und soll deshalb vermieden werden. Um das vor allem durch Videos erhöhte Datenvolumen in den Griff zu bekommen, werden Bilder und Videos auf *Amazon's Simple Storage Service (S3)* [2] gehostet. Dadurch können Probleme wie mangelnder Speicherplatz, Skalierbarkeit und hohe Verfügbarkeit der Daten ausgelagert werden und zugleich die Komplexität beim Deployment der Plattform reduziert werden.

2.5 Community-Funktionen

Community-Plattformen wie *Facebook*, *MySpace* und *StudiVZ* erfreuen sich in den letzten Jahren einer sehr großen Beliebtheit. Einige der dort bewährten Funktionalitäten sind auch in der hier entwickelten Web-Applikation sinnvoll und könnten einen Beitrag dazu leisten, den Nutzern einen Mehrwert zu bieten und sie enger an die Plattform zu binden. Beispielsweise sollen die gemeinsam erstellten Spotbeschreibungen, die eine objektive Sicht auf einen Spot darstellen, mit den subjektiven Kommentaren und Meinungen der Nutzer verknüpft werden, um eine Diskussion zu ermöglichen. Auf der

[2] http://aws.amazon.com/s3

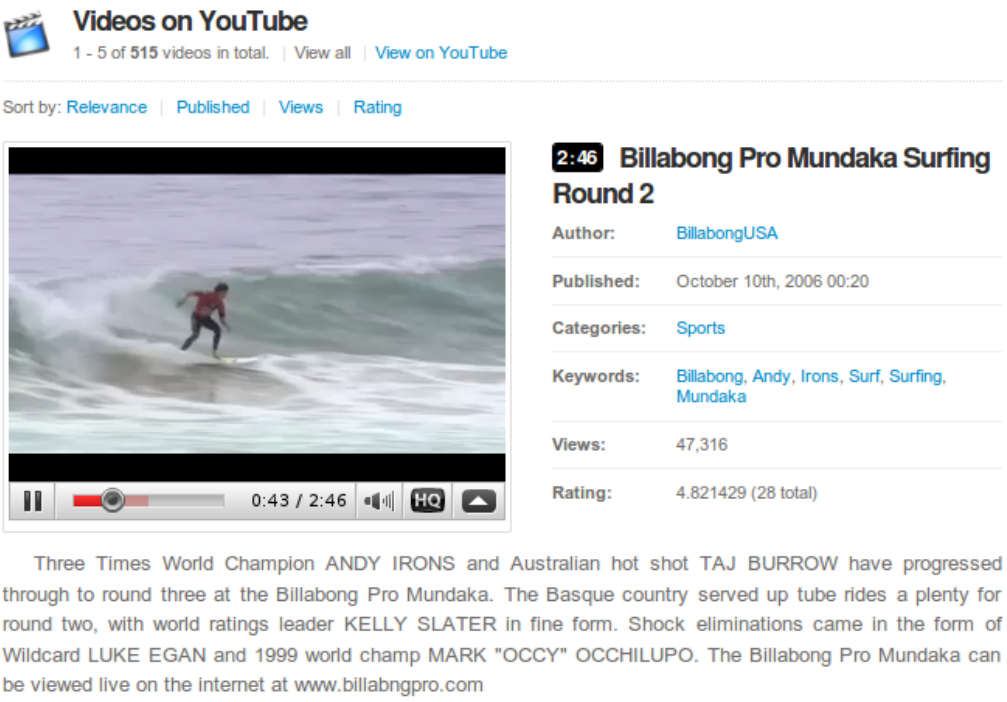


Abbildung 2.5: Integration von Videos durch den *YouTube* Webservice

Plattform publizierte Bilder und Videos sollen ebenfalls kommentiert und bewertet werden, damit diese nach verschiedenen Kriterien wie Relevanz, Beliebtheit usw. sortiert und angezeigt werden können. Damit die Plattform lebendiger wird, können Nutzer ihr eigenes Profilbild verwalten und Freundschaften mit anderen Nutzern eingehen. Darauf aufbauend soll es möglich sein, sowohl private als auch öffentliche Nachrichten an andere Nutzer zu senden und in den freigegebenen Profilseiten der registrierten Surfer zu stöbern. Um die Privatsphäre der Nutzer zu respektieren, müssen einige diese Funktionen von den Anwendern freigeschalten bzw. gesperrt werden können. Einige dieser Konzepte werden zwar von vielen als ”Spielerei” angesehen, für andere sind dies aber einige der Gründe, eine Community-Plattform zu nutzen und tragen wohl auch einen großen Teil zum Erfolg vieler bereits etablierter Community-Plattformen bei.

2.6 Nachhaltiges Publizieren

Ein relativ unschönes, aber teilweise auch verständliches Phänomen in der Surfszene ist der sogenannte Lokalismus oder *Localism* im Englischen. Ent-

gegen dem allgemeinen Klischee, dass Surfer friedliche und entspannte Menschen sind, wird dieser Eindruck an einigen Spots durch unsportliches Verhalten einiger weniger getrübt. Beim Surfen bekommt meist derjenige die Welle, der sie am schnellsten anpaddelt, sich die bessere Technik antrainiert hat und am meisten Erfahrung hat. Kurz gesagt, der Stärkere. Bei guten Surfbedingungen mit vielen Leuten führt dies unweigerlich dazu, dass einige Surfer die Wellen fast immer und andere fast nie bekommen. Insbesondere an Wochenenden und Feiertagen, an denen viele Surfer ihrem Hobby nachgehen, sind viele Spots überfüllt und es kann zu aggressivem Verhalten auf dem Wasser kommen.

Abbildung 2.6: *"Only Rodiles Locals"* - Lokalismus in Rodiles, Spanien

Vor Ort ansässige Surfer, sogenannte *Locals*, fühlen sich teilweise durch die vielen Besucher an "ihren" Spots bedrängt und reagieren auf diese mit aggressivem Verhalten. Wie in Abbildung 2.6 zu sehen ist, wird fremden Surfern deshalb oft durch Grafittis oder ähnliche Markierungen nahe gelegt, dass sie an einem Spot nichts zu suchen haben. Dies gelingt teilweise auch, da es zuweilen vorkommt, dass man einen Spot mit einem unguten Gefühl besucht. Grundsätzlich gilt deshalb, je weniger Surfer an einem Spot sind, desto entspannter ist die Atmosphäre und desto eher kommt jeder auf seine Kosten. Abgelegenere oder nur schwer zugängliche Spots werden deshalb von *Locals* oft als geheim gehandelt, gegenüber Fremden nicht erwähnt oder diese beim Suchen nach Spots in die Irre geführt.

Um die vor Ort ansässigen Surfer zu respektieren und nicht sofort jeden neuen Spot in die Welt hinaus zu posaunen, wurde hier deshalb das Konzept der *Secret Spots* eingeführt. Auf der Plattform sollen nur die allgemein bekannten Spots für jeden sichtbar sein. Diese Spots befinden sich meist an bekannten Stränden, sind ausgeschildert und auch in den Karten vieler Tourismusbüros verzeichnet. Andere Spots können beim Erstellen als *Secret Spots* markiert werden und sind dann nur für den Ersteller selbst und eingeladene Freunde sichtbar. Alle anderen Funktionen, wie Wetter- und Wellenvorher-

Abbildung 2.7: Ansicht der Einladungen zu einem Secret Spot

sagen stehen für diese Spots ebenfalls zur Verfügung, allerdings nur für einen eingeschränkten Nutzerkreis. Ein Prototyp der hier entwickelten Web Applikation ist unter der Adresse `http://staging.burningswell.com` zu finden.

KAPITEL 3

Implementierung der Web-Applikation

3.1 Vorstellung der verwendeten Technologien

3.1.1 Web Application Framework

Ein *Web Application Framework* ist eine Softwarebibliothek zur Entwicklung von dynamischen Webseiten, Web-Applikationen und Web Services. Ein solches Framework baut meist auf der in Abschnitt 3.2.1 beschriebenen *Model View Controller (MVC)* - Architektur auf und bietet Lösungen und Abstraktionen für im Bereich der Web-Programmierung häufig auftretenden Probleme und Konzepte. Hierzu gehört unter anderem der Zugriff auf Datenbanken, das Routing von URLs, Caching und Template-Systeme zur Generierung von Benutzeroberfächen.

Ruby on Rails - Web development that doesn't hurt

Ruby on Rails ist ein ursprünglich von David Heinemeier Hansson in der Programmiersprache *Ruby* geschriebenes Framework zur Entwicklung von Web-Applikationen. Es besteht aus den vier größeren Bibliotheken *ActionMailer*, *ActionPack*, *ActiveSupport* und *ActiveRecord*, mit denen Emails verarbeitet, Benutzeroberflächen erstellt, Daten persistent gespeichert und viele weitere Probleme bewältigt werden können. *Rails* basiert auf der bewährten *MVC*-Architektur, greift aber auch neuere Techniken, wie den in Abschnitt 3.2.2 beschriebenen *REST* -Architekturstil auf und ebnet so den Weg zu einer zukunftsorientierten Entwicklung von Web-Applikationen. Viele dieser bewährten Konzepte, aber auch neue und innovative Ideen haben dazu bei-

getragen, dass *Rails* in letzter Zeit viel Aufmerksamkeit auf sich gezogen hat. Durch Prinzipien wie *Don't Repeat Yourself (DRY)* und *Convention over Configuration* hat *Ruby on Rails* die Art und Weise, wie Web Anwendungen entwickelt werden, verändert. Die integrierten Testumgebungen für den in *Model*-, *View*-, *Controller*- und *Helper*-Klassen aufgeteilten Programmcode motivieren zum *Test Driven Development* und erleichtern den Einstieg. Nach der im Dezember 2008 angekündigten Verschmelzung mit *Merb*, dem zweitgrößten Framework zur Entwicklung von Web-Applikationen in *Ruby*, ist die Entscheidung, *Ruby on Rails* für dieses Projekt zu verwenden, noch zusätzlich untermauert worden.

Don't Repeat Yourself

Giving a computer two contradictory pieces of knowledge was Captain James T. Kirk's preferred way of disabling a marauding artificial intelligence. Mit diesem Zitat wird in [HT99][S.49] eine Diskussion über das *Don't Repeat Yourself* Prinzip eingeleitet. Die Duplikation von Informationen wird dort als eines der grundlegenden Probleme im Bereich der Softwareentwicklung identifiziert. Mit Information ist nicht nur der üblicherweise ohnehin zu normalisierende Inhalt von Datenbanken gemeint, sondern auch Konfigurationsdaten und vor allem Programmcode.

Convention over Configuration in ActiveRecord

In *Ruby on Rails* wird die Duplikation von Informationen vor allem durch die Anwendung von *Convention over Configuration* vermieden bzw. versucht, auf ein Minimum zu reduzieren. Das klassische Vorzeigebeispiel ist hier die Bibliothek *ActiveRecord*, der von *Ruby on Rails* verwendeter *objekt-relationale Mapper (ORM)*. Dieser ist dafür zuständig, in einer Programmiersprache verwendete Objekte auf das Schema einer relationalen Datenbank abzubilden. Dadurch können Daten persistent gespeichert und später wieder geladen werden. Als Beispiel soll hier die *One-To-Many* Beziehung zwischen den beiden zu persistierenden Klassen aus Auflistung 3.1 dienen: Ein Surfspot ist in einem bestimmten Land, und ein Land hat mehrere Surf Spots. Klassen repräsentieren hier die Tabellen einer Datenbank und Instanzen einer Klasse die Zeilen einer Tabelle.

Vorausgesetzt man hält sich an die durch *ActiveRecord* definierten Konventionen, reichen diese 6 Zeilen Code aus, um Länder und Spots zu speichern, zu finden, zu löschen und auf die Assoziationen zuzugreifen. Einzige Voraussetzung hierfür ist eine bestimmte, von *ActiveRecord* erwartete Struktur der Datenbank. Diese Struktur ist durch die folgenden Konventionen

```
1 class Country < ActiveRecord::Base
2   has_many :spots
3 end
4
5 class Spot < ActiveRecord::Base
6   belongs_to :country
7 end
```

Auflistung 3.1: *Convention over Configuration* in *ActiveRecord*

vorgegeben:

- Es müssen Tabellen in der Datenbank existieren, die nach dem Plural der kleingeschriebenen Klassennamen benannt sind. In diesem Beispiel also die beiden Tabellen *countries* und *spots*.
- Auf den Tabellen muss ein Primärschlüssel mit dem Namen *id* definiert sein. Instanzen einer Klasse können über dynamisch generierte Methoden auf diesen Primärschlüssel und alle zusätzlichen Attribute einer Tabelle zugreifen.
- Für eine *One-To-Many* Beziehung muss auf der Tabelle derjenigen Klasse ein Fremdschlüssel definiert sein, in der die Assoziation mit der *belongs_to* Methode definiert wurde. Der Fremdschlüssel muss aus dem kleingeschriebenen Namen der assoziierten Klasse und dem String *_id* zusammengesetzt sein. Hier also ein Fremdschlüssel mit dem Namen *country_id* auf der Tabelle *spots*.

Mit *ActiveRecord* lassen sich persistente Objekte und deren Beziehungen untereinander schnell und ohne größeren Konfigurationsaufwand modellieren. Im Vergleich zu den aus *Java* bekannten *Container Managed Entity Beans* wird der Entwicklungsaufwand hier erheblich reduziert. Die verwendeten Konventionen stammen aus den langjährigen Erfahrungen vieler Entwickler, treffen auf viele der Anwendunggsfälle zu und können als *"Best Practices"* verstanden und übernommen werden. *ActiveRecord* ist nur eine der von *Ruby on Rails* verwendeten Bibliotheken, die gut dokumentiert sind, innovative Konzepte bieten und somit eine einfache und flexible Entwicklung ermöglichen.

Template-Systeme

Dynamische Webseiten und Web-Applikation verwenden meist eine *Template Engine*, oder auch *Template System* genannt, um *Markup* [1] zu generieren. Als

[1] in Web-Applikationen üblicherweise HTML oder XML

Template bezeichnet man eine mit Platzhaltern und Instruktionen annotierte Datei, die vom Template System in ein Ausgabeformat transformiert wird. Platzhalter werden dabei durch dynamischen Inhalt ersetzt und Instruktionen bieten die Möglichkeit, den zu generierenden Inhalt durch Schleifen oder Bedingungen programmatisch zu kontrollieren. Einige bekannte Template Systeme sind z.B. *XSL/XSLT*, *Java Server Pages (JSP)* oder das aus der Ruby Standardbibliothek bekannte *Erb*.

Erb - Ruby Templating

Erb ist ein in die Ruby Standardbibliothek integriertes Template System, das beliebige Text Dokumente verarbeiten kann und zur Zeit noch das Standard Template System in *Ruby on Rails* Anwendungen ist. In Auflistung 3.2 ist ein *Erb* Template zu sehen, in dem Platzhalter verwendet werden, um *XHTML* mit dynamischem Inhalt zu generieren. Platzhalter werden in *Erb* Templates durch die speziellen Tags <%= und %> markiert. Der zwischen diesen Tags stehende Ruby Code wird vom Template System evaluiert und der Rückgabewert verwendet, um den Platzhalter zu ersetzen.

```
1  <div id="profile">
2    <div class="left column">
3      <div id="date"><%= print_date %></div>
4      <div id="address"><%= current_user.address %></div>
5    </div>
6    <div class="right column">
7      <div id="email"><%= current_user.email %></div>
8      <div id="bio"><%= current_user.bio %></div>
9    </div>
10 </div>
```

Auflistung 3.2: Beispiel eines *Erb*-Templates

Die Flexibilität von *Erb*, beliebige Eingabeformate verarbeiten zu können, ist gleichzeitig auch einer der großen Nachteile. Tippfehler, unbalancierte Tags und ähnliche Fehler bereiten oft Probleme. Durch inkorrektes *Markup* verursachte Layoutfehler lassen sich bei der Verwendung von *Erb* oft schwer finden und erforden oft ein zeitaufwendiges Debugging.

Haml - Markup Haiku

"Haml is 1.3x slower than straight ERB, but once you start writing code in Haml, it's highly addictive."

Eine sehr zu empfehlende und produktivitätssteigernde Alternative zu *Erb* ist *Haml*. *Haml* steht für *XHTML Abstraction Markup Language* und spielt mit seinem Untertitel berechtigterweise auf die kürzeste, aus dem japanischen stammende Gedichtsform *Haiku* an. Mit *Haml* lassen sich *XHTML*

Dokumente auf einer sehr viel kompakteren, lesbareren und somit übersichtlicheren Art und Weise beschreiben. In Auflistung 3.3 ist ein *Haml* Template zu sehen, das äquivalent zum Beispiel aus Auflistung 3.2 ist und dasselbe *Markup* generiert.

```
1 #profile
2   .left.column
3     #date= print_date
4     #address= current_user.address
5   .right.column
6     #email= current_user.email
7     #bio= current_user.bio
```

Auflistung 3.3: Beispiel eines *Haml*-Templates

Insbesondere in *XHTML* Dokumenten, die in Verbindung mit Javascript und *CSS* verwendet werden, lassen sich einige immer wiederkehrende Muster identifizieren, die in *Haml* durch eine kompaktere Syntax ersetzt wurden.

Ein weiterer Vorteil von *Haml* gegenüber *Erb* ist, dass der Parser das zu verarbeitende Template auf syntaktische Korrektheit überprüfen kann, mögliche Fehler sofort meldet und damit sicherstellt ist, dass immer valides *XHTML* erzeugt wird. Layoutfehler wie sie bei fehlerhaften *Erb* Templates gang und gäbe sind, werden durch den *Haml* Parser sofort erkannt und treten deshalb erst gar nicht auf.

Statt der in *XHTML* Dokumenten verwendeten und sehr viel Redundanz erzeugenden Start- und Ende-Tags wird in *Haml* Dokumenten nur ein Tag benötigt. *Haml* generiert den durch XML vorgeschriebenen und zum Start Tag gehörenden Ende Tag automatisch. Um Elemente zu verschachteln, wird eine Einrückung von zwei Leerzeichen verwendet. Die in *XHTML* Dokumenten häufig in Verbindung mit *id* oder *class* Attributen verwendeten $<div>$ Elemente wurden in *Haml* mit einer speziellen Syntax versehen. Ein # Zeichen gefolgt von einem String wird zu einem $<div>$ Element transformiert, dessen *id* Attribut den Wert des Strings enthält. Ein ähnliches Prinzip wird auch für den Punkt "." in Verbindung mit dem *class* Attribut verwendet, so dass Zeilen 1 und 2 in beiden Auflistungen äquivalent sind. Diese beiden Erweiterungen der Syntax erleichtern die Arbeit beim Erstellen von Templates erheblich und führen insgesamt zu wesentlich übersichtlicheren und kürzeren Dateien.

Das eng mit *Haml* verwandte Template System *Sass* bietet ähnliche Vorteile und wird dazu verwendet, *Cascading Style Sheets (CSS)* zu generieren. Das Template System definiert ebenfalls eine auf syntaktische Korrektheit überprüfbare Sprache und generiert als Ausgabeformat *CSS*. Zudem bietet es die in *CSS* nicht vorhandene Möglichkeit Variablen zu definieren, simple arithmetische Operationen auszuführen, Vererbung und einige weitere nützliche Funktionen.

Die durch *Haml* und *Sass* definierten Sprachen sind sehr einfach zu erlernen und die Verwendung dieser Systeme hat sich bei der Entwicklung der Web-Applikation als sehr hilfreich herausgestellt. Die übersichtlicheren und kompakteren Templates und deren Überprüfung auf Korrektheit überzeugten nach einiger Zeit so sehr, dass vor allem die Syntax von *CSS* leicht in Vergessenheit geriet. Diese beiden alternativen Template-Systeme sind für die Entwicklung von *Ruby on Rails* Anwendungen sehr zu empfehlen, können aber auch unabhängig von *Rails* eingesetzt werden.

3.1.2 Datenbankmanagementsystem

In Web-Applikationen wird für Anfragen an Daten und deren persistente Speicherung meist ein *objekt-relationaler Mapper (ORM)* verwendet. Dieser ist dafür zuständig in einer Programmiersprache verwendete Objekte auf das Schema einer relationalen Datenbank abzubilden. Dadurch können Daten persistent gespeichert und später wieder geladen werden. In *Ruby on Rails* Anwendungen kommt hier meistens das *ActiveRecord* Framework zum Einsatz, das die meisten der gängigen Datenbankmanagementsysteme (DMBS) unterstützt.

Bei der Auswahl des für die Web-Applikation verwendeten Datenbank Management Systems kamen kommerzielle Produkte wie Oracle's 11g oder DB2 von IBM aus Kostengründen nicht in Frage. Insbesondere die integrierten Möglichkeiten zur Auswertung von großen Datenbeständen [2] mit Data-Mining-Verfahren wären bei diesen Produkten eine interessante Anwendung. Leider sind diese Optionen in den kostenlosen Varianten dieser Produkte jedoch nur eingeschränkt oder überhaupt nicht nutzbar.

Bei den Open Source Datenbankmanagementsystemen fiel die Auswahl auf *PostgreSQL*. Zum einen wurden schon in der dieser Diplomarbeit vorausgegangenen Studienarbeit gute und tiefgründige Erfahrungen mit diesem *DBMS* gesammelt, zum anderen wird die Funktionalität von *PostgreSQL* selbst und einiger der zur Verfügung stehenden Erweiterungen den benötigten Anforderungen gerecht. Einige dieser Anforderungen sollen hier beschrieben werden.

- Ein sogenannter *Bulk Loader* wird benötigt, um größere Datenmengen zu exportieren bzw. zu importieren. *PostgreSQL* bietet hierfür die *COPY FROM/TO*-Befehle, mit denen Dateien in verschiedenen Formaten sehr viel schneller als mit den sonst üblichen *SELECT*-, *INSERT*- und *UPDATE*-Befehlen verarbeitet werden können. Diese Funktionali-

[2] hier die Historie der Vorhersagedaten und deren Klassifikation durch Benutzer

tät wird unter anderem für die Vorhersagedaten verarbeitenden *ETL* Prozesse benötigt.

- Das *PostGIS*-Projekt erweitert *PostgreSQL* mit geografischen Funktionen und Datentypen, die konform zu der *Simple Features Specification for SQL* des *Open Geospatial Consortium* sind. Hiermit können geografische Objekte in dem weit verbreiteten *Shapefile* Format verarbeitet werden. Insbesondere mit Hinblick auf die Integration von *Google Maps* bieten sich hier einige interessante Anwendungsmöglichkeiten.

- Aufbauend auf dem *PostGIS* Projekt stellt die *pgRouting* Erweiterung Funktionalitäten zur Routenplanung bereit. In das DMBS integrierte *Shortest Path* Algorithmen wie *Dijkstra*, *Shooting Star* und *Traveling Sales Person* können so effizient ausgeführt werden. Der *Dijkstra* Algorithmus wird beispielsweise in der Web-Applikation dazu verwendet, die aus vielen anderen Community-Plattformen bekannten Freundschaftsbeziehungen zwischen Benutzern zu berechnen.

3.2 Architektur der Web-Applikation

In diesem Abschnitt wird auf die Architektur der Web-Applikation eingegangen. Das *Ruby on Rails* Framework baut selbst auf der *Model View Controller* Architektur auf und ermöglicht es, Web Anwendungen in dem in letzter Zeit häufig diskutierten *REST* Architekturstil zu entwickeln, dessen Vorteile im folgenden vorgestellt werden.

3.2.1 Model View Controller

Die *Model View Controller (MVC)* Architektur [GHJV94][S.10] wurde erstmals in *Smalltalk-80* verwendet, um Benutzeroberflächen zu entwickeln. Sie basiert auf drei verschiedenen Arten von Objekten. Ein *Model* repräsentiert die Daten der Applikation und die darauf möglichen Operationen, die sogenannte Geschäftslogik. Die Benutzeroberfläche besteht aus *Views* bzw. Sichten, welche für die Darstellung der Daten auf verschieden Medien und in unterschiedlichen Formaten verantwortlich sind. *Controller* definieren schließlich die Art und Weise, wie eine Benutzeroberfläche auf Eingaben von Benutzern reagiert. Ziel dieser Architektur ist es, ein Softwaresystem in einzelne, möglichst voneinander unabhängige Komponenten aufzuteilen. Dadurch soll die Komplexität der Software reduziert werden und die Erweiterung und Wiederverwendung des Programmcodes erleichtert werden.

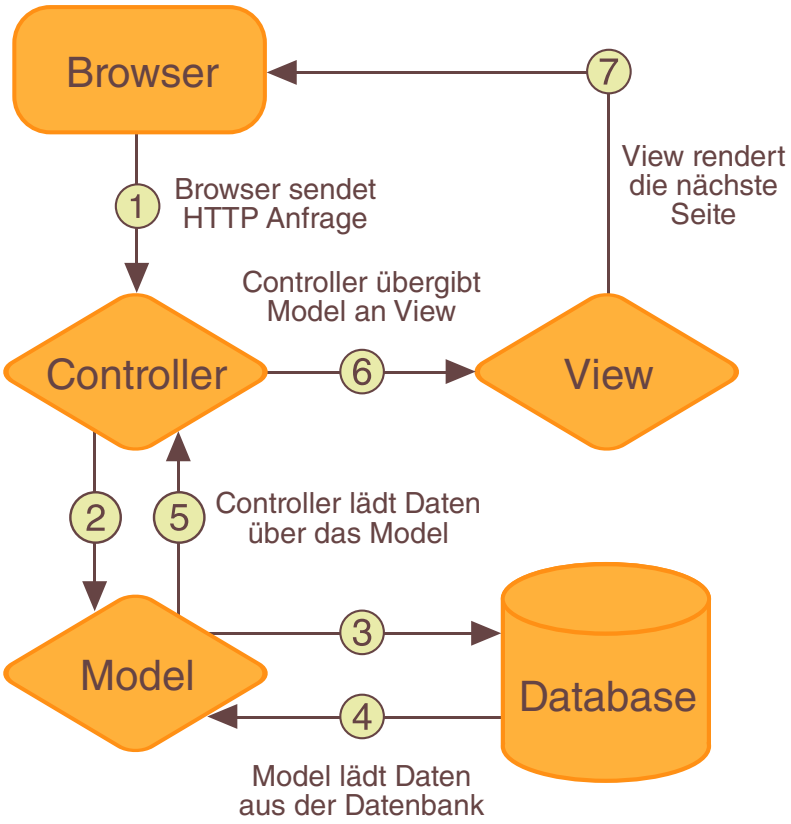


Abbildung 3.1: MVC-Architektur in *Ruby on Rails* Applikationen

3.2.2 Representational State Transfer

Das Akronym *REST* steht für *Representational State Transfer* und stammt ursprünglich aus der Doktorarbeit *Architectural Styles and the Design of Network-based Software Architectures* [Fie00] [3] von Roy T. Fielding. In dieser Arbeit werden verschiedene Architekturstile für das Design netzwerkbasierter Hypermedia Systeme vorgestellt und deren Verhalten unter anderem bezüglich Effizienz und Skalierbarkeit untersucht. In Kapitel 5 wird *REST* als Architekturstil vorgestellt, der bewährte Konzepte aus den zuvor analysierten Architekturen aufgreift und Regeln und Anforderungen für das Design moderner Web-Architekturen vorgibt. Fielding, der auch an der Spezifikation des *Hypertext Transfer Protocol (HTTP)* beteiligt war, führt den großen Erfolg des *World Wide Web (WWW)* unter anderem auf dessen Architektur und die verwendeten Technologien zurück, die sich in den von ihm vorge-

[3] `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`

schlagenen Regeln widerspiegeln.

Resource-Oriented Architecture

In *RESTful Web Services* [RR07] werden Fielding's Vorschläge auf die Architektur moderner *Web Services* übertragen und *Resource-Oriented Architecture (ROA)* als ein konkreter Architekturstil vorgestellt, der sich an den Konzepten von *REST* orientiert. Im Folgenden werden die zugrunde liegenden Konzepte und deren Anwendung anhand einiger Beispiele vorgestellt.

Ressourcen Der Begriff *Ressource* wird in der *Resource Oriented Architecture* als Abstraktion verwendet, um Dinge zu bezeichnen, die auf einem Computer gespeichert, in einer Folge von Bits repräsentiert und auf irgendeine Art und Weise identifiziert und referenziert werden können. Dokumente auf einem Web Server, Zeilen einer Datenbanktabelle oder das Ergebnis eines Algorithmus auf einer bestimmten Eingabe können als Ressource modelliert werden. Einige konkrete Beispiele für eine Ressource sind:

- Die Version 3.0.12 eines Softwarepakets.
- Die letzte stabile Version eines Softwarepakets.
- Die Beschreibung der Surfbedingungen in *Mundaka, Spanien.*
- Die Wettervorhersage für *Mundaka* am 26. Juli 2009 um 15:00h.

Identifizierung Damit man auf eine bestimmte Ressource zugreifen kann, müssen diese auf irgendeine Art und Weise identifiziert werden können. Im Web werden hierfür üblicherweise sogenannte *Uniform Resource Identifier (URI)* verwendet. Eine Ressource muß mindestens einen *URI* besitzen, die von Anwendungen verwendet wird um Informationen über eine Ressource zu bekommen oder diese zu verändern. Die eben als Beispiel erwähnten Ressourcen könnten z.B. über die folgenden *URI*s identifiziert werden:

- http://example.com/software/releases/3.0.12.tar.gz
- http://example.com/software/releases/latest.tar.gz
- http://example.com/spots/03-Mundaka/description.html
- http://example.com/spots/03-Mundaka/forecasts/2009-07-26/15h.html

Wie an den ersten beiden Beispielen zu sehen ist, kann eine Ressource zu einem bestimmten Zeitpunkt auch über mehrere *URI*s identifiziert werden: Zu einem bestimmten Zeitpunkt war Version 3.0.12 der Software auch die aktuellste Version. Der Aufbau oder die Struktur von *URI*s wird durch *REST* nicht strikt vorgegeben. Eine durch den Client vorhersehbare Struktur wird aber in der *Resource Oriented Architecture* empfohlen, da dies dem Client die Möglichkeiten gibt sehr viel flexibler auf den *Web Service* zuzugreifen.

Adressierbarkeit Ein *Web Service* ist addressierbar, wenn die zur Verfügung gestellten Informationen als Ressourcen publiziert werden und auf diese durch *URI*s zugegriffen werden kann. Die Anzahl der *URI*s, die von solch einem *Web Service* angeboten werden, ist durch die Vielzahl der möglichen Eingaben meist unbegrenzt. Vertreter eines solchen *Web Services* ist z.B. die Suche nach dem Wort "REST" bei *Google*, welche durch folgenden *URI* adressierbar ist: `http://www.google.de/search?q=REST`.

Addressierbarkeit ist eine wichtige Eigenschaft *REST*-basierter Web-Architekturen und bringt viel mehr Anwendungsmöglichkeiten mit sich als das "bloße Publizieren" von Informationen. Beispiele hierfür sind:

- Die Adressen, mit denen Informationen referenziert werden, können z.B. in Büchern abgedruckt, als Lesezeichen gespeichert, per Mail weitergeleitet und als Eingabe für Programme dienen, welche den Inhalt weiterverarbeiten. Dadurch muss nicht der eigentliche Inhalt verbreitet oder kommuniziert werden, sondern nur die Adresse, unter der dieser Inhalt zu finden ist.

- In *HTTP Proxy Caches* wird die Adressierbarkeit von Ressourcen ausgenutzt um den Zugriff darauf zu beschleunigen. Bei einer ersten Anfrage wird die Ressource vom Proxy zwischengespeichert und weitere Anfragen an diese aus dem Cache bedient.

- Durch *URI*s adressierbare *Web Services* bieten nicht nur den speziell entworfenen Programmen Zugang zu Ressourcen, sondern auch Menschen, die mit Standardwerkzeugen wie z.B. einem *Web Browser* oder einem *RSS Reader* darauf zugreifen, oder Agenten des *Semantic Web*.

Zustandslosigkeit Unter Zustandslosigkeit bzw. *Statelessness* versteht man in Protokollen zur Client-Server Kommunikation, dass alle Anfragen des Clients an den Server komplett unabhängig voneinander sind. Anfragen des Clients müssen alle nötigen Informationen beinhalten, die vom Server benötigt werden, um diese zu bearbeiten. Auf Seiten des Servers wird keinerlei

Information über den Zustand von Anfragen verwaltet. Dadurch wird dessen Implementierung vereinfacht, spart Ressourcen beim Verarbeiten von Anfragen ein und bietet die Möglichkeit *Load Balancing* in Form von horizontaler Skalierung zu betreiben.

Repräsentation Das abstrakte Konzept einer Ressource wird durch ihre Repräsentation bzw. Darstellung konkretisiert. Fordert ein Client Informationen über eine Ressource an, wird eine bestimmte Repräsentation dieser Ressource als Sequenz von Bytes übertragen. Eine Ressource kann viele verschiedene Repräsentationen haben, beispielsweise als *HTML*-, *XML*- oder *JSON*-Dokument kodiert. Darstellungen in unterschiedlichen konkreten Sprachen sind ebenfalls mögliche Repräsentationen einer Ressource.

In der *Resource Oriented Architecture* wird empfohlen, die Information darüber, welche Art der Repräsentation einer Ressource der Server ausliefern soll, in den *URI* oder in die Metadaten des verwendeten Übertragungsprotokolls einzubetten. Mögliche Repräsentationen der Wettervorhersage für Mundaka am 26. Juli 2009 um 15:00h könnten z.B. als Dateiendung im *URI* kodiert werden:

- http://example.com/spots/03-Mundaka/forecasts/2009-07-26/15h.**html**
- http://example.com/spots/03-Mundaka/forecasts/2009-07-26/15h.**xml**
- http://example.com/spots/03-Mundaka/forecasts/2009-07-26/15h.**json**

Durch Differenzierung von Repräsentationen einer Ressource kann ein *Web Service* Anwendungen mit verschiedenen Anforderungen bedienen. *Web Browsern* kann z.B. eine aufwendigere, auf Interaktivität fokussierte Darstellung gesendet werden, datenverarbeitenden Programmen eine Variante in *XML* und mobilen Endgeräten eine leicht gewichtige Version im *JSON* Format.

Einheitliche Schnittstelle Eine einheitliche Schnittstelle zwischen den Komponenten einer *REST*-basierten Architektur ist eine weitere Forderung Fieldings, mit dem Ziel die Komplexität des zu entwerfenden Systems zu reduzieren. Das Akronym *CRUD* steht für die mindestens zu implementierenden Operationen einer im Sinne der Theorie vollständigen relationalen Datenbank: *Create*, *Read*, *Update* und *Delete*. Mit diesen Basis Operationen können die Objekte einer Datenbank erstellt, gelesen, verändert und gelöscht werden. Ausgehend von der Annahme, dass Ressourcen in einer bestimmten Repräsentation auch in einer relationalen Datenbank gespeichert

werden können, bieten sich die *CRUD*-Operationen auch beim Design einer einheitlichen Schnittstelle für Ressourcen an. In [RR07][S.97-105] wird das *HTTP*-Protokoll zur Implementierung dieser Schnittstelle vorgeschlagen. Dabei werden die vier *CRUD*-Operationen auf die im *HTTP*-Protokoll definierten *GET*, *POST*, *PUT* und *DELETE* Methoden abgebildet.

- Die Repräsentationen existierender Ressourcen werden mit der *GET* Methode angefordert, beispielsweise die *HTML* Repräsentation der Beschreibung zu den Surfbedingungen in *Mundaka*.

  ```
  GET http://example.com/spots/03-Mundaka.html
  ```

- Existierende Ressourcen werden mit der *DELETE* Methode gelöscht. Im folgenden Beispiel wird der Surfspot *Mundaka* gelöscht.

  ```
  DELETE http://example.com/spots/03-Mundaka
  ```

- Der Zustand existierender Ressourcen wird mit der *PUT* Methode verändert. Im *HTTP-Body* der Anfrage wird eine Repräsentation [4] der Ressource mitgesendet, die den veränderten Zustand widerspiegelt. Im folgenden Beispiel wird die geographische Position des Spot *Mundaka* mit der im *HTTP-Body* enthaltenen Repräsentation aktualisiert.

  ```
  PUT http://example.com/spots/03-Mundaka
  ```

  ```
  HTTP Body: <lat>43.406</lat><lng>-2.691</lng>
  ```

- Neue Ressourcen werden entweder mit der *POST*- oder der *PUT* Methode erstellt. Eine den Zustand der Ressource widerspiegelnde Repräsentation wird hier ebenfalls im *HTTP-Body* kodiert.

  Mit welcher Methode und *URI* die Anfrage gesendet wird, ist abhängig davon, ob der Client (*PUT*) oder der Server (*POST*) den *URI* der neuen Ressource festlegt. Diese Unterscheidung ist auf die Einhaltung der im *RFC-2616* beschriebenen *HTTP/1.1*-Spezifikation zurückzuführen und wird in [RR07][S.99-102] ausführlicher erläutert. Die zwei folgenden Beispiele sollen beide Situationen verdeutlichen.

  POST: Der *URI* für Spots wird in der hier entwickelten Web-Applikation aus dem Datenbank-Identifikator und dem Namen des Spots zusammengesetzt. Da der Datenbank-Identifikator dem Benutzer beim Erstellen des neuen Spots nicht bekannt ist, muss hier die *POST* Methode verwendet werden.

  ```
  POST http://example.com/spots
  ```

[4] In Web-Applikationen üblicherweise in XML oder Form-Encoded

PUT: Bei der Registrierung von Benutzern hingegen wird die *PUT* Methode verwendet. Der *URI* eines Benutzer wird in der Web-Applikation nur aus dem von ihm gewählten Nicknamen konstruiert. Hier legt also der Client den *URI* der neuen Ressource fest.

```
PUT http://example.com/users/bob
```

Die Konzepte der *Resource Oriented Architecture* wurden bei der Implementierung der Web-Applikation angewendet und haben sich bei der Entwicklung als sehr hilfreich erwiesen. Insbesondere die Modellierung von Ressourcen unter Berücksichtigung der einheitlichen Schnittstelle hat sich bewährt. Das "Denken" in Ressourcen und deren Implementierung führt nicht nur zu sauberen Schnittstellen nach außen hin, sondern auch zu modularem Programmcode. Die implementierten Ressourcen werden hauptsächlich im *XHTML* Format, vereinzelt aber auch im *JSON* Format repräsentiert. Das *JSON* Format wird in Verbindung mit *Google Maps* eingesetzt, um mit Javascript asynchron auf die Ressourcen zugreifen zu können. Die Regeln der *Resource Oriented Architecture* können als *"Best Practices"* verstanden werden und sind bei der Konzeption und beim Design von Web-Anwendungen überaus hilfreich.

3.3 Behaviour & Test Driven Development

Behaviour Driven Development ist eine Erweiterung des *Test Driven Development* und eine Methode aus dem Bereich der *Agilen Softwareentwicklung*. Ziel der agilen Softwareentwicklung ist es den klassischen Softwareentwicklungsprozess schlanker und flexibler zu gestalten und sich mehr auf die zu erreichenden Ziele und Anforderungen zu fokussieren. Paarprogrammierung, ständige Refaktorisierung des Codes und testgetriebene Entwicklung sind nur einige der dabei verwendeten Methoden [Wik09a]. *Behaviour Driven Development* ist ein durch Dan North geprägter Begriff und eine Erweiterung der Philosophie des *Test Driven Developments* [Wik09b]. Durch die Einführung eines gemeinsamen Vokabulars soll die Zusammenarbeit zwischen Programmierern und nicht-technisch Beteiligten an einem Softwareprojekt erhöht werden. Das gemeinsame Vokabular wird benutzt, um das Verhalten bzw. die Anforderungen der Software gemeinsam zu beschreiben und eine Spezifikation zu erstellen. Diese Spezifikation wird in Form von automatischen Tests in einer domänenspezifischen Sprache vom Programmierer implementiert. Die Spezifikation und die eigentliche Implementierung des Verhaltens geschieht dabei zeitnah.

3.3.1 Red, Green, Refactor

Die Entwicklung der Software erfolgt, wie beim Test Driven Development, durch ständige Wiederholung des *Red, Green, Refactor* Zykluses. *Red* und *Green* stehen für fehlschlagende bzw. erfolgreiche Testfälle und *Refactor* für die Umstrukturierung, Verbesserung und Säuberung von schon implementiertem Programmcode. Die Entwicklung der Anwendung wird in kleinere Teile, sogenannte *Feature*, aufgeteilt und diese nacheinander abgearbeitet. Das Verhalten eines zu implementierenden Features wird durch mehrere Testfälle beschrieben und somit spezifiziert. Dabei werden die folgenden drei Schritte solange wiederholt bis das gewünschte Feature ausreichend spezifiziert und implementiert ist. Die Kunst ist hierbei, immer nur sehr kleine Schritte pro Zyklus zu spezifizieren und zu implementieren.

Red - Die Spezifikation des Verhaltens

In der *Red* Phase wird ein korrekter, aber fehlschlagender Test geschrieben, der ein bestimmtes Verhalten eines noch nicht implementiertes Features spezifiziert. Dieser Test sollte nur einen sehr geringen Teil des gesamten Features verifizieren. Beispielsweise, dass eine Methode ein Array zurück gibt, die Elemente eines Arrays Instanzen einer bestimmten Klasse sind oder die Distanz zwischen zwei gegebenen geografischen Positionen korrekt berechnet wird.

Green - Die Implementierung des Verhaltens

Ziel der *Green* Phase ist es, den soeben geschriebenen und fehlschlagenden Test (und alle bisher erfolgreichen) durch eine geeignete Implementierung zum Erfolg zu bringen. Hierbei ist zu Erwähnen, dass auch eine vorübergehend nicht korrekte Implementierung erlaubt ist, solange alle Tests erfolgreich sind. Beispielsweise kann eine Methode vorübergehend immer einen leeren Array zurückgeben, wenn im Test nur auf die Rückgabe eines Arrays geprüft wird. Diese inkorrekten Implementierungen sind gewollte Indizien für die als nächstes zu spezifizierenden Testfälle. In einem der nächsten Zyklen wird die inkorrekte Implementierung durch einen neuen fehlschlagenden Test aufgedeckt und durch eine korrekte Implementierung ausgetauscht.

Refactor - Besserer Programmcode durch Umstrukturierung

Die *Refactor* Phase dient dazu, bisher entwickelten Programmcode durch Umstrukturierung aufzuräumen. Dazu zählt die Eliminierung doppelten oder ähnlichen Codes, das Extrahieren von Hilfsmethoden und weiterer Refactor-Techniken, die ausführlich in [FBB+99] beschrieben werden. Insbesondere in

Softwareprojekten, die ohne automatische Tests entwickelt werden, ist diese Phase hoch problematisch. Zuvor funktionierender und durch Refactoring geänderter Programmcode muss nochmals manuell verifiziert werden. Probleme treten oft auch an nicht offensichtlich mit den Änderungen in Verbindung stehenden Teilen des Programms auf. Ein deshalb notwendiger, manueller Systemtest kann dann oft zu aufwendig sein, so dass die Vorteile des Refactoring in den Hintergrund rücken und oftmals darauf verzichtet wird.

3.3.2 Vorteile Behaviour & Test Driven Developments

Diese Art der Softwareentwicklung wird heute oft immer noch aus den verschiedensten Gründen abgelehnt. *"Nicht notwendig, bei gut entwickeltem Code"*, *"zu aufwendig"*, oder Barrieren, seine gewohnte Arbeitsweise zu ändern sind nur einige dieser Gründe. Der Klassiker *"man würde ja, aber habe keine Zeit Tests zu schreiben"* zeigt, dass die Vorteile des *Test Driven Development* oft nicht erkannt werden. Dies liegt unter anderem wohl daran, dass es einige Zeit in Anspruch nimmt seine gewohnte Arbeitsweise zu ändern, und die Früchte erst nach einiger Zeit und Praxis deutlich zu sehen sind. Einige der Vorteile sollen im Folgenden deutlich gemacht werden.

Steigerung der eigenen Produktivität

Martin Fowler beschreibt in [FBB+99][S.89], dass viele Programmierer ihre Zeit nicht mit der Entwicklung des eigentlichen Programmcodes verbringen, sondern den Großteil der Zeit für Debugging und das manuelle Testen des Codes verwendet wird. Wirklich produziert wird hier aber nur der Programmcode der Implementierung. Beim *Test Driven Development* wird diese Zeit für dieselben Ziele verwendet, doch als Nebenprodukt entstehen automatische Tests. Manuelles Testen und Debugging muss unter Umständen immer wieder von neuem wiederholt werden, während automatische Tests sicherstellen, dass bestimmte Dinge funktionieren und nicht nochmals verifiziert werden müssen. Die Entwicklung von Tests bedeutet mehr Code, was mit mehr Arbeit gleichzusetzen ist. Dieser Mehraufwand scheint auf den ersten Blick keinen Sinn zu machen, bis man selbst erfahren hat, wie sehr dieses Vorgehen die eigene Produktivität steigern kann.

Besserer Programmcode

Schreibt man Tests vor der eigentlichen Implementierung blickt man von vornherein aus der Rolle eines Anwenders auf die zu entwickelnde Schnittstelle. Die API des zu implementierenden Codes kann so während der Ent-

wicklung der Testfälle ”erforscht” werden. Die Struktur des Programmcodes wird zudem positiv beeinflusst, da vor der Implementierung überlegt werden muss, wie das Verhalten am besten getestet werden kann. Implementierungen, die vor Testfällen entwickelt wurden, müssen oft umstrukturiert werden, damit diese überhaupt getestet werden können. Zu lange Methoden sind hier das klassische Beispiel. Sie lassen sich schlecht testen, weil sie oft viel zu viel auf einmal erledigen. Darüber hinaus sind sie meist auch noch schwer zu verstehen. Durch Extraktion von Hilfsmethoden kann man diese Methoden verkürzen und verbessert dadurch die Lesbarkeit des Codes. Die extrahierten Methoden und die verkürzte Methode selbst lassen sich nun in Isolation meist viel besser und einfacher testen als zuvor. Zudem können die extrahierten Hilfsmethoden potentiell von anderem Programmcode wiederverwendet werden.

Qualitätsmanagment und Teamwork

Insbesondere bei Softwareprojekten in dynamischen Programmiersprachen, die erst zur Laufzeit interpretiert werden und keine statische Typprüfung besitzen, ist es von großem Vorteil eine automatische *Testsuite* zu haben. In diesen Sprachen gibt es keinen Compiler, der nach Änderungen überprüfen kann, dass zumindest syntaktisch noch alles korrekt ist. Änderungen an einer Stelle im System haben oft Konsequenzen an anderen nicht vorhersehbaren Stellen. Eine *Testsuite*, welche die gesamte Codebasis abdeckt, kann sicherstellen, dass Änderungen funktionieren und eingeschlichene Fehler so früh wie möglich aufgedeckt werden.

Auch bei Softwareprojekten, die von mehreren Personen gleichzeitig entwickelt werden, bietet eine *Testsuite* Schutz gegen das Einschleichen von Fehlern und hilft bei der Weiterentwicklung fremden Codes. Wird Programmcode vom nicht ursprünglichen Autor geändert, muss dieser zuerst verstanden, dann geändert und anschließend verifiziert werden. Testfälle dienen dabei nicht nur zur Verifizierung dieser Änderungen, sondern auch als Dokumentation und Spezifikation und helfen somit, das Problem besser und schneller zu verstehen.

3.3.3 Unit Tests mit RSpec

RSpec [rsp] ist ein *Behaviour Driven Development* Framework für die Entwicklung von Software in der Programmiersprache Ruby. Es implementiert eine interne bzw. eingebettete domänenspezifische Sprache (DSL) zur Spezifikation von Anforderungen in Form von Unit Tests. Spezifikationen können wie bei den diversen xUnit Derivaten gruppiert, verifiziert und deren Resul-

tate angezeigt werden. Als Beispiel soll hier die Implementierung einer Klasse dienen, welche die Distanz in Kilometern zwischen zwei geografischen Positionen berechnet. Vor der eigentlichen Implementierung wird das Verhalten der Klasse anhand von Beispielen in der *RSpec* DSL wie folgt spezifiziert:

```
describe Haversine do
  it "should calculate a distance of 0 km between the same location"
  it "should calculate a distance of 878.01 km between Berlin and Paris"
end
```

Diese Spezifikation der Haversine-Klasse ist Teil des oben erwähnten gemeinsamen Vokabulars und zugleich gültiger und ausführbarer Ruby Code. Die Entwicklung einer einfachen, verständlichen und aussagekräftigen DSL hatte zum Ziel, dass Spezifikationen wie diese von oder mit nicht-technischen Personen erstellt werden können. Die eigentliche Verifizierung und Implementierung des Verhaltens soll dann vom Programmierer erstellt werden. Bevor die Haversine-Formel durch eine Klasse implementiert wird, erweitert der Programmierer die Spezifikation durch Code, der das Verhalten der Klasse verifiziert. Da die Spezifikation in der Programmiersprache Ruby geschrieben ist, kann diese vom Programmierer einfach und flexibel erweitert werden. Auch hier kommen Konstrukte der *RSpec* DSL zum Einsatz, welche die Lesbarkeit vereinfachen und die Spezifikation verständlicher machen sollen.

```
describe Haversine do

  before(:each) do
    @berlin = Location.make(:berlin)
    @paris = Location.make(:paris)
  end

  it "should calculate a distance of 0 km between the same location" do
    Haversine.distance(@berlin, @berlin).should == 0.km
  end

  it "should calculate a distance of 878.01 km between Berlin and Paris" do
    Haversine.distance(@berlin, @paris).should == 878.01.km
  end

end
```

Das Ausführen dieser Spezifikation schlägt auf Grund der noch fehlenden Implementierung fehl. Durch die Wiederholung des Red, Green, Refactor Zykluses wird nun das Verhalten der Klasse implementiert. Der erste Testfall kann durch die folgende, noch inkorrekte Implementierung zum Erfolg gebracht werden:

```
def Haversine.distance(from, to)
  return 0 # A first, naive implementation.
end
```

Im nächsten Iterationszyklus muß nun die eigentliche Haversine-Formel [Wik09c] implementiert werden, um auch den zweiten Testfall zum Erfolg

zu bringen. Durch das Aufteilen der zu bewältigenden Aufgaben in kleine Iterationszyklen wird das spezifizierte Verhalten so Schritt für Schritt implementiert. Die nächste zu bearbeitende Teilaufgabe wird durch einen neuen Testfall beschrieben und das bisher erwartete Verhalten durch die bestehenden verifiziert.

Das im Umfeld von *RSpec* entwickelte *Spec::Rails*-Plugin bietet die nötige Infrastruktur für *Ruby on Rails* Projekte. An die MVC-Architektur von *Rails* angelehnt werden verschiedene Umgebungen und Hilfsmethoden bereit gestellt um Model, View, Controller und Helper zu spezifizieren. Während der Entwicklung des Prototypen der Web Applikation wurden insgesamt 4274 Testfälle spezifiziert, die 92.6% (C0 Code Coverage) des gesamten Codes abdecken und das korrekte Verhalten der Web-Applikation verifizieren.

Das Durchlaufen der gesamten *Testsuite* benötigt momentan ca. 15 Minuten, was zu einem Durchschnitt von 4,8 Sekunden pro Test führt. Diese hohe Laufzeit lässt sich auf die vielen Datenbankzugriffe zurückführen, die in vielen Testfällen gemacht werden. Durch den Einsatz von *Mocks* und *Stubs* können diese Datenbankzugriffe reduziert werden. Diese Techniken sollten jedoch vorsichtig eingesetzt werden. Eine übermäßige Anwendung resultiert meist in einer zu engen Kopplung zwischen Test und Implementierung. Diese Kopplung macht sich dann meist beim Refactoring bemerkbar, wenn Tests ebenfalls geändert werden müssen weil aus Performanzgründen Annahmen über die Implementierung getroffen wurden. Hier muss man einen Mittelweg zwischen der Wartbarkeit der *Testsuite* und deren Laufzeit finden.

In der Praxis hat sich herausgestellt, dass der Durchschnittswert von 4,8 Sekunden pro Test von geringer Bedeutung ist. Bei der Implementierung eines Features lässt man meist nur die gerade aktuellen Testfälle laufen, und deren Laufzeit ist im einzelnen meist besser als der erwähnte Durchschnittswert. Mindestens vor jedem Deployment sollte dann die gesamte *Testsuite* ausgeführt werden, um sicher zu stellen, dass das System auch in der Gesamtheit korrekt funktioniert.

3.3.4 Integrationstests mit Cucumber

Cucumber [cuc] ist ein weiteres Framework für *Behaviour Driven Development* in Ruby. Es bietet die Möglichkeit, das Verhalten von Anwendungen in einer natürlichen Sprache, wie z.B. Englisch oder Deutsch, zu beschreiben. Auch hier war das Ziel, dass nicht-technische Personen das Verhalten einer Anwendung durch ein gemeinsam verwendetes Vokabular beschreiben und spezifizieren können. Die Spezifikationen werden dann von einem Entwickler erweitert, um daraus automatische Tests zu erstellen. In *Ruby on Rails* Projekten wird *Cucumber* üblicherweise benutzt um durch Integrationstests

sicherzustellen, dass das Zusammenspiel zwischen Browser und Controllern funktioniert.

Ein Feature wird in *Cucumber*'s externer DSL *Gherkin* [5] beschrieben, einer sogenannten *Business Readable, Domain Specific Language.* [6] Die Beschreibung kann in einer der über 30 unterstützen natürlichen Sprachen verfasst werden, wobei nur eine durch *Gherkin* definierte Struktur eingehalten werden muss. Als Beispiel für solch eine Spezifikation soll hier die Authentifizierung eines Benutzers durch die Web-Applikation dienen.

```
1  Feature: Authentication
2    To ensure the safety of the application
3    A user of the system
4    Must authenticate before using the application
5
6    Scenario: Login as an activated user with wrong credentials
7      Given I am an activated user
8       When I log in with invalid credentials
9       Then I should see "Sorry, invalid nick/password
10           combination given."
11       And I should not be logged in
```

Auflistung 3.4: Spezifikation eines Features mit *Cucumber*

Die Struktur der Beschreibung ist durch die Reihenfolge einiger Schlüsselwörter vorgegeben. Mit dem Schlüsselwort *Feature* wird die Spezifikation eines Features eingeleitet, gefolgt von einer kurzen Beschreibung zu Dokumentationszwecken. Ein Abschnitt, der mit dem Schlüsselwort *Scenario* definiert wird, beschreibt das Verhalten der Anwendung in einer bestimmten Situation. Mit dem Schlüsselwort *Given* beschreibt man, welche Vorbedingungen in der gegebenen Situation herrschen müssen. *When* beschreibt auszuführende Aktionen, wie z.B. das Ausfüllen eines Formulars oder das Klicken eines Links. Das Schlüsselwort *Then* beschreibt den erwarteten Zustand nach dem Ausführen der Aktionen. Mit *And* wird eine weitere Zeile des selben Typs der vorigen Zeile definiert, und dient zur Verknüpfung und der besseren Lesbarkeit.

Cucumber verarbeitet solch eine Spezifikation zeilenweise und führt für bestimmte Zeilen Code aus, der die beschriebenen Vorbedingungen herstellt, Aktionen ausführt und Erwartungen verifiziert. Für jede Zeile, die mit einem der Schlüsselwörter *Given*, *When*, *Then* oder *And* anfängt, muss ein auf sie passender regulärer Ausdruck definiert werden, der mit einem Codeblock assoziiert wird.

Der folgende Programmcode definiert einen regulären Ausdruck der auf die erste *Given* Zeile passt. Der mit ihm assoziierte Codeblock erstellt einen neuen und aktivierten Benutzer, speichert ihn in der Datenbank und weist

[5] `http://wiki.github.com/aslakhellesoy/cucumber/gherkin`

[6] `http://martinfowler.com/bliki/BusinessReadableDSL.html`

ihn einer Variablen zu, auf die in anderen Codeblöcken zugegriffen werden kann.

```
Given /^I am an activated user$/ do
  @current_user = User.make(:activated)
end
```

Cucumber stellt eine Bibliothek einiger dieser häufig verwendeten Konstrukte als Makros zu Verfügung. Diese können in eigenen Codeblöcken benutzt werden. Der folgende Code verwendet einige dieser vordefinierten Konstrukte. Der Code, der dabei ausgeführt wird, simuliert das Besuchen der *Login* Seite, das Ausfüllen zweier Textfelder mit dem Benutzernamen und einem falschen Passwort sowie das Klicken des *Login* Buttons mit einem Browser.

```
When /^I log in with invalid credentials$/ do
  Given "I am on the login page"
  When  "I fill in \"login\" with \"#{@current_user.nick}\""
  When  "I fill in \"password\" with \"invalid-password\""
  When  "I press \"Login\""
end
```

Die Zeile *Then I should see "Sorry, invalid nick/password combination given."* verwendet ebenfalls ein von *Cucumber* definiertes Makro, das überprüft ob der in Hochkomma gesetzte Text in der vom Server empfangenen HTML Seite zu finden ist.

Die letzte Zeile ist ein selbst definiertes Makro, das wiederum Methoden des *RSpec* Frameworks verwendet, um zu überprüfen dass der Benutzer nicht angemeldet ist.

```
Then /^I should not be logged in$/ do
  controller.send(:current_user).should be_nil
end
```

Cucumber bietet die Möglichkeit, eigene Makros interaktiv zu entwickeln. Für Zeilen, die keinem regulären Ausdruck zugeordnet werden können, generiert *Cucumber* auf sie passende Codeblöcke, die sehr einfach angepasst und weiterentwickelt werden können. Durch die Verwendung der eigenen und der vielen vordefinierten Makros lässt sich so ein auf die eigene Domäne zugeschnittenes Vokabular definieren, mit dem das korrekte Verhalten einer Anwendung auf einem hohem Abstraktionsniveau verifiziert werden kann.

Da *Cucumber* relativ neu ist, wurden im hier entwickelten Prototypen bisher nur 17 solcher Szenarien spezifiziert, die vor allem die korrekte Interaktion mit externen Diensten verifizieren. Beispielsweise werden von Benutzern publizierte Videos und Bilder von der Web-Applikation auf *Amazon's S3*[7] gespeichert. Die Verwendung dieses *Web Services* wurde aus Performanzgründen in den *RSpec* Unit Tests durch Mock-Objekte ersetzt. Integrationstests

[7] Amazon Simple Storage Service `http://aws.amazon.com/s3`

mit *Cucumber* bieten hier eine gute Möglichkeit, das Zusammenspiel von Browser und *Controller* in einer realitätsnahen Umgebung zu verifizieren.

KAPITEL 4

Aufbau und Analyse der ETL-Prozesse

4.1 Überblick über die verwendeten Datenquellen

Eine der Anforderungen an die Web-Applikation ist den Nutzern aktuelle Informationen über die Wetter- und die Wellenverhältnisse in den nächsten Tagen zu bieten. Hierfür werden die frei erhältlichen Ergebnisse von zwei numerischen Wettermodellen verwendet, dem *Global Forecast System* und dem *Wave Watch III* Modell. Bevor diese beiden Wettermodelle näher beschrieben werden und wie auf die Ergebnisse der Modellberechnungen zugegriffen werden kann, wird zunächst kurz erklärt, wie heutige Wettervorhersagen zustande kommen und welche Rolle dabei numerische Wettermodelle spielen.

4.1.1 Wettervorhersagen

Wettervorhersagen haben das Ziel, den Zustand der Erdatmosphäre zu einer bestimmten Zeit an einem bestimmten Ort zu prognostizieren. Sie werden meist von staatlichen aber auch privaten Wetterdiensten erstellt, die sich den Erkenntnissen der Meteorologie bedienen. Heutige Wettervorhersagen basieren auf den aufwändig berechneten Ergebnissen numerischer Wettermodelle.

Die Vorhersage des Wetters ist ein Anfangswertproblem, das meist in drei Schritten gelöst wird. Im ersten Schritt, der Analyse, wird der Ausgangszustand der Atmosphäre bestimmt. Dieser Zustand wird durch verschiedene physikalische Größen festgelegt, die von Wetterstationen, Satelliten, Bojen oder Flugzeugen gemessen werden. Einige typische Größen sind Luftdruck,

Temperatur, Wind, Wasserdampf, Wolken und Niederschlag.

Die Modellberechnung ist der zweite Schritt. Mit ihr wird die zukünftige Entwicklung der Atmosphäre in Form der physikalischen Größen simuliert. Die Berechnung dieser Simulation ist sehr aufwändig und wird deshalb mit Hilfe von Supercomputern durchgeführt. Ergebnis der Modellberechnung ist der Zustand der Atmosphäre zu verschiedenen Zeitpunkten in der Zukunft, dargestellt durch die physikalischen Größen.

Im letzten Schritt, der Nachbereitung, werden die Ergebnisse der Simulation schließlich für die verschiedensten Nutzer aufbereitet. Dies beinhaltet die Generierung von Wetterkarten und Strömungsfilmen für das Fernsehen, das Erstellen von Warnhinweisen für Technisches Hilfswerk und Feuerwehr oder die Interpretation und Analyse verschiedenster Wetterphänomene.

4.1.2 Nummerische Wettermodelle

Numerische Wettermodelle versuchen, den Zustand der Erdatmosphäre und deren Veränderung im Laufe der Zeit als mathematisches Problem zu beschreiben. Dabei werden die physikalischen Größen und Beziehungen, die den Zustand und die Veränderung der Atmosphäre beschreiben, als System partieller Differentialgleichungen modelliert. Die meisten Modelle verwenden dabei dieselben physikalischen Gesetzmäßigkeiten, die auf den Erhaltungssätzen von Energie, Impuls und Masse beruhen. Meist unterscheiden sie sich aber in der konkreten mathematischen Formulierung und der numerischen Lösung der Gleichungssysteme, weshalb die Ergebnisse verschiedener Modelle voneinander abweichen können.

Operative Wettermodelle

Der Deutsche Wetterdienst betreibt drei verschiedene Wettermodelle. Die lokalen *COSMO-DE* und *COSMO-EU* Modelle [1] liefern Vorhersagen für Deutschland und Europa, das globale Modell *GME* [2] Vorhersagen für die ganze Welt. Weitere bekannte Modelle sind das von der US-amerikanischen *National Oceanic and Atmospheric Administration (NOAA)* betriebene *Global Forecast System (GFS)* und das neuere *Weather Research and Forecasting (WRF)* Modell, das vom *National Weather Service (NWS)* , vom amerikanischen Militär und einigen privaten meteorologischen Organisationen verwendet wird.

[1] COSMO - Consortium for Small-Scale Modelling

[2] Global Model

Diskretisierung von Raum und Zeit

Um die sich verändernde Atmosphäre der Erde auf ein Modell abbilden zu können wird eine Diskretisierung von Raum und Zeit vorgenommen. Dabei wird die Oberfläche der Erde mit einem aus Drei- oder Vierecken bestehenden Gitternetz überzogen, und die Atmosphäre vertikal in mehrere Luftschichten aufgeteilt. Damit werden die möglichen Vorhersagepunkte in der Atmosphäre auf die endliche Zahl der so entstandenen Kreuzungspunkte reduziert. In Abbildung 4.1 ist das dreieckige Gitternetz des zur Zeit vom Deutschen Wetterdienst und des Max-Planck-Instituts für Meteorologie neu entwickelten *ICON* [3] Wettermodells zu sehen. Das *Global Forecast System* und das *Wave Watch III* Modell verwendet ein aus Vierecken bestehendes Gitternetz.

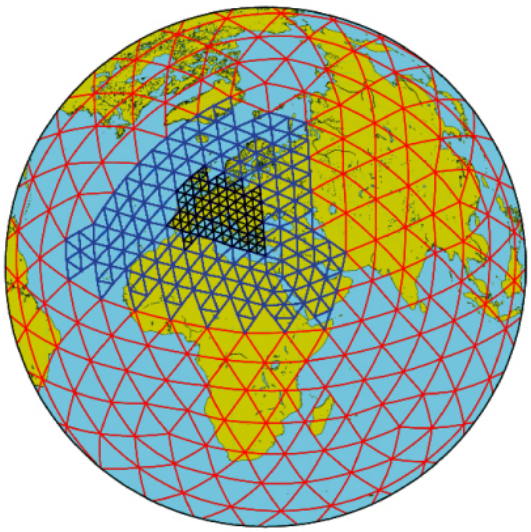


Abbildung 4.1: Dreiecksgitter des *ICON* Wettermodells (Quelle: Deutscher Wetterdienst)

Mit der Maschenweite bezeichnet man den Abstand zwischen zwei benachbarten Gitterpunkten. Je feiner das Gitter bzw. je höher die Auflösung des Modells ist, desto genauer können die Erdoberfläche und die darüber liegenden atmosphärischen Strukturen erfasst werden, was sich wiederum auf die Genauigkeit der Wettervorhersage auswirkt. Die benötigten Ressourcen zur Berechnung der Modellgleichungen steigt mit der Anzahl der verwendeten Gitterpunkte.

Da eine sehr hohe Auflösung selbst die Leistungsfähigkeit der schnellsten

[3] Icosahedral Non-hydrostatic General Circulation Model

Supercomputer übersteigt, werden von den Wetterdiensten meist verschiedene Modelle in unterschiedlichen Auflösungen berechnet. Globale Modelle, die den gesamten Globus umfassen, werden mit einer geringeren Auflösung berechnet als lokale Modelle, die nur einzelne Länder oder Kontinente abdecken. Je weiter aber in die Zukunft prognostiziert wird, desto mehr spielen aber wieder Wetterphänomene aus Gebieten, die nicht vom lokalen Modell abgedeckt werden eine Rolle. Für Vorhersagen ab 5 Tagen in die Zukunft benötigen die lokalen Modelle wiederum Informationen aus der gesamten Atmosphäre. Deshalb verwenden die höher auflösenden lokalen Modelle oft Informationen als Randwerte aus einem zuvor berechneten globalen Modell.

Die zeitliche Diskretisierung hingegen ist weniger problematisch. Die meisten Modelle bieten mindestens Prognosen um 12.00 und 24.00 Uhr für diejenigen Tage an, über die sich der Vorhersagezeitraum erstreckt. Das *Global Forecast System* und das *Wave Watch III* Modell bieten Vorhersagen im drei Stunden-Intervall an, das lokale *COSMO-DE* Modell sogar in einem Intervall von 25 Sekunden.

Rechenaufwand heutiger Modelle

In einer Präsentation [4] aus dem November 2008 wurde der Rechenaufwand für die vom Deutschen Wetterdienst betriebenen Modelle mit den dazugehörigen Kenngrößen veröffentlicht. Damals wurden die Wettervorhersagen auf einem IBM Power 5 System (p575) mit 52 Knoten, 416 Prozessoren und einer Spitzenleistung von 3,1 Teraflop/s berechnet.

- Für Deutschland wird das *COSMO-DE* Modell mit einer Maschenweite von 2,8 Kilometern betrieben und besteht aus ca. 10 Millionen Gitterpunkten. Die Berechnung dauert 30 Minuten und liefert Vorhersagen in einem 25 Sekunden Intervall für einen 21-stündigen Vorhersagezeitraum.

- Das Europa umfassende Modell *COSMO-EU* hat eine Maschenweite von 7 Kilometern, ca. 17 Millionen Gitterpunkten und wird mit einem Zeitintervall von 40 Sekunden erstellt. Die Berechnung einer 24-stündigen Vorhersage dauert 25 Minuten.

- Das globale, die gesamte Welt umfassende *GME* Modell hat eine Maschenweite von 40 Kilometern mit ca. 15 Millionen Gitterpunkten. Die Berechnung der 24-stündigen Vorhersage mit einem Zeitintervall von 133 Sekunden benötigt 15 Minuten.

[4] `http://www.initiative-wissenschaftsjournalismus.de/fileadmin/Downloads/WissensWerte2008/B3_Majewski.pdf`

Leider wurden während der Recherche keine genaueren Informationen gefunden, wie sich die Berechnung der hier erwähnten Modelle auf dem seit März 2009 beim Deutschen Wetterdienst in Betrieb genommenen Vektorsupercomputer SX-9 der Firma NEC verhält. Die Spitzenleistung dieses Systems beträgt momentan 4,5 Teraflop/s, die bis 2010 auf 11 Teraflop/s aufgestockt werden soll. Mit dieser neuen Anschaffung will der Deutsche Wetterdienst unter anderem auch Wettervorhersagen mit einer Auflösung von 2,8 Kilometern für Deutschlands Anrainerstaaten berechnen.

4.1.3 Geographische Breite und Länge

Zur Festlegung von Punkten auf der Erde wird meist ein Koordinatensystem genutzt, dessen Grundlage parallel zum Äquator verlaufende Breitenkreise und die beiden Pole verbindende Längenhalbkreise sind. Der am Äquator angesiedelte Breitenkreis teilt die Erde in die nördliche und die südliche Halbkugel. Sowohl auf dem nördlichen als auch auf dem südlichen Teil verlaufen jeweils weitere 90 Kreise, den Äquator mit eingeschlossen, also insgesamt 181 Breitenkreise. Die beiden Pole werden durch 360 nebeneinander liegende Längenhalbkreise verbunden, von denen der durch die Sternwarte in *Greenwich* verlaufende Halbkreis als Nullmeridian bezeichnet wird. Die Zählung fängt am Nullmeridian an, und geht jeweils um 180 Schritte in beide Richtungen. Nullpunkt des Koordinatensystems ist derjenige Punkt, an dem der Nullmeridian den Äquator kreuzt. In Abbildung 4.2 ist der durch *Greenwich* laufende Nullmeridian und der 30 ° nördlich verlaufende Breitenkreis zu sehen.

Die Position eines Punktes auf der Erde wird durch seine geographische Breite (*Latitude*) und Länge (*Longitude*) angegeben. Dies sind die in der Einheit Grad angegebenen, und vom Nullpunkt aus gesehenen Abstände der durch diesen Punkt verlaufenden Breiten- und Längenkreise. Die Position von *Greenwich* ist beispielsweise durch die Angabe *51.479 ° N, 0.0 ° E* festgelegt.

4.1.4 Angaben zur Modellauflösung

Das *Global Forecast System* und das *Wave Watch III* Modell beziehen sich ebenfalls auf dieses Koordinatensystem, und deren Gitterauflösung wird in der Einheit Grad angegeben. Da die meisten Menschen aber bei Distanzangaben in einer Längeneinheit denken an die sie gewöhnt sind und unter der sie sich auch etwas vorstellen können, stellt sich die Frage, wie viele Kilometer bzw. Meilen der Abstand zwischen zwei Knotenpunkten des Gitternetzes beträgt.

Aufgrund der elipsoiden Gestalt der Erdkugel kann hierzu jedoch keine allgemeingültige Aussage getroffen werden, da dies von der jeweils betrach-

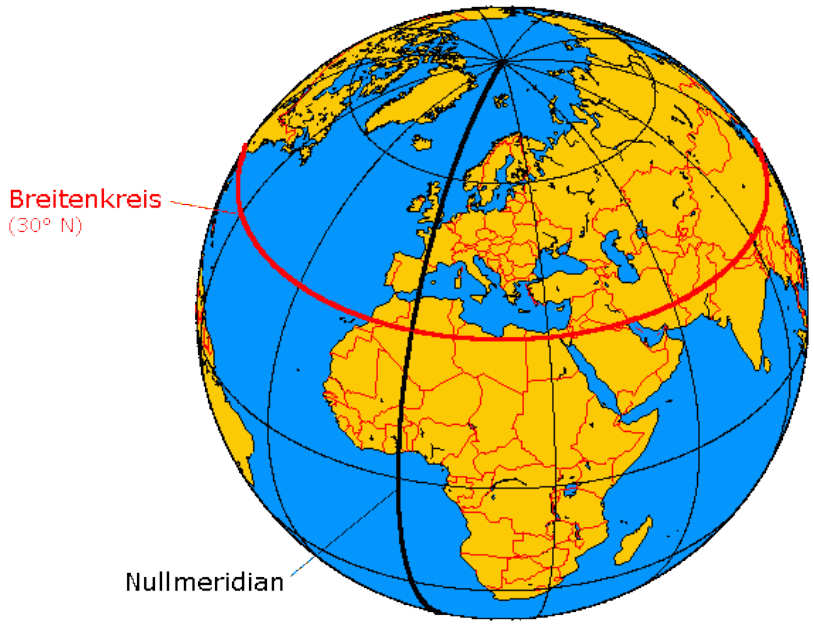


Abbildung 4.2: Die in Breiten- und Längenkreise unterteilte Erdkugel (Quelle: Wikipedia)

teten Region auf der Erdkugel und dem verwendeten Referenzellipsoiden abhängig ist. Ein Breitengrad (*Latitude*) entspricht durchschnittlich etwa 111 Kilometern und ist weitestgehend konstant. Der Abstand zwischen den Längengraden (*Longitude*) variiert allerdings erheblich. Am Äquator beträgt dieser Abstand ebenfalls ungefähr 111 Kilometer, konvergiert aber an den Polen in Richtung Null. Die hier teilweise in Kilometern angegebenen Gitterauflösungen sind demnach nur als grobe Richtwerte zur besseren Einschätzung der Gitterauflösung zu verstehen und beziehen sich auf die Regionen um den Äquator. Eine praktische Zusammenfassungen mit Formeln zur Berechnung von Problemen aus dem geographischen Bereich sind unter [Wil09] und [Ven09] zu finden.

4.1.5 Global Forecast System

Das *Global Forecast System* ist ein globales numerisches Wettermodell, das vom *National Weather Service* betrieben wird und den gesamten Erdball abdeckt. Da die Ergebnisse der Modellberechnung über das Internet [5] erhältlich sind und von jedem verwendet werden dürfen, erfreut es sich einer

[5] `http://nomad5.ncep.noaa.gov/pub/gfs`

großen Beliebtheit. Das Modell liefert Vorhersagen bis zu 384 Stunden (16 Tage) in die Zukunft und wird mit variierenden Auflösungen viermal täglich berechnet, jeweils um 6.00, 12.00 und 18.00 und 24.00 Uhr koordinierter Weltzeit (UTC). Die ersten 180 Stunden werden mit einer Maschenweite von ca. 40 Kilometern und einem Intervall von 3 Stunden berechnet, die restlichen Stunden im 12-Stunden-Intervall und einer Auflösung von ca. 80 Kilometern. Vertikal wird die Atmosphäre bei beiden Auflösungen in 64 unterschiedliche Luftschichten aufgeteilt.

Das Modell berechnet eine Vielzahl von physikalischen Größen, von denen hier hauptsächlich die Temperatur, die Gesamtbewölkung und das Niederschlagswasser von Bedeutung sind. Da weithin Einverständnis darüber herrscht, dass Vorhersagen über 180 Stunden hinaus sehr ungenau sind, verwenden die hier entwickelten *ETL* Prozesse nur die ersten 180 Stunden in der höchsten Auflösung.

4.1.6 Wave Watch III

Um für mehr Sicherheit auf hoher See und an Küstenregionen zu sorgen, betreibt der *National Weather Service* das *Wave Watch III* Wettermodell, um Wellen vorherzusagen. Es liefert ausschließlich Informationen über diejenigen Wellen, die durch den direkten Einfluss von Winden entstehen. Wellen, die durch andere Ereignisse wie z.B. Gewitter, Gezeiten oder Tsunamis verursacht werden, sind in diesem Modell nicht berücksichtigt. Da sich Wellen viel zu sehr voneinander unterscheiden, werden nicht Vorhersagen für einzelne Wellen getroffen, sondern über die Statistik von mehreren Wellen. Das Modell liefert sowohl Informationen über die Wellenhöhe, die Wellenperiode und die Wellenrichtung als auch über die Windstärke und die Windrichtung.

Das Modell wird wie das *Global Forecast System* viermal täglich neu berechnet und liefert Vorhersagen im 3-Stunden-Intervall für 180 Stunden in die Zukunft. Die Ergebnisse sind ebenfalls frei erhältlich [6]. Das globale Modell wird mit einer Maschenweite von ca. 80 Kilometern berechnet, die lokalen Modelle mit einer Maschenweite von bis zu 20 Kilometern. Die hier entwickelten *ETL* Prozesse verarbeiten bisher nur die Daten des globalen Modells. Die Integration der lokalen Modelle ist für eine der nächsten Versionen der Web-Applikation geplant, um insbesondere in den Küstenregionen bessere Ergebnisse anbieten zu können.

[6] `http://polar.ncep.noaa.gov/waves/index2.shtml`

4.2 Das Gridded Binary Datenformat

Die Abkürzung *Grib* steht für *GRIdded Binary* und ist ein bitorientiertes Datenformat zum Speichern und Übertragen von Wetterdaten. Das Format wurde von der *Kommission für Basissysteme* (*CBS*) der *Weltorganisation für Meteorologie* (*WMO*) standardisiert [7] und wird von vielen Wetterorganisationen dazu verwendet die Ergebnisse ihrer Modellberechnungen kompakt und plattformunabhängig zu speichern und auszutauschen. Insgesamt wurden drei verschiedene Versionen spezifiziert, von denen sich mittlerweile die Versionen 1 und 2 etabliert haben. Die mit der Nummer 0 bezeichnete Version gilt als veraltet und befindet sich bei den meisten Wetterorganisationen nicht mehr im operativen Einsatz.

4.2.1 Struktur von Grib-Dateien

Eine *Grib*-Datei besteht aus eigenständigen, sich selbst beschreibenden Datensätzen, den sogenannten *Grib* Nachrichten. Eine Nachricht enthält dabei alle Daten eines bestimmten Vorhersageelements, für eine auf ein Gitternetz diskretisierte geographische Region zu einem bestimmten Zeitpunkt oder Zeitraum. Beispielsweise die Temperaturen um 12.00 Uhr mittags der gesamten Erdkugel aus dem *Global Forecast System* oder die signifikanten Wellenhöhen des *Wave Watch III* Modells für einen Zeitraum von einer Woche für Europa.

Eine *Grib*-Nachricht besteht wiederum aus mehreren Sektionen, die den Inhalt einer Nachricht genauer beschreiben und in Tabelle 4.2 aufgelistet sind. Die Sektionen enthalten neben den eigentlichen Daten Informationen über die Dimension und Auflösung des verwendeten Gitternetzes, die Herkunft der Daten, die Art des verwendeten Komprimierungsverfahrens und die physikalische Einheit, in der die Daten gespeichert sind. *Grib*-Nachrichten können beliebig oft aneinander gereiht werden, was eine individuelle Komposition von unterschiedlichen Nachrichten in einer Datei erlaubt. Dies wird in Abschnitt 4.3.2 ausgenutzt, um nur ausgewählte Nachrichten aus einer größeren *Grib*-Datei zu beziehen.

4.2.2 Programme zum Verarbeiten von Grib-Dateien

Zum Verarbeiten und Lesen von *Grib*-Dateien werden spezielle Programme eingesetzt. Werkzeuge für die Kommandozeile, C-Bibliotheken und Program-

[7] `http://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/GRIB1-Contents.html`

Name der Sektion	Verwendungszweck
Indicator	Die Zeichenkette "GRIB", Versionsnummer, Länge der gesamten Nachricht
Identification	Charakteristiken, die auf alle Daten zutreffen, u.a. Herkunft der Daten, Referenzzeit und Typ des Produkts
Local Use (optional)	Abschnitt für beliebige zusätzliche Informationen
Grid Definition	Definition des Gitternetzes, u.a. die Dimension, die Anzahl der Datenpunkte und die verwendete Koordinatenprojektion
Product Definition	Beschreibung der Daten, Temperatur, Wellenhöhe, etc.
Data Representation	Beschreibung, wie die Daten repräsentiert werden. Art der Komprimierung
Bitmap	Eine Bitmap, welche die Anwesenheit bzw. Abwesenheit von Datenpunkten in der nächsten Sektion signalisiert
Data	Die komprimierten Daten. Für jeden in der Bitmap existierenden Gitterpunkt ein Wert.
End	Die Zeichenkette "7777" markiert das Ende der Nachricht

Tabelle 4.2: Die Sektionen einer *Grib*-Nachricht (Version 2)

me zur Visualisierung der *Grib*-Daten sind für verschiedene Plattformen und Programmiersprachen erhältlich. Eine Übersicht gängiger Software ist bei *Wikipedia* [8] zu finden. Zur Weiterverarbeitung sind insbesondere die Kommandozeilenprogramme `wgrib` und `degrib` zu empfehlen, da sie als typische *UNIX*-Filter konzipiert sind und in Kombination mit Standardwerkzeugen wie `cat`, `curl` oder `dd` flexibel eingesetzt werden können. Das Programm `degrib` bietet zudem die Möglichkeit, einen Index für eine *Grib*-Datei zu erstellen, mit dessen Hilfe ein wahlfreier Zugriff nach geographischen Positionen ermöglicht wird und die Zugriffszeiten erheblich beschleunigt.

4.2.3 Inventar einer Grib-Datei

Die beiden Kommandozeilenprogramme `degrib` und `wgrib` können dazu verwendet werden, Inventare von *Grib*-Datei zu erstellen. Ein Inventar ist eine Art Inhaltsverzeichnis und liefert Informationen über die in der Datei enthaltenen Nachrichten. Da *Grib*-Dateien oft mehrere Megabyte groß sind, stellen viele Wetterorganisationen aus praktischen Gründen zusätzlich die Inventare zur Verfügung. Ein Inventar ist zur Verarbeitung einer *Grib* Datei zwar nicht

[8] `http://en.wikipedia.org/wiki/GRIB#Applications`

zwingend erforderlich, vereinfacht aber den Umgang, da zur Inspektion nicht immer die kompletten *Grib*-Dateien übertragen werden müssen, sondern nur die sehr viel kleineren Inventare. In Abbildung 4.3 ist der Ausschnitt eines Inventars von einer *Grib*-Datei des *Global Forecast System* zu sehen. Pro Nachricht ist in diesem Inventar eine Zeile enthalten, die unter anderem Informationen über die Nummer der Nachricht, deren Position in Bytes und den Zeitpunkt als auch das Element der Vorhersage liefert.

```
1:0:d=2009081812:HGT:10 mb:anl:NAve=0
2:519924:d=2009081812:TMP:10 mb:anl:NAve=0
3:812418:d=2009081812:UGRD:10 mb:anl:NAve=0
...
568:211123530:d=2009081812:VWSH:-1500 pv units:anl:NAve=0
```

Abbildung 4.3: Inventar einer *Grib*-Datei des *Global Forecast System*

Die zweite Zeile aus Abbildung 4.3 gibt zum Beispiel Auskunft darüber, dass die Nachricht mit der Nummer 2 in der *Grib*-Datei an der Position 519.924 (Byte) zu finden ist, die Werte der Nachricht für den 18. August 2009 um 12.00 Uhr gelten und das Element die Temperatur (TMP) auf einer Höhe von 10 Hektopascal (10 mb) darstellt. Zudem handelt es sich bei den Werten nicht um eine Vorhersage (fcst), sondern um eine Analyse (anl) für die hier kein durchschnittlicher Wert (NAve=0) berechnet ist. Mit Analyse wird hier der Ausgangszustand der Atmosphäre zum Zeitpunkt der Modellberechnung bezeichnet und mit Vorhersage der simulierte Zustand in der Zukunft. Diese Inhaltsverzeichnisse werden in den *ETL*-Prozessen verwendet, um nur die benötigten Nachrichten einer *Grib*-Datei zu extrahieren.

4.3 Extraktion aus dem Quellsystem

Die Aufgabe der hier entwickelten Extraktionsprozesse besteht darin, die benötigten Daten des *Wave Watch III* Models und des *Global Forecast Systems* herunterzuladen und diese in einer einheitlichen Struktur für die Weiterverarbeitung im lokalen Dateisystem zu hinterlegen. Da insbesondere das Datenvolumen des *Global Forecast System* in seiner höchsten Auflösung sehr umfangreich ist (ca. 13 GB), wird ein Verfahren angewendet, um nur ausgewählte Daten zu beziehen. In diesem Abschnitt wird zuerst die Struktur des Quellsystems analysiert, dann das Verfahren zum Download der ausgewähl-

ten Daten vorgestellt und anschließend eine Aussage über die Performanz des Extraktionsprozesses getroffen.

4.3.1 Analyse der Quellsysteme

Sowohl das *Global Forecast System* als auch das *Wave Watch III* Modell werden viermal täglich, jeweils um , 6.00, 12.00, 18.00 und 24.00 Uhr koordinierter Weltzeit (UTC) berechnet. Danach werden die Ergebnisse in *Grib*-Dateien auf mehreren, teilweise von Unterorganisationen der *NOAA* betriebenen Servern veröffentlicht, die verschiedensten Anforderungen gerecht werden. Auf einigen Servern sind nur die aktuellen Ergebnisse verfügbar, andere wiederum dienen der Archivierung und bieten eine Historie über mehrere Jahre hinweg.

Die Ergebnisse der Modellberechnung werden in einem, meist nach Datum und Uhrzeit strukturiertem Dateisystem hinterlegt, das per *FTP* oder *HTTP* exportiert wird. Die Struktur und Benennung der Verzeichnisse und Dateien variiert dabei zwischen den Servern, ist aber innerhalb eines Servers konsistent und folgt einem vorhersehbaren Muster. Die *URI*s der benötigten Daten können so für einen bestimmten Server im voraus konstruiert werden und die Existenz der *Grib*-Daten mit einem der unterstützten Protokolle überprüft werden. Dieses Verfahren ist ein Beispiel zur Identifizierung von Ressourcen durch vorhersehbare *URI*s und ist einer der in Abschnitt 3.2.2 beschriebenen Vorschläge aus der *Ressource Oriented Architecture*.

Datenorganisation des Global Forecast System

Aktuelle Daten des *Global Forecast System* können in verschiedenen Auflösungen von den Servern des *National Oceanic and Atmospheric Administration Operational Model Archive and Distribution System (NOMADS)* [9] bezogen werden. In Tabelle 4.4 sind die Dateigrößen und Bezugsquellen der *Grib*-Dateien in den verschiedenen Auflösungen dargestellt. Die Dateigröße bezieht sich dabei immer auf eine einzelne *Grib*-Datei, die alle Vorhersageelemente des *Global Forecast System* für einen bestimmten Zeitpunkt in der Zukunft enthält.

Die in dieser Arbeit entwickelten *ETL*-Prozesse arbeiten alle mit den *Grib*-Dateien der höchsten Auflösung (0.5° x 0.5°). Unter der *URI* `http://nomad5.ncep.noaa.gov/pub/gfs_master/` sind Verzeichnisse zu finden, die nach dem Muster `gfs`***`YYYYMMDD`*** benannt sind. Dabei steht `YYYY` für das Jahr, `MM` für den Monat und `DD` für den Tag, an dem ein Modell berechnet wurde. Beispielsweise waren die *Grib*-Dateien aller Modellberechnungen, die

[9] `http://nomads.ncdc.noaa.gov`

Auflösung	Dateigrößen	URI der Bezugsquelle
2° x 5°	4 MB - 4.8 MB	`http://nomad5.ncep.noaa.gov/pub/gfs2p5`
1° x 1°	25 MB - 29 MB	`http://nomad5.ncep.noaa.gov/pub/gfs`
0.5° x 0.5°	200 MB - 215 MB	`http://nomad5.ncep.noaa.gov/pub/gfs_master`

Tabelle 4.4: Dateigrößen des *Global Forecast System*

am 16. August 2009 durchgeführt wurden, unter der *URI* `http://nomad5.ncep.noaa.gov/pub/gfs_master/gfs20090816/` aufgelistet. [10]

In den nach Tagen geordneten Verzeichnissen befinden sich *Grib*-Dateien, die nach dem Muster `gfs.tXXz.master.grbfYY` benannt sind. Die Zeichen `XX` stehen dabei für den Zeitpunkt der Modellberechnung (00, 06, 12 oder 18), und die Zeichen `YY` für die Stunde der Vorhersage in der Zukunft (00-180 im 3 Stunden Intervall). Die Vorhersagedaten des *GFS* für den 16. August 2009 um 18 Uhr abends, die am selben Tag um 6 Uhr morgens berechnet wurden, waren somit in der *Grib*-Datei mit dem *URI* `http://nomad5.ncep.noaa.gov/pub/gfs_master/gfs20090816/gfs.t06z.master.grbf18` zu finden.

Eine *Grib*-Datei des *Global Forecast System* enthält für einen bestimmten Zeitpunkt 63 verschiedene Vorhersageelemente und ist zwischen 200 und 215 Megabyte groß. Die Größe aller *Grib* Daten für einen Vorhersagezeitraum von 180 Stunden (und für den Berechnungszeitpunkt selbst) mit einem 3-stündigen Intervall beträgt somit ca. $(180h/3h + 1) * 215MB = 13.115MB$. Da hier aber nur sehr wenige Vorhersageelemente der *Grib*-Dateien benötigt werden, wird in Abschnitt 4.3.2 ein Verfahren vorgestellt, um nur die benötigten Elemente zu übertragen und somit das zu übertragende Datenvolumen zu reduzieren.

Datenorganisation des Wave Watch III Models

Die Daten des *Wave Watch III* Modells sind ähnlich strukturiert wie die des *Global Forecast System* und werden in einer Auflösung von 1.25° x 1° ebenfalls auf den *NOMADS* Servern veröffentlicht. Die *Grib*-Dateien werden in Verzeichnissen hinterlegt, die nach dem Muster `nww3YYYYMMDD` benannt und unter der *URI* `http://nomad5.ncep.noaa.gov/pub/waves/nww3` angeordnet sind. Auch hier ist das Datum der Modellberechnung im Verzeichnisnamen kodiert. Das *Wave Watch III* Modell hat im Vergleich zum *Global*

[10] Der Server `nomad5.ncep.noaa.gov` verwaltet keine historischen Daten, d.h. wenn dieses Dokument gelesen wird, sind höchstwahrscheinlich keine Daten mehr für die hier verwendeten Tage vorhanden.

Forecast System viel weniger Elemente, weshalb die kompletten Daten für eine 180-stündige Vorhersage in einer einzigen Datei gespeichert werden. Diese ist nach dem Muster `nww3.tXXz.grib` benannt, wobei `XX` hier ebenfalls für den Zeitpunkt der Modellberechnung (00, 06, 12 oder 18) steht. Die Ergebnisse des *Wave Watch III* Modells, das z.B. am 16. August 2009 um 18 Uhr berechnet wurde, konnten unter dem *URI* `http://nomad5.ncep.noaa.gov/pub/waves/nww3/nww320090816/nww3.t18z.grib` bezogen werden.

Eine *Grib*-Datei des *Wave Watch III* Models mit 11 verschiedenen Elementen ist bei einer Auflösung von 1.25° x 1° ca. 32 Megabyte groß und enthält Vorhersagedaten für 180 Stunden in die Zukunft.

4.3.2 Download einzelner Grib-Nachrichten

Die beiden Vorhersagemodelle bieten sehr viel mehr Informationen, als von der hier entwickelten Anwendung überhaupt benötigt werden. Vom *Global Forecast System* werden im Moment lediglich 4 der 63 verschiedenen Vorhersagelemente und vom *Wave Watch III* Modell 7 von 11 Elementen verwendet. Um nicht unnötig Bandbreite zu verschwenden und den Extraktionsprozess zu beschleunigen, wird hier ein Verfahren angewendet, das in dem Dokument *Fast Downloading of Grib Files* [11] beschrieben ist. Ziel ist es, nur die gewünschten Nachrichten einer *Grib*-Datei herunterzuladen. Voraussetzung dafür ist, dass die Dateien von einem Server bezogen werden, der das HTTP/1.1 Protokoll unterstützt und die Inhaltsverzeichnisse der Dateien zu Verfügung stehen. Das Verfahren besteht aus den folgende drei Schritten.

1. Download des Inhaltsverzeichnisses der entsprechenden Datei
2. Berechnung der Start- und Endpositionen aller relevanten Nachrichten
3. Download der entsprechenden Nachricht (*HTTP Range Header*)

Im ersten Schritt wird das Inhaltsverzeichnis der entsprechenden *Grib*-Datei heruntergeladen. Die Inhaltsverzeichnisse auf den *NOMADS* Servern sind mit dem Programm *wgrib* erstellt und deren *URI* lässt sich durch das Anhängen der Endung ".inv" an die *URI* der *Grib*-Datei konstruieren.

Anschließend wird im zweiten Schritt in Zweierpaaren über die Zeilen des Inhaltsverzeichnisses iteriert. Dabei wird die Startposition jeder Nachricht extrahiert und die dazugehörige Endposition berechnet. Die Startposition steht dabei an zweiter Stelle jeder Zeile und die Endposition berechnet sich aus der Startposition der nächsten Nachricht, von der ein Byte subtrahiert wird.

[11] `http://www.cpc.noaa.gov/products/wesley/fast_downloading_grib.html`

Ein Sonderfall ist die letzte Nachricht des Inhaltsverzeichnisses. Für diese Nachricht kann die Endposition nicht berechnet werden, da keine Information über die gesamte Länge der *Grib*-Datei im Inhaltsverzeichnis vorhanden ist. Tabelle 4.6 zeigt das Resultat dieser Berechnung, angewendet auf das Inhaltsverzeichnis aus Abbildung 4.3.

Nachricht	Startposition	Endposition	Referenzzeit	Element
1	0	519.923	2009-08-18 12:00	HGT
2	519.924	812.417	2009-08-18 12:00	TMP
3	812.418	...	2009-08-18 12:00	UGRD
...	...	...	...	...
568	211.123.530	-	2009-08-18 12:00	HGT

Tabelle 4.6: Berechnete Positionen aus einem Inhaltsverzeichnis

Im dritten Schritt werden schließlich nur die ausgewählten Nachrichten unter Verwendung des *HTTP/1.1* Protokolls heruntergeladen. Dabei wird pro Nachricht eine Anfrage an den Server gesendet, in der die Start- und Endposition im *Range* Feld des *HTTP* Headers eingetragen wird. Um beispielsweise nur die zweite Nachricht aus Tabelle 4.6 herunterzuladen, wird das *Range* Feld auf "bytes=519924-812417" gesetzt. Der im zweiten Schritt erwähnte Sonderfall, bei dem die Endposition für die letzte Nachricht nicht bekannt ist, wird dadurch abgedeckt, dass die Endposition im *Range* Header einfach weggelassen wird. Dies ist trotz der fehlenden Endposition eine gültige *Range* Angabe und veranlasst den Server dazu, den Rest der Datei ab der gegebenen Startposition zu senden. Zum Download der einzelnen *Grib*-Nachrichten wird das Programm *curl* [12] verwendet, da es das *HTTP/1.1* Protokoll unterstützt und man beliebige Felder im *HTTP* Header angeben kann.

4.3.3 Beschreibung der Extraktionsprozesse

Ziel der Extraktion ist es, alle relevanten *Grib*-Nachrichten der beiden Modelle in einer einheitlichen Struktur im lokalen Dateisystem zu hinterlegen. Pro Element wird dabei eine Datei erstellt, die alle Nachrichten des entsprechenden Elements für die verschiedenen Zeitpunkte der Vorhersage enthält. Beim Download der *Grib*-Dateien von den *NOMADS* Servern werden zwei

[12] `http://curl.haxx.se`

verschiedene Strategien angewendet, die im Folgenden beschrieben werden. Dies ist auf die unterschiedliche Datenorganisation der beiden Vorhersagemodelle auf der Serverseite zurückzuführen.

Die Performanz der Extraktionsprozesse hängt hauptsächlich von der Geschwindigkeit ab, mit der die *Grib*-Daten heruntergeladen werden. Um die durchschnittliche Übertragungsgeschwindigkeit über einen längeren Zeitraum zu ermitteln, wurden mehrere Messung durch einen *Cronjob* mit dem Kommandozeilenprogramm *ab* [13] durchgeführt, wobei jedes mal eine 32 MB große *Grib*-Datei heruntergeladen wurde. Die Messwerte wurden in einem Intervall von 6 Stunden über einen Zeitraum von einer Woche erhoben. Dieser Messintervall entspricht dem Intervall, in dem die beiden Modelle aktualisiert werden. Alle 28 Messungen ($24h/6h * 7$) sind auf einem dedizierten Server ausgeführt worden, der mit einer 100 Mbps Verbindung an das Internet angeschlossen ist. Das Ergebnis der Messung ergab eine eher mäßige durchschnittliche Übertragungsgeschwindigkeit von 166,22 KB/s mit einer Standardabweichung von 40,51 KB/s. Die schlechte Übertragungsgeschwindigkeit liegt vermutlich an der hohen Auslastung der *NOMADS* Server, auf die keinen Einfluss genommen werden kann.

Um eine Aussage über die Laufzeit der Extraktionsprozesse zu machen, wurden die Log-Dateien der beiden Prozesse ebenfalls über einen Zeitraum von einer Woche ausgewertet. Für jedes Element wurden die 28 Messwerte extrahiert und daraus die durchschnittliche Laufzeit pro Element berechnet. Die durchschnittliche Gesamtlaufzeit für den Extraktionsprozess eines Modells ergibt sich aus der Summe der durchschnittlichen Laufzeiten der einzelnen Elemente.

Downloadstrategie für das Wave Watch III Model

Alle Daten des *Wave Watch III* Modells sind auf Serverseite in einer *Grib*-Datei gespeichert. Der Extraktionsprozess für dieses Modell lädt zunächst das Inhaltsverzeichnis dieser Datei, um die Start- und Endpositionen der gewünschten Elemente zu berechnen. Anschließend wird pro Element eine *HTTP-Range* Anfrage an den Server gesendet und die in der Antwort enthaltene *Grib*-Nachricht in einer Datei im lokalen Dateisystem gespeichert. Nachdem die Datei gespeichert wurde, wird für den späteren Transformationsvorgang noch ein Index mit dem Programm *degrib* erstellt. Insgesamt werden dabei 8 Anfragen an den Server gesendet, eine für das Inhaltsverzeichnis und 7 für die *Grib*-Nachrichten der entsprechenden Elemente. In Tabelle 4.8 sind die durchschnittlichen Downloadzeiten für die einzelnen Elemente

[13] Apache HTTP Server Benchmarking Tool

Element	Beschreibung	Größe (Element)	Zeit (Element)
HTSGW	Wellenhöhe	2,5 MB	27 s
PERPW	Periode des Wellenkamms	2,7 MB	30 s
DIRPW	Richtung des Wellenkamms	3,5 MB	37 s
WVPER	Wellenperiode	2,5 MB	26 s
WVDIR	Wellenrichtung	3,5 MB	36 s
WIND	Windstärke	2,9 MB	34 s
WDIR	Windrichtung	3,8 MB	39 s
Gesamt:		21,4 MB	3 min 49 s

Tabelle 4.8: Durchschnittliche Downloadzeiten des *Wave Watch III* Modells

des *Wave Watch III* Modells und deren Gesamtlaufzeit aufgeführt. Der Extraktionsprozess dauert durchschnittlich ca. 4 Minuten und überträgt 21,4 MB an *Grib*-Daten.

Downloadstrategie für das Global Forecast System

Der Downloadvorgang für das *Global Forecast System* erfordert wesentlich mehr Anfragen als der des *Wave Watch III* Modells, da die Daten auf Serverseite in mehreren Dateien gespeichert sind. Für jeden Zeitpunkt der Vorhersage existiert auf dem Server eine Datei, in der die Daten aller Modellelemente für den entsprechenden Zeitpunkt enthalten sind. Dies sind bei einem Vorhersagezeitraum von 180 Stunden im 3-stündigen Intervall 61 Dateien, $180h/3h = 60$ für den Vorhersagezeitraum und eine weitere für den Zeitpunkt der Modellberechnung, der sogenannten Analyse.

Um alle *Grib*-Nachrichten für ein ausgewähltes Element herunterzuladen, müssen zunächst die 61 Inhaltsverzeichnisse aller *Grib*-Dateien angefordert werden, damit die Start- und Endpositionen des Elements berechnet werden können. Anschließend können die eigentlichen Daten des Elements mit *Http-Range*-Anfragen aus den 61 *Grib*-Dateien heruntergeladen werden. Die einzelnen *Grib*-Nachrichten aus den Dateien werden dann pro Element in einer Datei zusammengefasst. Dies kann Dank des selbstbeschreibenden Formats der einzelnen *Grib*-Nachrichten mit Standard-Unix-Kommandos bewerkstelligt werden, beispielsweise mit *cat TMP.00h.grib TMP.03h.grib ... TMP.180h.grib > TMP.0h-180h.grib*. Bis auf das Anfordern der Inhaltsverzeichnisse muss dieser Vorgang für alle gewünschten Elemente wiederholt werden. Insgesamt sind $61*(N+1)$ Anfragen nötig, 61 für die Inhaltsverzeichnisse

und pro Element weitere *61* Anfragen für die einzelnen *Grib*-Nachrichten. Der Extraktionsprozess für das *Global Forecast System* sendet somit für die zurzeit 4 verwendeten Elemente $61 * (4 + 1) = 305$ Anfragen. Tabelle 4.10 zeigt die durchschnittlichen Laufzeiten für die einzelnen Elemente und die durchschnittliche Gesamtlaufzeit. Der Extraktionsprozess für das *Global Forecast System* überträgt insgesamt 68,04 MB und benötigt durchschnittlich ca. 14 Minuten.

Element	**Beschreibung**	**Größe (Element)**	**Zeit (Element)**
TMP	Temperatur	18.90 MB	3 m 47 s
TCDC	Bewölkung	13.23 MB	2 m 56 s
PWAT	Niederschlag	18.90 MB	3 m 56 s
WEASD	Schnee	17.01 MB	3 m 34 s
Gesamt:		68,04 MB	14 m 13 s

Tabelle 4.10: Durchschnittliche Downloadzeiten des *Global Forecast System*

Lokales Grib Repository

Nachdem der Extraktionsvorgang erfolgreich abgeschlossen wurde, liegen die *Grib*-Daten beider Modelle in einem *Repository* auf dem lokalen Dateisystem. Das *Repository* ist nach dem Namen des Modells, dem Zeitpunkt, an dem das Modell erstellt wurde, und dem Element der Vorhersage organisiert. Diese in Tabelle 4.12 zu sehende Struktur bietet den anschließenden Transformationsprozessen einen einheitlichen Zugriff auf die Elemente der beiden Modelle.

Aktualisierung der Daten durch Polling

Zwar werden beide Modelle viermal täglich zu festen Zeitpunkten berechnet, wann genau die *Grib*-Dateien auf den *NOMADS* Servern zur Verfügung stehen variiert allerdings. Deshalb wird die Verfügbarkeit neuer Daten mittels *Polling* überprüft. Die durch *Cronjob* gesteuerten Prozesse überprüfen in einem bestimmten Intervall die Existenz der Quelldaten und starten den Extraktionsvorgang erst, wenn neue, noch nicht integrierte Daten zur Verfügung stehen.

Größe	Datum	Uhrzeit	Pfad im Repository
19 MB	2009-08-18	16:19	forecasts/gfs/20090818/t06z.PWAT.grib
14 MB	2009-08-18	16:13	forecasts/gfs/20090818/t06z.TCDC.grib
19 MB	2009-08-18	16:09	forecasts/gfs/20090818/t06z.TMP.grib
18 MB	2009-08-18	16:24	forecasts/gfs/20090818/t06z.WEASD.grib
3.5 MB	2009-08-18	15:58	forecasts/nww3/20090818/t06z.DIRPW.grib
2.4 MB	2009-08-18	15:57	forecasts/nww3/20090818/t06z.HTSGW.grib
2.8 MB	2009-08-18	15:57	forecasts/nww3/20090818/t06z.PERPW.grib
3.9 MB	2009-08-18	16:02	forecasts/nww3/20090818/t06z.WDIR.grib
3.0 MB	2009-08-18	16:01	forecasts/nww3/20090818/t06z.WIND.grib
3.5 MB	2009-08-18	16:00	forecasts/nww3/20090818/t06z.WVDIR.grib
2.5 MB	2009-08-18	15:59	forecasts/nww3/20090818/t06z.WVPER.grib

Tabelle 4.12: Verzeichnisstruktur und *Grib*-Daten im lokalen Repository

Beurteilung der Extraktionsprozesse

Trotz der eher mäßigen Übertragungsgeschwindigkeit kann die Performanz der beiden Extraktionsprozesse als unproblematisch eingestuft werden. Kritisch wird es, wenn die von den gesamten *ETL*-Prozessen beanspruchte Zeit sich dem 6-stündigen Intervall der Modellberechnung nähert. Die Downloadzeit von 18 Minuten ist für die zu übertragende Datenmenge zwar sehr schlecht, aber nicht weiter problematisch. Der Download der *Grib*-Daten beansprucht im Vergleich zu den anschließenden Transformations- und Ladeprozessen sehr wenig Ressourcen und wirkt sich nur negativ auf die Aktualität der Vorhersagen aus. Da gravierende Veränderungen in den Vorhersagen zwischen zwei Modellberechnungen eher selten sind und die Vorhersagen nicht in Echtzeit benötigt werden, kann diese Verzögerung in Kauf genommen werden.

4.3.4 Verbesserung der Extraktionsprozesse

Da es den Anschein macht, dass die Übertragungsgeschwindigkeit von den *NOMADS* Servern pro Verbindung reguliert wird, könnte versucht werden mehrere Anfragen gleichzeitig zu stellen, um die Daten parallel herunterzuladen und somit die Performanz zu verbessern. Wohin dies führt ist allerdings fraglich, da man davon ausgehen kann, dass die Anzahl der Verbindungen

von einer IP-Adresse ebenfalls beschränkt wird, falls schon die Übertragungsgeschwindigkeit gedrosselt wurde. Um die *NOMADS* Server nicht unnötig zu überlasten wurden Ansätze dieser Art nicht weiter verfolgt.

4.4 Transformation der Daten

Nachdem der Extraktionsvorgang erfolgreich abgeschlossen wurde und die Daten beider Modelle im lokalen *Repository* vorliegen, startet die Transformationsphase. In dieser Phase wird die bisher noch den gesamten Globus umfassende Datenmenge auf die in der Datenbank enthaltenen *Spots* reduziert. Alle relevanten Datensätze werden dabei in einer *CSV*-Datei gespeichert, die im nächsten Schritt mit dem *Bulk Loader* des Datenbankmanagementsystems in die Datenbank importiert wird.

4.4.1 Auslesen der Vorhersagedaten

Um die Vorhersagedaten aus einer *Grib*-Datei auszulesen, wird ein sogenannter *Grib Reader* benötigt. Hierfür wird das Kommandozeilenprogramm *degrib* verwendet, da mit ihm ein schneller und wahlfreier Zugriff nach geographischen Positionen auf die Vorhersagedaten einer *Grib*-Datei möglich ist. Vorhersagewerte für geographische Positionen, die nicht genau auf einem der Gitterpunkte des Modells liegen, werden von *degrib* entweder durch *bilineare* oder *nearest-neighbor* Interpolation berechnet. Durch das Erstellen eines Index kann der wahlfreie Zugriff auf die Vorhersagedaten einer *Grib*-Datei beschleunigt werden.

```
1  $ degrib gfs/20090904/t06z.TMP.grib -P -pnt 52.523,13.411 -Unit m
2  element, unit, refTime, validTime, (52.523000,13.411000)
3  TMP, [C], 200909040600, 200909040600, 14.650
4  TMP, [C], 200909040600, 200909040900, 18.250
5  TMP, [C], 200909040600, 200909041200, 21.050
6  TMP, [C], 200909040600, 200909041500, 20.750
7  TMP, [C], 200909040600, 200909041800, 17.150
8  TMP, [C], 200909040600, 200909042100, 13.950
9  ...
10 TMP, [C], 200909040600, 200909110000, 16.050
11 TMP, [C], 200909040600, 200909110300, 15.150
12 TMP, [C], 200909040600, 200909110600, 14.250
13 TMP, [C], 200909040600, 200909110900, 17.550
14 TMP, [C], 200909040600, 200909111200, 20.650
15 TMP, [C], 200909040600, 200909111500, 20.550
16 TMP, [C], 200909040600, 200909111800, 16.750
```

Auflistung 4.1: Auslesen der Temperatur in Berlin mit *degrib*

Die Temperatur in Berlin *(52.523, 13.411)* kann mit dem Befehl in Zeile 1 aus Auflistung 4.1 ermittelt werden. Die hier verwendete *Grib*-Datei stammt

aus der Modellberechnung des *Global Forecast System* vom 04. September 2009 um 6.00 Uhr und bietet Vorhersagewerte im 3-stündigen Intervall für 180 Stunden in die Zukunft. Werden Vorhersagedaten an einer geographischen Position ausgelesen, an der keine Daten [14] vorhanden sind, liefert *degrib* Datensätze mit dem als ungültig definierten Wert *9999.0*.

4.4.2 Alternative Vorhersageposition

Bevor der eigentliche Transformationsvorgang beschrieben wird, muss zuvor noch auf eine Eigenart des *Wave Watch III* Modells eingegangen werden. Bei einigen Spots wird das Auslesen der Wellenvorhersagen durch die zu grobe Gitterauflösung des Modells und daraus resultierenden Datenlücken erschwert. Mit einem zusätzlichen Berechnungsschritt, der für jeden Spot einmalig durchzuführen ist, kann dieses Problem jedoch für die meisten Spots zufriedenstellend gelöst werden. In diesem Abschnitt wird zunächst näher auf diese Datenlücken eingegangen und anschließend ein prototypischer Algorithmus zur Lösung des Problems vorgestellt.

Datenlücken im Wave Watch III Modell

Der Fokus des *Wave Watch III* Modells liegt auf der Vorhersage von Wellen und den damit verbundenen physikalischen Größen. Im Ergebnis der Modellberechnung sind deshalb nur Vorhersagedaten für geographische Positionen enthalten, die sich auch über dem Meer befinden. Für alle anderen Positionen stehen keine Daten zur Verfügung. Wegen der groben Gitterauflösung des *Wave Watch III* Modells verläuft die Grenze zwischen Land und Meer jedoch nicht wirklichkeitsgetreu, sondern in einer Zick-Zack-Linie, die sich am rechteckigen Gitternetz des Modells orientiert. In Abbildung 4.4 ist die Küstenregion um den spanischen Ort Mundaka mit dem darüber liegenden Gitternetz zu sehen. Die Abbildung soll verdeutlichen, dass nur an den grün eingefärbten Knotenpunkten des Gitters Vorhersagedaten existieren. An allen anderen Stellen sind keine Daten vorhanden, da diese Positionen im Modell als Land betrachtet werden. Deshalb würde man beim Auslesen der Vorhersagedaten des *Wave Watch III* Modells an der geographischen Position des Spots Mundaka nur ungültige Werte erhalten.

Vorhersagedaten aus der näheren Umgebung

Da sich die meisten Spots aber genau in solchen Küstenregionen befinden, muss eine zufriedenstellende Alternative gefunden werden, um für diese Spots

[14] Beispielsweise die Wellenhöhe in Berlin aus dem *Wave Watch III* Modell

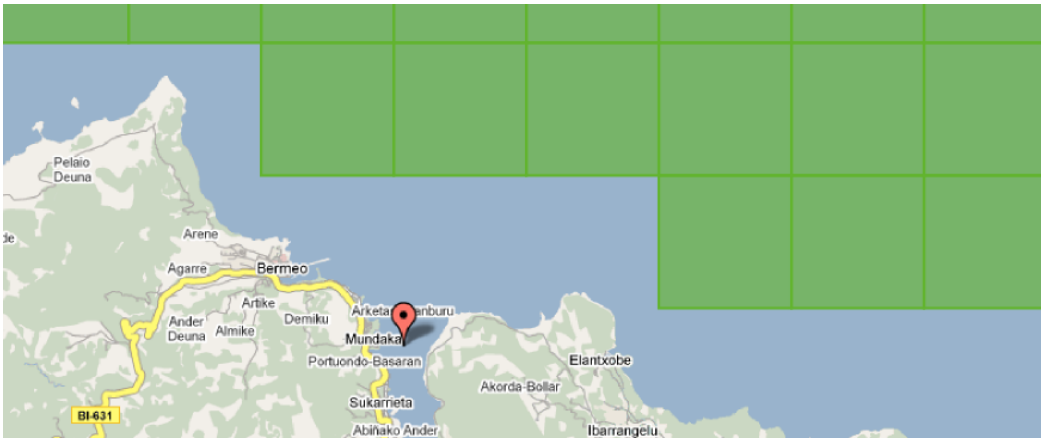


Abbildung 4.4: Visualisierung der Datenlücken im *Wave Watch III* Modell

trotzdem Vorhersagedaten anbieten zu können. Wie in der Einleitung erwähnt, werden die an den Spots brechenden Wellen durch den in weit entfernteren Regionen entstandenen und viele Kilometer weit gereisten Swell beeinflusst. Zieht man lokale Gegebenheiten, wie z.B. vorgelagerte Inseln, Hafenbecken oder abgeschirmte Buchten nicht in Betracht, dann sind die für Surfer wichtigen Eigenschaften eines großräumig eintreffenden Swells in der Umgebung eines Spots sehr ähnlich. Deshalb werden in der hier entwickelten Applikation Vorhersagedaten aus der näheren Umgebung eines Spots herangezogen, falls an der geographischen Position des Spots selbst keine Daten vorhanden sind. Das zu lösende Problem besteht darin, eine alternative Vorhersageposition für diejenigen Spots zu finden, an deren geographischer Position selbst keine Daten vorhanden sind.

Bestimmung der Vorhersageposition durch den Benutzer

Eine Möglichkeit dieses Problem zu Lösen, wäre den Benutzer beim Erstellen eines neuen Spots die Vorhersageposition selbst angeben zu lassen. Auf einer interaktiven Karte könnte man den Spot mit dem darüber liegenden Gitternetz des *Wave Watch III* Modells einzeichnen und die gültigen Positionen bzw. Gebiete ähnlich wie in Abbildung 4.4 einfärben. Mit einem Klick auf die Karte kann der Benutzer dann eine alternative Vorhersageposition auswählen. Wurde eine Position selektiert, für die keine Vorhersagedaten vorhanden sind, müsste dies dem Anwender unmittelbar signalisiert werden. Diese Idee wurde hier aber wegen ihrer Benutzerunfreundlichkeit nicht weiter verfolgt. Der Nutzer müsste zunächst über Sinn und Zweck der alternativen Vorhersageposition aufgeklärt werden. Dies verständlich zu kommunizieren und in

einer intuitiven Benutzeroberfläche darzustellen, wird als eines der Hauptprobleme bei diesem Ansatz angesehen. Benutzer der Plattform mit internen Details der Web-Applikation zu konfrontieren, die zudem nicht unbedingt sofort verständlich sind, wird als nicht zumutbar eingestuft. Eine bessere und zudem automatisierte Alternative, wäre einen Algorithmus zu entwickeln, der für einen gegebenen Spot eine alternative Vorhersageposition in dessen näherer Umgebung findet.

Algorithmus zur Bestimmung der Vorhersageposition

Der hier entwickelte Algorithmus sucht im näheren Umkreis eines Spots nach einer alternativen Vorhersageposition. Zur Ermittlung dieser Position wird eine *Grib*-Datei verwendet, in der die Wellenhöhen des *Wave Watch III* Modells gespeichert sind. Da die Entwicklung von speziellen Routinen zum Auslesen einer *Grib*-Datei und zur effizienteren Suche einer alternativen Vorhersageposition zu viel Zeit in Anspruch genommen hätte, wurde die *Grib*-Datei hier als *Black Box* betrachtet, auf die nur mit den zur Verfügung stehenden Werkzeugen zugegriffen werden kann. Zunächst wird das allgemeine Vorgehen des Algorithmus in einer vereinfachten Variante beschrieben, die anschließend durch Anwendung des *Divide and Conquer* Prinzips erweitert wird und gewisse Parallelen zu einer binären Suche aufweist. Ziel des Algorithmus ist es, diejenige geographische Position zu finden, die am nächsten an der Ursprungsposition des Spots liegt und für die gültige Vorhersagedaten existieren. Eingabe für den Algorithmus ist die geographische Position eines Spots, Ausgabe die gefundene Vorhersageposition. Eine erste Variante des Algorithmus arbeitet wie folgt.

1. Bei der Initialisierung des Algorithmus wird die maximale Distanz Δ festgelegt, die eine alternative Vorhersageposition von der Ursprungsposition des Spots entfernt sein darf. Weiterhin eine Schrittweite $\delta < \Delta$, um die der Radius ρ vom Ursprung aus pro Iteration erhöht wird und eine Schrittweite $\alpha < 360°$, mit der die Richtungen der zu untersuchenden Positionen berechnet werden.

2. Zunächst werden die Wellenhöhen an der Ursprungsposition des Spots mit dem Programm *degrib* ausgelesen. Falls der Vorhersagewert an dieser Position gültig ist (nicht 9999.0), ist der Algorithmus fertig und die ursprüngliche Position des Spots kann als Vorhersageposition verwendet werden.

3. Wurden an der/den bisher betrachteten Position(en) keine gültigen Werte gefunden, wird der Radius ρ um die Schrittweite δ erhöht. Falls

der Radius ρ den Maximalwert Δ überschritten hat, ist der Algorithmus fertig und es konnte keine alternative Vorhersageposition gefunden werden.

4. Nach der Erhöhung des Radius werden bestimmte geographische Positionen, die auf dem Umkreis des Spots liegen, auf Existenz gültiger Vorhersagewerte hin überprüft. Dabei werden im Uhrzeigersinn [15] nur diejenigen Positionen betrachtet, deren vom Ursprung aus in Grad angegebene Richtung durch die Schrittweise α teilbar ist. Sobald an einer dieser Positionen ein gültiger Wert gefunden wurde, kann diese als Vorhersageposition verwendet werden, und der Algorithmus ist fertig. Falls nicht, wird mit Schritt 3 weitergemacht.

In dieser und der folgenden Variante des Algorithmus werden bestimmte Positionen, die auf dem Umkreis des Radius liegen, auf Existenz gültiger Vorhersagedaten hin überprüft. Um die geographischen Koordinaten der alternativen Vorhersageposition (lat_2, lon_2) aus der Ursprungsposition (lat_1, lon_1), dem Radius ρ und der Richtung θ zu berechnen, werden die folgenden zwei Formeln [16] verwendet, wobei der Erdradius mit $R = 6371km$ festgelegt ist.

$$lat_2 = asin(sin(lat_1) * cos(\rho/R) + cos(lat_1) * sin(\rho/R) * cos(\theta))$$
$$lon_2 = lon_1 + atan2(sin(\theta) * sin(\rho/R) * cos(lat_1), cos(\rho/R) - sin(lat_1) * sin(lat_2))$$

Bei der Überprüfung einer Position auf gültige Werte wird das Programm *degrib* mit den entsprechenden Parametern aufgerufen, dessen Ausgabe geparst und auf Gültigkeit hin überprüft. Eine alternative Vorhersageposition wurde genau dann gefunden, wenn der Wert in der Ausgabe von *degrib* gültig (nicht 9999.0) ist. Die Anzahl der Aufrufe des Programms *degrib* ist mit den bei der Initialisierung festgelegten Parametern durch die Obergrenze $1 + (360°/\alpha) * (\Delta/\delta)$ beschränkt. Da die Wahl der Parameter die Laufzeit und die Genauigkeit des Algorithmus erheblich beeinflusst, wurde der Algorithmus auf einige Spots angewendet und die Ergebnisse untersucht.

Beobachtungen bei der Anwendung des Algorithmus

Der Algorithmus wurde auf 78 Spots angewendet, von denen zwei in Nord-Amerika, 4 in Indonesien und die restlichen in Europa liegen. Als Abbruch-

[15] In welche Reihenfolge die Positionen auf dem Umkreis am besten überprüft werden ist allerdings fraglich. Mögliche Alternativen wären gegen den Uhrzeigersinn, gegenüberliegend oder randomisiert.

[16] `http://www.movable-type.co.uk/scripts/latlong.html#destPoint`

bedingung wurde die maximale Distanz Δ bei der Initialisierung des Algorithmus auf 100 Kilometer gesetzt. Für die Schrittweite der Richtung wurde $\alpha = 10°$ gewählt und der Radius bei jeder Iteration um einen Kilometer erhöht. Bei 31 Spots wurden an der Ursprungsposition gültige Vorhersagewerte gefunden und der Algorithmus konnte bereits nach einem Aufruf des Programms *degrib* terminieren. Für zwei Spots in Italien konnten keine alternativen Vorhersageposition gefunden werden, da das *Wave Watch III* Modell für das Mittelmeer keine Vorhersagedaten enthält. Für diese beiden Spots wurde das Programm *degrib* vom Algorithmus $1+(360°/10°)*(100km/1km) = 3601$ mal ohne Erfolg aufgerufen. Für alle anderen Spots konnte mit dem Algorithmus eine alternative Vorhersageposition ermitteln werden, die zwischen 3 und 80 Kilometern von der Ursprungsposition entfernt war. In Abbildung 4.5 ist die Ursprungsposition des portugiesischen Spot *Esposende* mit einer roten Markierung und die vom Algorithmus 3,6 Kilometer weiter südlich gefundene alternative Vorhersageposition mit einer grünen Markierung gekennzeichnet.

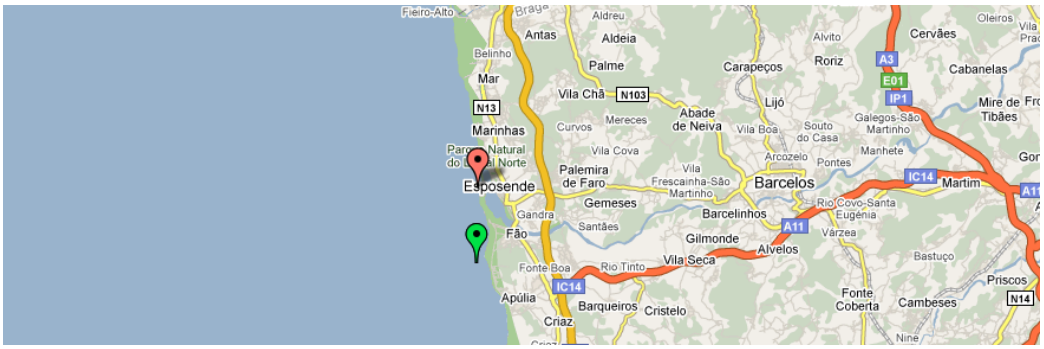


Abbildung 4.5: Alternative Vorhersageposition für Esposende, Portugal

Verbesserung des Algorithmus

Damit der Algorithmus für einen Spot auch wirklich die nächste alternative Vorhersageposition findet, muss die Schrittweite δ, um die der Radius pro Iteration erhöht wird, sehr klein gewählt werden. Ist diese Schrittweite zu groß, kann es vorkommen dass eine in Wirklichkeit viel näher am Ursprung liegende alternative Vorhersageposition nicht gefunden wird, da sie bei einer der Iterationen übersprungen wurde. Wählt man die Schrittweite zu klein, führt dies zu sehr vielen und teuren Aufrufen des Programms *degrib* und einer schlechten Laufzeit des Algorithmus. Dasselbe gilt für die Wahl der Schrittweite α, mit der die verschiedenen Richtungen um die Ursprungsposition herum berechnet werden. Um im Durchschnitt die Anzahl der *degrib*

Programmaufrufe zu reduzieren, wurde hier ein Verfahren angewendet, das dem einer binären Suche ähnelt und als *Divide and Conquer* Prinzip bekannt ist. Diese Variante des Algorithmus arbeitet folgendermaßen:

1. Bei der Initialisierung wird wieder die maximale Distanz Δ festgelegt, die eine alternative Vorhersageposition von der Ursprungsposition eines Spots entfernt sein darf, und die Schrittweite $\alpha < 360°$, mit der die Richtungen der zu untersuchenden Positionen berechnet werden. Außerdem wird ein Schwellenwert d benötigt, der als Abbruchkriterium dient und im 3. Schritt näher erklärt wird.

2. Da bei vielen Spots bereits an der Ursprungsposition Vorhersagedaten vorhanden sind, wird zunächst diese Position auf Gültigkeit überprüft. Wurde dort ein gültiger Vorhersagewert gefunden, kann die Ursprungsposition als Vorhersageposition verwendet werden und der Algorithmus ist fertig.

3. Falls nicht, wird in den nächsten beiden Schritten die alternative Vorhersageposition durch Anwendung des *Divide and Conquer* Prinzips gesucht. Hierzu wird eine untere Schranke mit $l = 0$ und eine obere Schranke mit $u = \Delta$ festgelegt. Der Abstand der unteren und oberen Schranke wird in den folgenden Iterationen halbiert, bis dieser kleiner ist als der zuvor angegebene Schwellenwert d.

4. Mit dem Programm *degrib* werden bestimmte Positionen auf gültige Vorhersagewerte hin überprüft, die auf dem Umkreis liegen, der durch den Radius $l+(u-l)/2$ definiert ist. Dabei werden wieder nur diejenigen Positionen betrachtet, deren in Grad angegebene Richtung durch die Schrittweise α teilbar ist.

   (a) Wurde auf dem Umkreis eine Position mit einem gültigen Vorhersagewert gefunden, wird diese als alternative Vorhersageposition vorgemerkt und die obere Schranke auf $u = u - (u - l)/2$ gesetzt.

   (b) Wurde auf dem Umkreis keine Position mit gültigen Vorhersagewerten gefunden, wird die untere Schranke auf $l = l + (u - l)/2$ gesetzt.

5. Solange $u - l > d$ ist, wird mit Schritt 4 weitergemacht. Ansonsten ist der Algorithmus fertig und die vorgemerkte Position kann als alternative Vorhersageposition verwendet werden. Wurde keine Position vorgemerkt, konnte der Algorithmus keine alternative Vorhersageposition finden und terminiert ohne Erfolg.

In Auflistung 4.2 ist die Ausgabe des Algorithmus für den Spot Esposende in Portugal zu sehen. Der Algorithmus wurde mit einer maximalen Distanz Δ von 100 Kilometern, einer Richtungsschrittweite α von 10° und einem Schwellenwert d von einem Meter initialisiert. Da an der Ursprungsposition keine gültigen Vorhersagewerte gefunden wurden, sucht der Algorithmus zunächst nach gültigen Position auf dem Umkreis mit einem Radius von 50 Kilometern. In Zeile 2 kann man erkennen, dass auf diesem Umkreis eine gültige Position gefunden wurde. Daher müssen Positionen, die mehr als 50 Kilometer von der Ursprungsposition entfernt sind, nicht weiter untersucht werden. In den folgenden Iterationen werden weitere Positionen untersucht und die untere oder obere Schranke entsprechend angepasst. Nachdem der Schwellenwert unterschritten wurde, terminiert der Algorithmus nach ca. 8 Sekunden mit einer alternativen Vorhersageposition, die 3,6 Kilometern weiter südlich liegt.

```
1  Locating wave watch position for Esposende, Portugal:
2  - Sample: Dist: 50.00 km, Deg: 100, Dest: 41.452404, -8.203167, Val: 1.20
3  - Sample: Dist: 25.00 km, Deg: 100, Dest: 41.492580, -8.498402, Val: 1.20
4  - Sample: Dist: 12.50 km, Deg: 110, Dest: 41.493466, -8.652970, Val: 1.20
5  - Sample: Dist:  6.25 km, Deg: 130, Dest: 41.495856, -8.736514, Val: 1.20
6  - Sample: Dist:  4.69 km, Deg: 150, Dest: 41.495485, -8.765856, Val: 1.20
7  - Sample: Dist:  3.91 km, Deg: 160, Dest: 41.498981, -8.777955, Val: 1.20
8  - Sample: Dist:  3.71 km, Deg: 170, Dest: 41.499124, -8.786260, Val: 1.20
9  - Sample: Dist:  3.61 km, Deg: 180, Dest: 41.499499, -8.794000, Val: 1.20
10 - Position: 41.499, -8.794, Dist: 3.61727461421652
11 - Done. (8.48 s)
```

Auflistung 4.2: Ausgabe bei der Suche einer alternativen Vorhersageposition

4.4.3 Transformation der Vorhersagedaten

Nachdem die alternativen Vorhersagepositionen für das *Wave Watch III* Modell ermittelt wurden, können die Vorhersagedaten der Spots mit dem Programm *degrib* ausgelesen und in das *CSV*-Format umgewandelt werden. Die Vorhersagedaten des *Wave Watch III* Modells werden dabei an der alternativen Vorhersageposition, die Daten des *Global Forecast System* an der Ursprungsposition eines Spots ausgelesen. Hierzu wird das Programm *degrib* mit den entsprechenden Parametern pro Spot n-mal aufgerufen, wobei n für die Anzahl der Vorhersageelemente steht. Die übergebenen Parameter sind der Name der *Grib*-Datei und die geographische Position, an der die Daten gelesen werden.

Die Ausgabe dieser Programmaufrufe entspricht der aus Auflistung 4.1 bekannten Struktur. Bis auf einige Feinheiten ist diese Ausgabe schon fast in dem gewünschten *CSV*-Format. Damit die in der Ausgabe enthaltenen Zeitangaben vom *Bulk* Loader des Datenbankmanagementsystems interpre-

tiert werden können, müssen diese noch in ein durch *ISO 8601* standardisiertes Datums- und Zeitformat konvertiert werden. Zudem wird pro Datensatz noch der Primärschlüssel des entsprechenden Spots hinzugefügt, damit die Vorhersagedaten im späteren Ladevorgang eindeutig mit einem Spot in Verbindung gebracht werden können. Das Ergebnis dieser Transformation ist in Auflistung 4.3 zu sehen.

```
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-18 06:00:00 0.65
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-18 09:00:00 0.67
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-18 12:00:00 0.7
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-18 15:00:00 0.76
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-18 18:00:00 1.08
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-18 21:00:00 1.0
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-19 00:00:00 0.96
24  43.500  -2.691  HTSGW m 2009-08-18 06:00:00 2009-08-19 03:00:00 0.9
...
```

Auflistung 4.3: Vorhersagedaten aus Mundaka im CSV-Format

Der hier beschriebene Vorgang wird für alle Spots ausgeführt und die Ausgabe der Transformation durch Anhängen in eine Datei umgeleitet. Die so entstandene *CSV*-Datei enthält schließlich alle relevanten Vorhersagedaten in einem Format, das im nächsten Schritt vom *Bulk Loader* des Datenbankmanagementsystem geladen werden kann. Die *Grib*-Dateien werden nach diesem Schritt nicht weiter benötigt und können gelöscht werden.

Element(e)	1 Spot	10 Spots	100 Sp.	1.000 Sp.	10.000 Sp.	100.000 Sp.
1	17 ms	170 ms	1,7 s	17 s	2 m 50 s	28 m 20 s
10	170 ms	1,7 s	17 s	2 m 50 s	28 m 20 s	4 h 43 m
11	**187 ms**	**1,87 s**	**18,7 s**	**3 m 7 s**	**31 m 10 s**	**5 h 12 m**
15	255 ms	2,55 s	25,5 s	4 m 15 s	42 m 30 s	7 h 5 min

Tabelle 4.14: Hochrechnung der Laufzeit des Transformationsvorgangs

Das Auslesen der Vorhersagedaten dauert für ein Element durchschnittlich 17 Millisekunden mit einer Standardabweichung von 4 Millisekunden. Diese Werte wurden ermittelt, indem für 78 Spots alle 11 Vorhersageelemente ausgelesen wurden und die Laufzeit für jedes Vorhersageelement protokolliert wurde. Das arithmetische Mittel und die Standardabweichung wurden also aus insgesamt 858 Messwerten berechnet. Die Messung wurde auf einem dedizierten Server [17] durchgeführt, der mit einem Intel Core 2 Duo (2x 2.66+ GHz) Prozessor und 4 GB Arbeitsspeicher ausgestattet ist. In Tabelle 4.14 ist

[17] `http://www.ovh.de/produkte/superplan_best_of.xml`

eine Hochrechnung dieser durchschnittlichen Laufzeit für mehrere Elemente und Spots zu sehen. Wie viele Spots weltweit existieren bzw. mit welcher Anzahl an zu verarbeitenden Spots zu rechnen ist, kann an dieser Stelle nur geschätzt werden. Auf einer ähnlichen Webseite [18] waren Mitte Oktober 2009 ungefähr 8.300 Spots weltweit gelistet. Das Auslesen der Vorhersagedaten für 11 Elemente würde für diese 8.300 Spots somit ca. 26 Minuten ($187ms * 8.300$) dauern.

4.4.4 Verbesserung des Transformationsvorgangs

Sowohl hinsichtlich der Geschwindigkeit als auch der Korrektheit ist das Auffinden der alternativen Vorhersageposition nicht optimal. Das eigentlich zur Extraktion von Daten konzipierte Programm *degrib* wird hier "missbraucht", um eine Position zu finden, an der Vorhersagedaten zur Verfügung stehen. Hierfür wird das Programm mehrmals aufgerufen, um an verschiedenen Positionen Vorhersagedaten zu extrahieren und diese auf Gültigkeit zu überprüfen. Diese vielen Programmaufrufe könnten durch die Entwicklung geeigneter Routinen, die direkt auf dem *Grib* Format arbeiten, vermieden werden und somit die Geschwindigkeit zum Auffinden der Vorhersageposition erhöht werden. Da diese Position für jeden Spot allerdings nur einmalig ermittelt werden muss, wurde dieser Weg hier erstmal nicht weiter verfolgt.

Zudem stellt sich die Frage nach der optimalen Vorhersageposition. Da der vorgestellte Algorithmus im Uhrzeigersinn sucht, wurde in Abbildung 4.5 für Esposende, eine Position südlich des eigentlichen Spots gefunden. Weiter westlich, auf gleicher Höhe wie Esposende, sind allerdings auch Vorhersagedaten vorhanden. Welche Position nun besser zur Extraktion der Vorhersagedaten geeignet ist bzw. welche Vorhersagewerte am ehesten dem Spot entsprechen, müsste im Detail noch untersucht werden. Die bisher erzielten Ergebnisse machen jedoch einen realistischen Eindruck und liefern nachvollziehbare Resultate, die bis auf kleinere Abweichungen den Vorhersagen einer ähnlichen Webseite [19] entsprechen, und somit erst einmal als zufriedenstellend bewertet werden.

Die Geschwindigkeit bei der Extraktion der Vorhersagedaten mit dem Programm *degrib* kann nicht bemängelt werden. Nachdem ein Index erstellt wurde, ist ein schneller und wahlfreier Zugriff auf die Daten einer *Grib*-Datei möglich. Zwar wurde das Programm nicht auf mögliche Optimierungen hin untersucht; es kann aber davon ausgegangen werden, dass die Implementierung durch das *Meteorological Development Laboratory* [20] sehr auf Effizi-

[18] `http://www.wannasurf.com`
[19] `http://magicseaweed.com`
[20] `http://www.weather.gov/mdl`

enz ausgerichtet ist. Stößt die Transformationsphase an ihre Grenzen, würde sich eine parallele Verarbeitung der *Grib*-Dateien anbieten. Da die Transformationsphase eines Spots nicht von anderen Spots abhängig ist, ist eine sequenzielle Verarbeitung nicht zwingend erforderlich. Durch eine parallele Verarbeitung könnte die Performanz der Transformationsphase erheblich verbessert werden und ein sehr viel größerer Geschwindigkeitsvorteil erzielt werden, als dies mit Optimierungen einer sequentiellen Verarbeitung möglich wäre. Das Auslesen und die Transformation der Vorhersagedaten der 8.300 Spots könnte man beispielsweise auf zwei Computern parallel durchführen, und somit die bisher benötigte Zeit von 26 Minuten auf 13 Minuten halbieren. Möglichkeiten zur parallelen Verarbeitung werden noch in den Verbesserungsvorschlägen des Ladevorgangs vorgestellt, denen deshalb hier nicht vorweggegriffen werden soll.

4.5 Laden der Daten

Die Aufgabe des Ladevorgangs besteht darin, die in den vorigen Schritten erhobenen Vorhersagedaten in die operative Datenbasis der Web-Applikation zu integrieren. Nachdem dieser Ladevorgang erfolgreich abgeschlossen wurde, stehen der Web-Applikation die aktuellen Vorhersagedaten zur Verfügung.

4.5.1 Datenbankschema der Vorhersagedaten

Das Datenbankschema der Web-Applikation orientiert sich an den von *ActiveRecord* erwarteten Konventionen, die sich hier zum Großteil auch bewährt haben. Beim Entwurf des Schemas wurden die üblichen Methoden zur Normalisierung von Relationen angewendet, Primär- und Fremdschlüssel definiert und Indizes zum schnelleren Auffinden von Datensätzen erstellt. Da für die *ETL*-Prozesse nur zwei Tabellen von Interesse sind, wird hier auf die komplette Darstellung des Datenbankschemas verzichtet und nur auf die für den Ladevorgang relevanten Relationen eingegangen.

Referenzielle Integrität in Ruby on Rails

Seltsamerweise scheint in der *Ruby on Rails* Community nicht allzuviel Wert auf die *ACID* Eigenschaften eines Datenbank Management Systems gelegt zu werden. 4 Jahren nach der Veröffentlichung von *Ruby on Rails* fehlen in der *API* von *ActiveRecord* leider immer noch Methoden zur Definition von Fremdschlüsseln. Auch in vielen Büchern und Internetartikeln zu *Ruby on Rails* wird auf die Verwendung von Fremdschlüsseln zur Sicherung der

referenziellen Integrität nicht eingegangen. Dieses Defizit wurde durch eine eigene Definition der entsprechender Methoden in der *ActiveRecord* Bibliothek behoben, so dass Fremdschlüssel direkt über die *API* definiert werden können. Diese stellen die referenzielle Integrität auf Datenbankebene sicher. Bei der Entwicklung der Web-Applikation traten so einige Fehler sehr viel schneller zum Vorschein, als wenn man auf Fremdschlüssel verzichtet hätte.

Repräsentation der Vorhersagedaten in der Datenbank

Die Vorhersagedaten der Spots werden in einer Relation mit dem Namen *forecasts* gespeichert, deren Aufbau in Tabelle 4.16 zu sehen ist. Die Spalten der Tabelle enthalten den Fremdschlüssel eines Spots, den Zeitpunkt der Vorhersage und die Werte der Vorhersageelemente. Die Attribute *spot_id* und *valid_time* bilden den Schlüsselkandidaten der Relation. Pro Spot (*spot_id*) existiert für jeden Vorhersagezeitpunkt (*valid_time*) genau ein Datensatz in der Tabelle. Um den *ActiveRecord*-Konventionen gerecht zu werden und Datensätze dieser Tabelle aus anderen Tabellen einfacher zu referenzieren, wird allerdings als künstlicher Primärschlüssel das Attribut *id* verwendet. Das Attribut *reference_time* enthält den Zeitpunkt, an dem die beiden Modelle erstellt wurden. Da das *Global Forecast System* und das *Wave Watch III* Modell im Moment zu den selben Zeitpunkten erstellt wird, gilt dieses Attribut für beide Modelle. Die beiden Attribute *created_at* und *updated_at* sind zwei von *ActiveRecord* automatisch verwaltete Attribute, welche die Zeitpunkte an dem ein Datensatz erstellt und an dem er zuletzt aktualisiert wurde, enthalten. Alle anderen Attribute repräsentieren die Vorhersageelemente der beiden Modelle und nehmen die mit *degrib* ausgelesenen Werte auf.

4.5.2 Tupelorientierte Aktualisierung

Eine offensichtliche Methode die Vorhersagedaten auf den neusten Stand zu bringen, ist die *tupelorientierte* Aktualisierung mit *SQL*. Tupelorientiert deshalb, weil für jedes Tupel, das in die Zielrelation eingefügt bzw. dort aktualisiert wird, ein *SQL* Befehl ausgeführt wird.

Hierzu müsste man zunächst die Zeilen der *CSV*-Datei zu Datensätzen transformieren, die der Struktur der *forecasts* Relation entsprechen. Anschließend könnte man die zu aktualisierenden Datensätze mit einem *UPDATE* Befehl ändern und die neu hinzugekommenen Datensätze mit einem *INSERT* Befehl einfügen. Bei einem Vorhersagezeitraum von 180 Stunden in einem 3 stündigen Intervall sind dies pro Spot $(180/3) + 1 = 61$ auszuführende *SQL* Befehle, den Zeitpunkt der Analyse mit berücksichtigt. Für die verwendeten 78 Spots wären das $61 * 78 = 4.758$, bei den zuvor erwähnten 8.300 Spots

Spaltenname	Datentyp	Modifikator
id	integer	not null default
spot_id	integer	not null
reference_time	timestamp with time zone	not null
valid_time	timestamp with time zone	not null
significant_wave_height	double precision	-
mean_wave_direction	double precision	-
mean_wave_period	double precision	-
peak_wave_direction	double precision	-
peak_wave_period	double precision	-
wind_direction	double precision	-
wind_speed	double precision	-
temperature	double precision	-
total_cloud_cover	double precision	-
precipitable_water	double precision	-
water_equivalent_snow_depth	double precision	-
created_at	timestamp with time zone	not null
updated_at	timestamp with time zone	not null

Tabelle 4.16: Schema der Datenbanktabelle *forecasts*

einer ähnlichen Webseite $61 * 8.300 = 506.300$ auszuführende *SQL* Befehle. Da sich bei einer größeren Datenmenge der vom Datenbankmanagementsystem betriebene Overhead beim Ausführen der vielen Befehle negativ auf die Performanz des Ladevorgangs auswirkt [Pos09b], wurde diese Methode zur Aktualisierung der Vorhersagedaten nicht weiter verfolgt.

Das hier beschriebene Verfahren eignet sich für Anwendungen, bei denen nur eine geringe Datenmenge zu verarbeiten und die Performanz des Ladevorgangs nicht kritisch ist. Der Vorteil der tupelorientierten Aktualisierung liegt in der Implementierung. Diese ist meist verständlicher und einfacher, als auf Performanz getrimmte Varianten.

Bevor im nächsten Abschnitt ein aus dem Bereich des *Data Warehousing* bekanntes Verfahren vorgestellt wird, sollen hier noch einige Verbesserungsvorschläge gemacht werden. Um die Laufzeit einer tupelorientierten Aktualisierung zu verbessern, sollte die Dokumentation des verwendeten Datenbankmanagementsystems herangezogen und mögliche Optimierungsvorschläge in

Betracht gezogen werden. Durch die Verwendung von *Prepared Statements*, dem Ausschalten der *Auto-Commit* Einstellung und der Einbettung aller Befehle in eine einzelne Transaktion kann die Performanz in vielen Anwendungen gesteigert werden. Reichen diese Optimierungen nicht aus, sind mit dem im folgenden Abschnitt beschriebenen Verfahren sehr viel bessere Ergebnisse zu erzielen.

4.5.3 Bulk Loading

Das grundlegende Problem der zuvor beschriebenen Methode besteht darin, dass die abgesetzten *SQL* Befehle tupelorientiert arbeiten. Die *SQL* Befehle zur Manipulation von Datensätzen können aber auch mengenorientiert eingesetzt werden, so dass pro Befehl mehrere Tupel geändert bzw. hinzugefügt werden. Voraussetzung hierfür ist, dass das Datenbankmanagementsystem auf alle zu verarbeitenden Daten zugreifen kann. Dies war bei der vorigen Methode nicht der Fall, da das Datenbankmanagementsystem bei jeder Operation nur einen Datensatz der Eingabe zu Gesicht bekam.

In diesem Abschnitt wird beschrieben wie die Vorhersagedaten der *CSV*-Datei effizient unter die Kontrolle des Datenbank Management Systems gebracht werden können. Die Daten werden zunächst in einen temporären Bereich der Datenbank geladen, der sogenannte *Staging Area*. Von dort aus werden sie in einem weiteren Schritt in die operative Datenbasis der Web-Applikation überführt. Die hier verwendete *Staging Area* besteht aus einer einzigen, in Tabelle 4.18 zu sehenden, Relation, deren Attribute die Spalten der *CSV*-Datei widerspiegeln.

Spaltenname	**Datentyp**	**Modifikator**
id	integer	not null default
spot_id	integer	not null
latitude	double precision	not null
longitude	double precision	not null
element	character varying(255)	not null
unit	character varying(255)	not null
reference_time	timestamp with time zone	not null
valid_time	timestamp with time zone	not null
value	double precision	-

Tabelle 4.18: Schema der Datenbanktabelle *grib_messages*

Bevor die Daten in diese Tabelle geladen werden, wird diese zunächst von den Datensätzen eines vorherigen Ladevorgangs bereinigt. Anschließend könnten die Vorhersagedaten mit *INSERT* Operationen in diese Relation eingefügt werden. Auch hier würde sich die Verwendung eines *Prepared Statement* anbieten, da eine hohe Anzahl sich ähnelnder Befehle verarbeitet werden müsste. Im Bereich des *Data Warehousing* wird beim Laden von Daten allerdings immer auf die sogenannten *Bulk Loader* der verwendeten Datenbankmanagementsysteme verwiesen. Dies sind meist datenbankspezifische Befehle oder Programme, mit denen größere Datenmengen effizienter als mit den standardisierten *SQL*-Befehlen geladen werden können. Der *Bulk Loader* von *PostgreSQL* ist durch die *COPY*-Befehlsfamilie implementiert, mit der Daten im Text-, *CSV*- oder Binärformat importiert und exportiert werden können. Der Befehl aus Auflistung 4.4 veranlasst *PostgreSQL* dazu, die über den Standard Eingabekanal gelesenen Datensätze in die Tabelle *grib_messages* zu importieren. Dieser Befehl wird mit *PostgreSQL*'s Kommandozeilenprogramm *psql* ausgeführt und die Datensätze der *CSV*-Datei über den Standard Eingabekanal weitergereicht. Nachdem mit dem *COPY*-Befehl alle Daten geladen wurden, wird ein zusätzlicher *ANALYZE*-Befehl ausgeführt, der die Statistiken von allen Tabellen aktualisiert. Diese Statistiken werden vom Anfrageoptimierer des Datenbankmanagementsystems verwendet, um einen geeigneten Ausführungsplan für einen *SQL*-Befehl zu finden.

```
1 COPY grib_messages (
2   spot_id, latitude, longitude, element, unit,
3   reference_time, valid_time, value
4 ) FROM STDIN;
```

Auflistung 4.4: Befehl zum Import von Datensätzen in *PostgreSQL*

Um die durchschnittliche Laufzeit des *COPY*-Befehls zu ermitteln, wurde eine *CSV*-Datei erstellt, in der Vorhersagedaten aus beiden Modellen für 78 Spots enthalten waren. Diese Datei war ca. 3,6 MB groß, hatte 52.338 Zeilen ($78 * 11 * 61$) und enthielt die Werte von 11 Vorhersageelementen für 61 Vorhersagezeitpunkte. Diese Datei wurde 28 mal mit dem *Bulk Loader* des Datenbankmanagementsystems in eine leere Tabelle geladen und die Laufzeit gemessen. Für die Vorhersagedaten der 78 Spots ergab sich aus diesen Messwerten eine durchschnittliche Laufzeit von 15,24 Sekunden mit einer Standardabweichung von 5,97 Sekunden. Aus der daraus berechneten durchschnittlichen Laufzeit von 195 Millisekunden für einen Spot, wurde eine Hochrechnung für mehrere Spots abgeleitet, die in Tabelle 4.20 zu sehen ist. In dieser Hochrechnung wurde angenommen, dass sich die Laufzeit des *COPY*-Befehls linear verhält.

Im Abschnitt *Populating a Database* [Pos09b] der *PostgreSQL* Dokumen-

	1 Spot	10 Spots	100 Sp.	1.000 Sp.	10.000 Sp.	100.000 Sp.
Zeit	195 ms	1,95 s	19,5 s	3 m 15 s	32 m 30 s	5 h 25 m
Größe	46,3 KB	463 KB	4,52 MB	45,21 MB	452,15 MB	4,42 GB

Tabelle 4.20: Hochrechnung der Laufzeit für den *COPY*-Befehl

tation sind Hinweise zu finden, die beim Verarbeiten größerer Datenmengen zu beachten sind. Auf der Zielrelation definierte Indizes, *Check Constraints* und *Trigger*, wirken sich negativ auf die Performanz des Ladevorgangs aus. Dort wird zum Beispiel vorgeschlagen, auf der Zielrelation definierte Indizes vor dem Laden zu entfernen und anschließend wieder neu zu erstellen. Da bei der zuvor durchgeführten Messung Indizes auf der Tabelle definiert waren, ist bei einer Umsetzung dieses Vorschlags noch eine leichte Steigerung der Performanz zu erwarten. Die Konfiguration des Datenbank Management Systems selbst ist auch nicht außer Acht zu lassen. Die Performanz von *PostgreSQL* lässt sich durch das Einstellen bestimmter Parametern in der Konfigurationsdatei verbessern. Da die Standardwerte in vielen Distributionen für heutige Hardware relativ niedrig angesetzt sind, ist eine Anpassung an das verwendete System unbedingt zu empfehlen. Eine optimale Systemkonfiguration benötigt allerdings einiges an Erfahrung und Geduld beim Ausprobieren der verschiedenen Konfigurationsparameter.

4.5.4 Mengenorientierte Aktualisierung

Nachdem die zu verarbeitenden Daten unter die Kontrolle des Datenbank Management Systems gebracht wurden, kann mit der mengenorientierten Aktualisierung der Vorhersagedaten begonnen werden. Ziel ist, die Vorhersagedaten der Relation *grib_messages* in die Relation *forecasts* zu überführen. Dabei werden existierende, aus einer früheren Modellberechnung stammende Datensätze der Tabelle *forecasts* mit den Werten der neueren Berechnung überschrieben und noch nicht existierende Datensätze hinzugefügt. Es werden nur diejenigen Datensätze aktualisiert bzw. hinzugefügt, die auch im Zeitraum der aktuellen Modellberechnung liegen. Dieser Zeitraum ist durch die Datensätze der Quellrelation *grib_messages* bestimmt und kann mit dem *SQL*-Befehl *SELECT MIN(valid_time), MAX(valid_time) FROM grib_messages* ermittelt werden.

Transposition von Spalten und Zeilen

Idealerweise würde man bei der mengenorientierten Aktualisierung genau zwei Befehle ausführen. Ein *UPDATE*-Befehl, der alle existierenden Datensätze der Relation *forecasts* aktualisiert und ein *INSERT*-Befehl, mit dem neue Datensätze hinzugefügt werden. Die Art und Weise wie die Daten aber in der Relation *grib_messages* vorliegen, würde zu sehr komplexen *SQL* Befehlen oder dem Einsatz datenbankspezifischer Erweiterungen führen. Das Problem besteht darin, dass mehrere Datensätze der Relation *grib_messages* zu einem Datensatz der Relation *forecasts* überführt werden müssen. Oder anders ausgedrückt: Mehrere Zeilen der Quellrelation müssen zu einer Spalte der Zielrelation transponiert werden. Die Konstruktion solcher Anfragen lässt sich nur umständlich in *SQL* ausdrücken und führt üblicherweise zu sehr komplexen Befehlen. Für diese Art von Anfragen werden in *PostgreSQL* üblicherweise die sogenannten *crosstab*-Funktionen aus dem Modul *tablefunc* verwendet. Da bei einem Engpass des Ladevorgangs eine Vorverarbeitung der *CSV*-Datei mit *awk* oder einem spezialisierten Programm als vielversprechender angesehen wird, wurden diese Alternativen hier nicht weiter verfolgt, sondern eine inkrementelle und leichter erweiterbare Strategie gewählt.

Hinzufügen von Datensätzen

Die gewählte Strategie zur Aktualisierung der Vorhersagedaten erfolgt in zwei Schritten. Im ersten Schritt wird durch das Hinzufügen entsprechender Datensätze sichergestellt, dass für jeden Spot und jeden Vorhersagezeitpunkt der Quellrelation ein Datensatz in der Zielrelation existiert. Diese hinzugefügten Datensätze dienen als Platzhalter und werden im zweiten Schritt mit den Vorhersagedaten der Quellrelation aktualisiert. Mit einer *SQL*-Anfrage wird diejenige Menge an Datensätze erzeugt, deren Schlüsselkandidaten noch nicht in der Zielrelation existieren, sich aber aus den Datensätzen der Quellrelation ableiten lassen. Diese Datensätze dienen vorübergehend als Platzhalter und enthalten nur gültige Werte in den Attributen *spot_id*, *valid_time*, *created_at*, *updated_at* und *reference_time*. Alle anderen Attribute repräsentieren die Vorhersageelemente und werden mit dem Wert *NULL* belegt. Diese Platzhalter Datensätze werden durch einen *INSERT* Befehl in die Tabelle *forecasts* eingefügt.

Beeinflussung des Anfrageoptimierers

Die Menge der hinzuzufügenden Datensätze kann durch Verwendung verschiedenster *SQL*-Konstrukte erzeugt werden. Eine der Aufgaben des Datenbankmanagementsystems besteht darin, einen guten Ausführungsplan für

eine Anfrage zu finden. Idealerweise sollten dabei zwei äquivalente Befehle in den optimalen Ausführungsplan übersetzt werden. Diese komplexe Aufgabe wird von den Optimierern der verschiedenen Datenbankmanagementsysteme allerdings unterschiedlich gut gelöst. Durch das Verwenden verschiedener *SQL*-Konstrukte kann der Optimierer eines Datenbankmanagementsystems teilweise beeinflusst werden. Hier wurde das Laufzeitverhalten von zwei äquivalenten *SQL*-Befehlen durch die Analyse ihrer Ausführungspläne untersucht. Der erste Befehl verwendet einen *Left Join* um das Ergebnis zu berechnen, im zweiten Befehl wird eine *Subquery* benutzt. Beide Befehle wurden auf einer Datenbank ausgeführt, in der sich bereits historische Vorhersagedaten aus einem Zeitraum von ca. 6 Monaten (108.169 Datensätze in der Relation *forecasts*) angesammelt haben. In der Relation *grib_messages* waren insgesamt 52.338 Datensätze aus dem *Wave Watch III Modell* und dem *Global Forecast System* enthalten, die mit den hier untersuchten Befehlen in die Relation *forecasts* überführt werden sollen.

Analyse der Ausführungspläne

Da der Optimierer von *PostgreSQL* bei der Wahl des Ausführungsplans Statistiken über Tabellen verwendet, ist es wichtig, diese regelmäßig zu aktualisieren. Üblicherweise wird dies in einem bestimmten Intervall durch einen Hintergrundprozess erledigt, sollte aber nach dem Einfügen oder Entfernen von vielen Datensätzen manuell mit dem *ANALYZE*-Befehl durchgeführt werden. Um den Ausführungsplan eines *SQL*-Befehls in *PostgreSQL* anzuzeigen, wird der *EXPLAIN*-Befehl benutzt. Durch die Angabe des Parameters *ANALYZE* wird der zu untersuchende Befehl ausgeführt und Informationen über dessen Laufzeit ausgegeben. Die hier gezeigten Ausführungspläne wurden zwar mit dem Parameter *ANALYZE* erstellt, zeigen der Übersichtlichkeit halber aber nur die für die Untersuchung relevanten Informationen. Vor der Analyse der Ausführungspläne wurden auf allen Attributen, die in Anfragen referenziert werden, Indizes definiert und die Statistiken der Tabellen aktualisiert. Auf den Schlüsselkandidaten *spot_id* und *valid_time* wurden zusätzlich noch zusammengesetzte Indizes definiert.

Ausführungsplan der Left Join Anfrage

In Auflistung 4.5 ist der *SQL*-Befehl als *Left Join* Variante zu sehen, mit der die Menge der hinzuzufügenden Datensätze erzeugt wird. Der Befehl verknüpft die Datensätze der Relation *grib_messages* über die Attribute *spot_id* und *valid_time* mit den Datensätzen der Relation *forecasts*. Durch die Bedingung in der *WHERE*-Klausel wird das Ergebnis auf diejenigen Daten-

sätze reduziert, deren Schlüsselkandidat noch nicht in der Relation *forecasts* existiert. Dies sind genau die Vorhersagezeitpunkte, die seit der letzten Modellberechnung neu hinzugekommen sind. Obwohl auf den entsprechenden Attributen beider Relationen Indizes definiert sind, kann man im Ausführungsplan in Auflistung 4.6 sehen, dass ein *Sequence Scan* auf der Relation *forecasts* durchgeführt wird. Falls viele Spots hinzukommen oder historische Vorhersagedaten für statistische Zwecke und *Data Mining*-Verfahren über einen längeren Zeitraum behalten werden, wird man mit diesem *Sequence Scan* auf Dauer Probleme bekommen.

```
1  INSERT INTO forecasts(spot_id, valid_time, created_at, updated_at,
2                        reference_time)
3      SELECT grib_messages.spot_id, grib_messages.valid_time,
4             NOW() AT TIME ZONE 'UTC', NOW() AT TIME ZONE 'UTC',
5             MAX(grib_messages.reference_time)
6        FROM grib_messages
7   LEFT JOIN forecasts
8          ON forecasts.spot_id = grib_messages.spot_id
9         AND forecasts.valid_time = grib_messages.valid_time
10       WHERE forecasts.id IS NULL
11    GROUP BY grib_messages.spot_id, grib_messages.valid_time;
```

Auflistung 4.5: Hinzufügen von Datensätze mittels *Left Join*

Der Optimierer scheint bei dieser Anfrage nicht zu erkennen, dass die Ergebnismenge im Vergleich zu der Anzahl von Datensätzen in der *forecasts* Relation sehr gering ist. Im Ergebnis befinden sich nämlich nur diejenigen Datensätze, die seit der letzten Modellberechnung hinzugekommen sind. Das Problem liegt hier in der Verwendung des *Left Joins*, der das Datenbankmanagementsystem dazu veranlasst, alle Datensätze der beiden Relationen zu verknüpfen und erst anschließend den Großteil der Datensätze wieder weg zu werfen.

```
1  Subquery Scan "*SELECT*" (cost=15636.15..23487.90 rows=4819 width=36)
2  -> GroupAggregate (cost=15636.15..23415.62 rows=4819 width=20)
3     -> Merge Left Join (cost=15636.15..23039.99 rows=35627 width=20)
4          Merge Cond: ((grib_messages.spot_id = public.forecasts.spot_id) AND
5                       (grib_messages.valid_time = public.forecasts.valid_time))
6          Filter: (public.forecasts.id IS NULL)
7          -> Index Scan using index_grib_messages_on_spot_id_valid_time
8              on grib_messages  (cost=0.00..5521.40 rows=71109 width=20)
9          -> Materialize (cost=15636.15..16993.40 rows=108580 width=16)
10            -> Sort (cost=15636.15..15907.60 rows=108580 width=16)
11                 Sort Key: public.forecasts.spot_id,
12                           public.forecasts.valid_time
13                 Sort Method: external merge  Disk: 3624kB
14                 ->  Seq Scan on forecasts  (cost=0.00..4695.80 rows=108580
15                       width=16)
16 Total runtime: 213.518 ms
```

Auflistung 4.6: Ausführungsplan des *Left Joins*

Die auf der Relation *forecasts* definierten Indizes erweisen sich bei dieser Anfrage als nutzlos. Der Optimierer von *PostgreSQL* kann durch das Setzen entsprechender Parameter allerdings beeinflusst werden. Die Ausführung des selben Befehls mit einem erzwungenen *Index Scan* ergab ein sehr viel schlechteres Ergebnis als der im Allgemeinen als teuer bewertete *Sequence Scan*. Das Problem dieser Anfrage lässt sich also nicht durch die geschickte Definition von Indizes lösen, sondern durch die Vermeidung des *Left Joins*. Die Ausführungszeit beträgt in diesem Ausführungsplan ca. 214 Millisekunden, wird aber mit einem Anwachsen der Relation *forecasts* zunehmen. Eine bessere Alternative ist, die Anfrage in eine *Subquery* umzuformen.

Ausführungsplan der Subquery Anfrage

Ein äquivalenter Befehl zum Einfügen der Platzhalter Datensätze ist in Auflistung 4.7 zu sehen. Bei dieser Variante wird eine *korrelierte Subquery* verwendet, die im Allgemeinen als kritisch hinsichtlich der Performanz eingestuft wird. Bei einer *korrelierten Subquery* werden in der inneren Anfrage Attribute der Äußeren referenziert, was dazu führt, dass die innere Anfrage für jeden Datensatz ausgeführt werden muss, der durch die Äußere erzeugt wird.

```
1  INSERT INTO forecasts(spot_id, valid_time, created_at, updated_at,
2                        reference_time)
3       SELECT grib_messages.spot_id, grib_messages.valid_time,
4              NOW() AT TIME ZONE 'UTC', NOW() AT TIME ZONE 'UTC',
5              MAX(grib_messages.reference_time)
6         FROM grib_messages
7        WHERE NOT EXISTS (
8                SELECT 1
9                  FROM forecasts
10                WHERE forecasts.spot_id = grib_messages.spot_id
11                  AND forecasts.valid_time = grib_messages.valid_time)
12    GROUP BY grib_messages.spot_id, grib_messages.valid_time;
```

Auflistung 4.7: Hinzufügen von Datensätzen mittels *Subquery*

Wie man aber am Ausführungsplan aus Auflistung 4.8 erkennen kann, ist diese Anfrage mit ca. 100 Millisekunden jedoch doppelt so schnell wie die eben untersuchte *Left Join* Variante. Der teure *Sequence Scan* auf der Relation *forecasts* konnte hier vermieden werden. Stattdessen wird für jeden Datensatz aus der Tabelle *grib_messages* der zusammengesetzte Index auf dem Schlüsselkandidaten benutzt, um die Existenz eines entsprechenden Datensatzes in der Tabelle *forecasts* zu überprüfen.

```
1  Subquery Scan "*SELECT*" (cost=0.00..436811.09 rows=4819 width=36)
2  -> GroupAggregate (cost=0.00..436774.94 rows=4819 width=20)
3     ->  Index Scan using index_grib_messages_on_spot_id_valid_time
4           on grib_messages (cost=0.00..436526.73 rows=25864 width=20)
5           Filter: (NOT (subplan))
6           SubPlan
7             ->  Index Scan using index_forecasts_on_spot_id_and_valid_time
```

```
8                  on forecasts (cost=0.00..8.34 rows=1 width=0)
9                  Index Cond: ((spot_id = $0) AND (valid_time = $1))
10 Total runtime: 98.575 ms
```

Auflistung 4.8: Ausführungsplan der *Subquery*

Die Performanz der *Subquery* Variante wird bei einer Zunahme von Datensätzen der *forecasts* Relation nur minimal beeinflusst, da die Anzahl der *Subquery*-Aufrufe durch die Zahl der Datensätze in der Tabelle *grib_messages* beschränkt ist. Die Größe der Relation *forecasts* kann sich hier nur beim Auffinden von Datensätze über den stetig wachsenden Index negativ auswirken. Diese Beeinflussung ist allerdings gerade wegen der Verwendung geeigneter Indexstrukturen gering.

Zusammenfassend kann man zu dieser Untersuchung sagen, dass durch die Wahl entsprechender *SQL*-Konstrukte der Anfrageoptimierer eines Datenbankmanagementsystems beeinflusst werden kann und sich dies auf die Performanz der Anfrage auswirken kann. Wie man sehen konnte, war die *Subquery*-Anfrage im Gegensatz zur *Left Join*-Anfrage durch die Anzahl der Datensätze in der Tabelle *grib_messages* beschränkt. Diese Beschränkung ist wiederum abhängig von der Anzahl der Spots, was auch genau so sein sollte. Wichtig ist, dass die Anfrage durch die Anzahl der Spots und nicht durch die Anzahl der historischen Daten beschränkt ist.

Um eine Aussage über die durchschnittliche Laufzeit der *Subquery* Variante für die Vorhersagedaten von 78 Spots zu treffen, wurden die Laufzeiten dieser Befehle für beide Modelle über einen Zeitraum von einer Woche aus den Log-Dateien extrahiert. Aus diesen 56 Messwerten ($24h/6h * 7 * 2$), ergab sich eine durchschnittliche Laufzeit von 131 Millisekunden mit einer Standardabweichung von 34 Millisekunden.

Aktualisierung der Vorhersagedaten

Die Aktualisierung der Vorhersagedaten wird wegen der erwähnten Transposition von Zeilen zu Spalten pro Vorhersageelement durchgeführt. Für jedes Element wird ein *UPDATE*-Befehl ausgeführt, der die entsprechende Spalte der Datensätze aus der Relation *forecasts* aktualisiert. In Auflistung 4.9 ist der *UPDATE*-Befehl zu sehen, mit dem die Werte der Wellenhöhen aktualisiert werden. Da sich die Befehle nur in der Angabe des Spaltennamen in Zeile 3 und der Angabe der Elementkennung in Zeile 6 unterscheiden, werden diese Befehle pro Modell dynamisch erzeugt und hintereinander ausgeführt.

```
1 UPDATE forecasts
2    SET reference_time = grib_messages.reference_time,
3        significant_wave_height = grib_messages.value,
4        updated_at = NOW() AT TIME ZONE 'UTC'
5   FROM grib_messages
```

```
6  WHERE grib_messages.element = 'HTSGW'
7    AND grib_messages.spot_id = forecasts.spot_id
8    AND grib_messages.valid_time = forecasts.valid_time;
```

Auflistung 4.9: Aktualisierung der Wellenhöhe

Der Ausführungsplan in Auflistung 4.10 zeigt, dass bei der Aktualisierung zwei Indizes genutzt werden. Zum einen wird der Index auf dem Attribut *element* der Tabelle *grib_messages* genutzt, um nur diejenigen Zeilen zu finden, in denen die Wellenhöhe gespeichert ist. Zum anderen wird der zusammengesetzte Index auf dem Schlüsselkandidaten (*spot_id, valid_time*) der Tabelle *forecasts* verwendet, um nur Vorhersagen zu aktualisieren, die auch im Zeitraum der aktuellen Modellberechnung liegen. Die Laufzeit zur Aktualisierung der Wellenhöhe beträgt in diesem Ausführungsplan ca. 128 Millisekunden und ist wegen der Verwendung des Index auf der Tabelle *forecasts* ebenfalls durch die Anzahl der Spots beschränkt. Über einen Zeitraum von einer Woche ergab sich für einen *UPDATE*-Befehl eine durchschnittliche Laufzeit von 195 Millisekunden mit einer Standardabweichung von 41 Millisekunden. Für diese Berechnung wurden insgesamt 308 Messwerte ($24h/6h * 7 * 11$) für die insgesamt 11 Vorhersageelemente aus den Log-Dateien extrahiert.

```
1  Nested Loop (cost=128.70..20824.68 rows=4555 width=134)
2  -> Bitmap Heap Scan on grib_messages (cost=128.70..958.48 rows=4702 width=28)
3      Recheck Cond: ((element)::text = 'HTSGW'::text)
4      -> Bitmap Index Scan on index_grib_messages_on_element
5              (cost=0.00..127.53 rows=4702 width=0)
6            Index Cond: ((element)::text = 'HTSGW'::text)
7  -> Index Scan using index_forecasts_on_spot_id_and_valid_time
8        on forecasts (cost=0.00..4.21 rows=1 width=118)
9      Index Cond: ((forecasts.spot_id = grib_messages.spot_id) AND
10                  (forecasts.valid_time = grib_messages.valid_time))
11 Total runtime: 127.865 ms
```

Auflistung 4.10: Ausführungsplan der Aktualisierung

Bei der mengenorientierten Aktualisierung der Vorhersagedaten werden insgesamt *N+1* Anfragen ausgeführt, wobei *N* für die Anzahl der Vorhersageelemente steht. Zunächst wird ein *INSERT*-Befehl ausgeführt und anschließend ein *UPDATE*-Befehl pro Vorhersageelement. Alle Befehle zur Aktualisierung der Vorhersagedaten laufen in einer Transaktion ab, durch die sichergestellt ist, dass der ursprüngliche Status der Datenbank bei einem eventuellen Fehler wieder hergestellt wird. Dies hat den zusätzlichen Vorteil, dass mehrere Befehle, die in einer Transaktion stattfinden, von *PostgreSQL* schneller ausgeführt werden können, als wenn jeder Befehl in eine eigene Transaktion eingebettet ist.

Laufzeitverhalten bei einer größeren Datenmenge

Eine konkrete Aussage über die Laufzeit des Ladevorgangs bei einer höheren Anzahl von Spots zu machen, ist schwer. Dies liegt hauptsächlich darin begründet, dass nicht bekannt ist, wie sich der Anfrageoptimierer und die Laufzeit der einzelnen Befehle bei einer größeren Datenmenge verhalten. Kommen mehr Daten hinzu, verändern sich die Statistiken, die vom Anfrageoptimierer dazu verwendet werden, einen geeigneten Ausführungsplan zu finden. Die dann verwendeten Ausführungspläne können sich von den hier Untersuchten unterscheiden und ein anderes Laufzeitverhalten haben. Um trotzdem einen Eindruck für die Laufzeit bei einer größeren Anzahl von Spots zu bekommen, wird hier eine Hochrechnung der Laufzeit vorgestellt, bei der angenommen wird, dass sich die durchschnittliche Laufzeit der *SQL*-Befehle bei ansteigendem Datenvolumen linear verhält. Zunächst werden aus den zuvor gemessenen Laufzeiten der *INSERT*- und *UPDATE*-Befehle für 78 Spots die durchschnittlichen Laufzeiten für einen Spot abgeleitet. Die durchschnittliche Laufzeit eines *INSERT*-Befehls für einen Spot beträgt somit $l_i = 131ms/78 = 1,68ms$, die eines *UPDATE*-Befehls $l_u = 195ms/78 = 2,5ms$. Daraus lässt sich die durchschnittliche Gesamtlaufzeit zur Aktualisierung von N Vorhersageelementen für einen Spot mit $l = l_i + N * l_u =$ berechnen. Für 11 Vorhersageelementen sind dies $l = 1,68ms + 11 * 2,5ms = 29,18ms$. Ausgehend von der durchschnittlichen Laufzeit von einem Spot für die 11 Vorhersageelemente beider Modelle, wurde eine Hochrechnung für mehrere Spots vorgenommen, die in Tabelle 4.22 zu sehen ist. Wie nahe die hier vorgestellte Hochrechnung an der Realität ist, bleibt allerdings fraglich.

	1 Spot	**100 Spots**	**1.000 Spots**	**10.000 Spots**	**100.000 Spots**
Laufzeit	29 ms	2,9 s	29 s	4 m 50 s	49 m 20 s

Tabelle 4.22: Hochrechnung der Laufzeit aller *SQL*-Befehle

4.5.5 Verbesserung des Ladevorgangs

Im Abschnitt "*Performance Tips*" [Pos09a] der *PostgreSQL* Dokumentation werden einige Vorschläge gemacht, die zur Steigerung der Performanz beitragen können. Beispielsweise kann der *COPY*-Befehl am schnellsten ausgeführt werden, wenn er in der selben Transaktion wie ein zuvor abgesetzter *CREATE TABLE*- oder *TRUNCATE*-Befehl ausgeführt wird. In diesem Fall kann

auf das sonst zu verwaltende *Write Ahead Log* verzichtet werden, da bei einem Fehler die Daten wieder gelöscht werden. Weiterhin wird vorgeschlagen, die Indizes einer zu ladenden Tabelle vor dem Laden zu löschen und erst danach wieder zu erstellen, da der Aufbau eines neuen Index schneller ist als eine zeilenweise Aktualisierung. Bei der jetzigen Datenmenge wird der Umsetzung dieser Verbesserungsvorschläge allerdings eine eher geringe Priorität eingeräumt, da der zu erzielende Geschwindigkeitsvorteil im Vergleich zu anderen Vorschlägen als nicht sehr hoch eingestuft wird.

Viel mehr Potential zur Beschleunigung des Ladevorgangs wird in einer Vorverarbeitung der Vorhersagedaten gesehen. Durch diesen zusätzlichen Schritt könnte man die Vorhersagedaten in eine *CSV*-Datei transformieren, die der Struktur der Tabelle *forecasts* entspricht. Die bisher in der Datenbank durchgeführte Transposition von Zeilen zu Spalten würde so in ein externes Programm verlagert werden, mit der Aussicht, diese effizienter durchführen zu können, als dies innerhalb des Datenbankmanagementsystems möglich ist. Die transformierte Datei könnte dann in eine entsprechende *Staging Area* geladen werden und mit einem *UPDATE*- und einem *INSERT*-Befehl in die Relation *forecasts* integriert werden.

Eine noch bessere Alternative wäre, die zu aktualisierenden Datensätze der Tabelle *forecasts* vor dem Laden zu löschen und die *CSV*-Datei mit einem *COPY*-Befehl direkt in diese Relation zu integrieren. Ein Nachteil an diesem Vorschlag ist, dass der Transformationsschritt vom Aktualisierungszeitplan der beiden Modelle abhängig ist. Da die Daten beider Modelle zu einem Datensatz der Relation *forecasts* transformiert werden, stellt sich die Frage, welche Strategie man bei der Transformation und beim späteren Laden wählt, falls die Modelle erst zu unterschiedlichen Zeitpunkten zur Verfügung stehen (was in der Regel der Fall ist). Hier wäre die einfachste Möglichkeit, den Transformationsschritt erst dann anzustoßen, wenn die Daten beider Modelle verfügbar sind. Andernfalls müsste man die aktuellen Daten des einen Modells mit den schon geladenen Daten des Anderen verknüpfen.

4.6 Zusammenfassung

Eine Hochrechnung der Gesamtlaufzeit der *ELT*-Prozesse für mehrere Spots ist in Tabelle 4.24 zusammengestellt. Die Extraktionsprozesse für das *Wave Watch III* Modell und das *Global Forecast System* sind dabei unabhängig von der Anzahl der Spots, da in diesen Schritten zunächst die benötigten Vorhersageelemente (für den gesamten Globus) der beiden Modelle heruntergeladen werden. Die Extraktion der Vorhersagedaten benötigt deshalb in allen Fällen durchschnittlich ca. 18 Minuten. In der Transformationsphase

Phase	**100 Spots**	**1.000 Spots**	**10.000 Spots**	**100.000 Spots**
Extraktion *(NWW3)*	4 m	4 m	4 m	4 m
Extraktion *(GFS)*	14 m	14 m	14 m	14 m
Transformation	18,7 s	3 m 7 s	31 m 10 s	5 h 12 m
Bulk Load	19,5 s	3 m 15 s	32 m 30 s	5 h 25 m
Aktualisierung	2,9 s	29 s	4 m 50 s	49 m 20 s
Gesamt:	18 m 41 s	24 m 51 s	1 h 26 m	11 h 44 m

Tabelle 4.24: Hochrechnung der Gesamtlaufzeit

wird diese Datenmenge dann auf die Anzahl der Spots reduziert, indem nur die Vorhersagedaten an den Spots in eine *CSV*-Datei transformiert werden. In diesem und den folgenden Schritten wirkt sich dann die Anzahl der Spots auf die Laufzeit der *ETL*-Prozesse aus. Der Ladevorgang ist schließlich dafür verantwortlich die neuen Vorhersagedaten aus der *CSV*-Datei in die operative Datenbasis zu überführen. Danach stehen die Daten der Web-Applikation zur Verfügung und können dem Nutzer der Plattform angezeigt werden.

Spätestens wenn sich die Laufzeit der *ETL*-Prozesse dem 6-stündigen Aktualisierungszeitplan der beiden Modelle nähert, sollte aber ein alternativer Weg zur Verarbeitung und Integration der Vorhersagedaten eingeschlagen werden. Wie schon in den Verbesserungsvorschlägen des Transformations- und Ladevorgangs beschrieben, wird eine Vorverarbeitung der Daten außerhalb des Datenbankmanagementsystems als mögliche Strategie angesehen. Zum einen wird dadurch das Datenbankmanagementsystem erheblich entlastet und zum anderen könnte man dazu übergehen die Vorhersagedaten parallel zu Verarbeiten. Insbesondere bei dem zur Zeit stattfindenden Trend in Richtung *Cloud Computing* bietet sich eine Vorverarbeitung der Daten an. Die Extraktion und die Transformation der Vorhersagedaten wären beispielsweise gute Kandidaten für das *Map Reduce* Verfahren. Das Verarbeiten der Vorhersagedaten könnte auf mehrere Systeme aufgeteilt und durch parallele Verarbeitung sehr viel schneller durchgeführt werden. Dieses Verfahren ist mittlerweile durch Web Services, wie z.B. *Amazon's Elastic Map Reduce* [21], auch für kleiner Projekte ohne Anschaffung der benötigten Hardware erschwinglich und interessant geworden. Die Verbesserung des gesamten *ETL*-Prozesses erfordert ein gründliches Abwägen der verschiedenen Vor-

[21] `http://aws.amazon.com/elasticmapreduce`

und Nachteile. Oft stellt sich heraus, dass man mit einem Geschwindigkeitsvorteil an einer Stelle, Flexibilität an einer anderen einbüßt und/oder zusätzliche Komplexität erzeugt. Stoßen die hier entwickelten *ETL*-Prozesse an ihre Grenzen, würde mit Hinblick auf das *Map Reduce* Verfahren der Weg in Richtung Vorverarbeitung eingeschlagen.

KAPITEL 5

Ausblick

5.1 Visualisierung von Wetter- und Wellendaten

Die visuelle Aufbereitung von Wetterdaten ist ein weiteres Anwendungsgebiet, bei dem die Daten numerischer Wettermodelle zum Einsatz kommen. Vielen sind solche Darstellungen meist aus den Wettersendungen im Fernsehen bekannt. Dort wird den Zuschauern die Gesamtwetterlage einer Region oft in Form von Wetterkarten oder Strömungsfilmen präsentiert. Der Wertebereich eines Vorhersageelements wird dabei auf eine Farbpalette reduziert und die unterschiedlichen Werte durch Einfärben der Wetterkarte dargestellt. Strömungsfilme werden wiederum aus mehreren Wetterkarten erstellt, die die Wetterlage zu verschiedenen Zeitpunkten zeigen und im Zeitraffer hintereinander abgespielt werden. In Strömungsfilmen wird beispielsweise die Ausbreitung von Hoch- und Tiefdruckgebieten oder die Bewegung von Wolken in den nächsten Stunden und Tagen gezeigt. Die Reduzierung des Wertebereichs und das Einfärben der Landkarte bietet eine globale Sichtweise auf die Vorhersageelemente und spielt eine wichtige Rolle bei der Betrachtung der Wetterlage.

Da surfbare Wellen oft durch den viele Kilometer weit gereisten Swell entstehen, ist eine globale Betrachtung dessen Ausbreitung recht aufschlussreich. In Abbildung 5.1 ist eine Wetterkarte mit den Wellenhöhen und einiger weiterer Vorhersageelemente über dem Atlantik zu sehen. Die verschiedenen Wellenhöhen sind dort in unterschiedlichen Farben gekennzeichnet. Durch diese Darstellung kann man sehr gut Trends erkennen, die bei einem Blick

auf bloße Zahlen gar nicht auffallen würden oder nur schwer erkennbar wären. Betrachtet man beispielsweise nur die Wellenhöhen an einem bestimmten Spot, kann man nur eine Aussage für die nächsten 180 Stunden machen. Zieht man jedoch die visuelle Darstellung der Wellenausbreitung hinzu, kann man oft einen dort erst sehr viel später eintreffenden Swell erkennen und eine Einschätzung weit über 180 Stunden hinaus bekommen. Eine Darstellung solcher Strömungsfilme für einige ausgewählte Regionen wäre eine wünschenswerte Erweiterung, mit der den Nutzern der Web-Applikation ein Mehrwert geboten werden könnte. Einige Webseiten [1] bieten bereits sehr gute [2] Aufbereitungen solcher Vorhersagen an, was auch für die hier entwickelte Web-Applikation wünschenswert wäre. Um Plots wie in Abbildung 5.1 und Strömungsfilme zu erstellen, kann das Open Source Programm *Grads* [3] verwendet werden.

5.2 Verbesserung der Vorhersagen

Die aus dem *Global Forecast System* und dem *Wave Watch III* Modell stammenden Wetter- und Wellenvorhersagen beruhen zwar auf einem numerisch berechneten Wettermodell, können aber nur bedingt zur Vorhersage der tatsächlichen Surfbedingungen verwendet werden. In Kombination mit Erfahrungswerten und der Kenntnis lokaler Gegebenheiten dienen sie eher als Indikator zur Einschätzung dieser Bedingungen. In welcher Verbindung die Vorhersagewerte mit der Qualität der vor Ort brechenden Wellen stehen ist unklar und variiert wohl von Spot zu Spot. Hinzu kommt, dass wegen der groben Gitterauflösung der Modelle die Vorhersagewerte teilweise aus der näheren Umgebung eines Spots herangezogen werden müssen. Trotzdem sind aber vor allem die Wellenhöhe, die Wellenperiode, die Windstärke und die Windrichtung der Modelle wichtige Indizien, die bei der Einschätzung der Surfbedingungen eine große Rolle spielen. Was noch fehlt, ist eine Komponente mit der die lokalen Gegebenheiten eines Surfspots mit ins Spiel gebracht werden. Außerdem wäre es wünschenswert, eine Aussage über die Qualität der Surfbedingungen an einem Spot treffen zu können.

Beobachtung lokaler Gegebenheiten

Würde man die Qualität der Surfbedingungen an einem Spot mit den dort prognostizierten Wetter- und Wellenvorhersagen über einen längeren Zeit-

[1] `http://nomad5.ncep.noaa.gov/cgi-bin/pdisp_wave.sh`
[2] `http://magicseaweed.com`
[3] `http://www.iges.org/grads/`

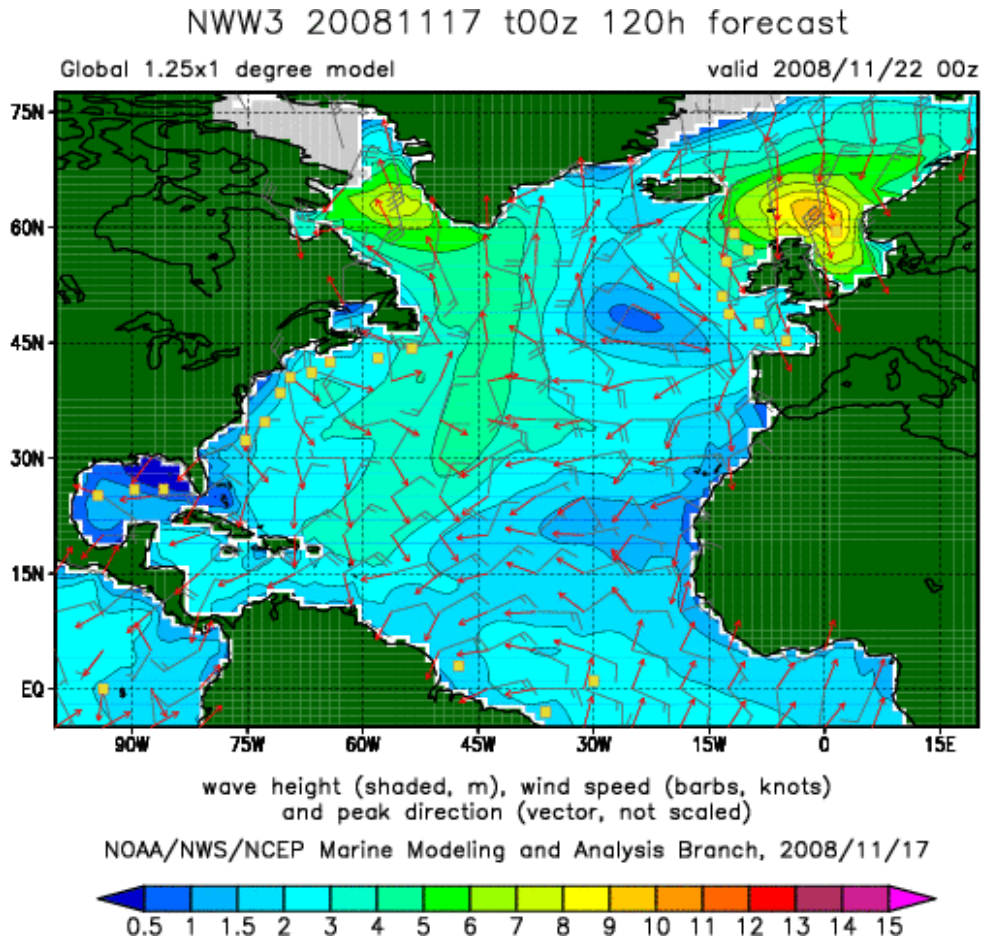


Abbildung 5.1: Plot einiger Elemente des Wave Watch III Modells

raum hinweg vergleichen, könnte man wahrscheinlich bestimmte Zusammenhänge feststellen. Beispielsweise ist es durchaus üblich, dass zwei benachbarte Spots sehr ähnlichen Wetter- und Wellenverhältnissen ausgesetzt sind, die Qualität der brechenden Wellen sich aber erheblich unterscheidet. Dies kann z.B. daran liegen, dass der eine Spot besser bei Flut und der andere besser bei Ebbe bricht. Manche Spots wiederum fangen erst bei einer bestimmten Wellenhöhe an zu brechen, andere hingegen sind bei zu großen Wellen nicht mehr surfbar, da die Wellen sofort in sich zusammenfallen. Vor Ort ansässige Surfer, sogenannte *Locals*, haben oft ein feines Gespür für diese Zusammenhänge, was wohl auf deren langjährige Beobachtungen zurückzuführen ist.

Diese Beobachtungen von den Benutzern der Web-Applikation vornehmen zu lassen, ist eher unrealistisch. Zum einen sind solche Beobachtungen

sehr aufwändig, und zum anderen müsste sichergestellt sein, dass diese diszipliniert und nach denselben Kriterien durchgeführt werden. Zudem stellt sich die Frage, wie diese Beobachtungen maschinell verwertet und in Verbindung mit den Vorhersagen gebracht werden können.

Abstimmung durch die Benutzer

Eine im Rahmen dieser Arbeit leider nicht weiter verfolgbare Idee, welche diese Problematik vielleicht lösen könnte, wäre die Einführung eines Abstimmungssystems in Kombination mit *Data-Mining*-Verfahren. Informiert sich ein Benutzer auf der Community Plattform über die Surfbedingungen an einem bestimmten Spot, könnte man ihn zu den Bedingungen in den letzten Tagen befragen. Mit etwas Glück kann es nämlich durchaus sein, dass der Benutzer dort surfen war und eine Aussage über die damalige Qualität der Wellen treffen kann. Diese Befragung sollte benutzerfreundlich und schnell vonstatten gehen, damit sich die Besucher nicht belästigt fühlen, sondern konstruktiv daran teilnehmen. Hier würde sich z.B. eine Bewertungsskala von 1 bis 5 "Sternen" mit einer Auswahl des entsprechenden Zeitpunktes anbieten. Die so erhobenen Datensätze sollten Informationen über den Spot, den Zeitpunkt zu dem der Benutzer surfen war, die Qualität der Wellen und den Benutzer selbst (optional) enthalten. Sammelt man viele dieser Bewertungen können diese statistisch ausgewertet und eine Aussage darüber getroffen werden, wie gut die Surfbedingungen in der Vergangenheit waren. Mit diesen Informationen könnte man z.B. die besten Spots in einer Region bestimmen oder die Konsistenz der Surfbedingungen an einem Spot ermitteln. Eine weitaus interessantere Verwendung dieser Daten wäre aber, sie für die Bewertung von zukünftigen Surfbedingungen einzusetzen.

Bewertung zukünftiger Surfbedingungen

Es wäre wünschenswert, wenn man einem Benutzer neben den Wetter- und Wellenvorhersagen auch noch eine Einschätzung zur Qualität der Surfbedingungen geben könnte, in der die lokalen Gegebenheiten eines Spots mit berücksichtigt sind. Diese Einschätzung sollte dem Benutzer in der ihm bekannten Bewertungsskala aus dem Abstimmungssystem präsentiert werden. Je besser die Surfbedingungen, desto mehr "Sterne". Der zugrunde liegende Ansatz besteht in der Vermutung, dass ähnliche Wetter- und Wellenvorhersagen auch ähnliche Surfbedingungen provozieren. Wurden für einen Zeitpunkt in der Vergangenheit positive Bewertung abgegeben, sind die Surfbedingungen zu einem zukünftigen Zeitpunkt mit ähnlichen Wetter- und Wellenverhältnissen vielleicht auch gut, und umgekehrt. Durch den Vergleich historischer

Vorhersagedaten mit zukünftigen, und einer Strategie aus den abgegebenen Bewertungen, welche für die Zukunft abzuleiten, könnte man dem Nutzer Aussagen zur Qualität zukünftiger Surfbedingungen anbieten.

Bewertungen als Klassifikationsproblem

Die hier zu lösende Aufgabe ist ein sogenanntes Klassifikationsproblem. Objekte einer Menge, deren Klassenzugehörigkeit nicht bekannt ist, müssen dabei einer bestimmten Klasse zugeordnet werden. Als Klassifikator bezeichnet man den Algorithmus oder das Verfahren, nach dem diese Zuordnung durchgeführt wird. Klassen würden hier durch die in mehrere Abschnitte unterteilte Bewertungsskala repräsentiert, Objekte durch die Vorhersagen an den Spots. Die Bewertung der Surfbedingungen würde sich dann aus der Klasse ergeben, der eine Vorhersage zugeordnet wurde. Wüsste man, wie sich die Variablen einer Vorhersage auf die Qualität der Wellen auswirken, könnte man für jeden Spot einen Klassifikator konstruieren, der idealerweise auch noch die lokalen Gegebenheiten mit einbezieht. Die Vorschriften, wie solch ein Klassifikator jedoch konstruiert wird, sind jedoch meistens nicht bekannt bzw. nur mit hohem Aufwand zu ermitteln.

Anwendung von Data-Mining-Verfahren

Unter *Data Mining* versteht man die Anwendung verschiedenster Algorithmen auf einem Datenbestand, mit dem Ziel meist noch nicht bekannte Muster zu entdecken. Neben vielen Anwendungsbeispielen ist in [Seg07] nicht nur eine Zusammenfassung der in diesem Gebiet häufig verwendeten Algorithmen zu finden, sondern auch eine Diskussion über deren Vor- und Nachteile. Einige der dort vorgestellten Algorithmen können dazu benutzt werden, Klassifikationsprobleme zu lösen und bedienen sich Techniken aus dem Bereich des maschinellen Lernens. Mit diesen Verfahren könnte versucht werden die Zusammenhänge zwischen den Vorhersagen, den lokalen Gegebenheiten und der Qualität der Surfbedingungen zu finden und daraus zukünftige Bewertungen abzuleiten. *Bayesian Classifier*, *Decision Trees*, und *Neuronale Netzwerke* sind nur einige der dort vorgestellten Algorithmen, mit denen man versuchen könnte, dieses Problem zu lösen.

Supervised Learning

Bei Algorithmen aus dem Bereich des maschinellen Lernens unterscheidet man zwischen Verfahren des *Supervised-* und *Unsupervised Learning*. Beim *Supervised Learning* werden sogenannte Trainingsdaten benötigt, die von

den Algorithmen analysiert und zur Lösung des Problems herangezogen werden. Die Trainingsdaten für Klassifikationsprobleme bestehen dabei aus einer Menge von Objekten, deren Klassenzugehörigkeit bekannt ist. Die zuvor durch ein Abstimmungssystem erhobenen Daten können nicht nur für statistische Zwecke verwendet werden, sondern würden sich auch als Trainingsdaten für diese Art von Algorithmen anbieten. Die Vorhersagedaten wären dabei die Attribute eines Objekts, und die Klassenzugehörigkeit des Objekts ist durch die Bewertung des Benutzers festgelegt.

Support Vector Machine

Der unter dem Namen *Support Vector Machine* bekannte Algorithmus scheint ein viel versprechender Kandidat zu sein, um das hier beschriebene Klassifikationsproblem zu lösen. Zum einen ist der Algorithmus bei der Klassifikation von Objekten sehr schnell, und zum anderen liefert er auch bei einer hohen Anzahl von Dimensionen recht zuverlässige Ergebnisse. Als Dimension bezeichnet man die Attribute eines zu klassifizierenden Objekts. Bei dem hier zu lösenden Problem wären dies die Elemente einer Vorhersage und evtl. weitere bekannte Eigenschaften eines Spots, die sich auf die Qualität der Surfbedingungen auswirken könnten.

Um nun die Bewertungen zur Qualität der Surfbedingungen zu berechnen, würde man für jeden Spot eine separate Instanz des Algorithmus erzeugen und diese mit den bereits abgegebenen Bewertungen aus der Vergangenheit trainieren. Jede Instanz wird dabei nur mit denjenigen Vorhersagedaten und Bewertungen trainiert, die auch zu dem entsprechenden Spot gehören. Je mehr Trainingsdaten vorhanden sind, desto besser sind die Ergebnisse einer solchen Instanz. Da die Trainingsphase aufwändiger als die eigentliche Klassifikation ist und bei hinzukommenden Trainingsdaten die Instanz erneut mit allen Trainingsdaten initialisiert werden muss, bieten die meisten Implementierungen des Algorithmus die Möglichkeit, das Ergebnis der Trainingsphase in einem sogenannten Modell zu speichern und wieder zu laden.

Zur Berechnung einer Bewertung wird eine Vorhersage der entsprechenden Instanz des Algorithmus übergeben, der dann das Ergebnis unter Berücksichtigung der Trainingsdaten berechnet. Um dies Berechnung in die Web-Applikation zu integrieren würde man beispielsweise einmal täglich für jeden Spot eine Instanz des Algorithmus mit den zugehörigen Trainingsdaten initialisieren und abspeichern. Da die Klassifikation einer Vorhersage schnell zu berechnen ist, könnte die Bewertung sogar bei einem Seitenaufruf der Web-Applikation dynamisch generiert werden. Alternativ könnte man die Berechnung auch in den *ETL*-Prozess integrieren und alle Bewertungen in der Datenbank vorhalten.

Anzumerken ist hier noch, dass der *Support Vector Machine* Algorithmus ein sogenanntes *Black-Box* Verfahren [Seg07][S.292] ist, d.h. die lokalen Gegebenheiten eines Spots fließen hier indirekt über die erhobenen Abstimmungsdaten mit in die Bewertung ein. Welche Zusammenhänge allerdings zwischen den Vorhersagedaten und den lokalen Gegebenheiten bestehen, können mit diesem Verfahren nicht ermittelt werden.

Erfolgsaussichten und Hindernisse

Damit mit dem Algorithmus gute Ergebnisse erzielt werden, benötigt man allerdings eine größere Menge an Trainingsdaten. Da diese zum jetzigen Zeitpunkt nicht vorhanden sind, kann dieser Ansatz hier leider nicht weiter verfolgt werden. Diese müssten im produktiven Betrieb der Web Applikation über einen längeren Zeitraum durch die Benutzer erstellt werden.

Ob dieses Verfahren überhaupt zum Ziel führt, müsste durch eine Auswertung der Ergebnisse an verschiedenen Spots überprüft werden. Außerdem werden dem Algorithmus bei der Initialisierung noch einige Parameter übergeben, die sich auf die Berechnung der Bewertungen auswirken. Diese Parameter variieren von Problem zu Problem, können aber durch ein Verfahren das sich *Crossvalidation* nennt ermittelt werden. Eine ausführliche Auswertung und Probephase ist deshalb auf jeden Fall erforderlich, um die Erfolgsaussichten einschätzen zu können. Falls die Ergebnisse einer solchen Auswertung jedoch positiv ausfallen, hätte man einen Weg gefunden, die lokalen Gegebenheiten eines Spots indirekt über die Trainingsdaten in Form einer Bewertung in die Vorhersagen mit einfließen zu lassen.

Selbstständigkeits- und Einverständniserklärung

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, 7. Dezember 2009

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des *Instituts für Informatik* der *Humboldt-Universität zu Berlin* ausgestellt werden darf.

Berlin, 7. Dezember 2009

Abkürzungsverzeichnis

ACID	Atomicity, Consistency, Isolation, Durability
CBS	Commission for Basic Systems
CRUD	Create, Read, Update, Delete
CSV	Comma-Separated Values
DRY	Don't Repeat Yourself
GFS	Global Forecast System
GRIB	Gridded Binary
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICON	Icosahedral Non-hydrostatic General Circulation Model
JSON	Javascript Object Notation
MVC	Model View Controller
NOMADS	NOAA Operational Model Archive and Distribution System
NWS	National Weather Service
REST	Representational State Transfer
RFC	Request for Comments
URI	Uniform Resource Identifier
WMO	World Meteorological Organization
WRF	Weather Research and Forecasting
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language

Literaturverzeichnis

[BS98] B. Sutherland, O. Fitzjones und D. Haylock: *The Stormrider Guide Europe*. Low Pressure Publishing Ltd., 1998.

[cuc] *Cucumber - Behaviour Driven Development with elegance and joy*. `http://cukes.info`.

[FBB+99] Fowler, Martin, Kent Beck, John Brant, William Opdyke und Don Roberts: *Refactoring: Improving the Design of Existing Code (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, Juli 1999.

[Fie00] Fielding, Roy Thomas: *Architectural Styles and the Design of Network-based Software Architectures*. Doktorarbeit, University of California, Irvine, 2000.

[GHJV94] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Januar 1994.

[HT99] Hunt, Andrew und David Thomas: *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, Oktober 1999.

[Pos09a] PostgreSQL, Global Development Group: *PostgreSQL 8.4.1 Documentation - Performance Tips*. `http://www.postgresql.org/docs/8.4/interactive/performance-tips.html`, 2009.

[Pos09b] PostgreSQL, Global Development Group: *PostgreSQL 8.4.1 Documentation - Populating a Database*. `http://www.postgresql.org/docs/8.4/interactive/populate.html`, 2009.

[RR07] Richardson, Leonard und Sam Ruby: *RESTful Web Services*. O'Reilly Media, Inc., Mai 2007.

[rsp] *RSpec - The original Behaviour Driven Development framework for Ruby*. http://rspec.info.

[Seg07] Segaran, Toby: *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly Media, Inc., August 2007.

[Ven09] Veness, Chris: *Movable Type Scripts*. http://www.movable-type.co.uk/scripts/latlong.html, 2009.

[Wik09a] Wikipedia: *Agile Softwareentwicklung — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=Agile_Softwareentwicklung&oldid=61264582, 2009.

[Wik09b] Wikipedia: *Behavior Driven Development — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Behavior_Driven_Development&oldid=296178408, 2009.

[Wik09c] Wikipedia: *Haversine formula — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Haversine_formula&oldid=294524876, 2009.

[Wil09] Williams, Ed: *Aviation Formulary*. http://williams.best.vwh.net/avform.htm, 2009.