



Entwicklung einer Schnittstelle zwischen SAP Sailing Analytics und SAP Standardanalysetools

Bachelorarbeit

des Studienganges Angewandte Informatik
an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Lennart Hensler

26. August 2013

Kurs
Betreuer
Gutachter

TAI10B1
Daniel Lindner
Dr. Axel Uhl

Erklärung

gemäß § 5 (2) der „Studien- und Prüfungsordnung DHBW Technik“ vom 18. Mai 2009.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Karlsruhe, 26. August 2013

Lennart Hensler

Sperrvermerk

Die nachfolgende Praxisarbeit enthält vertrauliche Daten der:

SAP AG

Dietmar-Hopp-Allee 16

69190 Walldorf, Germany

Sie darf als Leistungsnachweis des Studiengangs Angewandte Informatik 2010 an der Duale Hochschule Karlsruhe verwendet werden und nur zu Prüfungszwecken zugänglich gemacht werden.

Über den Inhalt ist Stillschweigen zu bewahren. Veröffentlichungen oder Vervielfältigungen der Praxisarbeit – auch auszugsweise – sind ohne ausdrückliche Genehmigung der SAP AG nicht gestattet.

SAP und die SAP Logos sind eingetragene Warenzeichen der SAP AG. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in dieser Arbeit berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedem benutzt werden dürfen.

Karlsruhe, 26. August 2013

Lennart Hensler

Zusammenfassung

In dieser arbeit geht es darum wie man das SAP Standardanalysetool *BusinessObjects Explorer* mit den SAP Sailing Analytics zu verbinden. Dabei ist zu beachten, dass die Daten der Sailing Analytics nicht in einer Datenbank liegen, sondern nur zur Laufzeit in Form eines Objektfeldes vorhanden sind. Zunächst gebe ich einen kurzen Einblick in den Segelsport und in die für diese Arbeit wichtigen Komponenten der Sailing Analytics. Anschließend stelle ich einen theoretischen Entwurf für die Umsetzung der Aufgabe vor und gehe daraufhin kurz auf meine Arbeitsweise ein. Zuletzt stelle ich das Ergebnis meiner Arbeit - ein Datenanalyse-Framework für die Sailing Analytics - vor und gebe einen Ausblick, wie damit in Zukunft weitergearbeitet werden könnte.

Summary

This bachelor thesis is about how a connection between the standard SAP data mining tool *BusinessObjects Explorer* and the SAP Sailing Analytics could be established. The data of the Sailing Analytics isn't stored in a database, but is only available in through an object field during runtime. At first I will deliver insight into sailing as a sport and the components of the Sailing Analytics, which are relevant in the context of this paper. Afterwards I introduce a theoretical draft for the implementation of this task and I explain, how I am going to do my work. At last I will present the result - a data mining framework for the Sailing Analytics - and I give a forecast, what can be done with this framework in the future.

Inhaltsverzeichnis

1. Einleitung	1
1.1. Der Segelsport	1
1.2. SAP Sailing Analytics	11
1.3. SAP BusinessObjects Explorer	26
2. Motivation	29
3. Entwurf	30
3.1. Verbindung mit dem BOBJ-Explorer	30
3.2. Prinzip der Datenanalyse	32
4. Umsetzung	35
5. Ergebnis	37
5.1. Datenbeschaffung, Filterung und Gruppierung	39
5.2. Extraktion und Aggregation der Statistiken	44
5.3. Datenanalyse über die GPS-Daten	47
5.4. Performance des Frameworks	52
6. Ausblick	57
6.1. Erweiterung der Analysemöglichkeiten	57
6.2. Optimierung der Performance	59
6.3. Verbindung zwischen den Sailing Analytics und dem BOBJ-Explorer . .	60
6.4. Erweiterung der Benutzeroberfläche	60
6.5. Weitere dynamische Komponenten	62
7. Fazit	65
Abbildungsverzeichnis	i
Listings	iii
Literaturverzeichnis	iv

A. Appendix	v
A.1. Implementierung der Komponenten	v
A.2. Dimensionen der GPS-Daten	vii
A.3. Ergebnisse der Datenanalyse-Benchmarks	viii

1. Einleitung

In diesem Kapitel gebe ich eine Einführung in die Umgebung dieser Arbeit. Dabei gehe zunächst auf den Segelsport ein und erläutere was im Kontext dieser Arbeit darunter verstanden wird. Anschließend wird die SAP Sailing Analytics betrachtet, wobei ich deren Komponenten und das zu Grunde liegende Datenmodell vorstellen werde. Zuletzt wird das Datenanalysewerkzeug *BusinessObjects Explorer* gezeigt.

1.1. Der Segelsport

In diesem Abschnitt wird der Kontext des Segelsports im Zusammenhang mit den Sailing Analytics erläutert. Man unterscheidet dabei zwischen Regattasegeln und Fahrtensegeln. Unter letzterem versteht man längere Fahrten, deren Dauer von wenigen Tagen bis zu mehreren Jahren sein kann. Diese sind auch nicht immer im Format eines Wettkampfes, sondern können auch als Herausforderungen gesehen werden (wie zum Beispiel bei einer Weltumsegelung) oder als Freizeit praktiziert werden. Diese Form des Segelns wird weniger im Zusammenhang mit der Sailing Analytics betrachtet. Lediglich kürzere Fahrten mit Wettkampfcharakter werden zur Zeit von der Software unterstützt. Diese ähneln dabei stark der Form des Regattasegeln, weswegen ich diese zukünftig als eine Art des Segelns betrachten werde.

Das Regattasegeln ist eine Wettkampfsportart, bei der mehrere Teilnehmer dieselbe Strecke befahren und um die beste Platzierung kämpfen. Die Sailing Analytics nimmt dabei eine unterstützende Rolle ein worauf ich in Abschnitt 1.2 *SAP Sailing Analytics* genauer eingehen werde.

1.1.1. Struktur einer Regatta

Eine Regatta wird von einer Menge von Teilnehmern bestritten, wobei in der Regel alle Segler Boote der selben Bootsklasse führen. Ausnahmen stellen hier Regatten auf offener See dar (Fahrtensegeln mit Wettkampfcharakter, welche auch als *Offshore*-Regatten bezeichnet werden), bei denen unterschiedliche Bootsklassen teilnehmen dürfen, die dieselben (von der Wettfahrtleitung definierten) Kriterien erfüllen. Alle Segler fahren auf einem vorher festgelegten Kurs, der durch Bojen abgesteckt ist. Diesen versuchen die Teilnehmer so schnell wie möglich zu absolvieren. Beispiele der unterschiedlichen Kurse und Bootsklassen befinden sich im nachfolgenden Unterabschnitt. Eine Durchfahrt des Kurses stellt ein Rennen dar, welches jeder Segler mit einer Platzierung abschließt, der bei der Bepunktung (s. Unterabschnitt 1.1.2 *Punktesysteme beim Regattasegeln* weiter unten) eine Rolle spielt. Je nach Kurs kann eine Durchfahrt zwischen unter einer Stunde und mehreren Tagen (*Offshore*-Rennen) beanspruchen. Eine Regatta besteht nun aus mehreren Rennen, die hintereinander durchgeführt werden, wobei deren Anzahl unterschiedlich ist. Bei langen *Offshore*-Rennen kann eine Regatta auch nur aus einem einzigen bestehen, während bei normalen Regatten eher 12 Rennen gefahren werden. In der Regel werden diese an mehreren Tagen ausgeführt. Dabei ist es möglich die Rennen in mehrere Serien zu unterteilen. Zum Beispiel in die Serie *Qualifikation* und *Finale* welche jeweils aus mehreren Rennen bestehen (mehr Qualifikationsrennen als Finalrennen). Der Sinn dahinter ist es, die Anzahl der Teilnehmer welche am *Finale* teilnehmen zu reduzieren, da zum Beispiel nur die besten 50% der Qualifikationsrennen bei den Finalrennen starten dürfen.

Bei vielen Teilnehmern besteht weiterhin die Möglichkeit, diese in unterschiedliche Flotten zu unterteilen, da der Kurs sonst zu voll werden würde und ein Rennen aus Platzgründen nicht zustande kommen würde. Gibt es dabei mehrere Serien, hat das Auswirkungen auf die Zuordnung der Teilnehmer auf das *Finale*. Man nehme an, die Serie *Qualifikationen* hätte nun die Flotten *Blau* und *Gelb*, während die Finalrennen in *Gold* und *Silber* unterteilt sind. Dabei soll die goldene Flotte höherwertig sein als die silberne. Deswegen kommen jeweils die oberen 50% von *Blau* und *Gelb* in die goldene und die unteren in die silberne Flotte. Des weiteren wird der beste Teilnehmer in *Silber* auf jeden Fall als schlechter gewertet, als der schlechteste in *Gold*.

Zusammenfassend besteht eine Regatta aus mindestens einer Serie, welche eine oder mehrere Flotten enthält, denen die Teilnehmer zugeordnet werden. In jeder Serie wird für jede Flotte die festgelegte Anzahl an Rennen durchgeführt. Ein schematische Dar-

stellung dieser Struktur findet sich in der nachfolgenden Abbildung.

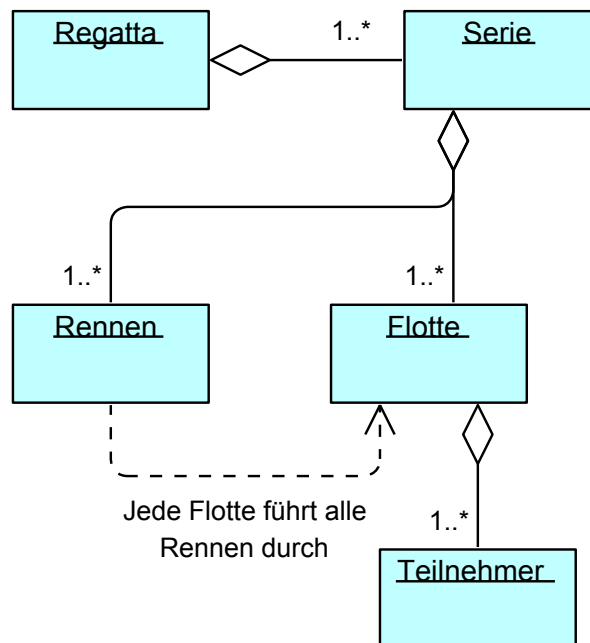


Abbildung 1.1.: Schematische Struktur einer Regatta

1.1.2. Punktesysteme beim Regattasegeln

In diesem Unterabschnitt werden die grundlegenden Regeln zur Bepunktung beim Regattasegeln erläutert. Das vollständige Regelwerk findet sich in [ISAF12], welches regelmäßig von der *International Sailing Federation*¹ (ISAF) überarbeitet und herausgegeben wird.

Je nach der Platzierung mit der ein Teilnehmer ein Rennen abgeschlossen hat, erhält er eine gewisse Punktzahl. Dabei unterscheidet man zwischen Niedrigpunkt-Systemen und Hochpunkt-Systemen. Bei ersterem erhält der Erstplatzierte einen Punkt, der Zweitplatzierte zwei Punkte und so weiter. Bei Hochpunkt-Systemen würde der Erste stattdessen so viele Punkte erhalten, wie es Teilnehmer gibt und der Zweite die Anzahl abzüglich Eins, was für alle nachfolgenden Teilnehmer so fortgeführt wird. Dabei kann es weitere Sonderregeln geben, wie zum Beispiel, dass der Erstplatzierte statt einem Punkt, null Punkte erhält oder dass man nach einer gewissen Anzahl an Rennen das

¹ Homepage: <http://www.sailing.org/>
Die neuste Version des Regelwerks findet sich unter <http://www.sailing.org/documents/racingrules/index.php>

schlechteste Ergebnis streichen kann, sodass es nicht in der Gesamtplatzierung berücksichtigt wird. Diese ergibt sich aus der Summe der Punkte aller Rennen, wobei die Streicher ignoriert werden. Der Sieger ist der Teilnehmer, welcher nach allen Rennen - je nach System - die wenigsten/meisten Punkte gesammelt hat.

Die Strukturierung der Regatten nach Serien kann dabei Auswirkungen auf die Gesamtplatzierung haben. Zum Beispiel kann man ab der Serie *Finale* erneut bei null Punkten beginnen, sodass die Punkte der Qualifikationsrennen nicht in die Gesamtplatzierung zählen. Weiterhin kann es eine Übertragspalte nach der *Qualifikation* geben, in der die aktuelle Gesamtplatzierung nach den Qualifikationsrennen als Punkte eingetragen werden. Diese Übertragspalte wird zusammen mit den anderen Ergebnissen aufsummiert, woraus sich die Gesamtplatzierung ergibt. Dadurch fließt die Leistung der Teilnehmer in den Qualifikationsrennen stärker in die Gesamtplatzierung ein. Diese beiden Sonderregeln lassen sich auch kombinieren, sodass sich die erreichte Leistung der *Qualifikation* ausschließlich durch die Übertragspalte auf die Gesamtplatzierung auswirkt. Des weiteren gibt es Regeln die gelten, falls ein Teilnehmer nicht gestartet ist oder das Ziel nicht erreicht hat, da er zum Beispiel gekentert ist. Ist dies der Fall erhält der Teilnehmer in der Regel so viele Punkte, wie sich Teilnehmer für die Regatta registriert haben. Dabei werden nicht die konkreten Punkte in die Tabelle eingetragen, sondern es werden Abkürzungen verwendet, die beschreiben was vorgefallen ist. Beispiele dafür sind *DNF*, falls der Teilnehmer das Rennen nicht beenden konnte, *DNS*, falls er nicht gestartet ist oder *DSQ*, falls er disqualifiziert wurde. Eine vollständige Liste der Abkürzungen findet sich in [ISAF12, Seite 51].

Beispiele für Regatten

Nachdem nun die Struktur einer Regatta und die Bepunktungsregeln definiert wurden, möchte ich zum besseren Verständnis konkrete Beispiele für Regatten geben:

Regatta Kieler Woche 2013 - Internationale Bootsklasse Hobie 16

Serie Standard

Erster Streicher nach dem fünften Rennen.

Rennen 11 Rennen

Flotte Standard

Einfachste Form einer Regatta mit einer Serie, die 11 Rennen enthält. Nach dem fünften Rennen darf das Schlechteste gestrichen werden.

Regatta Kieler Woche 2013 - Internationale Bootsklasse 29er

Serie Qualifikation

Erster Streicher nach dem fünften Rennen.

Rennen 7 Qualifikationsrennen

Flotte Blau

Gelb

Serie Finale

Rennen 7 Finalrennen

Flotte Gold

Silber

Regatta mit den Serien *Qualifikation* und *Finale*, in denen jeweils sieben Rennen stattfinden. Nach fünf abgeschlossenen Qualifikationsrennen darf das schlechteste gestrichen werden. Im Finale sind keine Streicher erlaubt. Jede Serie hat zwei Flotten, welche nach dem oben beschriebenen Schema funktionieren.

Regatta Kieler Woche 2013 - Olympische Bootsklasse 49er

Serie Qualifikation

Erster Streicher nach dem fünften Rennen.

Rennen 8 Qualifikationsrennen

Flotte Blau

Gelb

Serie Finale

Erster Streicher nach dem fünften Rennen.

Hat Übertragungspalte der Qualifikationsrennen.

Rennen 8 Finalrennen

Flotte Gold

Silber

Serie Medaille

Hat Übertragspalte der Finalrennen.

Rennen 3 Medaillenrennen

Flotte Standard

Ähnlich zu der *29er*-Regatta, allerdings ist in der Final-Serie nun auch ein Streicher nach dem fünften Rennen erlaubt und die Qualifikationsrennen werden zusätzlich mit einer Übertragspalte berücksichtigt. Außerdem gibt es eine dritte Serie mit drei Rennen und der Übertragspalte der Finalrennen. Dadurch, dass diese nur noch die Standard-Flotte besitzt werden nur die besten Teilnehmer der goldenen und silbernen Finalrennen zum Start zugelassen. Hier gilt wieder, dass der schlechteste Segler von *Medaille* vor dem besten nicht in zu den Medaillenrennen zugelassenen Teilnehmer platziert ist.

1.1.3. Kurse

Die Form des Rennkurses entscheidet welche Taktiken die Teilnehmer anwenden sollten, um möglichst gut abzuschneiden. Um eine bessere Vorstellung zu bekommen, wie ein Rennen aussieht werden in diesem Unterabschnitt einige typische Kurse vorgestellt.

Ein Kurs wird durch mehrere Bojen markiert, welche in einer bestimmten Reihenfolge und auf einer festgelegten Seite umfahren werden müssen. Der Abschnitt zwischen zwei Bojen wird als Schenkel bezeichnet, welche das Rennen weiter unterteilen. Ein Schenkel beginnt sobald der erste Teilnehmer dessen Start-Boje umrundet hat.

Üblicherweise werden die Bojen so gelegt, dass die Teilnehmer gegen den Wind starten und somit den ersten Schenkel durch Kreuzen¹ absolvieren müssen. Zu den Bojen kommen noch ein Start- und optional² ein Ziel-Schiff, welche in unmittelbarer Nähe zur ersten/letzten Boje vor Anker liegen. Aus Start-/Ziel-Schiff und der ersten/letzten Boje ergeben sich die Start- und Ziellinie. Die Aufgabe der Mannschaft auf dem Schiff ist es, Frühstarts und andere Fehler in der Startphase zu erkennen und entsprechend einzugreifen, beziehungsweise die Reihenfolge der ins Ziel einlaufenden Teilnehmer und somit deren Punkte zu erfassen.

¹ Kreuzen bedeutet ein Ziel im „Zickzackkurs“ anzulaufen, welches im Wind liegt und deshalb nicht geradeaus angefahren werden kann (s. [DENK04, Seite 169]).

² Der Kurs kann auch so gelegt werden, dass Start und Ziel durch das gleiche Schiff markiert werden.

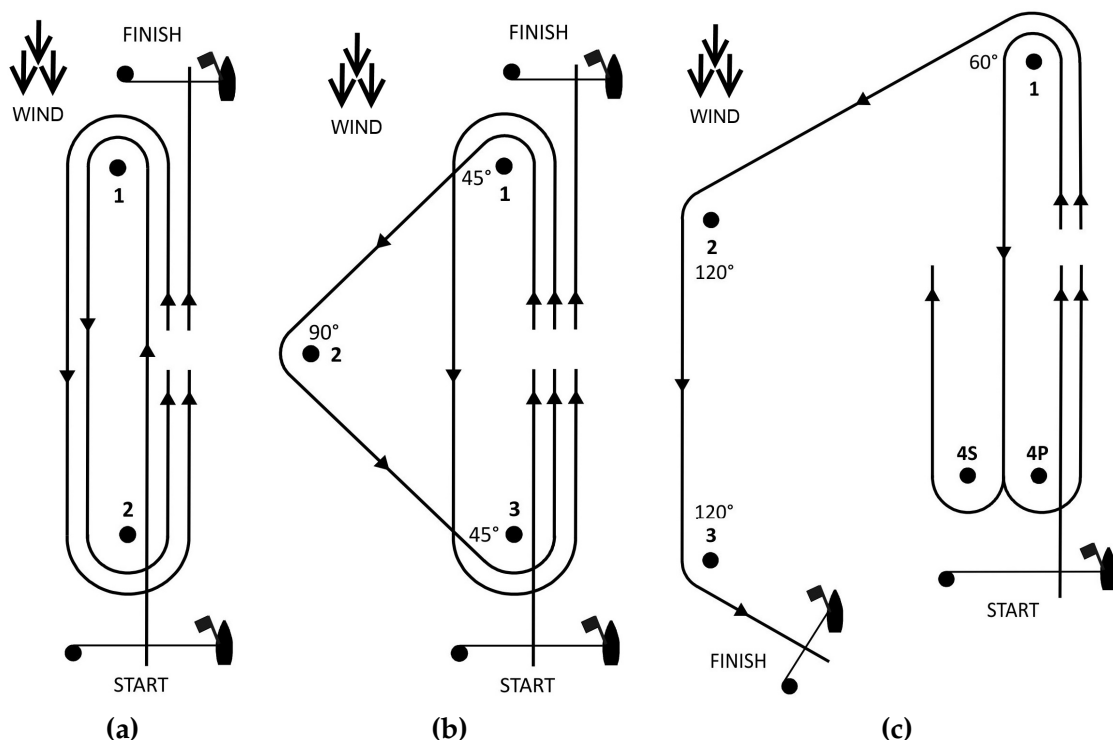


Abbildung 1.2.: Beispiele möglicher Rennkurse

(a) Windwärtiger-Leewärtiger-Kurs

(b) Dreiecks-Kurs

(c) Trapez-Kurs

Quelle: [ISAF12], Windrichtung nicht im Original enthalten

Eine sehr einfache Kursauslage ist der *Windwärtige-Leewärtige*¹- oder auch *Up-and-Down*-Kurs, welcher zum Beispiel aus vier Bojen, sowie dem Start- und Ziel-Schiff besteht. Zwei Bojen markieren zusammen mit den Schiffen Start- und Ziellinie. Die übrigen Bojen werden so auf einer Linie übereinander verlegt, sodass diese dieselbe Richtung wie der Wind haben. Die Bojen welche nicht die Start- und Ziellinie bilden werden zukünftig als kursgebende Bojen bezeichnet. Eine möglicher Kurs mit dieser Auslage wäre nun *Start - 1 - 2 - 1 - 2 - Ziel*, woraus sich fünf Schenkel ergeben. Drei davon erfolgen entgegen dem und zwei mit dem Wind.

Eine Abwandlung davon nennt sich *Dreiecks-Kurs* (früher auch *Olympisches Dreieck* genannt, da er bei Rennen olympischer Bootsklassen verwendet wurde). Hierbei wird eine dritte kursgebende Boje ausgelegt, sodass diese mit den anderen beiden ein (in diesem Fall gleichschenkliges) Dreieck bilden. Bei dieser Kursauslage wäre der Kurs zum Beispiel *Start - 1 - 2 - 3 - 1 - 3 - Ziel*. Damit hat man einen Schenkel mehr als mit

¹ Windwärtig: Zum Wind gerichtet
Leewärtig: Vom Wind weg gerichtet

einem *Windwärtig-Leewärtigen*-Kurs und die Schenkel von Boje 1 zu 2 und von 2 zu 3 liegen eher seitlich zum Wind, was mehr taktische Tiefe in das Rennen bringt.

Bei dem *Trapez*-Kurs werden die kursgebenden Bojen trapezförmig ausgelegt. Dadurch ist der Winkel des auf den Schenkeln zwischen den Bojen 1 und 2, sowie 3 und *Ziel* zu fahrendem Kurs steiler zum Wind als beim *Dreiecks*-Kurs. Außerdem liegt die Ziellinie nun in der Nähe von der Startlinie und nicht mehr gegenüber. Den *Trapez*-Kurs gibt es in zwei unterschiedlichen Varianten, wobei die hier Gezeigte ein sogenanntes Tor (im englischen *Gate*) verwendet (eine Abbildung der anderen Variante findet sich in [ISAF12, Seite 143]). Ein Tor ist ein Wegpunkt, welcher aus zwei Bojen besteht. Die Teilnehmer haben dabei die Wahl, ob sie diesen auf der rechten oder linken Seite verlassen.

1.1.4. Bootsklassen

Boote gleicher Bauart werden in Klassen unterteilt. Damit ein Boot zu einer bestimmten Klasse gehört, muss dieses festgelegte Kriterien erfüllen. Diese können unterschiedlich streng sein und von Richtlinien für die einzelnen Bauteile bis zu kompletten Bauplänen reichen. Bootsklassen wiederum werden in Gruppen zusammengefasst, welche zum Beispiel *Einhand-/Zweimann-Jollen*, *Mehrrumpfboote*, *Kielboote* oder *Yachten* sind. Zudem können Bootsklassen einen internationalen oder olympischen Status erlangen. Im Kontext dieser Arbeit haben Boote mit diesen Status die größte Relevanz, da die meisten Regatten mit diesen Klassen ausgeführt werden.

Die verschiedenen Bootsklassen unterscheiden sich in Form und Größe des Rumpfes, der Länge des Mastes (und damit dem Segel) sowie weiteren Merkmalen, wovon ich die markantesten kurz vorstellen möchte (eine schematische Darstellung findet sich in Abbildung 1.3 unter den Beschreibungen):

- **Kielboote** haben an ihrer Unterseite einen schweren Ballastkiel, der meist aus Blei oder Eisen besteht und durchaus mehr als 50% des Gewichts ausmachen kann. Dadurch wird der Schwerpunkt des Bootes nach unten verlagert, was für mehr Stabilität sorgt.
- **Jollen** haben einen breiten Rumpf, an dessen Unterseite ein Schwert ins Wasser ragt. Ihr Schwerpunkt liegt dadurch über der Wasseroberfläche, weswegen Jollen bei starkem Wind oder in Böen anfangen zu krängen¹. Damit lassen sich höhere

¹ Ein Teil des Rumpfes hebt sich von der Wasseroberfläche ab.

Geschwindigkeiten erreichen, da die Reibung reduziert wird, allerdings geht das zu Lasten der Stabilität, sodass Jollen leichter kentern, wenn die Mannschaft nicht rechtzeitig reagiert und den Schwerpunkt mit dem eigenen Körpergewicht stabilisiert.

- **Mehrrumpfboote** haben wie der Name schon sagt mehr als einen Rumpf, welche parallel zueinander stehen und schmaler sind als der von Kielbooten oder Jollen. Boote mit zwei zueinander symmetrischen Rümpfen werden als Katamaran bezeichnet. Des weiteren gibt es Trimarane, welche einen Hauptrumpf und zwei schmalere Auslegerrümpfe besitzen. Durch die geringe Kontaktfläche mit dem Wasser können Mehrrumpfboote hohe Geschwindigkeiten erreichen. Da durch den Abstand der Rümpfe eine große Breite erreicht wird, sind sie dabei dennoch stabil.

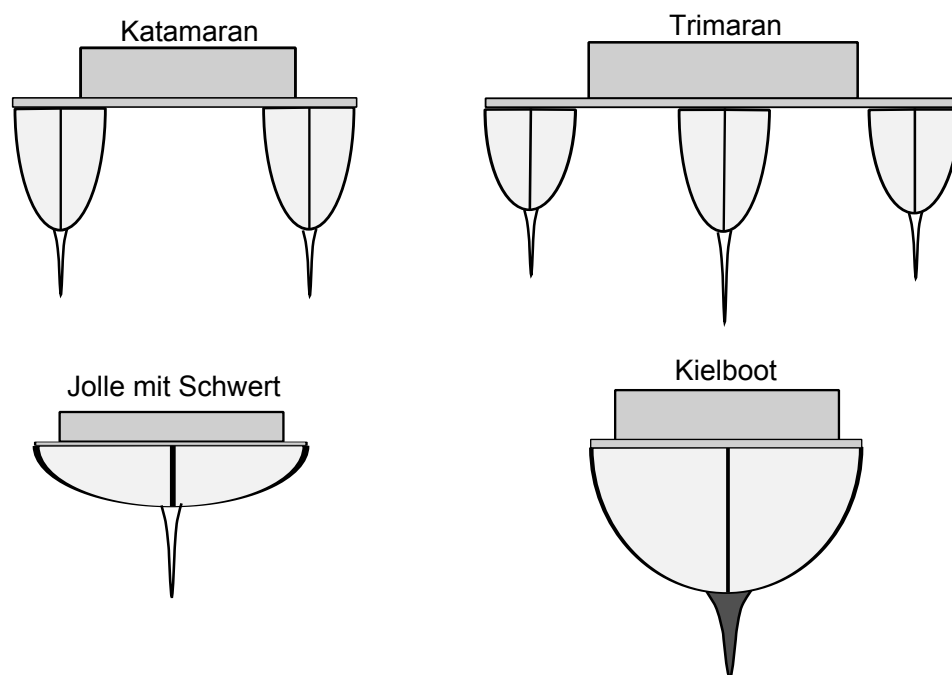


Abbildung 1.3.: Schematischer Querschnitt der Rümpfe von unterschiedlichen Bootsklassen

Quelle der ursprünglichen Grafik: https://commons.wikimedia.org/wiki/File:Multihull_de.svg

Internationale Bootsklassen müssen von der *ISAF* anerkannt werden und deren Kriterien [ISAF13] erfüllen. Zum Beispiel müssen mit Booten dieser Klassen regelmäßig Regatten durchgeführt werden, was jährlich nachzuweisen ist. Diese Regatten müssen zudem in unterschiedlichen Ländern und Kontinenten stattfinden. Klassen die diesen Regularien entsprechen, sind unter anderen *29er* (Zweimann-Jolle), *Hobie 16* (Katamaran) oder *Star* (Kielboot).

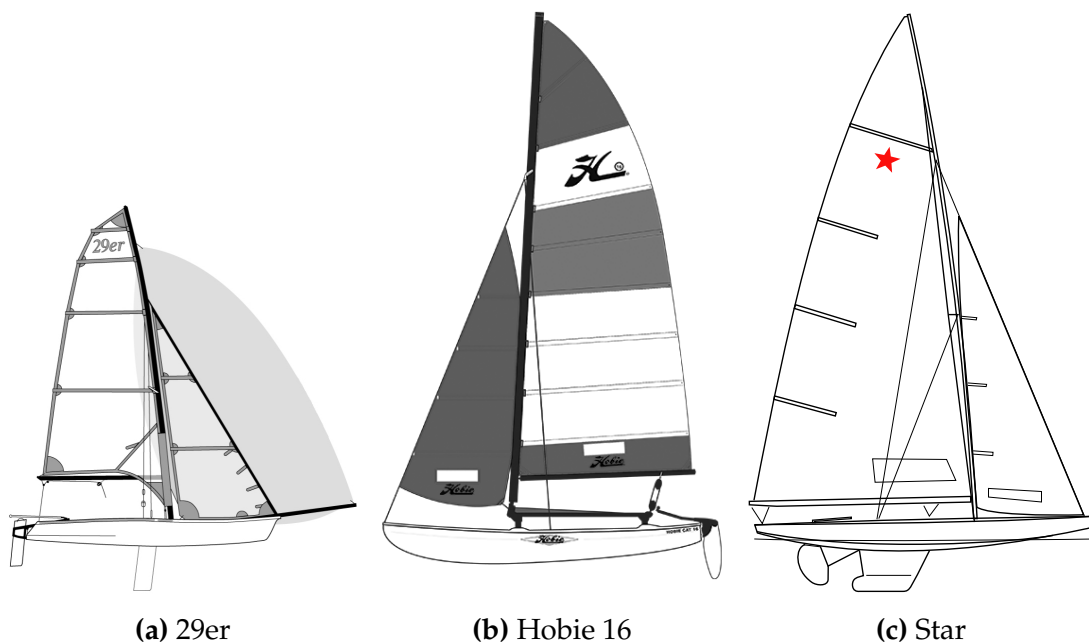


Abbildung 1.4.: Segelrisse internationaler Bootsklassen

Quellen:

(a) <http://www.js-steckborn.ch/index.php/flotte.html>

(b) <http://www.hobiecat.com/sail/hobie-16/specs/>

(c) [https://commons.wikimedia.org/wiki/File:Star_\(keelboat\).svg](https://commons.wikimedia.org/wiki/File:Star_(keelboat).svg)

Olympische Bootsklassen werden von der *ISAF* benannt und dürfen olympische Wettkämpfe austragen. Diese müssen nicht zwingend den internationalen Status haben. Eine Klasse wird dabei entweder von Herren oder Damen geführt oder darf von gemischten Teams gesegelt werden. Einige olympische Bootsklassen sind *Laser* (Einhand-Jolle, Herren) und *Laser Radial* (Einhand-Jolle, Damen), sowie *49er* (Zweimann-Jolle, Herren) und *Nacra 17* (Mehrrumpfboot, Gemischt).

Eine vollständige Liste aller Bootsklassen findet sich auf der ISAF-Webseite¹.

¹ <http://www.sailing.org/classes/index.php>

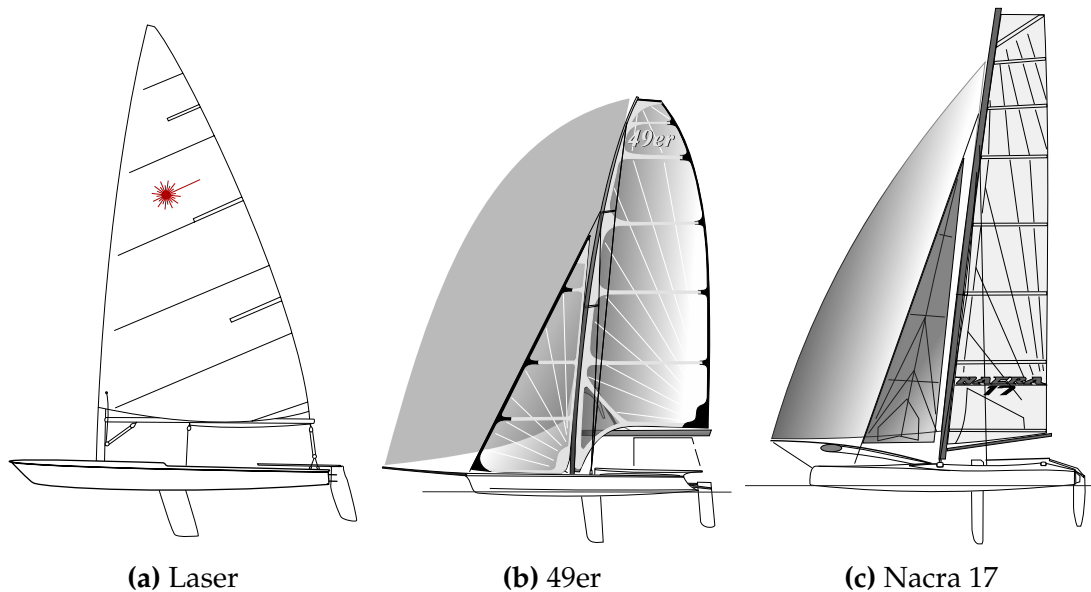


Abbildung 1.5.: Segelrisse olympischer Bootsklassen

Quellen:

(a) http://commons.wikimedia.org/wiki/File:Laser_dinghy.svg

(b) http://en.wikipedia.org/wiki/File:49er_skiff.svg

(c) http://commons.wikimedia.org/wiki/File:Nacra_17.svg

1.2. SAP Sailing Analytics

Die *SAP Sailing Analytics* ist eine Softwarelösung, welche den Segelsport den Zuschauern näher bringen soll und es den Seglern ermöglicht ihre Rennen nachträglich zu analysieren. Sie wird von einer kleinen Abteilung des Marketing-Bereichs von SAP entwickelt und wird den Anwendern kostenlos zur Verfügung gestellt. Das wirtschaftliche Interesse der Software ist es nicht Umsatz zu generieren, sondern den Namen SAPs zu verbreiten. Mit der Anwendung soll verdeutlicht werden, welche komplexen Statistiken SAP aus einfachsten Daten erzeugen kann.

Dabei wird jedes teilnehmende Boot und jede Boje mit einem GPS-Tracker ausgestattet, welcher regelmäßig die Position, Richtung und Geschwindigkeit an die Anwendung überträgt. Zusätzlich wird mit mehreren Windmessanlagen der Wind gemessen. Diese Daten werden gesammelt, analysiert, angereichert und präsentiert. Die Präsentation erfolgt über einen Browser mit den in Unterabschnitt 1.2.1 vorgestellten Komponenten. Dadurch sind die Sailing Analytics plattformunabhängig und es ist keine Installation nötig, um sie zu verwenden. Sie wird mit der Programmiersprache Java entwickelt,

wobei das *Google Web Toolkit*¹ (GWT) verwendet wird. Dieses wandelt die Klassen der Benutzeroberfläche in JavaScript Code um und bettet diesen in eine HTML-Seite ein. Außerdem stellt das GWT eine einfache Möglichkeit zur Kommunikation zwischen Client und Server zur Verfügung. Dadurch wird es einem ermöglicht komplexe Web-Anwendungen mit Java zu entwickeln, ohne sich mit HTML oder JavaScript auseinander setzen zu müssen.

Zwei mögliche Anwendungsfälle können wie folgt aussehen:

- Gelaufene Rennen lassen sich beliebig oft ansehen, was sich zum Beispiel für Segler und deren Trainer anbietet, um die eigene Leistung in dem Rennen zu analysieren. Die dabei gesammelten Daten und Statistiken geben einem tiefe Analysemöglichkeiten.
- Gerade stattfindende Rennen lassen sich mit den Sailing Analytics live verfolgen, um einen Einblick in den aktuelle Rennstand zu erhalten. Dadurch bringt man den Segelsport näher zu den Zuschauern, da man von Land meist wenig bis nichts von einem Rennen erkennen kann. Auch Moderatoren nutzen die Analysemöglichkeiten und die Visualisierung des Rennens, um das Renngeschehen besser beschreiben zu können.

1.2.1. Komponenten

Die Sailing Analytics bestehen aus mehreren Komponenten, welche einem unterschiedliche Analysemöglichkeiten bieten. Die nachfolgende Liste führt die wichtigsten Komponenten auf und gibt zu jeder eine kurze Beschreibung. Diese werden anschließend detaillierter vorgestellt.

- **Leaderboard-Gruppen**
Bieten eine Übersicht der gelaufenen oder gerade stattfindenden Rennen, von der der Anwender zu den eigentlichen Analysetools gelangt.
- **Leaderboard**
Eine große Tabelle mit Einträgen zu jedem Teilnehmer einer Regatta. Gibt Informationen über die aktuelle Platzierung und stellt weitere Statistiken dar, wie zum Beispiel die gesegelte Distanz oder die durchschnittliche Geschwindigkeit.

¹ Homepage: <http://www.gwtproject.org/>

- **Charts**

Stellen den zeitlichen Verlauf bestimmter Statistiken (z.B. dem Abstand zum aktuell führenden Teilnehmer oder die Platzierung über mehrere Rennen) als Diagramm dar.

- **Karte**

Stellt den Kurs des Rennens, die aktuelle Positionierung der Teilnehmer und Informationen über die Windrichtung und -stärke mit Hilfe einer Google Maps dar.

- **Raceboard**

Analysetool für ein einzelnes Rennen. Verbindet Leaderboard, Chart und Karte in einer Komponente.

Leaderboard-Gruppen

Die Ansicht einer Leaderboard-Gruppe dient als Einstiegspunkt für den Anwender, von dem er zu den eigentlichen Analysetools gelangt. Intern werden die einzelnen Rennen einem Leaderboard zugeordnet. Diese wiederum werden zu Gruppen zusammengefasst. Eine genaue Erläuterung dieser Strukturierung erfolgt im Unterabschnitt 1.2.2 *Datenmodell* auf Seite 21. Mit Hilfe der Leaderboards und den Gruppen lassen sich nun Veranstaltungen abbilden. Alle Rennen einer Regatta werden einem Leaderboard zugeordnet, sodass es eines für jede Regatta gibt. Diese werden einer Gruppe hinzugefügt, welche die Veranstaltung repräsentiert.

Einen Ausschnitt der Leaderboard-Gruppe der Kieler Woche 2013 sieht man in Abbildung 1.6. Für jedes Leaderboard der Gruppe gibt es eine Zeile in der Tabelle. Diese enthält den Namen des Leaderboard, einen Knopf zum Absprung in die Ansicht für das Leaderboard und eine Übersicht der enthaltenen Rennen. Diese werden dabei anhand ihrer Serie und Flotte angeordnet. Über die Knöpfe der einzelnen Rennen gelangt man zum Raceboard, sofern es aufgezeichnet wurde.

Leaderboards		Legend: Untracked tracked live	
Regatta		Races	
2.4mR open	Leaderboard	Qualifications	Q1 Q2 Q3 Q4 Q5
		Finals	CF1 F1 F2 F3 F4 F5 F6
470 - Women	Leaderboard	Qualifications	Q1 Q2 Q3 Q4 Q5 Q6
		Finals	CF1 F1 F2 F3 F4 F5 F6
		Medals	CF2 M
470 - Men	Leaderboard	Qualifications	Q1 Q2 Q3 Q4 Q5 Q6
		Finals	 Gold CF1 F1 F2 F3 F4 F5 F6
			 Silver CF1 F1 F2 F3 F4 F5 F6
		Medals	CF2 M
49er	Leaderboard	Qualifications	 Blue Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
			 Yellow Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
		Finals	 Gold CF1 F1 F2 F3 F4 F5 F6 F7 F8
			 Silver CF1 F1 F2 F3 F4 F5 F6 F7 F8
		Medal	CF2 M1 M2 M3

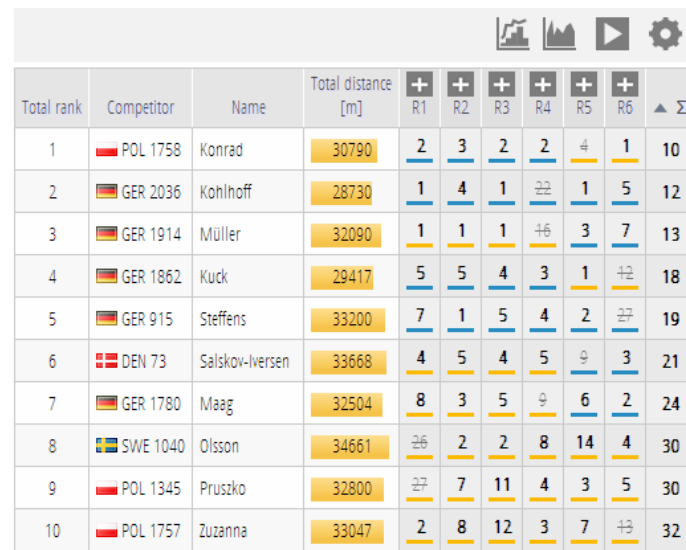
Abbildung 1.6.: Ausschnitt aus der Leaderboard-Gruppe für die Kieler Woche 2013.

Leaderboard

Mit dem Leaderboard kann man sich Daten und Statistiken über die Teilnehmer einer Regatta anzeigen lassen. Im einfachsten Fall wird es verwendet, um die aktuelle Platzierung der Segler zu erhalten. Dies ist im Fall einer gerade stattfindenden Regatta besonders praktisch, da es dauert bis die offiziellen Ergebnisse der Wettfahrtleitung veröffentlicht werden. Für jedes Rennen einer Regatta enthält das Leaderboard eine Spalte, in welche die erreichte Punktzahl jedes Teilnehmers eingetragen wird. Die letzte Spalte enthält die Summe der Punkte eines Teilnehmers unter Berücksichtigung der gestrichenen Rennen, woraus sich die aktuelle Platzierung ergibt, welche in der Spalte ganz links eingetragen ist.

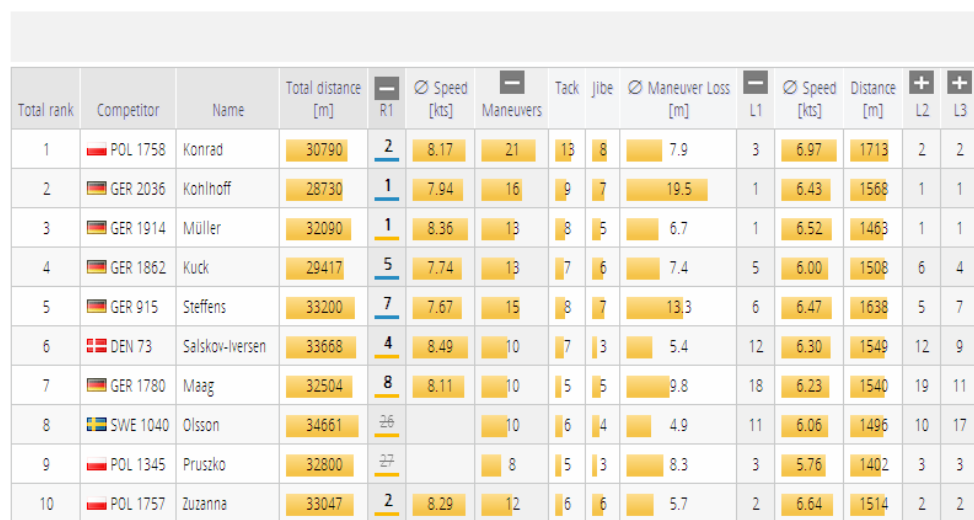
Des weiteren kann man sich Statistiken wie die durchschnittliche Geschwindigkeit, die insgesamt zurückgelegte Distanz, die Art und Anzahl der gefahrenen Manöver, den durchschnittlichen Distanzverlust bei einem Manöver und viele mehr anzeigen lassen. Diese Statistiken werden immer im Kontext eines bestimmten Teils einer Regatta berechnet. So kann man sich die zurückgelegte Distanz bei der gesamten Regatta, einem einzelnen Rennen oder eines Schenkels anzeigen lassen, indem man die entsprechenden

Spalten aufklappt. Welche Statistiken angezeigt werden sollen, kann der Anwender über einen Einstellungsdialog festlegen. Selektiert man einige Spalten und klickt auf einen der beiden linken Knöpfe im Kopf der Tabelle, öffnet sich ein Chart über den Verlauf der Platzierung der selektierten Teilnehmer oder über den zeitlichen Verlauf bestimmter Statistiken.



Total rank	Competitor	Name	Total distance [m]	R1	R2	R3	R4	R5	R6	Σ
1	POL 1758	Konrad	30790	2	3	2	2	4	1	10
2	GER 2036	Kohlhoff	28730	1	4	1	22	1	5	12
3	GER 1914	Müller	32090	1	1	1	16	3	7	13
4	GER 1862	Kuck	29417	5	5	4	3	1	12	18
5	GER 915	Steffens	33200	7	1	5	4	2	27	19
6	DEN 73	Salskov-Iversen	33668	4	5	4	5	9	3	21
7	GER 1780	Maag	32504	8	3	5	9	6	2	24
8	SWE 1040	Olsson	34661	26	2	2	8	14	4	30
9	POL 1345	Prusko	32800	27	7	11	4	3	5	30
10	POL 1757	Zuzanna	33047	2	8	12	3	7	13	32

Abbildung 1.7.: Ausschnitt eines Leaderboards mit vollständig eingeklappten Spalten. Durchgestrichene Einträge repräsentieren die Streicher des Teilnehmers. Die farbige Markierung gibt an in welcher Flotte (Blau oder Gelb) gesegelt wurde.



Total rank	Competitor	Name	Total distance [m]	R1	Speed [kts]	Maneuvers	Tack	Jibe	Maneuver Loss [m]	L1	Speed [kts]	Distance [m]	L2	L3
1	POL 1758	Konrad	30790	2	8.17	21	13	8	7.9	3	6.97	1713	2	2
2	GER 2036	Kohlhoff	28730	1	7.94	16	9	7	19.5	1	6.43	1568	1	1
3	GER 1914	Müller	32090	1	8.36	13	8	5	6.7	1	6.52	1463	1	1
4	GER 1862	Kuck	29417	5	7.74	13	7	6	7.4	5	6.00	1508	6	4
5	GER 915	Steffens	33200	7	7.67	15	8	7	13.3	6	6.47	1638	5	7
6	DEN 73	Salskov-Iversen	33668	4	8.49	10	7	3	5.4	12	6.30	1549	12	9
7	GER 1780	Maag	32504	8	8.11	10	5	5	9.8	18	6.23	1540	19	11
8	SWE 1040	Olsson	34661	26		10	6	4	4.9	11	6.06	1496	10	17
9	POL 1345	Prusko	32800	27		8	5	3	8.3	3	5.76	1402	3	3
10	POL 1757	Zuzanna	33047	2	8.29	12	6	6	5.7	2	6.64	1514	2	2

Abbildung 1.8.: Ausschnitt eines Leaderboards mit aufgeklappten Spalten

Charts

Mit den Charts lässt sich der zeitliche Verlauf statistischer Kennzahlen von einer Menge an Teilnehmern als Diagramm anzeigen lassen. Zur Zeit kann dieses Teilnehmer-Chart Statistiken für die windwärtige und zeitliche Distanz zum führenden Teilnehmer, die gesegelte Distanz, die effektive Geschwindigkeit zum Ziel, die Geschwindigkeit über Grund und die Platzierung im aktuellen Rennen oder der gesamten Regatta anzeigen.

Des weiteren gibt es Charts für den Verlauf der Windgeschwindigkeit und -richtung, sowie der Platzierung nach den jeweiligen Rennen. Die Präsentation erfolgt mit der GWT-Verschaltung von Highcharts¹, einer in JavaScript und HTML5 geschriebenen Diagramm-Bibliothek. Die GWT-Verschaltung ermöglicht den Einsatz dieser Bibliothek mit dem Google Web Toolkit.

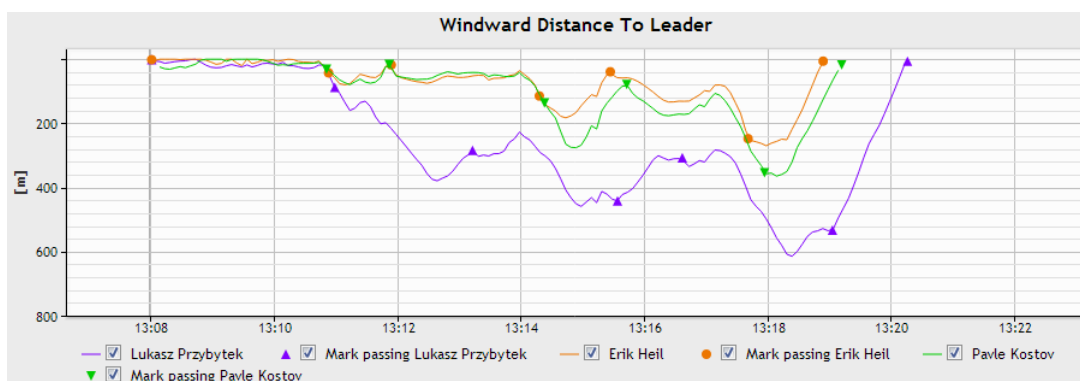


Abbildung 1.9.: Chart für die windwärtige Distanz zum Führenden. Die Symbole markieren den Zeitpunkt an dem der Teilnehmer den nächsten Schenkel begonnen hat.

¹ Homepage Highcharts: <http://www.highcharts.com/>
Homepage GWT-Verschaltung: <http://www.moxiegroup.com/moxieapps/gwt-highcharts/>

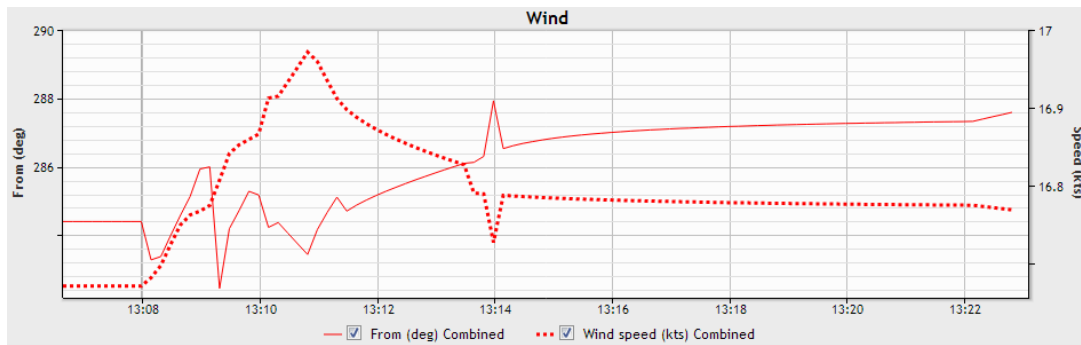


Abbildung 1.10.: Chart für den Verlauf der Windgeschwindigkeit (gepunktete Linie) und -richtung

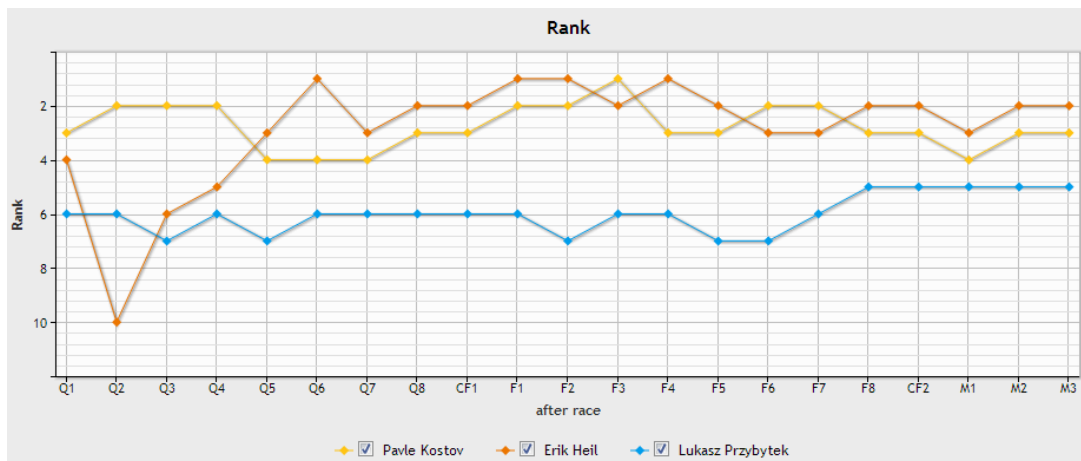


Abbildung 1.11.: Chart den Verlauf der Platzierung nach den jeweiligen Rennen

Karte

Auf der Karte wird der Kurs und die Position der Teilnehmer zu einem gegebenen Zeitpunkt visualisiert. Dazu kommen der gefahrene Kurs jedes Teilnehmers, welcher als Linie hinter der aktuellen Position dargestellt wird. Außerdem werden mobile Windmessenanlagen auf der Karte verzeichnet, wobei die Richtung der Pfeile die Richtung des Windes repräsentieren. Hierzu wird die GWT-Verschaltung der Google Maps verwendet, auf die die entsprechenden Elemente als Symbole oder Linien eingezeichnet werden. Klickt der Anwender auf ein Boot werden Informationen (z.B. die aktuelle Platzierung oder Geschwindigkeit) über dem entsprechenden Teilnehmer angezeigt.

Dabei können auch die Positionen der gefahrenen Manöver auf dem gefahrenen Kurs des Teilnehmers markiert werden.

Über den Einstellungsdialog der Karte kann der Anwender einstellen, welche Informationen angezeigt werden sollen. Darunter ist zum Beispiel eine Auswahl von Hilfslinien, wie der Führungslinie¹ oder der Kursmittellinie. Hier lässt sich auch einstellen, welche Manöver bei der Selektion eines Bootes auf dem gefahrenen Kurs markiert werden sollen und wie lange in die Vergangenheit (in Sekunden) der gefahrene Kurs angezeigt werden soll. Zusätzlich gibt es die Möglichkeit die Zoomstufe und den anzuzeigenden Bereich der Karte automatisch an bestimmte Elemente (z.B. Boote oder Bojen) anzupassen.

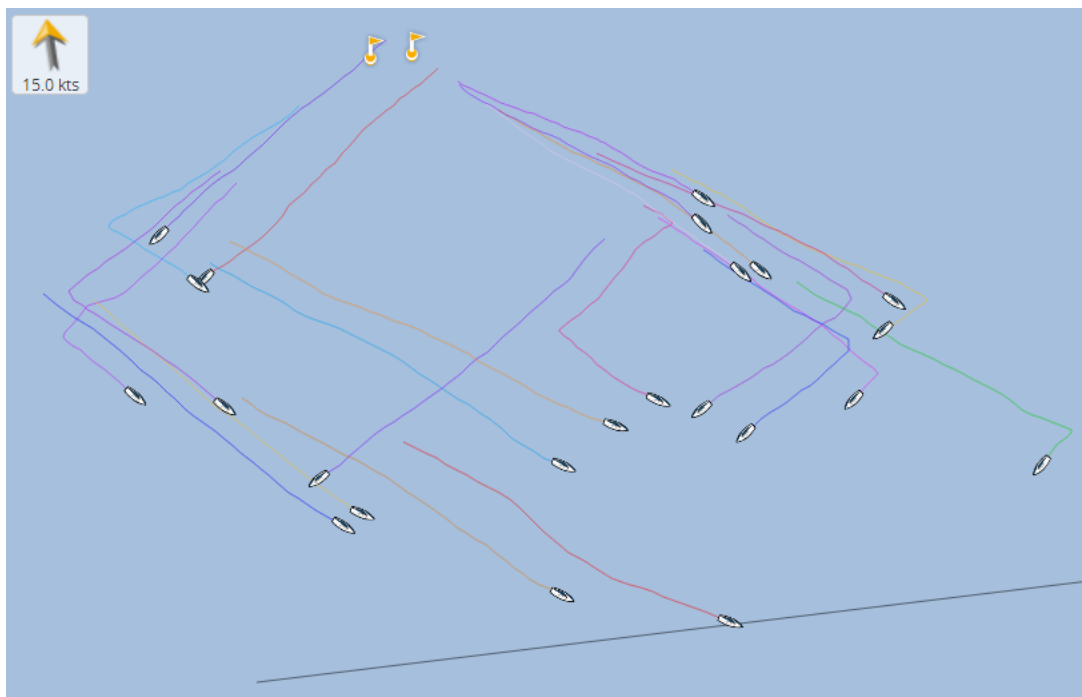


Abbildung 1.12.: Ausschnitt der Karte mit den beschriebenen Elementen. In diesem Fall sind keine mobilen Windmessenanlagen zu sehen, Der Pfeil oben rechts stellt den Wind dar, der aus unterschiedlichen Windquellen berechnet wird.

¹ Zum Wind orthogonale Linie, welche an den Führenden angelegt wird. Damit lässt sich der Abstand der anderen Teilnehmer zum Führenden besser erfassen.

Raceboard

Mit dem Raceboard lassen sich einzelne Rennen betrachten und analysieren. Dazu verwendet es die Karte, das Leaderboard und die Charts, sodass alle Komponenten zusammenspielen und gleichzeitig betrachtet werden können. Das Leaderboard wird zur besseren Übersicht auf die Spalten des betrachteten Rennens reduziert. Alle Komponenten außer der Karte lassen sich über Kontrollelemente ausblenden. Zum Beispiel sind die Charts für den Wind und die statistischen Kennzahlen der Teilnehmer zu Beginn unsichtbar. Zusätzlich gibt es einen Zeitregler, der den aktuell anzuzeigenden Zeitpunkt festlegt. Mit diesem ist es möglich das Rennen wie eine Video zu betrachten, es also automatisch (mit einer frei wählbaren Geschwindigkeit) ablaufen zu lassen, zu pausieren oder an eine bestimmte Stelle zu springen. Wichtige Zeitpunkte des Rennens werden auf dem Zeitregler markiert. Diese sind der Start (S), sowie Beginn jedes Schenkels (M), der Einlauf des ersten Teilnehmers ins Ziel (F) und des Ende des Rennens (E). Die anderen Komponenten zeigen immer den Zustand zu dem Zeitpunkt des Zeitreglers an, so ändert sich die Reihenfolge der Spalten des Leaderboards anhand der aktuellen Platzierung und die Darstellung der Karte passt sich entsprechend an. Des weiteren wirkt sich die Selektion der Teilnehmer des Leaderboards auf die anderen Komponenten aus. So werden deren Boote auf der Karte hervorgehoben und das Teilnehmer-Chart zeigt nur die Daten der selektierten Teilnehmer an.



Abbildung 1.13.: Raceboard mit eingblendetem Leaderboard und Wind-Chart.

1.2.2. Datenmodell

In diesem Unterabschnitt gehe ich näher auf die Repräsentation der Daten innerhalb der Sailing Analytics ein. Die kleinsten Datenelemente sind GPS-Fixes und Winddaten. Diese enthalten einen Zeitstempel für den Zeitpunkt zu dem sie erfasst, eine Position und eine Geschwindigkeit, sowie Informationen über die Richtung in die sich das gemessene Objekt bewegt. Anhand dieser Daten ist alles aufgebaut und aus ihnen werden alle Statistiken berechnet. Für die komplexeren Datenstrukturen gibt es zwei Hierarchien. Die eine bildet die einer in Unterabschnitt 1.1.1 erläuterten Regatta ab und die andere repräsentiert Leaderboards und Leaderboard-Gruppen, wobei diese beiden Hierarchien zum Teil miteinander verbunden sind. Wie diese aufgebaut sind möchte ich in diesem Unterabschnitt darstellen.

Die GPS-Daten werden dabei von einem Drittanbieter gesammelt und gespeichert. In einer Administrator-Oberfläche lässt sich das tracken von Rennen starten, woraufhin die Daten von dem Drittanbieter geladen werden. Das bedeutet, dass die Daten auf Seite der Sailing Analytics nicht in einer Datenbank vorliegen, sondern immer nur zur Laufzeit in Form eines Objektfeldes des Datenmodells zur Verfügung stehen.

Abbildung einer Regatta

Wie oben beschrieben besteht eine Regatta aus Serien, Flotten, Teilnehmern und Rennen. Dementsprechend gibt es für jedes dieser Elemente eine Klasse, die dieses repräsentiert. Einige davon werden auch von zwei Klassen abgebildet, da sie sowohl im ungetrackten als auch im getrackten Zustand vorliegen können. Getrackt bedeutet in diesem Fall, dass sie konkrete GPS- oder Winddaten enthalten (oder getrackte Unterstrukturen, welche die Daten enthalten). Die ungetrackte Repräsentation besitzt Metadaten über das entsprechende Element, wie zum Beispiel den Namen oder einen eindeutigen Identifikator. Die getrackte Klasse wird durch den Präfix `Tracked` markiert und hat eine Referenz auf die Ungetrackte, um Zugriff auf die Metadaten zu erhalten.

Das größte Element der Regatta-Hierarchie ist die `Regatta`, welche unter anderem Informationen über die Bootsklasse und das Punktesystem enthält. Außerdem hat sie Referenzen auf eine Menge von Serien und Rennen. Daneben gibt es die `TrackedRegatta`, welche nach dem oben beschriebenen Schema fungiert.

Eine `Series` besitzt Informationen bezüglich der Streicher, die innerhalb der Serie erlaubt sind, ob sie eine Übertragspalte hat und weiterer bepunktungsspezifischer Daten. Des weiteren enthält sie eine Menge von Rennen und Flotten.

`Fleets` (welche Flotten repräsentieren) besitzen einen Namen, eine Farbe und eine Ordinalzahl, welche benötigt wird, um eine Flotte vor der anderen zu bevorzugen (wie es bei Gold- und Silber-Flotten benötigt wird).

Ein Rennen wird von der Klasse `RaceDefinition` repräsentiert, welche einen Namen, eine Bootsklasse, einen Kurs und eine Menge von Teilnehmern beinhaltet. Die getrackte Variante ist `TrackedRace`, welche wesentlich mehr Informationen enthält und auf die ich später zurück kommen möchte. Ein `Competitor` (also Teilnehmer) besitzt einen Namen, sowie ein Boot und gehört zu einem Team, woraus man weitere Informationen des Teilnehmers ableiten kann (z.B. die Nationalität).

Die einen Kurs abbildende Klasse `Course`, hat eine Menge von Wegpunkten und Schenkeln. Ein Schenkel (`Leg`) besteht aus zwei dieser Wegpunkte, welche Beginn und Ende des Schenkels bestimmen. Von `Leg` gibt es wieder eine getrackte Variante.

Die Klasse `TrackedRace` ist das Kernelement für den Zugang zu den eigentlichen Daten und Statistiken eines Rennens. Es enthält Informationen über wichtige Zeitpunkte, wie den Start oder das Ende. Über das `TrackedRace` gelangt man an die Platzierung eines Teilnehmers zu einem bestimmten Zeitpunkt, deren Tonnenrundungen und Manöver, sowie deren statistischen Kennzahlen, welche sich auf ein ganzes Rennen beziehen (z.B. die windwärtige Distanz zum Führenden). Außerdem bietet es Methoden um an Informationen über den Wind an einer bestimmten Stelle zu einem bestimmten Zeitpunkt zu gelangen. Mit der `getTrack`-Methode erhält man sämtliche GPS-Daten des Teilnehmers innerhalb dieses Rennens. Es enthält außerdem eine Menge von `TrackedLegs` welche unter anderem Informationen über den Typ (mit, gegen oder quer zum Wind) des Schenkels enthalten. Diese getrackten Schenkel wiederum haben für jeden am Rennen teilnehmenden Segler einen eigenes Schenkel-Unterobjekt. Dieses `TrackedLegOfCompetitor` fungiert als Schnittstelle für die statistischen Kennzahlen eines Schenkels für den jeweiligen Teilnehmer.

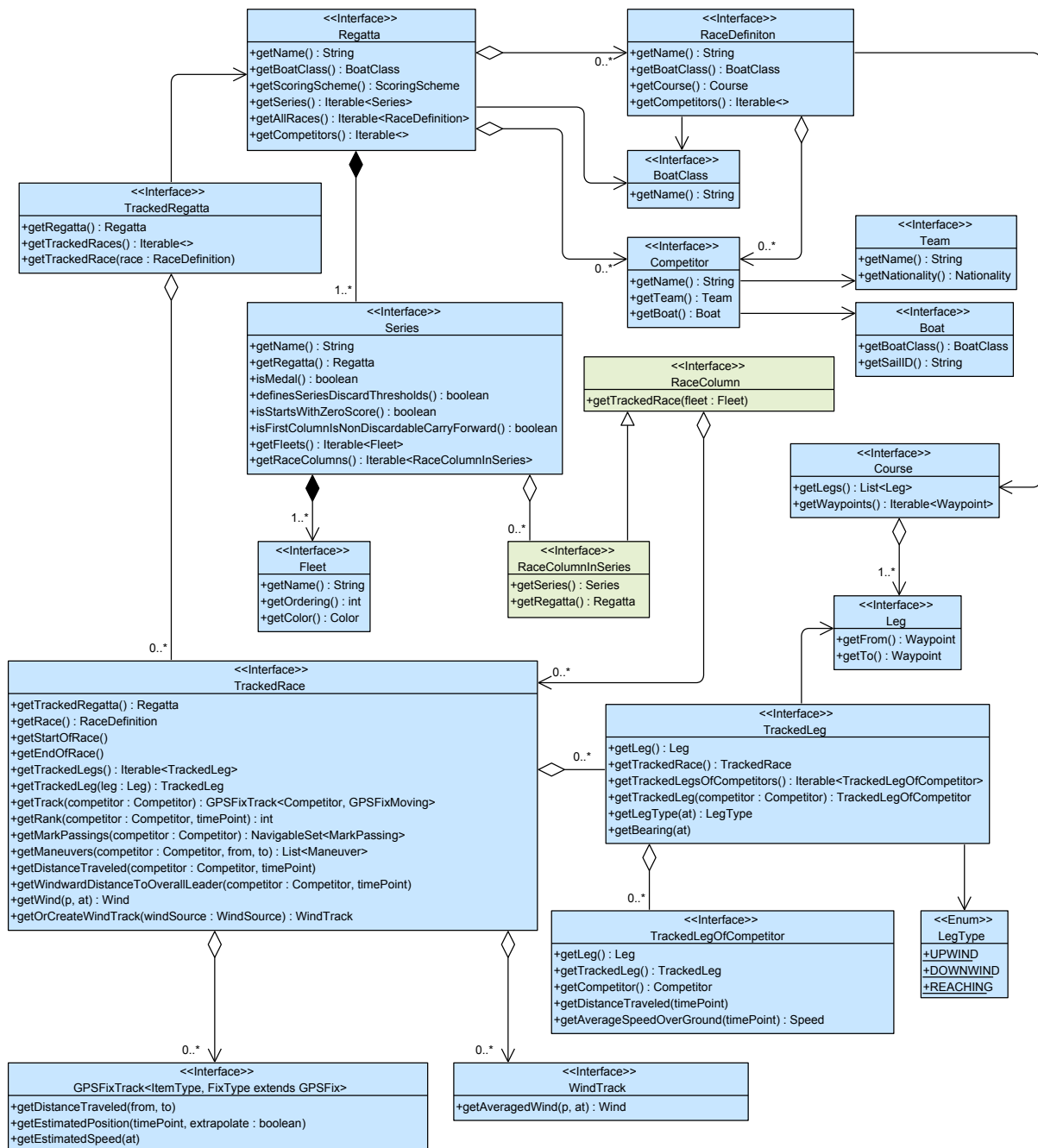


Abbildung 1.14.: Datenmodell von Regatten

Enthält aus Gründen der Übersicht nur die wichtigsten Interfaces und Methoden. Bei manchen Methoden wurde zudem die Signatur gekürzt.

Abbildung von Leaderboard-Gruppen und Leaderboards

Leaderboard-Gruppen sind Container für eine Menge von Leaderboards, wobei diese in keinem Zusammenhang stehen müssen oder bestimmte Bedingungen erfüllen müssen. Sie werden dazu genutzt alle Leaderboards einer Veranstaltung zusammenzutragen und diese dem Anwender zu präsentieren. Neben den Leaderboards hat jede `LeaderboardGroup` einen Namen und eine Beschreibung.

Ein `Leaderboard` ist die Datenbasis der Leaderboard-Komponente. Es enthält neben einem Namen und Informationen über möglicher Streicher noch eine Menge von Teilnehmern, Flotten und getrackten Rennen. Auch das `Leaderboard` bietet Methoden um statistische Kennzahlen zu berechnen, welche an die enthaltenen `TrackedRaces` delegieren. Das `Leaderboard` hat zwei konkrete Implementierungen. Die eine ist kontextfrei und kann beliebige Rennen enthalten, auch wenn diese in keinerlei Zusammenhang stehen. So könnte man Rennen unterschiedlicher Regatten und Bootsklassen an ein `Leaderboard` binden, wobei die resultierende Platzierung der Teilnehmer keinen Sinn mehr ergeben würde. Dieses `FlexibleLeaderboard` wird in der Praxis nicht mehr verwendet, sondern dient zum Testen während der Entwicklung. Die andere Implementierung ist an eine konkrete Regatta gebunden und enthält genau die in dieser Regatta definierten Rennen, an welche nur `TrackedRaces` gebunden werden können, die zu der Regatta des `RegattaLeaderboards` gehören.

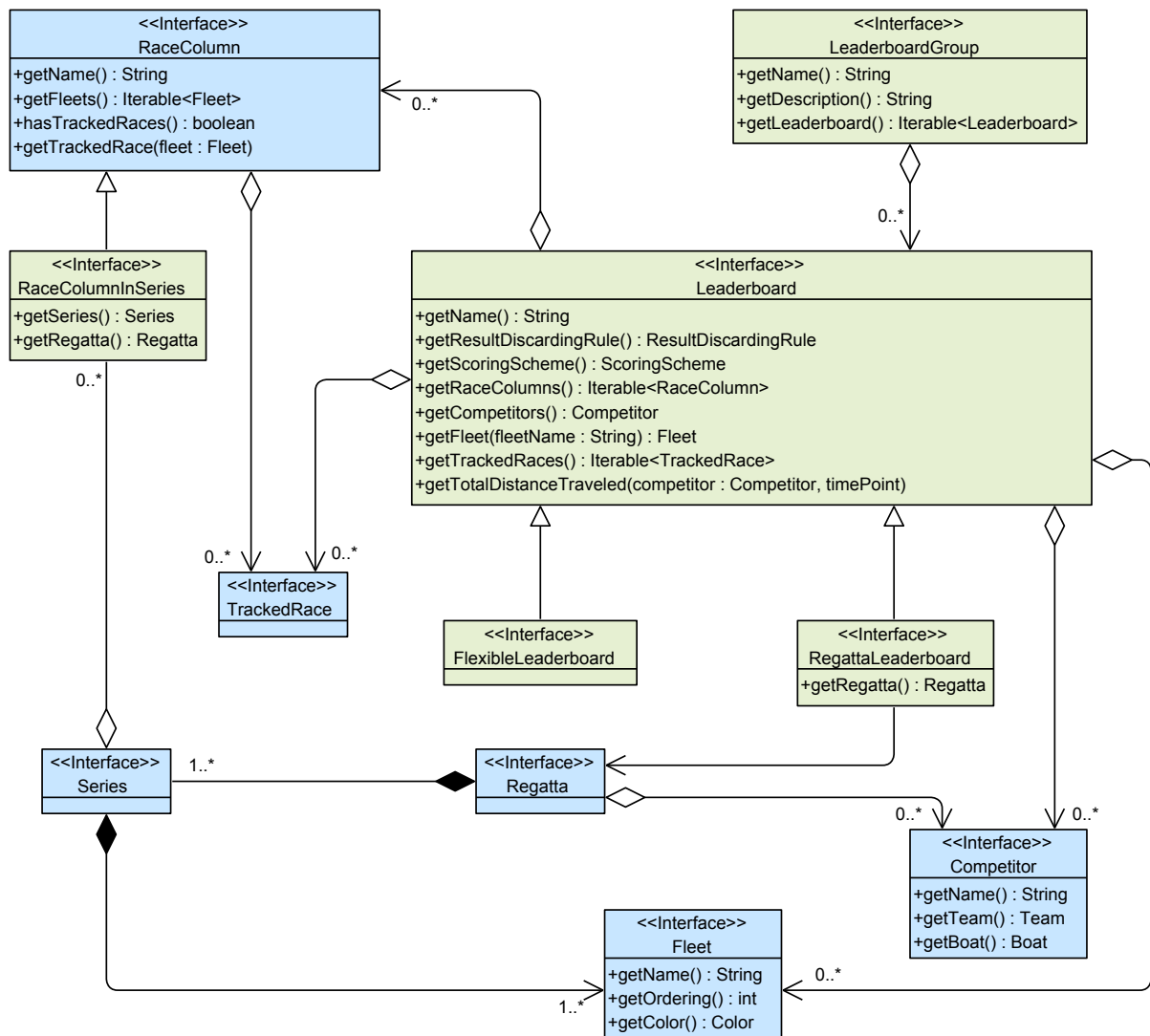


Abbildung 1.15.: Datenmodell von Leaderboard-Gruppen
 Enthält aus Gründen der Übersicht nur die wichtigsten Interfaces und Methoden.
 Bei manchen Methoden wurde zudem die Signatur gekürzt.

1.3. SAP BusinessObjects Explorer

Der SAP *BusinessObjects Explorer* (zukünftig als BOBJ-Explorer bezeichnet) ist eine Business-Intelligence-Lösung zur Analyse großer Datenmengen. Es bietet die Möglichkeit bestimmte Kennzahlen anhand der zur Verfügung stehenden Datenbasis (im BOBJ-Explorer als *Information Space* bezeichnet) zu ermitteln. Im Kontext des Segelns sind solche Kennzahlen zum Beispiel die durchschnittliche Geschwindigkeit über Grund, der durchschnittliche zeitliche Abstand zum Führenden oder die Summe bestimmter Manöver. Dabei lassen sich die zu berücksichtigenden Daten anhand von definierten *Facetten* filtern, sodass zum Beispiel nur Dateneinträge die zu einer bestimmten Regatta gehören berücksichtigt werden. Die Ergebnisse werden anhand von bis zu zwei *Facetten* gruppiert und in einem Diagramm dargestellt. Zu sehen ist das in Abbildung 1.17 *Ausschnitt der Explorationsansicht des BOBJ-Explorers* auf Seite 28.

Welche Kennzahlen und *Facetten* dem Nutzer dabei zur Verfügung stehen, werden von einem Administrator beim Anlegen des *Information Spaces* ausgewählt, wobei diese *Information Spaces* eine Zwischenschicht zu der eigentlichen Datenbasis darstellen. In dieser sind die zur Verfügung stehenden Kennzahlen und *Facetten* definiert. In Abbildung 1.16 *Erstellung eines neuen Information Spaces* sieht man auf der linken Seite die Kennzahlen (oranges Symbol) und *Facetten* (blaues Symbol) der Datenbasis, während in der Mitte die ausgewählten Elemente aufgeführt sind, die dem Nutzer beim Betrachten des *Information Spaces* bereitstehen. Damit lassen sich mehrere spezialisierte Ansichten einer Datenbasis erstellen, was dazu genutzt werden kann die *Information Spaces* für bestimmte Anwendergruppen übersichtlicher zu gestalten.

Die Datenbasen werden in einer anderen Anwendung, dem *BusinessObjects Information Design Tool*, definiert. Für deren Definition gibt es mehrere Möglichkeiten, von denen ich zwei kurz vorstellen möchte. Zum einen lässt sich direkt eine Datenbank als Basis verwenden. Hierbei definiert man zunächst die Verbindung zu dieser. Die Spalten der Datenbank werden anschließend automatisch als *Facetten* erkannt. Die Kennzahlen basieren auf den Werten einer Datenbankspalte, auf die ein Aggregationsoperator (zum Beispiel Summe oder Durchschnitt) angewendet wird. Des weiteren können Grundrechenarten auf das Aggregat angewendet werden, um zum Beispiel die Einheit des Wertes anzupassen (z.B. Umrechnung von Millisekunden zu Sekunden). Die zweite Variante einer Datenbasis wird als *Universum* bezeichnet und stellt eine semantische Zwischenschicht zwischen der Datenbank und dem Datenanalysetool dar. So kann

man zum Beispiel den Datenbankspalten sprechende Namen geben, welcher daraufhin im BOBJ-Explorer als *Facette* angezeigt wird.

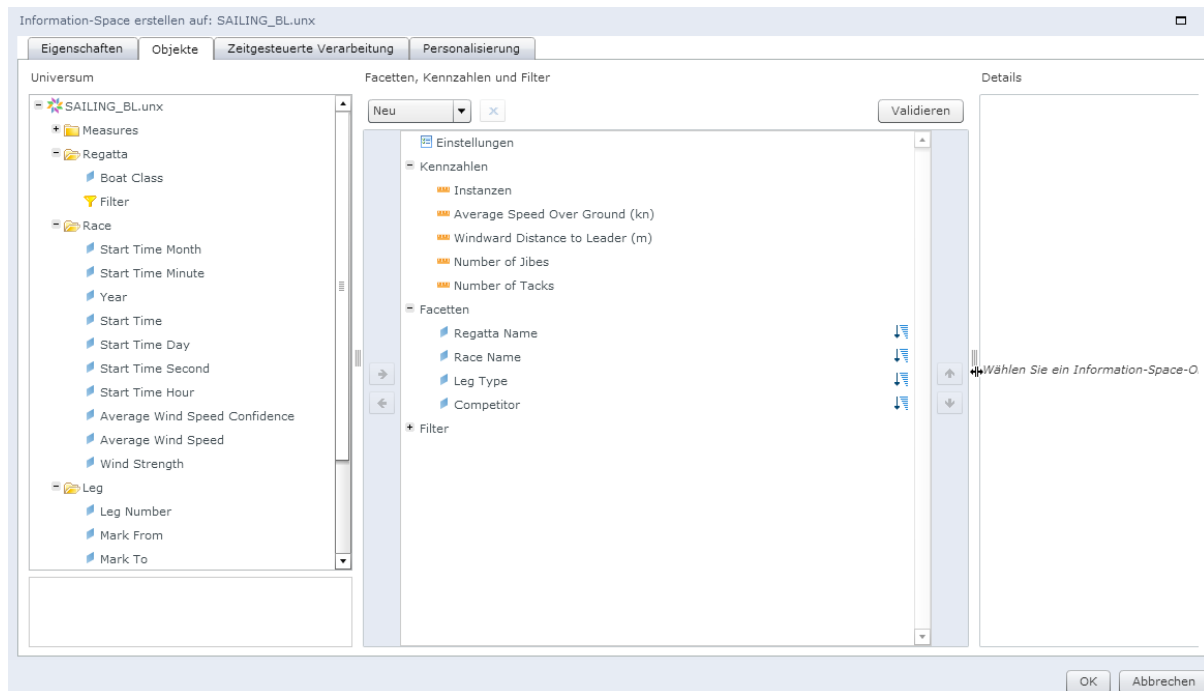


Abbildung 1.16.: Erstellung eines neuen *Information Spaces* basierend auf einem *Universum*

Mit dem BOBJ-Explorer lassen sich keine dynamischen Daten analysieren, da die für die Analyse vorliegenden Daten immer in statischer Form sind. Basiert der *Information Space* auf einem *Universum*, müssen die in der Datenbank enthaltenen Daten indiziert werden, damit sie in der Analyse zur Verfügung stehen. Der BOBJ-Explorer gibt einem hier die Möglichkeit die Indizierung in regelmäßigen Abständen automatisch durchzuführen. Falls der *Information Space* direkt auf der Datenbank aufbaut ist diese Indizierung einmalig vor der Verwendung des *Spaces* nötig, damit der BOBJ-Explorer die Form der Daten erkennt. Ruft der Anwender nun die Explorationsansicht des *Information Spaces* auf werden alle zu diesem Zeitpunkt in der Datenbank vorliegenden Daten in die Analyse aufgenommen. Damit Änderungen der Daten wirksam werden, muss die Explorationsansicht (s. Abbildung 1.17) geschlossen und neu geöffnet werden.

In der oberen Abbildung sieht man einen Ausschnitt der Explorationsansicht des BOBJ-Explorers, in der gerade die Kieler Woche 2012 analysiert wird. Mit der oberen linken Liste kann man die zu berechnende Statistik auswählen. Rechts davon ist eine Tabelle der Facetten mit den konkreten Werten anhand denen gefiltert werden kann. In der mittleren Leiste sind die aktuell angewendeten Filter aufgelistet. In diesem Fall muss das Datenelement als Bootsklasse den Wert *29er* haben. Unten sieht man die

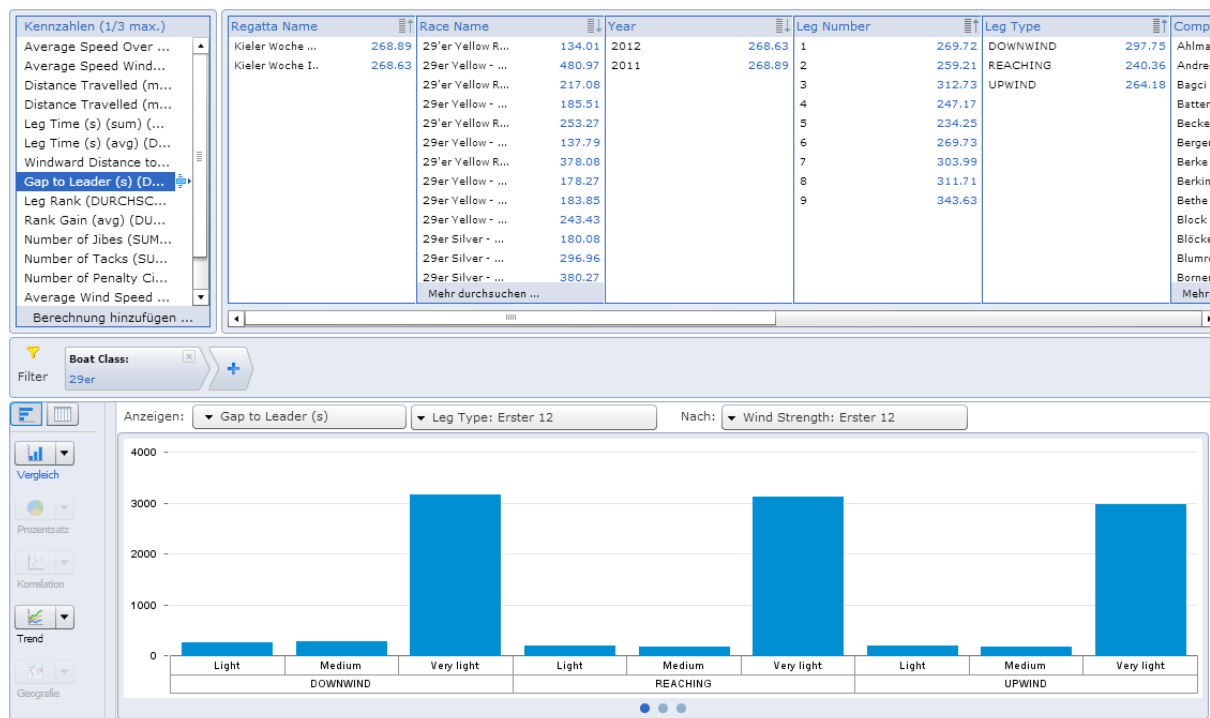


Abbildung 1.17.: Ausschnitt der Explorationsansicht des SAP BOBJ-Explorers bei der Datenanalyse.

anhand des Schenkel-Typs und der Windstärke gruppierten Ergebnisse in Form eines Balkendiagramms.

Aktuelle Verbindung der Sailing Analytics mit dem BOBJ-Explorer

Zur Realisierung eines Show-Cases der Datenanalysemöglichkeiten SAPs hat ein Arbeitskollege bereits einen Weg entwickelt, mit dem sich die Daten der Sailing Analytics mit dem BOBJ-Explorer analysieren lassen. Hierfür hat er ein Servlet geschrieben, welches das Datenmodell abarbeitet, die relevanten Daten sammelt und als XML-Dokument zurückliefert. Dieses Dokument wird über die *BusinessObjects Data Services* in eine *HANA*-Datenbank geladen. Auf Basis dieser Daten gibt es ein *Universum* auf das der BOBJ-Explorer zugreift. Durch die Verwendung des Servlets einer Datenbank als „*Transportschicht*“ für die Daten, lassen sich nur die Daten analysieren, welche zuvor aus den Sailing Analytics exportiert und in die Datenbank importiert wurden.

2. Motivation

Wie oben beschrieben, ist es bereits möglich mit Hilfe von BOBJ Data Mining über die Statistiken der Sailing Analytics zu betreiben. Allerdings ist man dabei auf die bereits extrahierten Daten beschränkt und auch die *Facetten* und Kennzahlen müssen zuvor von einem Administrator definiert werden. Mit dieser Arbeit möchte ich die Grundlage schaffen, die Analysemöglichkeiten bezüglich des Data Minings zu erweitern. Im Optimalfall wäre es dann möglich mit allen auf einer Server-Instanz befindlichen Daten zu arbeiten, ohne dass diese vorher extrahiert werden müssten.

Dadurch wären die Sailing Analytics und das SAP Business-Intelligence-Tool BOBJ stärker miteinander verknüpft. Damit wird der Business Showcase, mit dem SAP verdeutlichen möchte, welche Analysemöglichkeiten SAP selbst bei einfachsten Datenelementen zur Verfügung stellen kann, verbessert und noch eindrucksvoller, da ein Anwender direkt im Anschluss oder sogar während eines Rennens, dessen Daten analysieren kann. Die Segler können die erweiterten Analysemöglichkeiten nutzen, um die Statistiken in einem anderen Kontext als einem einzelnen Rennen oder einer Regatta zu betrachten und damit neue Erkenntnisse gewinnen und ihre Leistung weiter zu verbessern.

3. Entwurf

In diesem Kapitel möchte ich einen Entwurf für die spätere Umsetzung vorstellen. Dabei werde ich zunächst erläutern, wie man den *BusinessObjects Explorer* mit den Sailing Analytics verbinden könnte und anschließend gehe ich auf den vorgesehenen Ablauf zur Datenanalyse ein.

3.1. Verbindung mit dem BOBJ-Explorer

Mit dem BOBJ-*Information Design Tool* ist es möglich mit einem generischen JDBC-Treiber eine Verbindung zu einer Datenbank herzustellen. Dafür muss man die URL, unter der die Datenbank zu erreichen ist, sowie die Klasse des Treibers angeben. Damit wäre es möglich, dass der BOBJ-Explorer SQL-Anfragen direkt an die Sailing Analytics sendet, in dem man einen eigenen JDBC-Treiber implementiert. Dieser würde die ankommenden SQL-Anfragen interpretieren, die verlangten Informationen aus dem Datenmodell der Sailing Analytics extrahieren und als Ergebnistabelle zurückliefern. Aus dieser Tabelle ließen sich nun die Statistiken berechnen und mit dem BOBJ-Explorer anzeigen.

Damit wären die Daten, die man analysieren könnte, immer auf dem aktuellsten Stand, ohne eine vorherige Extraktion und Importierung durchführen zu müssen. Der aufwendigste Schritt dieses Ablaufs ist es die Daten aus den Sailing Analytics zu ermitteln. Wie dieser Schritt funktionieren könnte, wird im nächsten Abschnitt erläutert, wobei sich das dabei verwendete Prinzip nicht auf den konkreten Fall für die Verbindung mit dem BOBJ-Explorer beschränkt.

3.2. Prinzip der Datenanalyse

Das Prinzip der Datenanalyse soll wie folgt aussehen. Im ersten Schritt werden alle verfügbaren Daten anhand eines oder mehrerer Filter-Kriterien gefiltert. Bei BOBJ entsprechen diese Kriterien den ausgewählten Elementen der *Facetten*, im allgemeinen Fall sollen allerdings beliebige Filter eingesetzt werden können. Anschließend werden die gefilterten Daten in Gruppen zusammengefasst, sodass jeder Dateneintrag einer Gruppe eine bestimmte Eigenschaft hat, welche durch die Gruppierungs-Kriterien definiert sind. Im letzten Schritt wird für jede Gruppe die ausgewählte Kennzahl berechnet. Zurückgegeben wird eine Menge von Ergebnissen, wovon jedes Ergebnis zu einer bestimmten Gruppe gehört. Eine Darstellung dieser Vorgehensweise ist in Abbildung 3.2 auf Seite 33 zu sehen.

Dieses Prinzip ist nicht ohne weiteres auf die Sailing Analytics zu übertragen, worauf ich im nachfolgenden Unterabschnitt eingehen werde.

3.2.1. Sammlung und Aufbereitung der Daten

Um das oben dargestellte Prinzip bei den Sailing Analytics anzuwenden, muss dieses angepasst werden. Das liegt daran, dass die Daten nicht in einer Datenbank oder einem großen Datenpool vorliegen, sondern in einem Objektfeld des in Unterabschnitt 1.2.2 beschriebenen Datenmodells enthalten sind. Hinzu kommt, dass die atomaren Daten wenig bis gar keine Informationen über ihre Umgebung haben. Hat man nur eine Referenz auf so ein Datenelement, lässt sich zum Beispiel nicht ermitteln zu welchem Rennen oder Teilnehmer dieser gehört.

Nun muss in den Ablauf zur Datenanalyse ein Schritt hinzugefügt werden. Dabei wird die Datenbasis der Sailing Analytics traversiert und die atomaren Daten werden gesammelt. Während der Abarbeitung der Datenbasis lässt sich der Kontext der zu sammelnden Daten erfassen. Im Falle der GPS-Daten gehören zu dem Kontext Elemente wie die Regatta, das Rennen, die Flotte oder der Teilnehmer zu dem das einzelne GPS-Objekt gehört. Dieser Kontext wird an das Datenelement angehängt, worauf dann die nachfolgenden Schritte angewendet werden können.

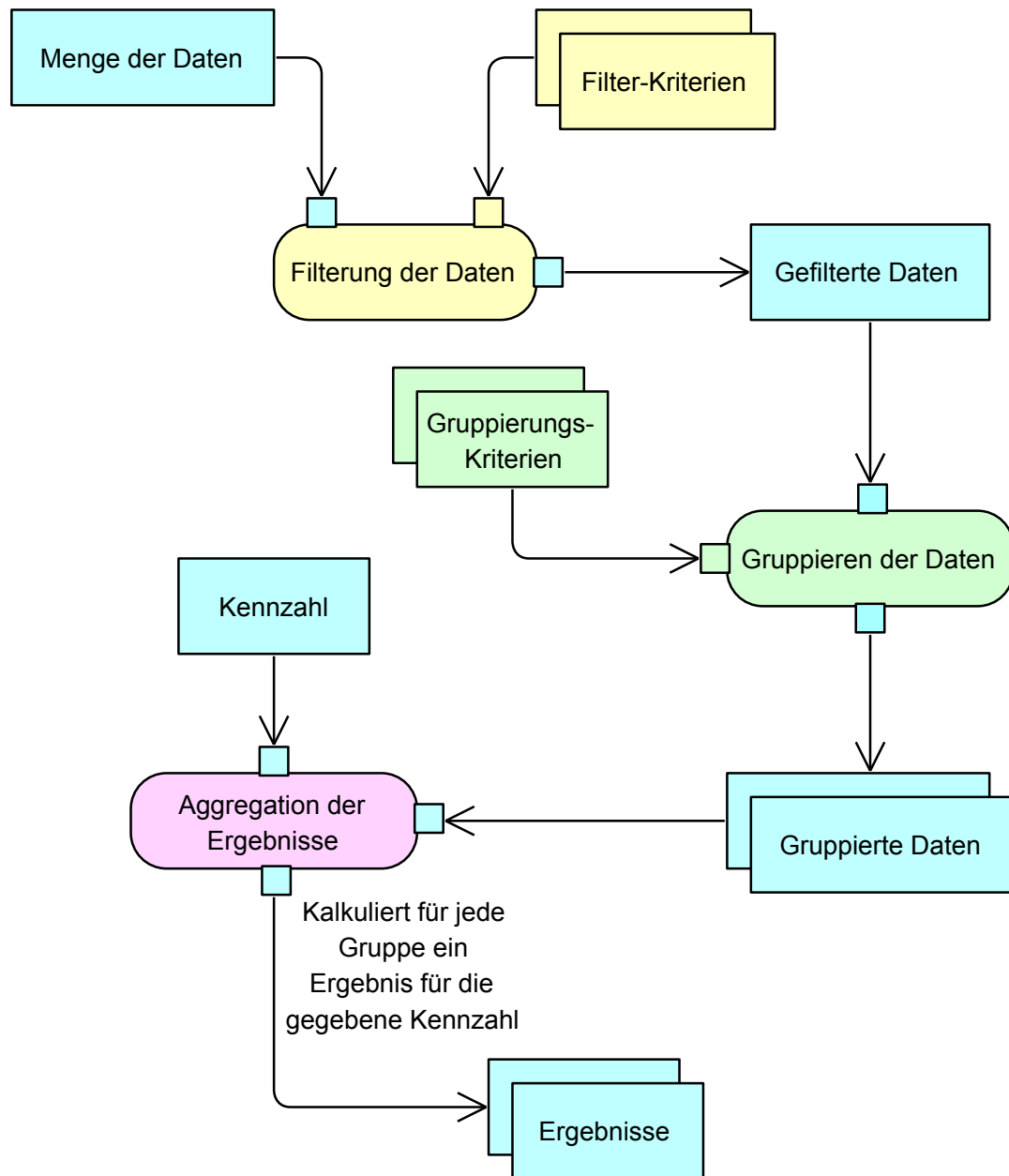


Abbildung 3.2.: Aktivitätsdiagramm zum Ablauf der Datenanalyse

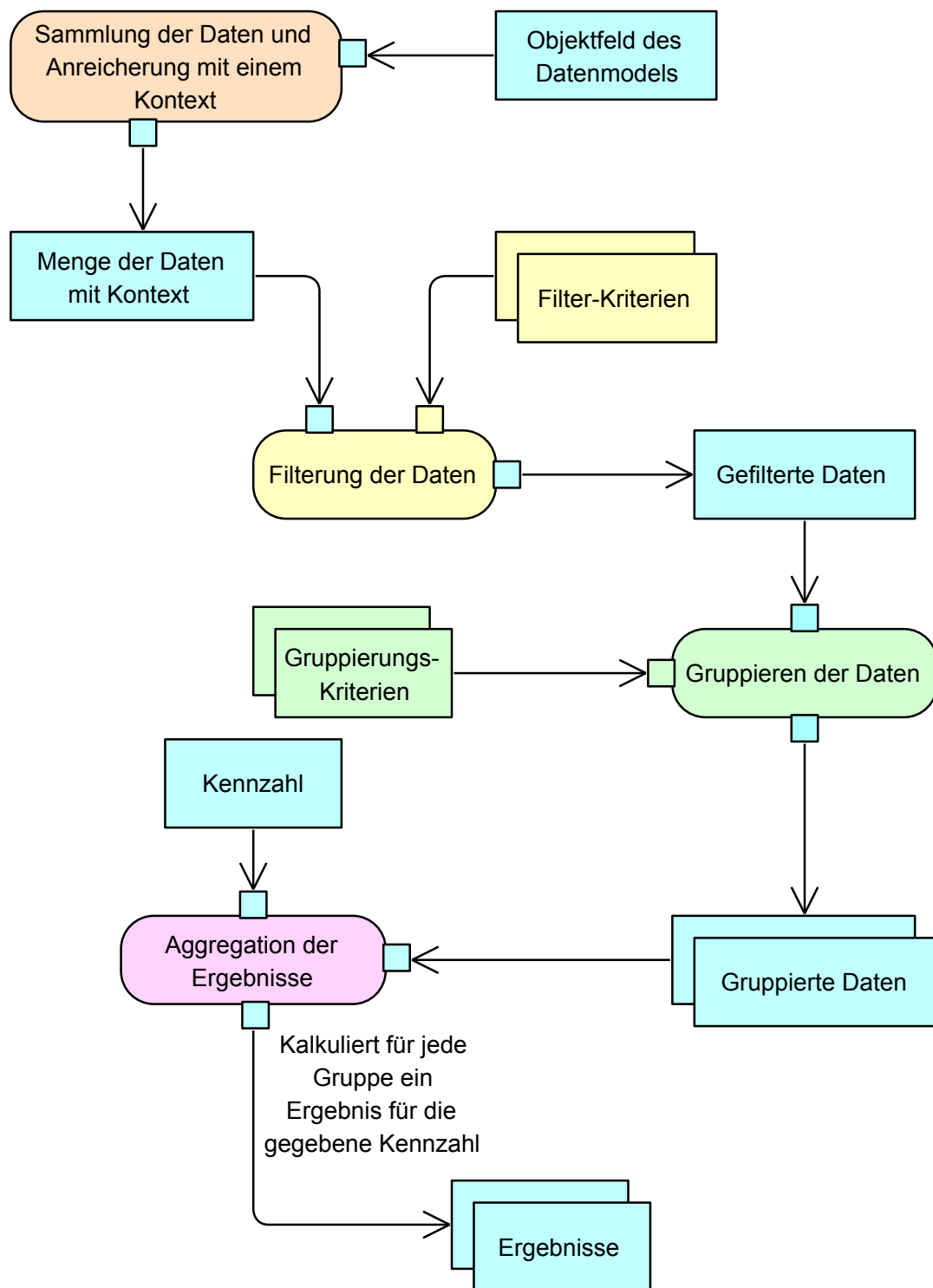


Abbildung 3.3.: Aktivitätsdiagramm zum Ablauf der Datenanalyse mit Kontext

4. Umsetzung

Um eine Verbindung mit dem BOBJ-Explorer herstellen zu können, muss zunächst die Grundlage geschaffen werden, die Daten aus den Sailing Analytics auszulesen und in eine geeignete Form zu überführen. Wie diese Grundlage funktionieren könnte, wurde im vorherigen Kapitel beschrieben. Da die Entwicklung eines eigenen Datenbanktreibers keine Neuerungen mit sich bringt, möchte ich mich in dieser Arbeit auf die Umsetzung dieser Grundlage konzentrieren. Dabei wäre es wünschenswert, wenn diese möglichst allgemein umgesetzt wird, um eine Datenanalyse unabhängig von dem BOBJ-Explorer zu ermöglichen. Dadurch ließen sich auch dessen Beschränkungen überwinden. Diese sind zum Beispiel, dass man lediglich vordefinierte Kennzahlen berechnen kann, sich die Daten nur nach bestimmten Werten bestimmter *Facetten* filtern lassen oder dass sich die Ergebnisse bloß anhand zweier Dimensionen gruppiert werden können. Dabei ist es zwar möglich neue Kennzahlen zu definieren, dies muss allerdings entweder durch einen Administrator im BOBJ *Information Design Tool* vorgenommen werden oder der Anwender kann lediglich zwei der definierten Kennzahlen mit Grundrechenarten kombinieren.

Ziel dieser Arbeit soll es demnach sein, eine flexible und leicht zu erweiternde Grundlage zu schaffen, um die in Form eines Objektfeldes vorhandenen Daten der Sailing Analytics zu analysieren und mit der es möglich ist eine Verbindung nach dem zuvor beschriebenen Konzepts zu dem BOBJ-Explorer herzustellen.

Arbeitsweise

Die Implementierung erfolgt in der Programmiersprache Java und soll als weitere Komponente der Sailing Analytics entwickelt werden. Dabei wird mit Eclipse als Entwicklungsumgebung und Git als Versionierungstool gearbeitet.

Dabei sollen die einzelnen Komponenten an den Schritten des in Abbildung 3.3 dargestellten Ablaufs orientiert sein und modular entwickelt werden, um ein möglichst fle-

xinles Ergebnis zu erreichen. Nach der Implementierung jeder einzelnen Komponente, wird deren Funktion mit Modultests¹ überprüft und gegen degenerative Änderungen abgesichert. Bei der Entwicklung des Tests soll dieser zunächst absichtlich fehlschlagen, um die Genauigkeit und Qualität der Fehlermeldungen zu überprüfen. Anschließend wird dieser so abgeändert, dass er erfolgreich abläuft. Nachdem alle Komponenten fertiggestellt wurden, werden diese in ihrer Gesamtheit auf Fehler untersucht, indem konkrete Datenanalysen ausgeführt werden. Funktionieren diese einfachsten Datenanalysen fehlerfrei, werde soll der Funktionsumfang der Grundlage in einem iterativen Prozess erweitert werden.

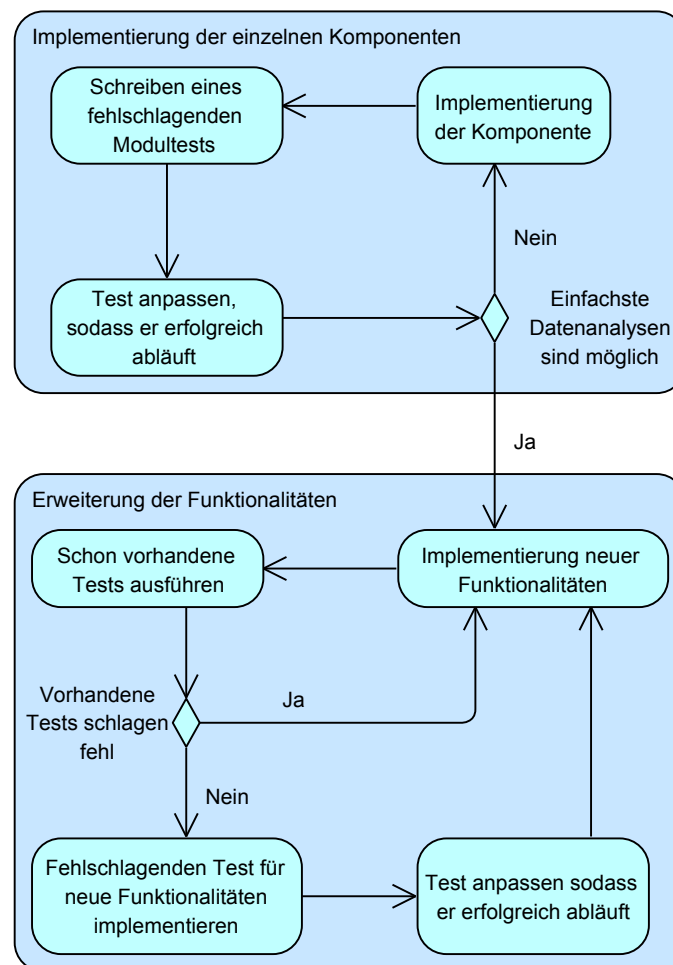


Abbildung 4.1.: Arbeitsweise bei der Implementierung

¹ Modultests sollen die korrekte Funktionalität einer Komponente abgekapselt vom Rest des Systems gewährleisten.

5. Ergebnis

In diesem Kapitel werde ich das Ergebnis meines Datenanalyse-Frameworks für die Sailing Analytics vorstellen. Dabei gebe ich zunächst einen Überblick über den generellen Aufbau und gehe anschließend genauer auf die einzelnen Komponenten und deren Aufgaben ein. Zum Schluss erläutere ich wie die Datenanalyse über alle vorhandenen GPS-Daten mit Hilfe des Frameworks funktioniert.

Die zentrale Komponente des Frameworks ist die `Query`, welche alle übrigen Komponenten in sich bündelt. Sie enthält eine Methode an die ein Service übergeben wird, mit dem man an das Datenmodell der Sailing Analytics gelangt. Die `Query` verwendet die enthaltenen Komponenten, um den in Abbildung 3.3 beschriebenen Ablauf durchzuführen. Das Ergebnis der Datenanalyse wird in Form einer `QueryResult`-Instanz zurückgegeben. Diese enthält neben Metadaten über die Analyse, wie zum Beispiel die zur Ausführung benötigte Zeit oder die Anzahl der abgearbeiteten Datenelemente, die eigentlichen Ergebnisse in Form einer `Map`, welche als Schlüssel einen eindeutigen Identifikator der zu dem Ergebnis gehörigen Gruppe verwendet.

Die Ermittlung des `QueryResults` läuft dabei folgendermaßen ab. Im ersten Schritt werden mit dem `DataRetriever` alle `Datenelement` aus dem erhaltenen Service gesammelt und als große Menge zurückgeliefert. Als nächstes arbeitet der `Filter` diese Datenmenge ab und reduziert diese auf die Elemente, welche den Filterkriterien entsprechen. Diese gefilterten Daten werden nun an den `Groupier` übergeben, der diese den entsprechenden Gruppen zuweist. Den letzten Schritt - die Aggregation der Ergebnisse - habe ich in zwei Schritte aufgeteilt. Dabei werden zunächst die benötigten Daten aus den einzelnen `Datenelementen` einer Gruppe durch den `Extractor` extrahiert und anschließend vom `Aggregator` aggregiert. Den Grund für diese Entscheidung erläutere ich in Abschnitt 5.2 *Extraktion und Aggregation der Statistiken* ab Seite 44. Die so gesammelten Ergebnisse werden in der `QueryResult`-Instanz abgespeichert und diese wird an den Aufrufer zurückgegeben.

```

1 public QueryResult<AggregatedType> run(RacingEventService
   racingEventService) {
2     final long startTime = System.nanoTime();
3
4     Collection<DataType> retrievedData = retriever.retrieveData(
       racingEventService);
5     Collection<DataType> filteredData = filter.filter(retrievedData);
6     QueryResultImpl<AggregatedType> result = new QueryResultImpl<
       AggregatedType>(retrievedData.size(), filteredData.size(),
       createResultSignifier());
7     Map<GroupKey, Collection<DataType>> groupedFixes = grouper.group(
       filteredData);
8     for (Entry<GroupKey, Collection<DataType>> groupEntry : groupedFixes.
       entrySet()) {
9         Collection<ExtractedType> extractedData = extractor.extract(
       groupEntry.getValue());
10        AggregatedType aggregatedData = aggregator.aggregate(extractedData);
11        result.addResult(groupEntry.getKey(), aggregatedData);
12    }
13
14    final long endTime = System.nanoTime();
15    result.setCalculationTimeInNanos(endTime - startTime);
16    return result;
17 }

```

Code 5.1: run-Methode der Query-Implementierung

Alle Schnittstellen des Frameworks haben generische Typangaben, um die Flexibilität des Systems zu maximieren. `DataType` stellt dabei den Typ dar, der als atomarer Datentyp verwendet werden, wie zum Beispiel GPS-Daten oder Teilnehmer. Der `ExtractedType` bestimmt den Typ der durch den `Extractor` aus dem Datenelement extrahiert werden soll. Dies wäre zum Beispiel die Geschwindigkeit in Form von `Speed`-Objekten oder die Platzierung des Teilnehmers in einem Rennen. Der `AggregatedType` ist der Typ der Ergebnisse, also zum Beispiel `Double` für die durchschnittliche Geschwindigkeit in Knoten.

Hinzu kommen weitere Schnittstellen, welche die Datenanalyse unterstützen. Dazu gehört die `Dimension`, die es ermöglicht den Wert der Dimension aus einem Datenelement auszulesen. Weitere sind `FilterCriteria`, welche von bestimmten Implementierungen der `Filter`-Schnittstelle verwendet werden und `GroupKey`, die der eindeutigen Identifikation von Gruppen dienen.

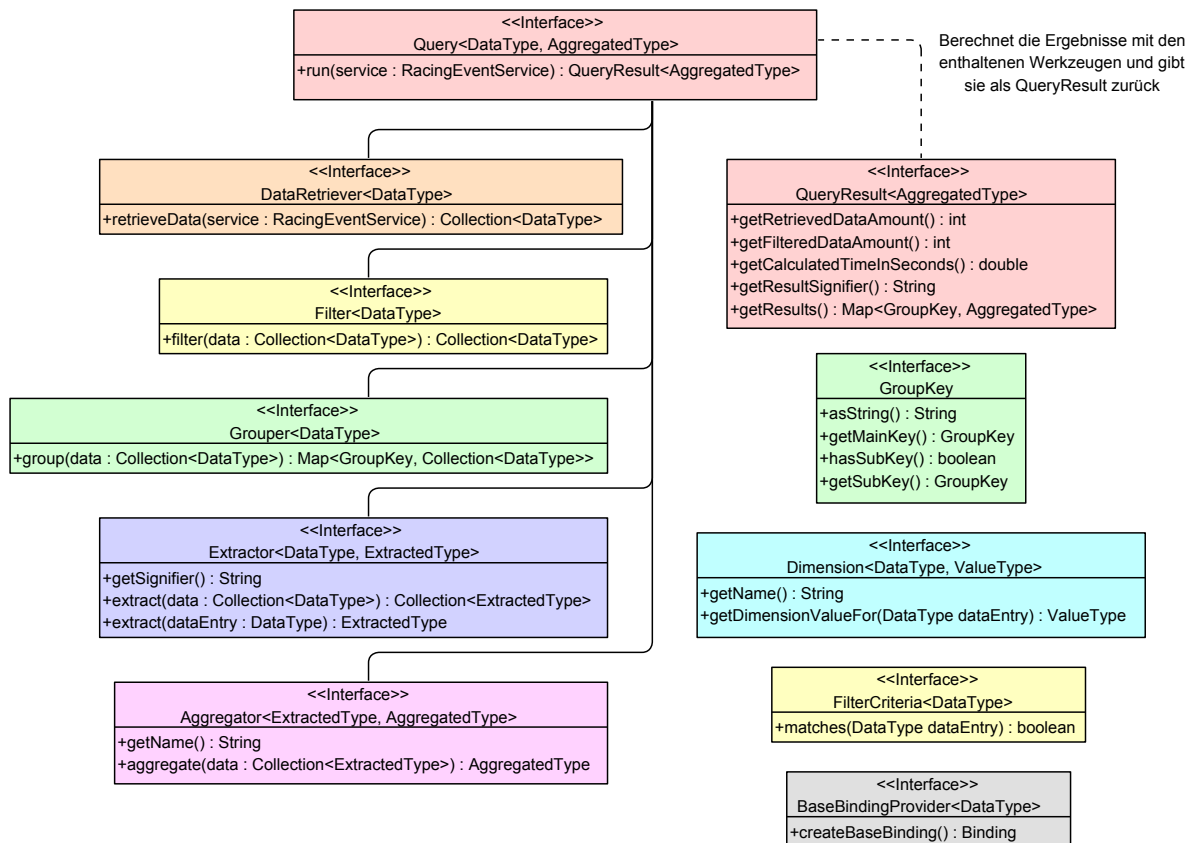


Abbildung 5.1.: Komponenten des Datenanalyse-Frameworks

5.1. Datenbeschaffung, Filterung und Gruppierung

Die Datenbeschaffung ist stark abhängig vom jeweiligen `DataType`, weswegen es keine allgemeinen Klassen gibt, die die `DataRetriever`-Schnittstelle implementieren. Eine konkrete Erläuterung wie die Datenbeschaffung von GPS-Daten abläuft folgt in Abschnitt 5.3 ab Seite 47. Das allgemeine Prinzip ist, dass das Datenmodell der Sailing Analytics nach dem `DataType` abgesucht wird und währenddessen Informationen über den Kontext des Datenelements gesammelt werden. Das Ergebnis der Datenbeschaffung soll eine Menge der Datenelemente mit ihrem Kontext sein.

Für die anschließende Filterung der Daten gibt es die Klasse `FilterByCriteria`, die diese anhand eines `FilterCriteria`s auswählt. Dabei wird über die Eingabemenge iteriert und für jedes Element die `matches`-Methode des Kriteriums ausgeführt. Ist der Rückgabewert *wahr*, wird das Element der Ausgabemenge hinzugefügt. Für die Kriterien gibt es mehrere Implementierungen. Die erste ist die abstrakte Klasse `RegexFilterCriteria`, welche einen String auf einen gegebenen regulären Aus-

druck überprüfen. Da nicht bekannt ist, um welchen Typ es sich bei dem Datenelement handelt, muss für das Element der zu überprüfende String ermittelt werden, was die Aufgabe der abstrakten Methode `getValueToMatch` ist. Eine einfache Möglichkeit, diese Ermittlung auszulagern, ist es, Dimensionen zu verwenden, welche als `ValueType` die Klasse `String` haben. Dann delegiert `getValueToMatch` an die Methode `getDimensionValueFor` der konkreten `Dimension`. Das zweite Filter-Kriterium wird durch `DimensionValuesFilterCriteria` realisiert. Dieses überprüft den Dimensions-Wert des übergebenen Datenelements auf Gleichheit mit einer Menge von Werten. Dieser Filter entspricht dem Filtern über bestimmten Werten einer *Facette*, wie es beim BOBJ-Explorer angewendet wird. Damit man nun nicht nur anhand eines Filter-Kriteriums filtern kann, gibt es das `CompoundFilterCriteria`, welches dem Kompositum-Pattern [GAM94, Seite 163] entspricht. Durch die beiden Unterklassen, welche die Menge der Kriterien verodern, beziehungsweise verunden, lassen sich beliebige logische Ausdrücke über die Kriterien bilden.

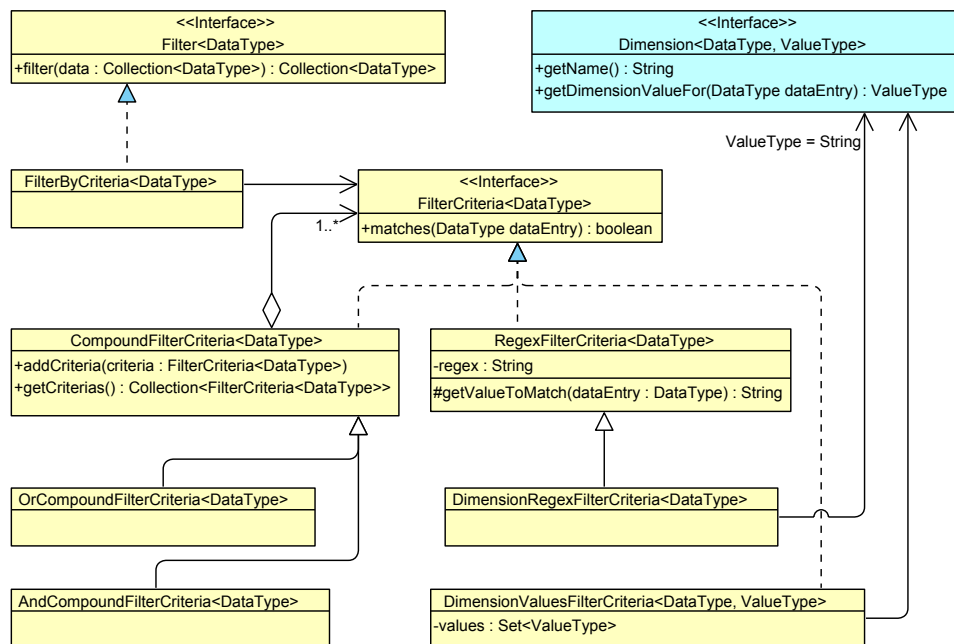


Abbildung 5.2.: Implementierung der Filter

Zur Gruppierung der gefilterten Daten werden die `Groupers` eingesetzt. Da der Vorgang des Gruppierens unabhängig von dem Typ der Daten ist, lässt sich dieser mit der abstrakten Oberklasse aller konkreten `Groupers` implementieren. Dabei wird für jedes Datenelement ein `GroupKey` erzeugt und überprüft, ob es schon eine Gruppe mit diesem Schlüssel gibt. Ist dies der Fall, wird das Element zu dieser Gruppe hinzugefügt; andernfalls wird zunächst eine neue Gruppe angelegt. Die Erstellung des `GroupKeys`

wird von den konkreten Unterklassen durch die Methode `getGroupKeyFor` implementiert.

```
1 public Map<GroupKey, Collection<DataType>> group(Collection<DataType> data)
    {
2     Map<GroupKey, Collection<DataType>> groupedData = new HashMap<GroupKey,
        Collection<DataType>>();
3     for (DataType dataEntry : data) {
4         GroupKey key = getGroupKeyFor(dataEntry);
5         if (key != null) {
6             if (!groupedData.containsKey(key)) {
7                 groupedData.put(key, new ArrayList<DataType>());
8             }
9             groupedData.get(key).add(dataEntry);
10        }
11    }
12    return groupedData;
13 }
14
15 protected abstract GroupKey getGroupKeyFor(DataType dataEntry);
```

Code 5.2: Implementierung der `group`-Methode von `AbstractGrouper`

Um komplexere Gruppierungen zu ermöglichen, besitzt jeder `GroupKey` das Potential aus einem Haupt- und einem Unterschlüssel zu bestehen, was durch entsprechende Methoden der Schnittstelle definiert wird. Deren Verhalten ist wie folgt festgelegt. Besteht ein Schlüssel aus mehreren Schlüssel, liefern die Methoden die erwarteten Elemente. Falls nicht liefert `getMainKey` sich selbst und `getSubKey` null zurück. Die abstrakte Oberklasse, von der alle konkreten Implementierungen erben müssen, gewährleistet durch die explizite Deklaration der `equals`- und `hashCode`-Methode, dass alle Schlüssel diese implementieren müssen. Ansonsten kann es zu Problemen bei der Gruppierung kommen, da diese durch die Verwendung einer `HashMap` auf dem *HashCode* der Schlüssel basiert. Nun gibt es zwei konkrete `GroupKey`-Implementierung, von denen der `GenericGroupKey` einen einfachen Schlüssel (also ohne Haupt- und Unterschlüssel) darstellt und mit dem sich ein beliebiges Objekt zur Gruppierung nutzen lässt. Die `equals` und `hashCode` Methoden basieren nun auf dem enthaltenen Objekt, was zu Problemen führt, sollten die Methoden des Objekts nicht sauber implementiert sein. Allerdings lässt sich mit sprachlichen Mitteln nicht erzwingen, dass Typen diese zwei Methoden implementieren müssen und auch das würde noch nicht gewährleisten, dass sie auch korrekt funktionieren. Allerdings würde es zu erheblichen

Einschränkungen führen, wenn für jeden Typ ein eigener `GroupKey` implementiert werden müsste, da man diese nicht einfach in den anderen Komponenten verwenden kann, da in diesen der Typ der für die Gruppierung verwendet wird in der Regel nicht definiert ist. Dadurch muss bei der Entwicklung darauf geachtet werden, dass nur korrekte Schlüssel erzeugt werden. Mit dem `CompoundGroupKey` lassen sich hierarchische Schlüssel bilden. Da so ein zusammengesetzter Schlüssel aus maximal zwei `GroupKeys` bestehen kann, ist deren Reihenfolge eindeutig und es lassen sich beliebig Tiefe Hierarchien bilden, indem man mehrere `CompoundGroupKeys` ineinander verschachtelt.

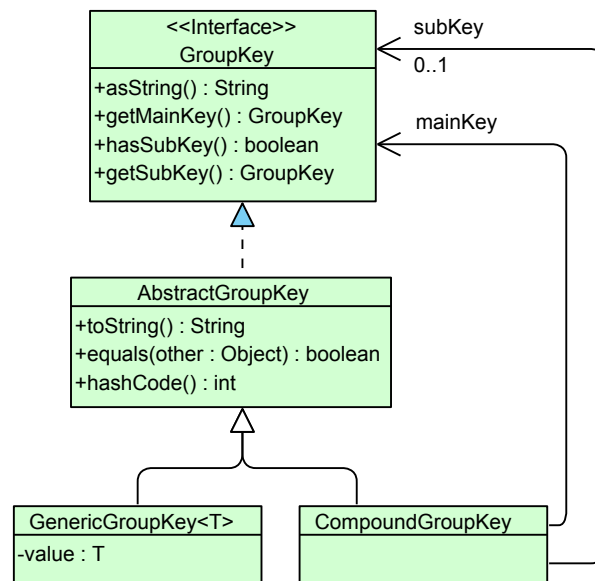


Abbildung 5.3.: Implementierung von `GroupKey`

Zur Gruppierung der Daten gibt es zwei konkrete `Grouper`. Die erste Implementierung `GroupByDimension` hat eine Referenz auf eine Menge von Dimensionen aus denen er den `GroupKey` erzeugt. Dabei wird für jede Dimension ein `GenericGroupKey` erzeugt, der als Wert den Dimensions-Wert des Dateneintrages erhält. Aus diesen Schlüsseln wird, falls es mehr als einer sind, ein `CompoundKey` zusammengesetzt. Die Reihenfolge der referenzierten Dimensionen bestimmt die Hierarchie des `CompoundKeys`.

Eine weitere `Grouper`-Implementierung ist der `DynamicGrouper`. Dieser ist der erste Ansatz, um dem Anwender die Möglichkeit zu geben direkt auf den Daten zu arbeiten und selbst zu bestimmen, wie mit diesen umgegangen werden soll. Dadurch wird die Flexibilität und die Möglichkeiten der Datenanalyse massiv erhöht. Ein konkreter Anwendungsfall dieses `DynamicGroupers` folgt in Abschnitt 5.3 und zukünftige Verwendungszwecke werden in Kapitel 6 *Ausblick* erläutert. Das Prinzip des

DynamicGroupers ist es, dass er vom Anwender eine Reihe von Befehlen erhält (in diesem Fall in Form eines Groovy-Skripts), die einen Wert zurückliefern. Mit diesem wird ein GenericGroupKey erzeugt, anhand dem die Daten gruppiert werden. Zur Konstruktion des DynamicGrouper wird das Skript in Form eines Strings benötigt, sowie ein BaseBindingProvider. Der String wird durch die Klasse GroovyShell zu einem Script-Objekt parsed. Der BaseBindingProvider erzeugt dem Grouper eine Binding-Instanz mit allen für den entsprechenden Datentyp benötigten Elementen. Das Binding verbindet das Groovy-Skript mit der Java-Umgebung des Frameworks. So wird der aktuelle Dateneintrag dem Skript als Variable data zur Verfügung gestellt. Ansonsten würde es zu einem Fehler kommen, sobald der Anwender ein Skript schreibt, welches auf den Dateneintrag zugreifen möchte. Den Code der Implementierung des DynamicGroupers findet sich in Code A.1 des Appendix.

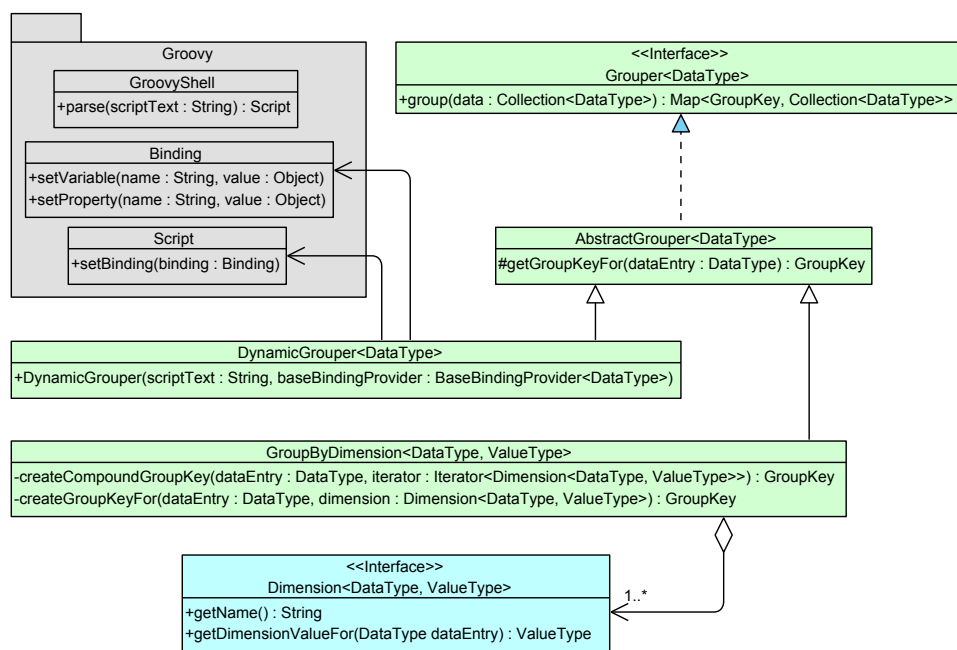


Abbildung 5.4.: Implementierung der Grouper

5.2. Extraktion und Aggregation der Statistiken

Der in Abbildung 3.3 als Aggregation bezeichnete Schritt wird im Framework durch zwei einzelne Schritte durchgeführt. Als erstes werden aus den gruppierten Daten die gesuchten Informationen extrahiert, woraufhin diese von einem Aggregator aggregiert werden. Dies hat den Vorteil, dass man allgemeine Aggregationsverfahren (wie zum Beispiel das bilden der Summe oder des arithmetischen Mittels) implementieren kann und nicht für jede Statistik einen eigenen Aggregator benötigt, der mit dem Typ der berechneten Statistik umgehen kann. Voraussetzung dafür ist, dass der Ausgabotyp des `Extractors` von diesen allgemeingültigen Aggregatoren verarbeitet werden kann.

Die Extraktoren sind wie die `DataRetriever` in der Regel stark abhängig vom Typ der atomaren Daten. Unabhängig davon ist lediglich der `DataAmountExtractor`, der als extrahierten Wert die Größe der übergebenen Datenmenge zurückgibt. Daneben gibt es den `SpeedInKnotsExtractor`, welcher als Eingabe eine Menge von `Movings` erwartet. Diese Schnittstelle ist Teil des Datenmodells der Sailing Analytics und definiert, dass deren Implementierungen eine Methode haben, welche ein `SpeedWithBearing`-Objekt zurückgibt. Daraus lässt sich wiederum die Geschwindigkeit in Knoten als Gleitkommazahl ermitteln. Zum Beispiel verwendet `GPSFixMoving` dieses Interface. Die abstrakte Oberklasse dieser beiden Extraktoren enthält keine Logik, sondern verwaltet einen Bezeichner, der den extrahierten Wert beschreibt. Dieser wird für die spätere Präsentation der Ergebnisse verwendet.

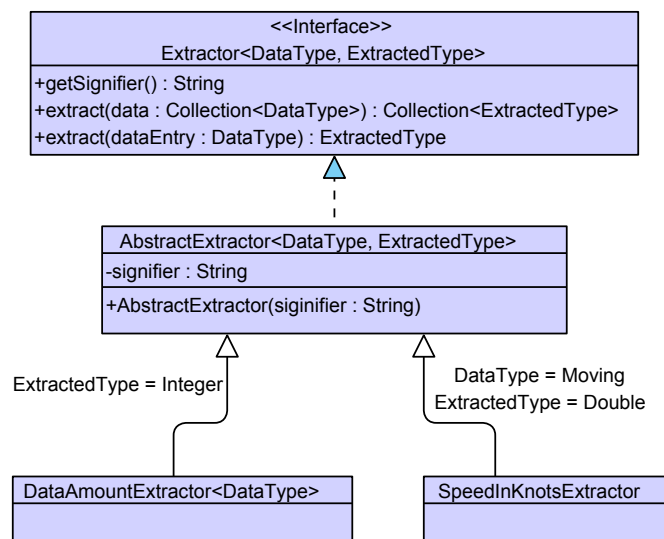


Abbildung 5.5.: Implementierung der Extraktoren

Zur Zeit gibt es Aggregatoren zur Bildung der Summe und des arithmetischen Mittels. Die abstrakte Oberklasse verwaltet lediglich den Namen des Aggregators, der auch für die Präsentation der Ergebnisse genutzt wird. Die Summe wird nach dem unten abgebildeten Algorithmus berechnet. Damit Summen über eine Menge beliebiger Typen gebildet werden können, definiert der abstrakte `SumAggregator` zwei Methoden. Die Erste holt aus dem Typ der extrahierten Daten den Wert des zu aggregierenden Typs, während die Zweite zwei Werte des zu aggregierenden Typs addiert. Da der extrahierte und der zu aggregierende Typ nicht immer unterschiedlich sind, gibt es den `SimpleSumAggregator`. Dieser definiert, dass diese beiden Typen identisch sind und implementiert die `getValueFor`-Methode so, dass der eingegebene Wert direkt zurückgegeben wird. Die zwei konkreten Unterklassen implementieren noch die fehlende `add`-Methode für `Integer` und `Double`.

```
1 @Override
2 public AggregatedType aggregate(Collection<ExtractedType> data) {
3     Iterator<ExtractedType> dataIterator = data.iterator();
4     AggregatedType sum = null;
5
6     while (dataIterator.hasNext()) {
7         if (sum == null) {
8             sum = getValueFor(dataIterator.next());
9             continue;
10        }
11
12        sum = add(sum, getValueFor(dataIterator.next()));
13    }
14    return sum;
15 }
16
17 protected abstract AggregatedType add(AggregatedType value1, AggregatedType
    value2);
18 protected abstract AggregatedType getValueFor(ExtractedType extractedValue)
    ;
```

Code 5.3: Algorithmus zu Bildung des Summen-Aggregats

Der Aggregator zur Bildung des arithmetischen Durchschnitts verwendet einen entsprechenden `SumAggregator` zur Berechnung der Summe. Das Ergebnis wird anschließend mit der abstrakten Methode `divide` durch die Größe der eingegebenen Datenmenge dividiert, was dem arithmetischen Mittel entspricht. Wie bei den Summen-

Aggregatoren gibt es konkrete Implementierungen in denen der `ExtractedType` und `AggregatedType` vom Typ `Integer` beziehungsweise `Double` ist. Diese verwenden ihren entsprechenden `SumAggregator` zur Bildung der Summe.

```

1 @Override
2 public AggregatedType aggregate(Collection<ExtractedType> data) {
3     return divide(sumAggregator.aggregate(data), data.size());
4 }
5
6 protected abstract AggregatedType divide(AggregatedType sum, int dataAmount
    );

```

Code 5.4: Algorithmus zu Bildung des Durchschnitts-Aggregats

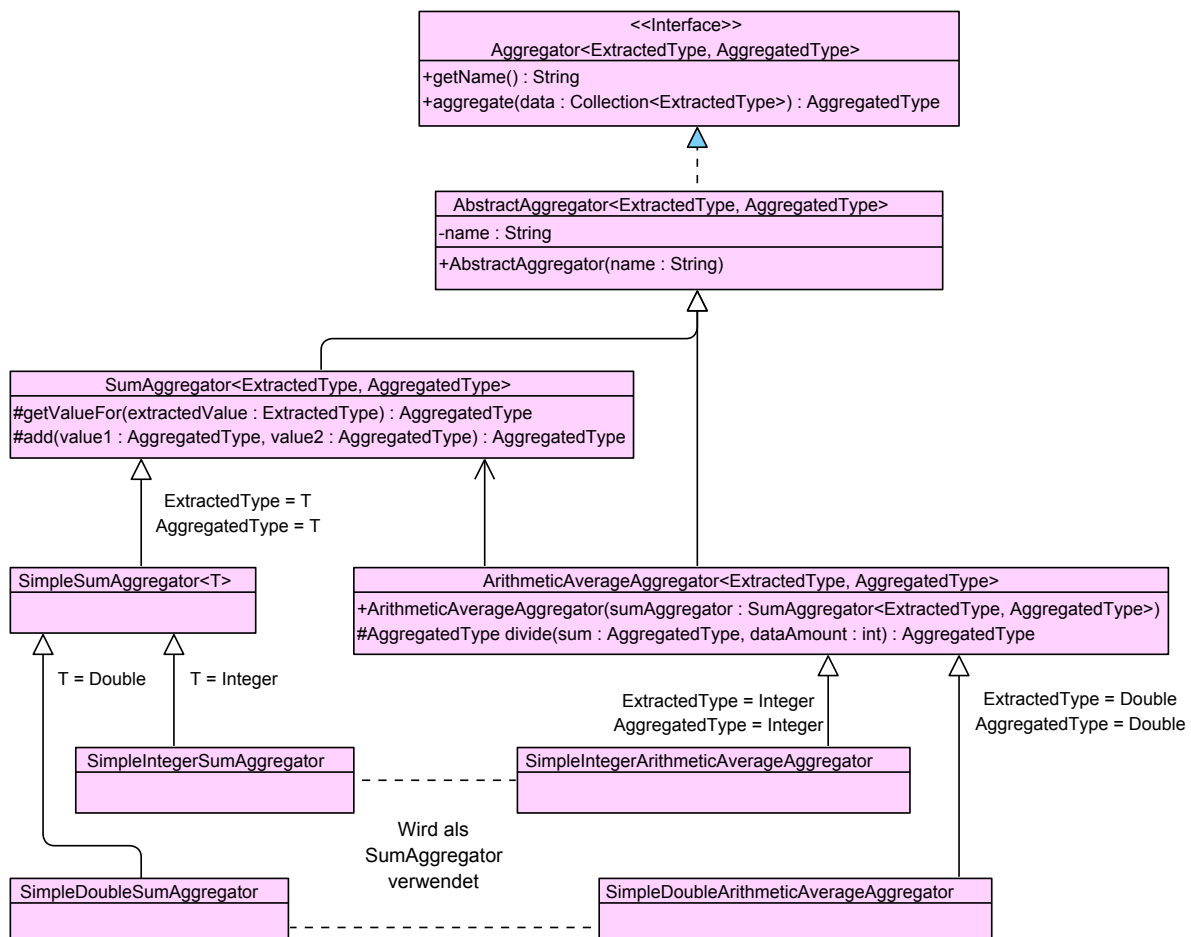


Abbildung 5.6.: Implementierung der Aggregatoren

5.3. Datenanalyse über die GPS-Daten

In diesem Abschnitt möchte ich vorstellen, wie man das oben beschriebene Framework zur Datenanalyse über den GPS-Daten verwenden kann. Als atomarer Datentyp, und demnach der generische Typ-Parameter `DataType`, wird `GPSFixWithContext` verwendet, welcher den allgemeinen `GPSFixMoving` um einen Kontext erweitert. Dadurch wird das über eine Position, eine Geschwindigkeit und dem Zeitpunkt der Messung verfügende GPS-Datenobjekt mit weiteren Daten angereichert. Der Kontext wird in einem eigenständigen Objekt gekapselt und enthält Referenzen auf alle relevanten Elemente des Datenmodells, wie zum Beispiel der Regatta, dem Rennen und dem Teilnehmer. Die Sammlung der GPS-Daten und deren Kontextes wird durch den `SimpleGPSFixRetriever` durchgeführt. Dieser beginnt bei allen auf dem Server vorhandenen Leaderboard-Gruppen und wandert das Datenmodell bis zu den getrackten Schenkeln eines Teilnehmers herunter. Auf diesem Weg erhält er alle Informationen des Kontextes, die benötigt werden, um einen `GPSFixWithContext` zu erzeugen.

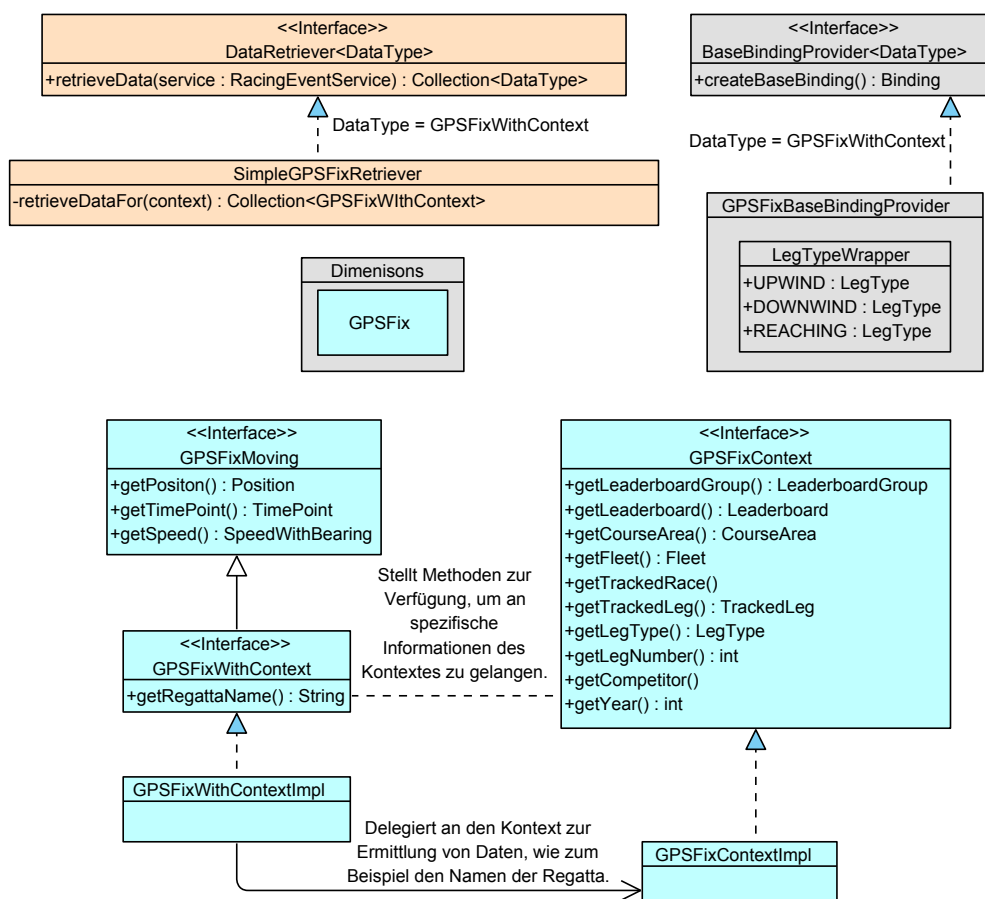


Abbildung 5.7.: Implementierung der GPS-spezifischen Framework-Komponenten

Zu Testzwecken habe ich eine einfache Oberfläche entwickelt, um Datenanalyse-Anfragen zu definieren und sich deren Ergebnisse anzeigen zu lassen. Dabei habe ich mich an der Oberfläche des BOBJ-Explorers orientiert. So sind im oberen Teil eine Reihe von Tabellen, anhand denen man definieren kann welche Daten bei der Analyse berücksichtigt werden sollen. Werden keine Einträge der Tabellen selektiert, findet keine Filterung der Daten statt. In der Mitte wird ausgewählt anhand welcher Kriterien die Daten gruppiert werden sollen. Dabei kann man wählen ob die Gruppierung mit einer Menge von Dimensionen (welche Dimensionen für GPS-Daten definiert sind, ist in Abbildung A.1 im Appendix zu sehen) durchgeführt werden soll oder ob ein eigenes Skript die Gruppen bestimmt. Rechts daneben wird aus einer Liste die Statistik ausgewählt, die berechnet werden soll. In diesem Fall steht die Anzahl der Daten jeder Gruppe oder die durchschnittliche Geschwindigkeit in Knoten zur Verfügung. Die vom Nutzer ausgewählten Elemente werden nun an den Server übertragen. Dort wird mit Hilfe einer Factory-Klasse eine *Query* erzeugt und die Anfrage durchgeführt. Das Ergebnis wird zurück an den Client gesendet und im unteren Teil der Oberfläche als Balkendiagramm dargestellt, was in Abbildung 5.10 zu sehen ist. Dabei verwende ich wie bei den anderen Diagrammen der Sailing Analytics die GWT-Verschaltung der *Highcharts*.

Regatta	Boat Class	Race	Leg	Leg Type	Competitor	Sail Number	Nationality
KW 2013 International (29ER)	29ER	29er blue race 1	1	UPWIND	Marcos Vivian	AUS 9105	HKG
KW 2013 International (505)	505	29er blue race 2	2	DOWNWIND	Katharina LEUKEL	GER 606	IND
KW 2013 International (EUR)	EUR	29er blue race 3	3	REACHING	Sven JÜRGENSEN	GER 215	AUS
KW 2013 International (F18)	F18	29er blue race 4	4		THEO Vallet	FRA 1747	FRA
KW 2013 International (Folkeboot)	Folkeboot	29er blue race 5	5		Denis KHASHINA	UKR 8205	DEN
KW 2013 International (H-Boat)	H-Boat	29er blue race 6	6		Marius LOEKEN	NOR 7	HUN
KW 2013 International (H16)	H16	29er blue race 7	7		Ulla BECKER	GER 294	GBR
KW 2013 International (L4.7)	L4.7	29er gold race 10	8		Christian MÜLLEJANS	GER 8400	SLO
KW 2013 International (STR)	STR	29er gold race 11	9		Helge SACH	GER 277	POL
		29er gold race 12	10		Max Stingele	GER 2180	GER
		29er gold race 13			Helge SPEHR	GER 7747	USA
		29er gold race 14			Marie Dominique SCHMIDT	GER 1728	NED
		29er gold race 8			BERTRAND LOYAL	FRA 9076	THA
		29er gold race 9			Finja PRIESE	GER 1832	NOR
		29er silver race 10			SCHMEINK Harry	GER 236	SUI
		29er silver race 11			Hans GENEKE	DEN 9109	SWE
		29er silver race 8			Finn Heeg	GER 232	CZE
		29er silver race 9			Jacob Clasen	GER 2159	AUT

Clear Selection
Group by: Dimensions
Calculate statistic: Speed (Average)

RegattaName
LegType

Abbildung 5.8.: Selektionsbereich der Datenanalyse-Oberfläche

Würde der Anwender in der Tabelle für die Bootsklassen die Einträge *29er* und *Star* und in der Tabelle für die Nationalität der Teilnehmer *AUS*, *FRA* und *GER* selektieren, würden im Server die nachfolgende Objektstruktur erzeugt werden, wobei für die Datenanalyse irrelevante Attribute (wie der *signifier* des Extraktors) vernachlässigt werden.

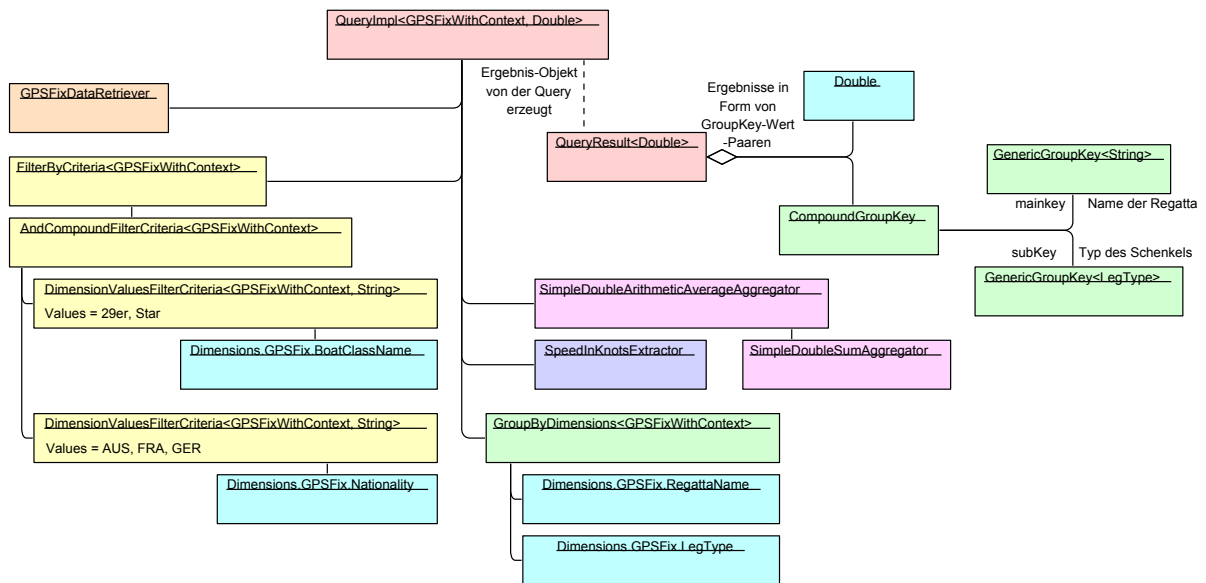


Abbildung 5.9.: Objektdiagramm einer Beispiel-Query

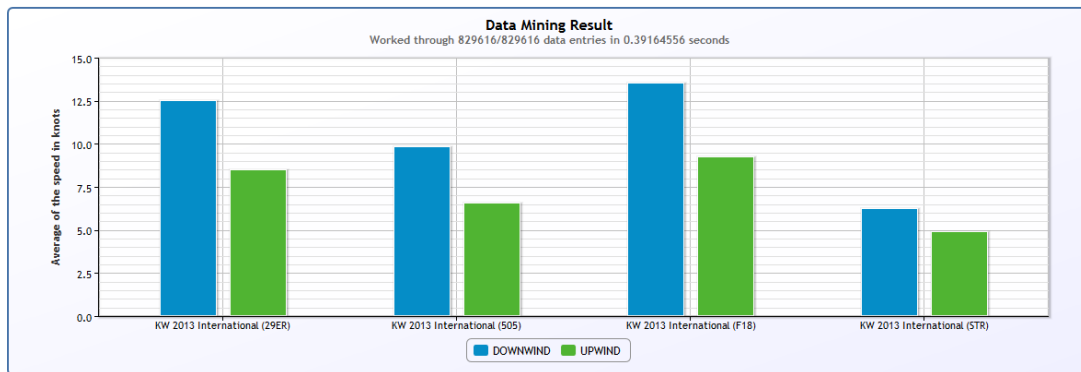


Abbildung 5.10.: Darstellung der Ergebnisse einer Datenanalyse

Anhand der Datenanalyse über den GPS-Daten möchte ich die Funktionsweise des `DynamicGroupers` näher erläutern. Wählt der Anwender aus, dass er die Gruppierung der Daten selbst bestimmen möchte, wird ein Textfeld angezeigt in welches das Skript eingegeben wird (s. Abbildung 5.11 auf der nächsten Seite). Der Anwender kann in dem Skript auf alle Methoden der Schnittstelle `GPSFixWithContext` zugreifen und mit deren Rückgabewerten arbeiten. Weitere Klassen werden durch den entsprechenden `BaseBindingProvider` in das `Binding` des Skripts eingefügt. Im Falle von GPS-Daten ist das der in Abbildung 5.7 dargestellte `GPSFixBasBindingProvider` (Implementierung s. Code A.2 im Appendix), der eine Instanz des `LegTypeWrappers` mit dem Bezeichner `LegType` einbindet. Der `LegTypeWrapper` hat drei statische Konstanten, welche auf die entsprechenden `LegTypes` verweisen. Andernfalls würde das in der Abbildung eingegebene Skript beim parsen einen Fehler erzeugen, da `LegType` zu keiner Variable oder Klasse zugeordnet werden kann. Anstatt einen Wrapper zu verwenden, ließen sich die Schenkel-Typen auch einzeln einbinden, wobei das rein funktional keinen Unterschied macht.

Durch den `DynamicGrouper` ist es mit dem Skript „`return data.getCompetitorName().length();`“ zum Beispiel möglich herauszufinden, dass Teilnehmer der Bootsklasse *29er* mit einem 11 Ziffern langen Vor- und Nachnamen (einschließlich der Leerzeichen) mit einem Ergebnis von ungefähr 16,2 Knoten die höchste durchschnittliche Geschwindigkeit haben (s. Abbildung 5.12 auf der nächsten Seite), während der Durchschnitt aller Teilnehmer derselben Datenmenge nur 9,86 Knoten beträgt. Der Sinn hinter dieser Datenanalyse ist fraglich, allerdings wird dadurch verdeutlicht, dass dem Anwender durch die Verwendung dynamische ausgeführter Skripte alle Möglichkeiten offen sind.

Clear Selection

Group by: Custom

Calculate statistic: Speed (Average)

```

public Object getValueToGroupByFrom(GPSFix data) {
    if (data.getLegType == LegType.REACHING) {
        return "Reaching";
    } else {
        return "Up- or Downwind";
    }
}

```

Abbildung 5.11.: Oberfläche zur eigenen Gruppierung

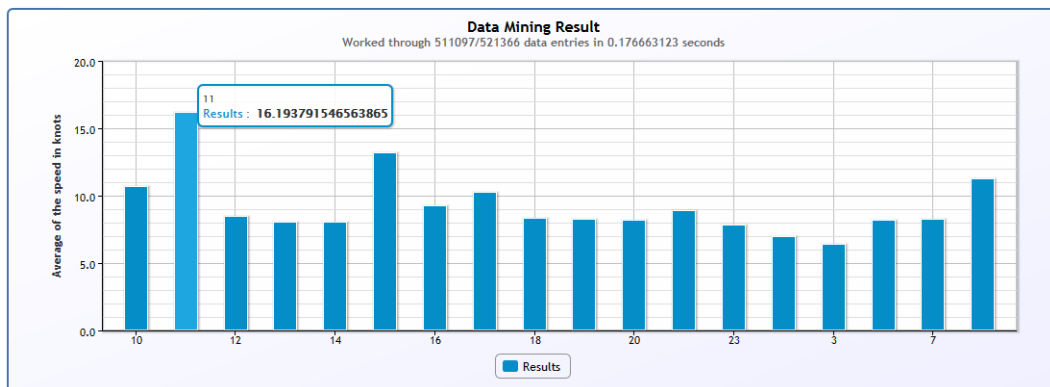


Abbildung 5.12.: Durchschnittliche Geschwindigkeit in Knoten gruppiert nach Länge des Teilnehmer-Namens

5.4. Performance des Frameworks

Zur Performance-Analyse meines Frameworks habe ich die Oberfläche so erweitert, dass man die im oberen Teil definierte Anfrage eine angegebene Anzahl oft durchführen kann. Dabei wird die benötigte Zeit und die Anzahl der bearbeiteten Datenelemente jedes Durchgangs gespeichert. Die Ergebnisse werden mit einem Highcharts-Diagramm dargestellt (s. Abbildung 5.13 auf der nächsten Seite), wobei die grüne Linie die Gesamtzeit (also einschließlich dem Senden der Anfrage und Empfangen des Ergebnisses) und die blaue Linie die Dauer der Berechnung auf dem Server repräsentiert. Dabei traten des öfteren extreme Ausreißer auf, welche durchaus den zehnfachen Wert des Durchschnitts hatten, was auf die Ressourcenvergabe des laufenden Systems zurückzuführen ist. Damit die durchschnittlich benötigte Zeit nicht verfälscht wird, werden diese mit folgendem Verfahren aus der Ergebnismenge entfernt. Der logische Hintergrund des Verfahrens wird auch bei der Erstellung von Kastengrafiken¹ angewendet. Dabei berechne ich zunächst das untere und obere Quartil² der benötigten Zeit und damit den Interquartilsabstand. Aus diesem Wert und dem Median ergeben sich folgende Grenzwerte:

$$\text{UntererGrenzwert} = \text{Median} - (\text{Interquartilsabstand} * \text{GrenzwertFaktor})$$

$$\text{ObererGrenzwert} = \text{Median} + (\text{Interquartilsabstand} * \text{GrenzwertFaktor})$$

Der Grenzwert-Faktor bestimmt dabei wie weit ein Punkt vom Median entfernt sein darf, bevor er als Ausreißer klassifiziert wird. In der Praxis hat sich ein Wert von 5 als sinnvoll erwiesen, damit nur die extremen Messwerte ausgeschlossen werden.

¹ Für eine grafische Darstellung und nähere Informationen zu Kastengrafiken siehe <https://de.wikipedia.org/wiki/Boxplot>

² [STATISTA13]: „Ein Quantil legt fest, wie viele Wert einer Verteilung über oder unter einer bestimmten Grenze liegen.“ Quartile sind spezielle Quantile, die die Verteilung vierteln. Demnach sind 25%/75% der Wertemenge kleiner als das untere/obere Quartil.

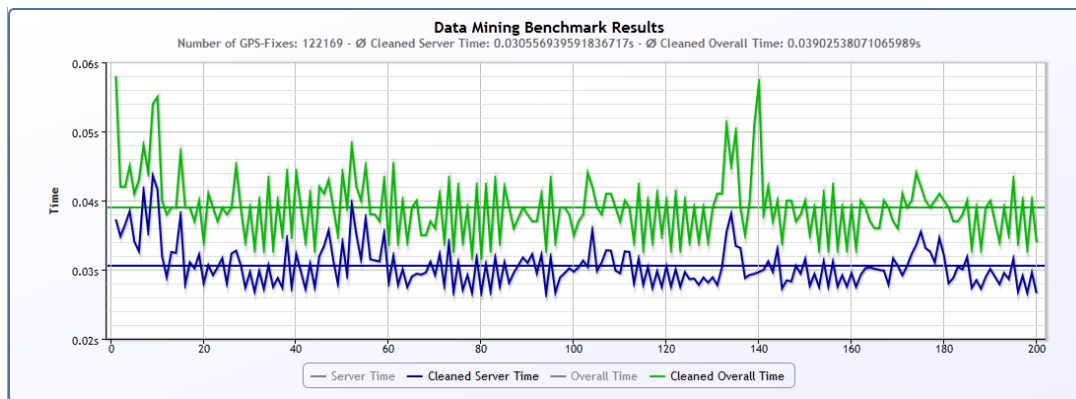


Abbildung 5.13.: Benchmark-Ergebnisse bei 200 Anfragen mit jeweils ca. 122.000 Dateneinträgen. Die Daten wurden anhand des Regattenamens durch eine `GroupByDimension`-Instanz gruppiert. Der Benchmark wurde mit dem Laptop durchgeführt.

Die nachfolgenden Ergebnisse wurden im lokalen Testmodus von GWT gemessen. Das bedeutet, dass sowohl Client als auch der Server auf der selben physischen Maschine liefen. Dabei wurden folgende Rechnerkonfigurationen verwendet:

- **PC** Vier-Kerner i7-2600k mit 3,8 GHz und Hyperthreading
- **Laptop** Vier-Kerner i7-2620M mit 2,7 GHz

Alle nachfolgenden Angaben der Zeiten beziehen sich auf die Rechenzeit des Servers die der PC zur Durchführung der Anfrage benötigt hat. Die benötigten Zeiten des Laptops sind dahinter in Klammern angegeben. Dabei ist zu beachten, dass die Zeiten im Produktivbetrieb kürzer ausfallen, da der lokale Testmodus von GWT den Ablauf verlangsamt. Die Grafiken der einzelnen Geschwindigkeitsmessungen befinden sich in Abschnitt A.3 des Appendix.

Bei der Messung der Performance hat sich herausgestellt, dass die Geschwindigkeit in einem frühen Stadium des Frameworks nicht sehr hoch war (ungefähr 2,5 Sekunden bei 300.000 Dateneinträgen), da auch rechenintensive Elemente des Kontextes, wie zum Beispiel der Typ des Schenkels oder der Wind, direkt bei der Erstellung des Kontextes berechnet wurden, auch wenn diese in der Anfrage gar nicht verwendet wurden. Nachdem ich diese Elemente *verzögert initialisiert*¹ habe (s. Code A.3 im Appendix), ist die Performance in allen Abfragen, die diese Elemente nicht benötigen, wesentlich besser. In diesen Fällen werden 1,25 Millionen Dateneinträge in 0,267 (0,311) Sekunden

¹ Englisch lazy initialization: Bezeichnet die Vorgehensweise die Erstellung eines Objektes so weit hinauszuzögern, bis es wirklich benötigt wird.

abgearbeitet. Diese Datenmenge entspricht 58 Rennen von neun Bootsklassen¹. Für die übrigen Anfragen lässt sich die Performance weiter verbessern, in dem man einzelne Instanzen des Kontextes möglichst oft wiederverwendet und so viele Datenelemente wie möglich in diesem verzögert initialisiert. So ist es bei der aktuellen Implementierung der Fall, dass alle GPS-Daten eines Schenkels eines Teilnehmers eine Referenz auf dieselbe Kontext-Instanz haben. Dadurch wird eine Anfrage, welche den Schenkel-Typ benötigt, bei 1,25 Millionen Dateneinträgen in durchschnittlich 0,526 (0,687) Sekunden verarbeitet. Hätte jeder Dateneintrag ein eigenes Kontext-Objekt werden für die gleiche Anfrage 31,11 (39,225) Sekunden benötigt. Allerdings bleibt das Performance-Problem für Anfragen, welche den Wind verwenden bestehen, da der Wert des Windes abhängig von der Position und dem Zeitpunkt ist. Dies sind Eigenschaften der GPS-Daten und können somit nicht in den Kontext ausgelagert werden. Dazu kommt, dass die Berechnung der Winddaten sehr rechenintensiv ist. Allerdings werden die Ergebnisse der Windberechnung innerhalb der Sailing Analytics zwischengespeichert, sodass nur die erste Anfrage oder eine Anfrage nach weiträumigen Änderungen der Daten sehr langsam ist. Diese benötigt bei 1,25 Millionen Dateneinträgen ungefähr 20 (25) Sekunden. Alle darauf folgenden werden in durchschnittlich 6,432 (10,221) Sekunden abgearbeitet.

Besonders bei der Implementierung von Komponenten, die Groovy-Skripte verwenden sollen, ist darauf zu achten, wie die Skripte geparsed und ausgeführt werden. Denn je nachdem wie diese implementiert wurden, kann dies massive Auswirkungen auf die Laufzeit haben. Besonders deutlich wird dies anhand der beiden nachfolgenden `DynamicGrouper` Implementierungen. Die erste Variante ist sehr langsam, da der eingegebene Skript-String für jeden Dateneintrag neu geparsed und ausgeführt wird. Dadurch beträgt die Rechenzeit für eine Anfrage mit dem Skript „`return data.getRegattaName();`“ über ungefähr 10.000 Dateneinträge bereits 26,445 (38,704) Sekunden.

¹ Zum Vergleich: Bei der Kieler Woche 2013 wurden 64 Rennen der internationalen Bootsklassen von der Sailing Analytics aufgezeichnet.

```

1 private String scriptText;
2
3 public DynamicGrouper(String scriptText) {
4     super();
5     this.scriptText = scriptText;
6 }
7
8 @Override
9 protected GroupKey getGroupKeyFor(DataType dataEntry) {
10     Binding binding = new Binding();
11     binding.setVariable("data", dataEntry);
12     GroovyShell shell = new GroovyShell(binding);
13     return new GenericGroupKey<Object>(shell.evaluate(scriptText));
14 }

```

Code 5.5: Implementierung der langsamen Groovy-Skript Ausführung

Die zweite Variante hingegen parsed den Skript-String direkt bei der Konstruktion des `DynamicGroupers` und speichert das Resultat in einer `Script`-Instanz ab. Zudem wird auch direkt ein `Binding`-Objekt erzeugt und bei dem Skript registriert. Nun wird für jeden Dateneintrag lediglich das aktuelle Datenobjekt eingebunden und das Skript ausgeführt. Damit wird erreicht, dass eine identische Anfrage über 1,25 Millionen Dateneinträgen in durchschnittlich 0,291 (0,340) Sekunden abgearbeitet wird. Dies ist lediglich 0,0236 (0,029) Sekunden langsamer, wie wenn die Anfrage mit einem nativ implementierten `Dimensions-Grouper` durchgeführt worden wäre.

```

1 private Script script;
2 private Binding binding;
3
4 public DynamicGrouper(String scriptText) {
5     super();
6     script = new GroovyShell().parse(scriptText);
7     binding = new Binding();
8     script.setBinding(binding);
9 }
10
11 @Override
12 protected GroupKey getGroupKeyFor(DataType dataEntry) {
13     binding.setVariable("data", dataEntry);
14     return new GenericGroupKey<Object>(script.run());
15 }

```

Code 5.6: Implementierung der performanten Groovy-Skript Ausführung

6. Ausblick

In diesem Kapitel möchte ich vorstellen, wie die Arbeit an dem Framework fortgeführt werden könnte und welche Erweiterungen zukünftig vorstellbar sind.

6.1. Erweiterung der Analysemöglichkeiten

Der nächstliegende Schritt ist es die Analysemöglichkeiten des Frameworks zu erweitern. Dabei geht es sowohl darum mehr Kennzahlen über die GPS-Daten oder Gruppierungs-Verfahren dieser zur Verfügung zu stellen, als auch weitere atomare Datentypen zu implementieren.

Zum Beispiel ließen sich die GPS-Daten anhand ihrer Positionierung gruppieren, wobei der `Group` Gebiete in denen die Daten konzentriert sind automatisch erkennen sollte und diese Gebiete den `GroupKey` darstellen. Ein Ansatz hierfür wäre die Positionen der Daten in einen `QuadTree`¹ einzutragen und durch diesen die Gebiete zu erkennen (s. Abbildung 6.1 auf der nächsten Seite). Eine weitere Option die Analysemöglichkeiten der GPS-Daten zu erweitern, wäre eine Filterung anhand der Position der Daten. Dabei könnte der Anwender angeben von welchen Positionen und mit welchem Radius er die Daten untersuchen möchte. Hierfür ließe sich einfach ein neues Filter-Kriterium implementieren.

¹ Ein `QuadTree` ist eine spezielle Baumstruktur, in der jeder Knoten genau vier Kinder hat (s. <https://de.wikipedia.org/wiki/Quadtree>).

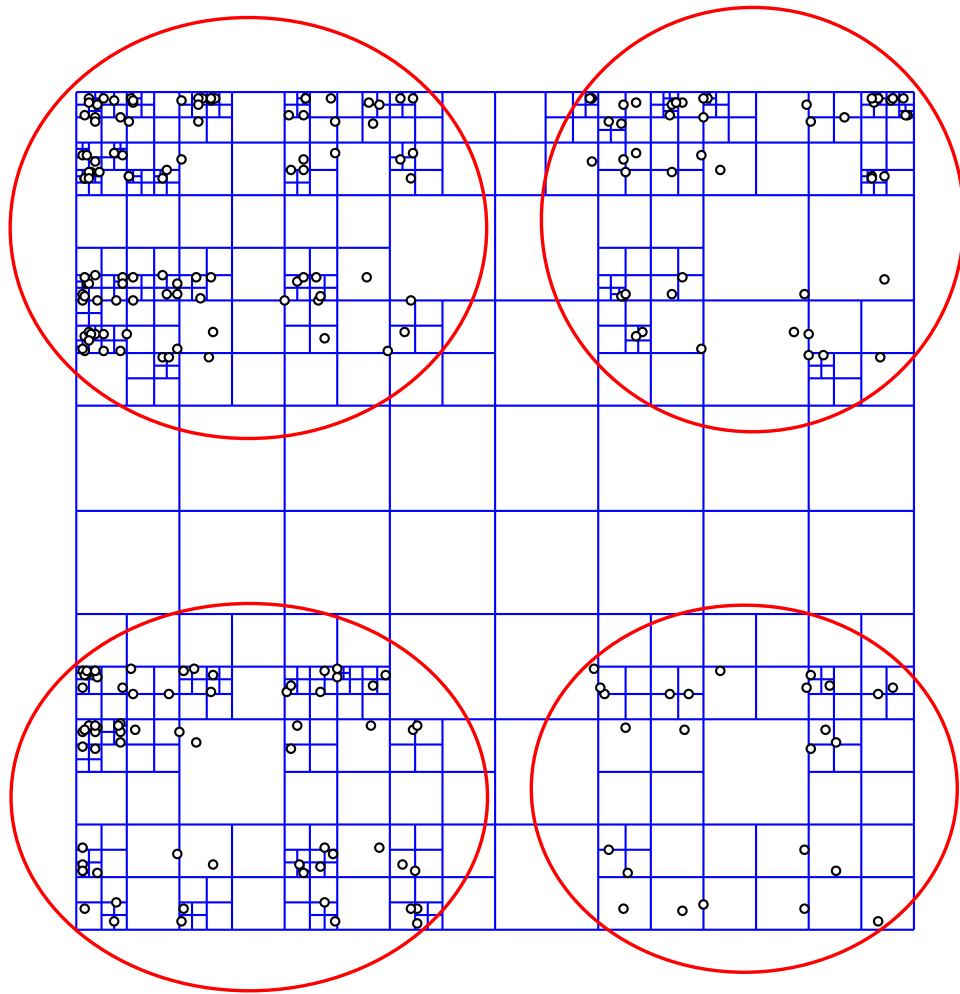


Abbildung 6.1.: Schematische Darstellung eines QuadTrees. Die eingekreisten Teile sollen jeweils eine Gruppe ergeben.

Quelle: https://en.wikipedia.org/wiki/File:Point_quadtree.svg

Als weitere atomare Datentypen wären prinzipiell alle Elemente des Datenmodells der Sailing Analytics möglich. Für die Umsetzung müsste ein `DataRetriever` dieses Typs und die Extraktoren der gewünschten Kennzahlen implementiert werden. Für eine einfache Form der Filterung und Gruppierung müssten lediglich neue Dimensionen definiert werden, sodass man den `Dimensions-Groupier` und das `DimensionsValue-FilterCriteria` wiederverwenden könnte. Interessante neue Datentypen wäre zum Beispiel der Wind, da er ein für Sailing Analytics grundlegendes Datenelement darstellt. Des weiteren könnte man auch ganze Schenkel eines Teilnehmers oder Rennen als atomare Datentypen verwenden, da diese bereits viele Methoden zur Verfügung stellen, um statistische Kennzahlen zu berechnen, womit die Implementierung von Extraktoren vereinfacht wäre.

Ein weiterer für die Zukunft relevanter Punkt ist die Behandlung von Daten gerade stattfindender Rennen. In der aktuellen Version des Frameworks würden einfach die vorhandenen Daten gesammelt werden, sodass nur ein Bruchteil der Renndaten zur Verfügung stehen. Für die Behandlung von Live-Rennen gäbe es zwei Möglichkeiten. Zum einen könnten die Daten dieser Rennen ignoriert werden, indem der `DataRetriever` überprüft, ob ein Rennen gerade stattfindet und dieses gegebenenfalls überspringt. Dabei wäre auch denkbar, dass der Anwender einstellen kann, ob er Live-Rennen überspringen oder deren vorhandenen Daten in die Analyse einbeziehen möchte. Eine wesentlich elegantere, aber auch aufwendigere Behandlung wäre, die Präsentation der Ergebnisse regelmäßig zu aktualisieren. Dabei wäre es von Vorteil, wenn es möglich wäre die Ergebnisse einer Anfrage mit den neuen Daten anzupassen, ohne die komplette Anfrage erneut ausführen zu müssen. Die neuen Ergebnisse würden von der Benutzeroberfläche regelmäßig abgefragt werden, woraufhin sich diese aktualisiert.

6.2. Optimierung der Performance

Obwohl die Performance des Frameworks in den meisten Fällen bereits sehr gut ist, ließe sie sich besonders im Bezug auf größere Datenmengen verbessern, indem man einen komplexeren `DataRetriever` entwickelt. Dabei gäbe es die Möglichkeit die gesammelten Daten vorheriger Anfragen abzuspeichern, sodass man für neue Anfragen welche die gleiche Teilmenge von Daten betrifft das Datenmodell nicht erneut durchsuchen muss, sondern die schon vorhandenen Daten zurückgibt. Die Erkennung ob die Daten bereits gesammelt wurden ließe sich damit realisieren, dass man abspeichert zu welchen Werten der jeweiligen Dimension bereits Daten vorhanden sind. Dies vergleicht man mit den Werten der Filter-Kriterien und holt sich anschließend lediglich das Delta aus dem Datenmodell. Eine weitere Möglichkeit wäre es ständig alle Daten vorzuhalten. Dabei würde man das Datenmodell beobachten (zum Beispiel mit einem *Beobachter*-Pattern [GAM94, Seite 293]) und auf Änderungen reagieren. Hierbei würde es sich anbieten die Verbindung zwischen getrackten Rennen und den Spalten eines Leaderboards zu beobachten. Wird ein Rennen neu an eine Spalte gebunden, werden die Daten dieses Rennens gesammelt und mit einem Kontext ausgestattet. Bei der Entfernung eines Rennens aus der Leaderboard-Spalte werden die entsprechenden Daten entfernt. Diese Vorgehensweise ist vor allem bei langlebigen Server-Instanzen von Vorteil, da Änderungen am Datenfeld nach dessen Instanziierung relativ selten sind.

6.3. Verbindung zwischen den Sailing Analytics und dem BOBJ-Explorer

Eine weitere zukünftige Erweiterung ist die Verbindung zwischen dem BOBJ-Explorer und den Sailing Analytics nach dem in Abschnitt 3.1 beschriebenen Prinzip. Wie man das Gerüst für den speziellen JDBC-Treiber entwickelt, ist in [NANDA02] beschrieben. Des weiteren müssten vor allem die `Statement`- und `ResultSet`-Implementierungen des Treibers angepasst werden. Bei dem `Statement` läge das Hauptaugenmerk auf der `executeQuery`-Methode, durch diese Daten aus der Datenbank geladen werden. Die übrigen `Query`-Methoden können vernachlässigt werden, da zum Beispiel die Datenbank ändernde Befehle (wie `executeUpdate`) nicht zulässig sein sollten. In der `executeQuery`-Methode würde der gegebene SQL-String analysiert¹ und eine entsprechende Objektstruktur des Datenanalyse-Frameworks erzeugt werden. Dabei würde der `Extractor` für jeden Dateneintrag ein Objekt erzeugen, welches eine Zeile in der Ergebnis-Tabelle repräsentiert. Der `Aggregator` würde diese Objekte anschließend zu einem `ResultSet` aggregieren. Dieses würde an den Aufrufer der `executeQuery`-Methode zurückgegeben werden und von der BOBJ-Infrastruktur weiterverarbeitet werden.

6.4. Erweiterung der Benutzeroberfläche

Da das Arbeiten mit dem BOBJ-Explorer gewissen Einschränkungen unterliegt, würde es Sinn machen parallel zu der BOBJ-Integration eine eigene Datenanalyse-Oberfläche zu entwickeln, in der die erweiterten Analysemöglichkeiten einfach anzuwenden sind. Neben der allgemeinen Anordnung der Elemente zu Definition der Anfrage betrifft dies, insbesondere die Definition eigener Analyse-Komponenten mit Groovy-Skripten, da der normale Anwender hierfür viel Hilfestellung benötigt. Eine schnell und einfach zu implementierende Möglichkeit wäre eine Art Hilfefenster, in dem die grundlegenden Elemente (z.B. Deklaration von Variablen, Umgang mit primitiven Datentypen oder Kontrollstrukturen) von Groovy erläutert werden und beschrieben ist, welche domänenspezifischen Elemente zur Verfügung stehen. Dies bedeutet insbesondere den zu untersuchenden Datentyp, dessen verfügbare Methoden und was diese zurückgeben. Außerdem sollten weitere, bezüglich des Datenelements relevante Elemente aufgeführt

¹ Zum Beispiel mit dem `JSqlParser` <http://jsqlparser.sourceforge.net/>

werden. Im Falle von GPS-Daten wäre das zum Beispiel der `LegTypeWrapper`. Neben dieser Hilfeseite wäre es möglich die Eingabefelder für Skripte benutzerfreundlicher zu gestalten, indem man eine Hervorhebung und Überprüfung der Syntax oder eine automatische Vervollständigung von Befehlen integriert. Diese Erweiterungen wären allerdings mit erheblich mehr Aufwand verbunden als eine Hilfeseite. Des weiteren sollte die Rückmeldung, falls das Skript des Anwenders einen Fehler enthält, verständlich sein und helfen diesen zu korrigieren.

Neben den Erweiterungen der Oberfläche zur einfacheren Benutzung von Skripten, sollte zudem die Präsentation der Ergebnisse überarbeitet werden, da diese bei der Gruppierung nach vielen Dimensionen nahezu unkenntlich werden kann. Dies liegt daran, wie Highcharts die einzelnen Punkte darstellt, wobei tiefgreifende Änderungen notwendig wären dieses Verhalten zu ändern. Außerdem ist für die oben beschriebenen positionsbezogenen Analysen eine anderer Form der Präsentation notwendig, da sich deren Ergebnisse schlecht über ein Balkendiagramm visualisieren lassen. Stattdessen könnte eine Google-Maps verwendet werden, in die die Gebiete eingezeichnet und farblich gefüllt werden. Mit der Deckkraft der Füllung ließe sich die relative Größe des Ergebnisses darstellen.

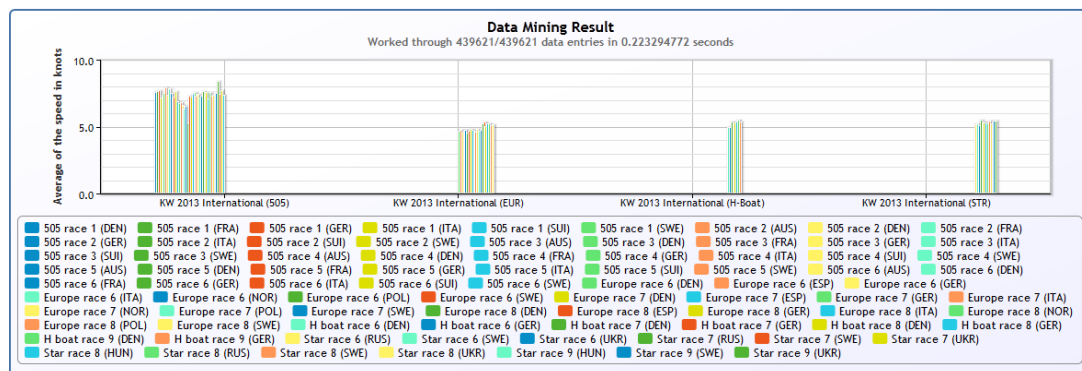


Abbildung 6.2.: Beispiel einer unkenntlichen Ergebnis-Präsentation. Die Daten wurden hierbei anhand des Reggattanamens, dem Rennnamen und der Nationalität der Teilnehmer gruppiert.

6.5. Weitere dynamische Komponenten

Zur Maximierung der Flexibilität wäre es sinnvoll für jede Komponente des Frameworks eine dynamische Implementierung zu entwickeln. So wäre es möglich eine dynamische Filterung der Daten mit einem `DynamicFilterCriteria` zu realisieren. Hierbei müsste der Anwender mit seinem Skript definieren welche Dateneinträge dem Kriterium entsprechen. Dies kommt der `matches`-Methode gleich. Der `DynamicGroupier` müsste so erweitert werden, dass es möglich ist die Daten mit einem `CompoundGroupKey` zu gruppieren. Um das zu realisieren könnte man dem Anwender ermöglichen mit dem Skript eine Menge von Objekten zurückzugeben, welche durch den `DynamicGroupier` in einen Schlüssel umgewandelt werden. Hierfür müsste sich der Anwender mit der Deklaration von `Collections` in Groovy auseinandersetzen. Eine weitere Möglichkeit wäre, dass der Anwender mehrere Skripte definiert, die jeweils ein Objekt zurückgeben. Dabei muss die Oberfläche verdeutlichen, welches Skript für welche Hierarchiestufe des Schlüssels verantwortlich ist. Welche der beiden Möglichkeiten besser ist, ist eine Frage der Performance (da das Ausführen mehrere Skripte länger dauert) und der Verständlichkeit und Benutzerfreundlichkeit. Für eine Entscheidung wäre es essentiell, sich mit einigen potentiellen Anwendern zu unterhalten, um deren Meinung zu erfahren.

Die Bereitstellung eines dynamischen `Extractors` und `Aggregators` ist am schwierigsten, da es einige Dinge zu beachten gilt. Zum einen kommen, im Gegensatz zu den anderen Komponenten, bei diesen mehrere generische Typ-Parameter zum Einsatz, welche zudem untereinander kompatibel sein müssen. So muss der `ExtractedType` des `Extractors` mit dem des `Aggregators` übereinstimmen, da sonst ein Fehler bei der Erzeugung der `Query`-Instanz auftritt. Zum anderen ist die Bildung des Aggregats komplexer, als zum Beispiel die Entscheidung ob ein Dateneintrag ein Kriterium erfüllt. So ist nicht jeder Anwender in der Lage korrekt über eine Menge von Objekten zu iterieren und auf die einzelnen Elemente eine Operation zur Bildung des Aggregats auszuführen. Deswegen könnte man dynamische Aggregatoren unterschiedlicher Komplexität anbieten. So wäre es möglich nach der Extraktion primitiver Datentypen die Standard-Aggregatoren zu verwenden. Hierbei müsste der Anwender kein eigenes Skript zur Aggregation definieren. Wird ein nicht primitiver Datentyp extrahiert könnte man dem Anwender die Wahl zwischen einem vereinfachten und einem komplexen Aggregator geben. Die simple Variante würde zwei Skripte erwarten. Das eine Skript soll aus einem extrahierten Objekt den entsprechenden Wert für die Aggregation zurückgeben und das andere definiert die Operation, welche auf je zwei zu aggregierende

Werte angewendet werden soll. Entspricht der Typ der extrahierten Objekte dem Typ der zu aggregierenden Werte, wäre nur das zweite Skript nötig. Damit wäre der simple dynamische Aggregator ähnlich dem SumAggregator (s. Abbildung 5.6). Die komplexe Variante würde als Skript eine Implementierung der aggregate-Methode erwarten. Dafür wäre mehr Fachwissen nötig, allerdings erhält der Anwender dadurch die Möglichkeit einen beliebigen Wert aus der Menge der extrahierten Objekte zu berechnen.

6.5.1. Absicherung der dynamischen Komponenten

Ein anderer Aspekt bezüglich der dynamischen Ausführung der Skripte ist die Sicherheit. Denn das Annehmen und Ausführen fremden Codes ist an und für sich eine Sicherheitslücke, durch die ein Angreifer viel Schaden anrichten kann. So konnte ich mit folgenden Skript meinen Rechner zum Herunterfahren zwingen, als ich den dynamischen Grouper lokal getestet habe.

```
1 "shutdown -s".execute();  
2 return "Diese Zeile wird nicht erreicht.";
```

Code 6.1: Skript zum Herunterfahren des Rechners

Dabei habe ich als Betriebssystem Windows 7 verwendet und war mit Administratorrechten angemeldet. Dadurch, dass das Skript als Grouper ausgeführt wird, wird es für jeden Dateneintrag angewendet, wodurch sich das Herunterfahren nicht durch eingeben des Befehls `shutdown -a` in der Kommandozeile abbrechen lässt, da die nächste Ausführung des Skriptes das Herunterfahren erneut erzwingt. Dieser Angriff ist nur unter den oben genannten Voraussetzungen möglich, allerdings gibt es immer die Möglichkeit mit dem nachfolgenden Skript das aktuell laufende Programm (in diesem Fall die Server-Instanz der Sailing Analytics) zu beenden. Das hätte zur Folge, dass das Tracking aller Rennen neu gestartet werden müsste, wobei es durchaus 15 Minuten dauern kann, bis der Server wieder auf dem vorherigen Stand ist.

```
1 System.exit(0);  
2 return "Diese Zeile wird nicht erreicht.";
```

Code 6.2: Skript zum Beenden des aktuellen Programms

Solche Angriffe gilt es zu unterbinden. Dabei sollte man allerdings die Verwendung bestimmter Klassen und Methoden nicht verbietet, sondern explizit erlauben. Die Definition dieser weißen Liste könnte ähnlich wie das erstellen des Basis-Bindings in

einer Klasse pro atomaren Datentyps definiert werden und von den einzelnen dynamischen Komponenten spezialisiert werden. Realisieren ließe sich so etwas zum Beispiel mit dem Open-Source-Projekt *Groovy Sandbox*¹, welches es einem ermöglicht Skripte in einem isolierten Bereich auszuführen, in dem nur bestimmte Klassen verwendet werden dürfen. Andernfalls führt dies zu einem Fehler bei der Ausführung. Eine andere Möglichkeit wäre die Verwendung eines eigenen `Java SecurityManagers` und eigenen `Permissions`, wie es in [SMIT12] beschrieben wird. Beide Möglichkeiten hätten negative Auswirkungen auf die Performance der Datenanalyse, sind allerdings unbedingt notwendig, damit Skripte überhaupt verwendet werden können.

¹ Homepage: <http://groovy-sandbox.kohsuke.org/>

7. Fazit

Mit dem in dieser Arbeit vorgestellten Framework, wurde die Grundlage für komplexe Datenanalysen der SAP Sailing Analytics geschaffen, wobei die Daten in Form eines Objektfeldes vorliegen, welches nur zur Laufzeit vorhanden ist. Dadurch, dass die einzelnen Komponenten generisch sind, lassen sich beliebige Daten extrahieren und verarbeiten. Dabei kann das Framework nicht nur für die Sailing Analytics, sondern auch in beliebigen anderen Programmen verwendet werden, solange diese mit Java kompatibel sind. Aufgrund des modularen Aufbaus lassen sich die Komponenten in ähnlichen Anwendungsfällen wiederverwenden und neue Ideen oder Erweiterungen lassen sich mit wenig Aufwand umsetzen. So wäre es mit etwas zusätzlichen Arbeitsaufwand für einen speziellen JDBC-Treiber möglich, eine Verbindung zu dem SAP Standardanalysetool *BusinessObjects Explorer* herzustellen und dessen ausgereiftes Bedienkonzept zur Datenanalyse zu verwenden. Durch die Entwicklung einer eigenen Benutzeroberfläche in der Sailing Analytics lassen sich die Grenzen des BOBJ-Explorers überwinden, um eine noch flexiblere Datenanalyse zu ermöglichen. Zum Beispiel kann der Anwender so das Verhalten der Framework-Komponenten selbst definieren und damit exploratorisch mit den vorhandenen Daten arbeiten. Dabei ist die Performance des Frameworks auch bei über einer Millionen Dateneinträgen in einem Großteil der Fälle so gut, dass Analysen in unter einer Sekunde ausgeführt werden, obwohl die Daten nicht als optimierte Datenbanktabellen vorliegen.

Abbildungsverzeichnis

1.1. Schematische Struktur einer Regatta	3
1.2. Beispiele möglicher Rennkurse	7
1.3. Rümpfe unterschiedlicher Bootsklassen	9
1.4. Segelrisse internationaler Bootsklassen	10
1.5. Segelrisse olympischer Bootsklassen	11
1.6. Ansicht einer Leaderboard-Gruppe	14
1.7. Ausschnitt eines kompakten Leaderboards	15
1.8. Ausschnitt eines Leaderboards mit aufgeklappten Spalten	15
1.9. Chart für die windwärtige Distanz zum Führenden	16
1.10. Wind-Chart	17
1.11. Platzierungs-Chart	17
1.12. Ausschnitt der Karte	18
1.13. Raceboard	20
1.14. Datenmodell von Regatten	23
1.15. Datenmodell von Leaderboard-Gruppen	25
1.16. Erstellung eines neuen <i>Information Spaces</i>	27
1.17. Ausschnitt der Explorationsansicht des BOBJ-Explorers	28
3.1. Verbindung zwischen den Sailing Analytics und dem BOBJ-Explorer	31
3.2. Aktivitätsdiagramm zum Ablauf der Datenanalyse	33
3.3. Aktivitätsdiagramm zum Ablauf der Datenanalyse mit Kontext	34
4.1. Arbeitsweise bei der Implementierung	36
5.1. Komponenten des Datenanalyse-Frameworks	39
5.2. Implementierung der <i>Filter</i>	40
5.3. Implementierung von <i>GroupKey</i>	42
5.4. Implementierung der <i>Grouper</i>	43
5.5. Implementierung der Extraktoren	44

5.6. Implementierung der Aggregatoren	46
5.7. Implementierung der GPS-spezifischen Framework-Komponenten . . .	47
5.8. Selektionsbereich der Datenanalyse-Oberfläche	48
5.9. Objektdiagramm einer Beispiel-Query	49
5.10. Darstellung der Ergebnisse einer Datenanalyse	49
5.11. Oberfläche zur eigenen Gruppierung	51
5.12. Durchschnittliche Geschwindigkeit in Knoten gruppiert nach Länge des Teilnehmer-Namens	51
5.13. Benchmark-Ergebnisse bei 200 Anfragen mit jeweils ca. 122.000 Daten- einträgen	53
6.1. Schematische Darstellung eines QuadTrees	58
6.2. Beispiel einer unkenntlichen Ergebnis-Präsentation	61
A.1. Vordefinierte Dimensionen der GPS-Daten	vii
A.2. Benchmark-Ergebnis für Anfragen ohne rechenintensive Datenelemente	viii
A.3. Benchmark-Ergebnis für Anfragen über den Schenkel-Typ ohne Opti- mierung	viii
A.4. Benchmark-Ergebnis für Anfragen über den Schenkel-Typ mit Optimie- rung	ix
A.5. Benchmark-Ergebnis für Anfragen mit einem geskripteten Grouper ohne Optimierung	ix
A.6. Benchmark-Ergebnis für Anfragen mit einem geskripteten Grouper mit Optimierung	ix

Codeverzeichnis

5.1. run-Methode der Query-Implementierung	38
5.2. Implementierung der group-Methode von AbstractGrouper	41
5.3. Algorithmus zu Bildung des Summen-Aggregats	45
5.4. Algorithmus zu Bildung des Durchschnitts-Aggregats	46
5.5. Implementierung der langsamen Groovy-Skript Ausführung	55
5.6. Implementierung der performanten Groovy-Skript Ausführung	56
6.1. Skript zum Herunterfahren des Rechners	63
6.2. Skript zum Beenden des aktuellen Programms	63
A.1. Implementierung des DynamicGrouper	v
A.2. Implementierung des GPSFixBaseBindingProviders	vi
A.3. getLegType-Methode der Klasse GPSFixContextImpl	vi

Literaturverzeichnis

- [GAM94] ERICH GAMMA U.A.: **Design Patterns – Elements of Reusable Object-Oriented Software**. 1994, Addison-Wesley Professional, Auflage 39
- [NANDA02] NITIN NANDA UND SUNIL KUMAR: **Create your own type 3 JDBC driver**. 17.05.2002, JavaWorld.com, <http://www.javaworld.com/javaworld/jw-05-2002/jw-0517-jdbcdriver.html> abgerufen am 18.08.2013
- [SAAKE03] GUNTER SAAKE UND KAI-UWE SATTLER: **Datenbanken & Java. JDBC, SQLJ, ODMG und JDO**. 2003, Dpunkt, Auflage 2
- [DENK04] ROLAND DENK: **Handbuch Segeln**. 2004, Delius Klasing
- [PHIL05] MALTE PHILIPP: **Regattasegeln : Strategie und Taktik**. 2005, Delius Klasing
- [ISAF12] ISAF: **The Racing Rules Of Sailing for 2013-2016**. 2012, [http://www.sailing.org/tools/documents/ISAFRRS20132016Final-\[13376\].pdf](http://www.sailing.org/tools/documents/ISAFRRS20132016Final-[13376].pdf) abgerufen am 16.08.2013
- [SMIT12] RENÉ SMIT: **Groovy DSL: Executing scripts in a sandbox**. 18.12.2012, SDI's Code Comments, <http://sdicc.blogspot.de/2012/12/groovy-dsl-executing-scripts-in-sandbox.html> abgerufen am 19.08.2013
- [ISAF13] ISAF: **ISAF Class Entry Guidelines**. 2013, [http://www.sailing.org/tools/documents/ISAFClassEntryGuidelines2012-\[12838\].pdf](http://www.sailing.org/tools/documents/ISAFClassEntryGuidelines2012-[12838].pdf) abgerufen am 16.08.2013
- [STATISTA13] STATISTA: **Quantil**. <http://de.statista.com/statistik/lexikon/definition/106/quantil/> abgerufen am 22.08.2013

A. Appendix

A.1. Implementierung der Komponenten

```
1 public class DynamicGrouper<DataType> extends AbstractGrouper<DataType> {
2
3     private Script script;
4     private Binding binding;
5
6     public DynamicGrouper(String scriptText, BaseBindingProvider<DataType>
7         baseBindingProvider) {
8         super();
9         script = new GroovyShell().parse(scriptText);
10        binding = baseBindingProvider.createBaseBinding();
11    }
12    @Override
13    protected GroupKey getGroupKeyFor(DataType dataEntry) {
14        binding.setVariable("data", dataEntry);
15        script.setBinding(binding);
16        return new GenericGroupKey<Object>(script.run());
17    }
18 }
```

Code A.1: Implementierung des DynamicGrouper

```

1 @Override
2 public Binding createBaseBinding() {
3     Binding binding = new Binding();
4     binding.setProperty("LegType", new LegTypeWrapper());
5     return binding;
6 }
7
8 @SuppressWarnings("unused")
9 private static class LegTypeWrapper {
10     public static LegType UPWIND = LegType.UPWIND;
11     public static LegType DOWNWIND = LegType.DOWNWIND;
12     public static LegType REACHING = LegType.REACHING;
13 }

```

Code A.2: Implementierung des GPSFixBaseBindingProviders

```

1 public LegType getLegType() {
2     if (!legTypeHasBeenInitialized) {
3         initializeLegType();
4     }
5     return legType;
6 }

```

Code A.3: getLegType-Methode der Klasse GPSFixContextImpl

A.2. Dimensionen der GPS-Daten

Dimenisons
GPSFix
+RegattaName : Dimension<GPSFixWithContext, String>
+RaceName : Dimension<GPSFixWithContext, String>
+LegNumber : Dimension<GPSFixWithContext, Integer>
+CourseAreaName : Dimension<GPSFixWithContext, String>
+FleetName : Dimension<GPSFixWithContext, String>
+BoatClassName : Dimension<GPSFixWithContext, String>
+Year : Dimension<GPSFixWithContext, Integer>
+LegType : Dimension<GPSFixWithContext, LegType>
+CompetitorName : Dimension<GPSFixWithContext, String>
+SailID : Dimension<GPSFixWithContext, String>
+Nationality : Dimension<GPSFixWithContext, String>
+getDimensionFor(dimenison : SharedDimensions GPSFix) : Dimension<GPSFixWithContext, ValueType>

Abbildung A.1.: Vordefinierte Dimensionen der GPS-Daten

A.3. Ergebnisse der Datenanalyse-Benchmarks

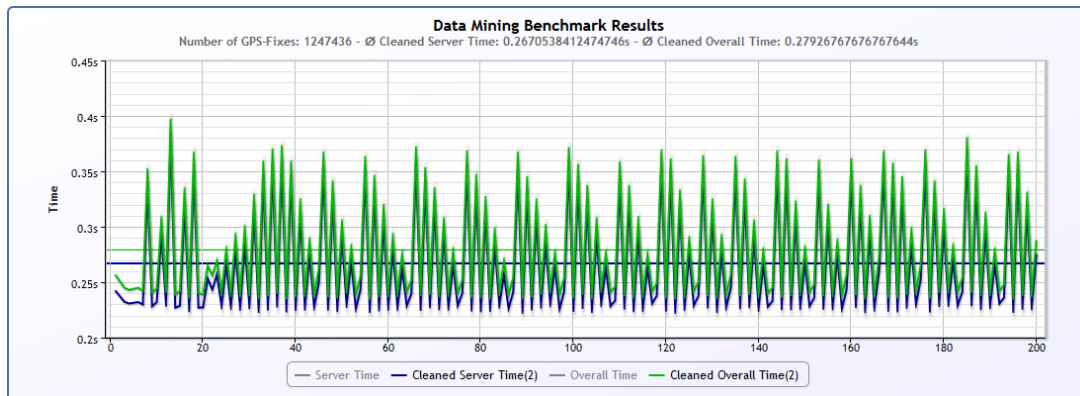


Abbildung A.2.: Benchmark-Ergebnis bei 200 Anfragen mit jeweils ca. 1,25 Millionen Dateneinträgen. Die Daten wurden anhand des Regattenamens durch eine GroupByDimension-Instanz gruppiert. Der Benchmark wurde mit dem PC durchgeführt.

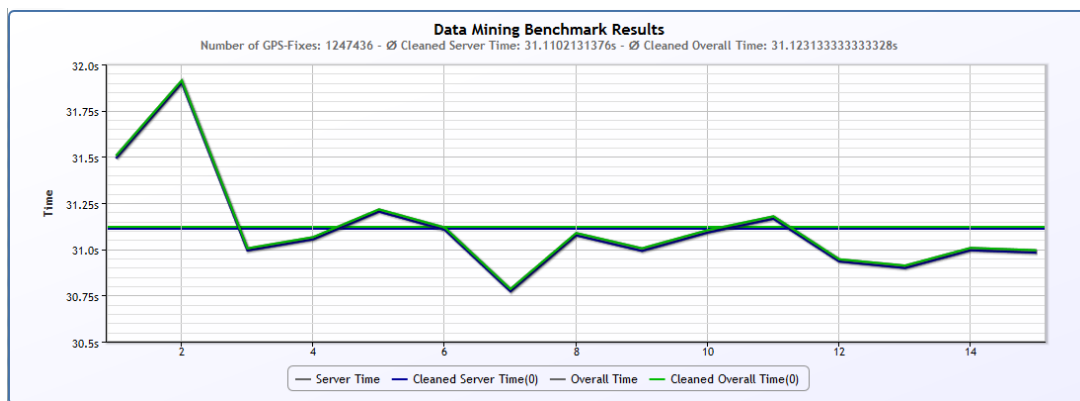


Abbildung A.3.: Benchmark-Ergebnis bei 15 Anfragen mit jeweils ca. 1,25 Millionen Dateneinträgen. Die Daten wurden anhand des Schenkel-Typs durch eine GroupByDimension-Instanz gruppiert. Der Benchmark wurde mit dem PC durchgeführt.

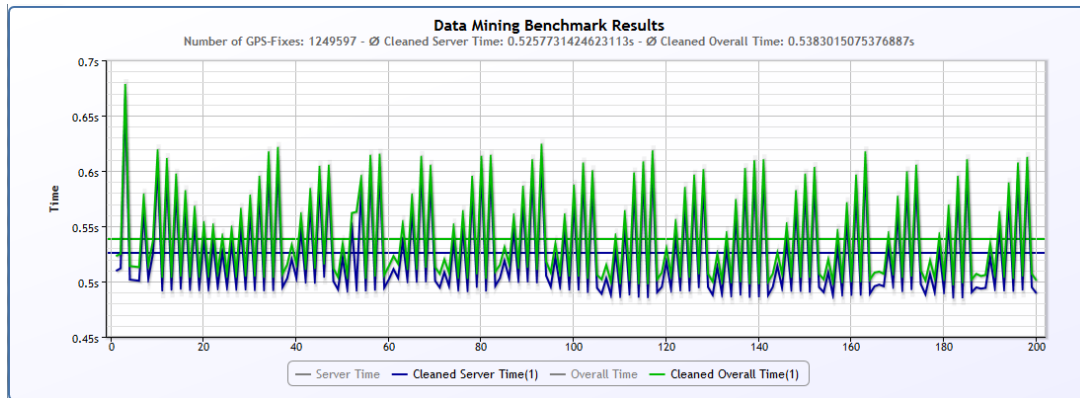


Abbildung A.4.: Benchmark-Ergebnis bei 200 Anfragen mit jeweils ca. 1,25 Millionen Dateneinträgen. Die Daten wurden anhand des Schenkel-Typs durch eine GroupByDimension-Instanz gruppiert. Der Benchmark wurde mit dem PC durchgeführt.

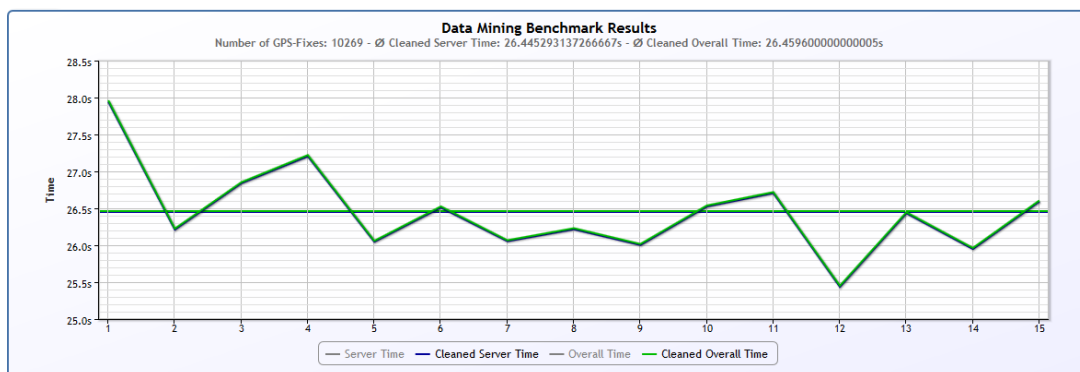


Abbildung A.5.: Benchmark-Ergebnis bei 15 Anfragen mit jeweils ca. 10.000 Dateneinträgen. Die Daten wurden anhand des Regattenamens durch ein Skript gruppiert. Der Benchmark wurde mit dem PC durchgeführt.

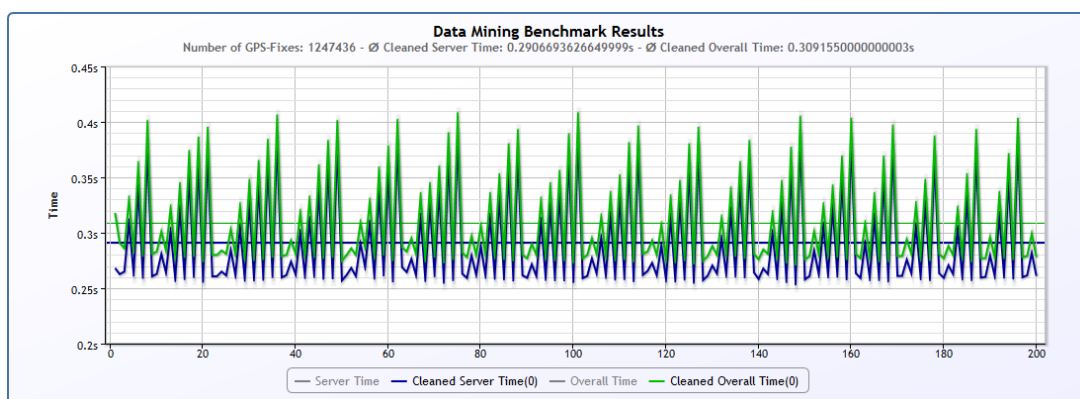


Abbildung A.6.: Benchmark-Ergebnis bei 200 Anfragen mit jeweils ca. 1,25 Millionen Dateneinträgen. Die Daten wurden anhand des Regattenamens durch ein Skript gruppiert. Der Benchmark wurde mit dem PC durchgeführt.