Verteiltes Genom Browsing

Projektspezifikation

1. Dezember 2015

Inhaltsverzeichnis

1 Integration **2**
- 1.1 Integrationsprozess . . . 2
  - 1.1.1 Ablauf der Integration . . . 2
  - 1.1.2 Quellenauswahl . . . 2
  - 1.1.3 Attributauswahl und -mapping . . . 2
  - 1.1.4 Mengengerüst . . . 2
  - 1.1.5 Inputfile-Format . . . 3
  - 1.1.6 Sequenzdiagramm . . . 3
- 1.2 Datenbankentwurf . . . 4
- 1.3 Entwurf des Parsers . . . 4
- 1.4 Klassendigramm . . . 5
- 1.5 Schnittstellenspezifikation . . . 5
  - 1.5.1 Schnittstelle: Integration - Middleware . . . 5
  - 1.5.2 Schnittstelle: Integration - Benutzer . . . 5
- 1.6 Tests . . . 5
  - 1.6.1 Unit-Tests . . . 5

2 Middleware **7**
- 2.1 Indexstruktur . . . 7
  - 2.1.1 Anforderungen . . . 7
  - 2.1.2 Funktionen und Datenstrukturen . . . 7
- 2.2 IndexController . . . 11
  - 2.2.1 Anforderungen . . . 11
  - 2.2.2 Funktionen . . . 11
- 2.3 Klassen-Diagramm . . . 12
- 2.4 Sequenzdiagramme . . . 13
  - 2.4.1 Intervall-Suche . . . 13
  - 2.4.2 Names-Suche . . . 13
  - 2.4.3 Suche nach Gennamen . . . 14
- 2.5 Schnittstellenspezifikation: Middleware - Frontend . . . 15
  - 2.5.1 Use-Cases . . . 15
- 2.6 Kommunikation . . . 17
- 2.7 Unit-Tests . . . 19
  - 2.7.1 QueryReceiver . . . 19
  - 2.7.2 GeneTranslator . . . 22
  - 2.7.3 IndexController . . . 23
  - 2.7.4 Intervallbaum . . . 24
- 2.8 Stresstests . . . 25

3 Frontend **26**
- 3.1 Mock-Ups der Benutzerschnittstelle . . . 26
- 3.2 Klassen-Diagramm . . . 27

3.3 Sequenzdiagramm 28
3.4 Use Cases 29
3.5 Unit-Tests 30
3.5.1 Suchfunktion 30
3.5.2 Quellen-Button 30
3.5.3 Quellen-Scroller 30
3.5.4 Zoom-slider 30
3.5.5 Chromosom-Auswahl 30
3.5.6 Allgemein 31

1 Integration

1.1 Integrationsprozess

1.1.1 Ablauf der Integration

Für die Integration von weiteren Quellen in die eigene Datenbank werden einige weitere Softwarekomponenten benötigt, um diese einzubinden, dazu zählen eine Downloadmöglichkeit (Skript, o.ä.) und ein lokaler, auf die Quelle zugeschnittener, Parser. Diese müssen dann in den Programmcode an die entsprechende Stelle eingebunden werden. Der lokale Parser wird die heruntergeladenen Dateien entpacken, entschlüsseln und in einem einheitlichen Format abspeichern. Die weiteren Schritte danach wird der feste, globale Parser übernehmen, den man nicht modifizieren muss. Er wird aus den Endprodukten der lokalen Parser die Datensätze rauslesen und diese dann in der Datenbank abspeichern.

1.1.2 Quellenauswahl

Unsere derzeitig benutzten Quellen werden die dbSNP und das 1000GenomeProject sein. Die HGMD werden wir nicht benutzen können, da diese Quelle ein Entgelt zur Benutzung der Daten verlangt, was unser Budget übersteigt. Die TCGA Datenbank werden wir vorerst nicht nicht beachten, da uns derzeit noch nicht klar ist, ob deren Daten verwendbar sind. Zu einem späteren Zeitpunkt, wenn die Integration von dbSNP und 1000GenomeProject abgeschlossen ist, werden wir uns mit TCGA befassen und, sollten die Daten erreichbar und verwendbar sein, werden wir diese nehmen um den Erweiterbarkeitsprozess zu verdeutlichen.

1.1.3 Attributauswahl und -mapping

Unsere verwendeten Datenbanken stellen uns die Mutationsdaten in .vcf-Dateien bereit. Die Metadaten sind in seperaten .txt-Dateien abgespeichert. Für die Metadaten werden uns die Daten Gender und Population bereitgestellt. Für die Mutation sind die relevanten Daten das Chromosom, in dem die Mutation auftaucht, die Position der Mutation im jeweiligen Chromosom und die vollständige Mutationssequenz. Mit diesen Daten werden wir auch im Weiteren arbeiten. Die Referenzgenome werden in einer separaten Datei abgespeichert, da diese sich nie verändern, und somit ein schnellerer Zugriff auf die Daten gewährleistet wird, und auch ein schnellerer Aufbau der Datenbank selber.

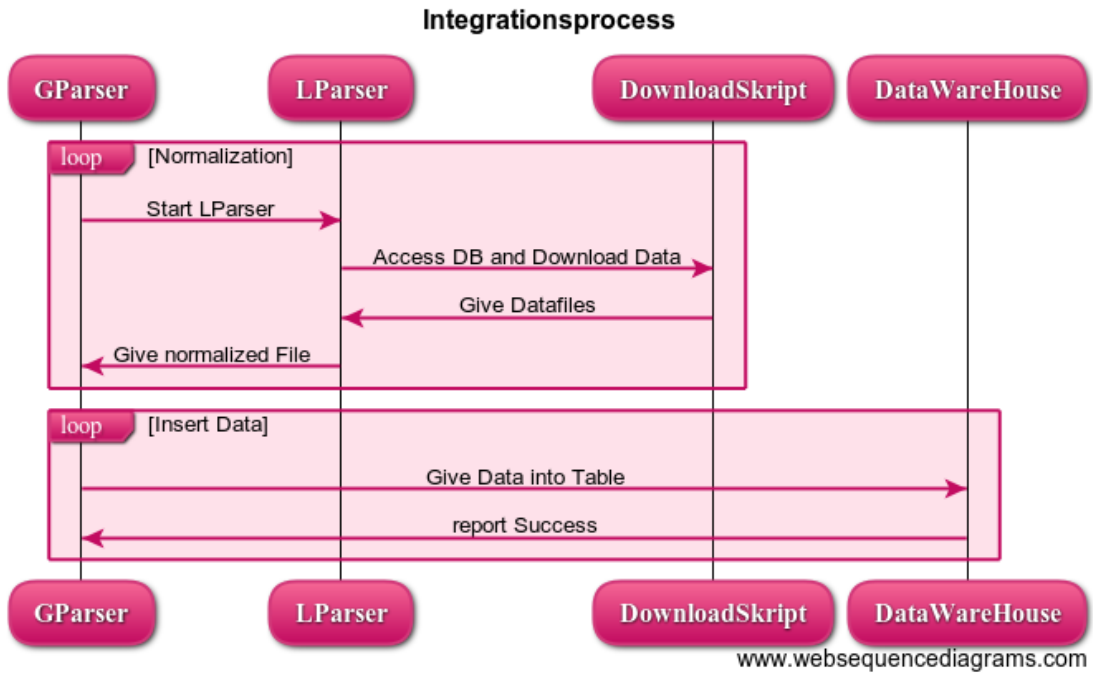
1.1.4 Mengengerüst

Der derzeitige Stand des benötigten Speichers beläuft sich auf einige hundert (aber weniger als 500) Gigabyte. Diese Angabe gilt nur für dbSNP und 1000GenomeProject, weitere Quellen werden dementsprechend mehr Speicher benötigen. Weitere Mengenangaben können wir derzeit noch nicht genau machen, diese werden im weiteren Verlauf konkreter und können dann gemacht werden.

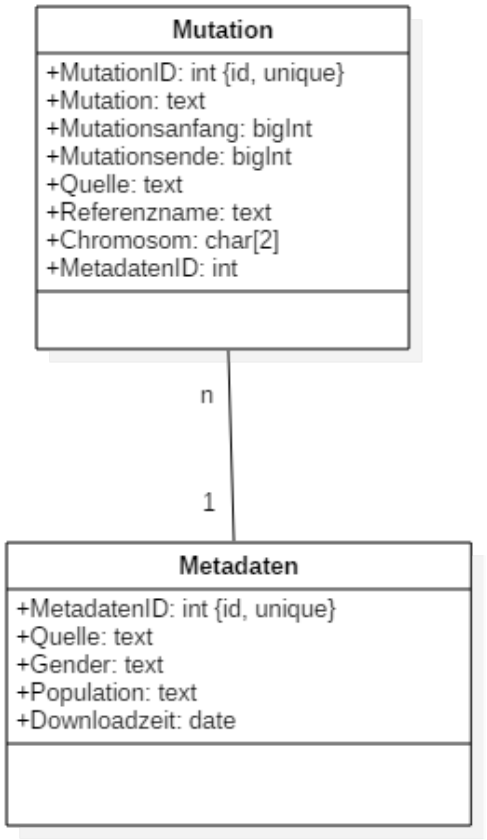
1.1.5 Inputfile-Format

Referenzgenomname:„Name des Referenzgenoms Bsp: GRCh38“
Quelle:„hier die Quelle angeben“
$$
SampleID:„hier Samplename“
Genkoordinaten:„Angabe der Koordinaten“
Mutationssequenz:„Sequenz“
$$
SampleID:„hier SampleID aus der Datenbank“
Gender:„m oder f“
Population:„drei Buchstaben bsp: GBR“
EOF

1.1.6 Sequenzdiagramm



1.2 Datenbankentwurf



1.3 Entwurf des Parsers

Der lokale Parser wird auf die jeweilige Quelle zugeschnitten sein. Er wird die vorher heruntergeladenen Dateien entpacken, entschlüsseln und danach die relevanten Daten aus den Dateien herauslesen und in einem einheitlichen Format für den globalen Parser abspeichern.
Der globale Parser wird die Einheitsdateien der lokalen Parser nehmen, die beinhaltenden Daten in einzelne Datensätze aufteilen und diese dann in der Datenbank abspeichern und sie somit der Middleware bereitstellen.

1.4 Klassendigramm

1.5 Schnittstellenspezifikation

1.5.1 Schnittstelle: Integration - Middleware

Die Schnittstelle zwischen der Integration und der Middleware ist die, im Modell sichtbare, Datenbank. Sie ist der Ort, an dem die Integration die Daten bereitstellt und von wo die Middleware sich die Daten für die Anfragen abholt.

1.5.2 Schnittstelle: Integration - Benutzer

Die Schnittstelle zwischen der Integration und dem Benutzer ist das Hinzufügen neuer Quellen. Der Nutzer wird angehalten sein, zu wissen, wie seine neue Quelle aufgebaut ist, da er selber ein Downloadskript o.ä. dafür schreiben muss, sowie einen lokalen Parser. Diese werden an den entsprechenden Stellen im Programmcode eingefügt. Die lokalen Parser werden durch ein Interface vereinheitlicht.

1.6 Tests

1.6.1 Unit-Tests

Konkrete Tests konnten wir bisher nicht durchführen, jedoch gibt es einige Dinge zu testen: Es muss getestet werden, ob der lokale Parser arbeitet wie gewünscht, also mindestens 2 Testläufe für ihn: Bei einem, für ihn korrekten Inputfile, muss er ein entsprechend richtiges Outputfile für den globalen Parser erstellen. Sollte er ein Inputfile parsen, was nicht für ihn gedacht ist, soll er das Inputfile verwerfen oder eine Fehlermeldung ausgeben, aber auf alle Fälle das Outputfile nicht mit diesem Input erweitern. Natürlich kann das Inputfile auf verschiedene Weisen korrupt sein, was dort mehrere Testfälle notwendig macht.
Der globale Parser muss auf ähnliche Weisen getestet werden, jedoch kann man bei ihm als Voraussetzung annehmen, dass die lokalen Parser korrekt arbeiten und auch ein korrektes Outputfile erstellt haben. Somit müsste nur überprüft werden, dass der globale Parser die Daten korrekt ausliest und korrekt in die Datenbank einfügt.

Korrektes Inputfile:

Referenzgenomname:GRCh38
Quelle:1000Genom
$$ SampelID:HG0094
Genkoordinaten:6:19:19
Mutationssequenz: AGTCTAGTA
$$ SampelID:HG0094
Gender:m
Population:GRB
Download:01:01:2001

Fehlerhaftes Inputfile

Referenzgenomname:KeinFehlerMöglich
Quelle:KeinFehlerMöglich
$
SapelID:KeinFehlerMöglich
Genkoordinaten:67:21:15
Mutationsequenz:ATCERROR
$$
SampelID:KeinFehlerMöglich?
Gender:h
Population:XXXX
Download:64:64:2045

2 Middleware

2.1 Indexstruktur

2.1.1 Anforderungen

Der Index soll eine effiziente Suche nach Mutationen für das Frontend ermöglichen. Während des Programmstarts wird der Index aus der vorhandenen Datenbank aufgebaut und steht dann so lange zur Verfügung, bis das Programm beendet wird. Im Index wird mit Intervallgrenzen gesucht und der Index gibt alle Intervalle zurück, die komplett innerhalb des angegebenen Intervalls liegen.
Der Index wird verteilt aufgebaut und liegt auf 4 virtuellen Maschinen verteilt. Die für den Index spezifizierten Funktionen sprechen immer einen Teilindex an. Der Aufruf der Funktionen wird über den IndexController erfolgen, der immer alle 4 Teilindizes ansprechen wird.

2.1.2 Funktionen und Datenstrukturen

Die Funktionen des Indexes variieren in ihrerem Ablauf je nach gewählter Indexstruktur. Momentan existieren 3 Varianten, die getestet werden. Es ist nicht ausgeschlossen, dass weitere Strukturen im Laufe der Entwicklung getestet werden. Da die Dauer des Indexaufbaus für den Endnutzer nicht relevant ist hängt die Auswahl der letztendlich genutzten Struktur lediglich von der Geschwindigkeit der Suchanfragen ab. Im folgenden werden die Such-und Einfüge-Operationen basierend auf den jeweiligen Indexstrukturen beschrieben

IntervallTree
In diesem Fall basiert die Indexstruktur auf einem Intervallbaum.
Hierfür wird die frei zugängliche Bibliothek İntervallST.java"der Universität Princeton genutzt. Beide Funktionen haben hier lediglich die Aufgabe als Interface zu den zugehörigen Bibliotheksfunktionen zu dienen: contains() zur Suche und put() zum Einfügen.
Das Suchergebnis wird nach möglichen Filtern gefiltert. Bei n Intervallen und einer Such-Ergebnisliste der Größe m ergibt sich ein Komplexität von O(log n + m)
Die Bibliothek muss noch angepasst werden, damit das Einfügen von gleichen Intervallen möglich ist ohne, dass das zuerst eingefügte Intervall gelöscht wird.

```
search()
found intervalls = contains(intervall);
forall the elements in found intervalls do
    if element corresponds to specified filters then
        add element to answer list;
    end
    return answer list;
end
```

```
addMutation()
if mutation is already in index then
    return ïndex already contains mutation";
end
put();
return „added mutation“;
```

Suchbaumbasierter Index
Da ein Großteil der Intervalle einstellig bzw sehr kurz sind bietet sich ein einfacher binärer Suchbaum als Datenstruktur an.
Dieser wird um einen Iterator erweitert, damit effizient eine Menge an Knoten ausgewählt werden kann.
Hierfür wird die Java-Klasse TreeMap verwendet, die eine Suche in logarithmischer Zeit ermöglicht. Die Ergebnismenge wird einmal durchlaufen, Mutationen deren Endpunkt außerhalb des gesuchten Intervalls liegen werden dabei entfernt und jeder Knoten wird nach den angegebenen Filtern gefiltert. Bei n Intervallen und einer Such-Ergebnismenge von m Intervallen ergibt sich eine Laufzeit von O(log n +m).
Auch in dieser Implementation dienen die Funktionen als Interface zu den jeweiligen Funktionen der genutzten Klasse: submap() zur Suche und put() zum Einfügen von Objekten).

```
search()
found intervalls = submap(intervall);
forall the elements in found intervalls do
    if element corresponds to specified filters then
        add element to answer list;
    end
    return answer list;
end
```

```
addMutation()
if mutation is already in index then
    return „index already contains mutation“;
end
put();
return „added mutation“";
```

Arraybasierter Index
Sollte die Anzahl der Mutationen groß genug sein, dass sie mit Integer-Variablen darstellbar ist, so bietet sich unter Umständen auch ein arraybasierter Index an.
Dieser speichert alle Mutationen aufsteigend sortiert nach ihrem Anfangspunkt. Wird nun nach einem Intervall gesucht, so iteriert er über das Array beginnend bei der Mutation, deren Startwert noch im gesuchten Intervall liegt. Dabei wird für jede Mutation überprüft, ob ihr Endwert noch im gesuchten Intervall liegt. Ist dem so wird sie zur Ergenismenge hinzugefügt. Hierbei können auch direkt die Filter überprüft werden.
Es so lange iteriert, bis der Startwert aller folgenden Mutationen größer, als der vom Nutzer angegebene Endwert ist.
Falls Mutationen an der gleichen Stelle beginnen, so verschieben sich alle folgenden Mutationen in der Liste, da in aufeinanderfolgenden Zellen gleiche Startintervalle gespeichert werden müssen. Es muss also ermittelt werden, wo sich die erste im Intervall liegende Mutation befindet. Ein Verfahren hierfür wird noch ermittelt
Bei m Mutationen, deren Startwert sich im gesuchten Intervall befinden, liegt die Laufzeit bei O(m + ϵ), wobei ϵ davon abhängt, wie die erste Mutation ermittelt wird. Es kann aber davon ausgegangen werden, dass ϵ einen geringen Anteil an der Laufzeit ausmachen wird.

```
search()
find index x of first mutation that lies in intervall;
while starting point of mutation at index x lies in search intervall do
    if endpoint of mutation at index x lies in search intervall and mutation
    corresponds to specified filters then
        add mutation to answer list;
    end
    x=x+1;
end
return answer list;
```

```
addMutation()
if mutation is already in index then
    return ïndex already contains mutation";
end
insert mutation at corresponding index and adjust array properly;
return „added mutation“;
```

2.2 IndexController

2.2.1 Anforderungen

Der IndexController nimmt Suchanfragen entgegen, leitet diese an die 4 Teilindizes weiter und fügt die Teilergebnisse wieder zusammen.
Falls eine Anfrage intern in mehrere Teilanfragen aufgeteilt werden sollte, weil z.B. ein Gen, nach dem gesucht wird an mehreren Stellen auftreten kann, leutet der IndexController alle Teilanfragen sequentiell an die Indizes weiter, fügt die Teilergebnisse zusammen und schickt die Ergebnismenge an den QueryReceiver zurück.

2.2.2 Funktionen

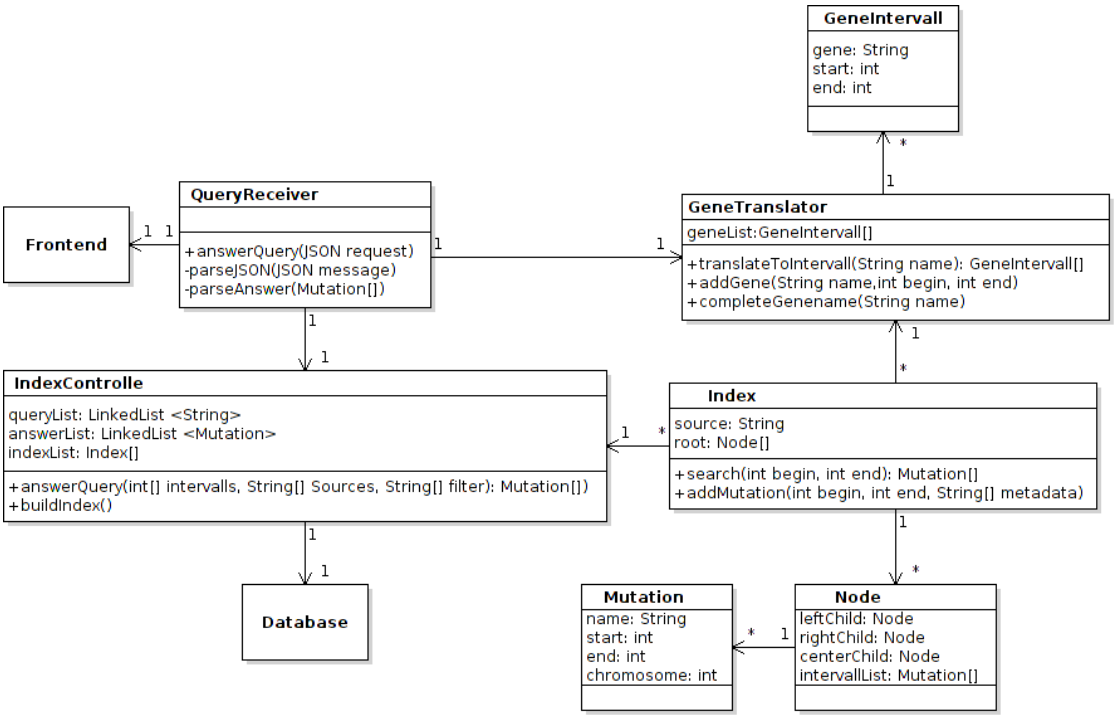
answerQuery(int[] intervals,String[] Sources,int[]filter) Die Funktion erhält mehrere Listen als Parameter, die die nötigen Informationen für die einzelnen Anfragen beinhalten. Jeweils 2 aufeinanderfolgende Einträge in der Intervall-Liste beschreiben den Start-und Endpunkt der gesuchten Intervalle. Die Einträge in den anderen Listen werden für alle Anfragen genutzt
Die Anfragen werden sequentiell an die 4 Teilindizes weitergeleitet und einzelnen Ergebnisse konkateniert und in einer Liste zurückgegeben.

```
forall the queries in parameter list do
    forall the sub indices do
        answer list = index.search();
    end
    concatenate all answer lists;
end
concatenate answer lists of each sub query;
return answer list;
```

buildIndex() Die Funktion wird bei Programmstart ausgeführt und baut auf Basis der Datenbank die 4 Teilinidzes auf.
Für jede Mutation wird zufällig entschieden in welchen Index sie eingefügt wird.

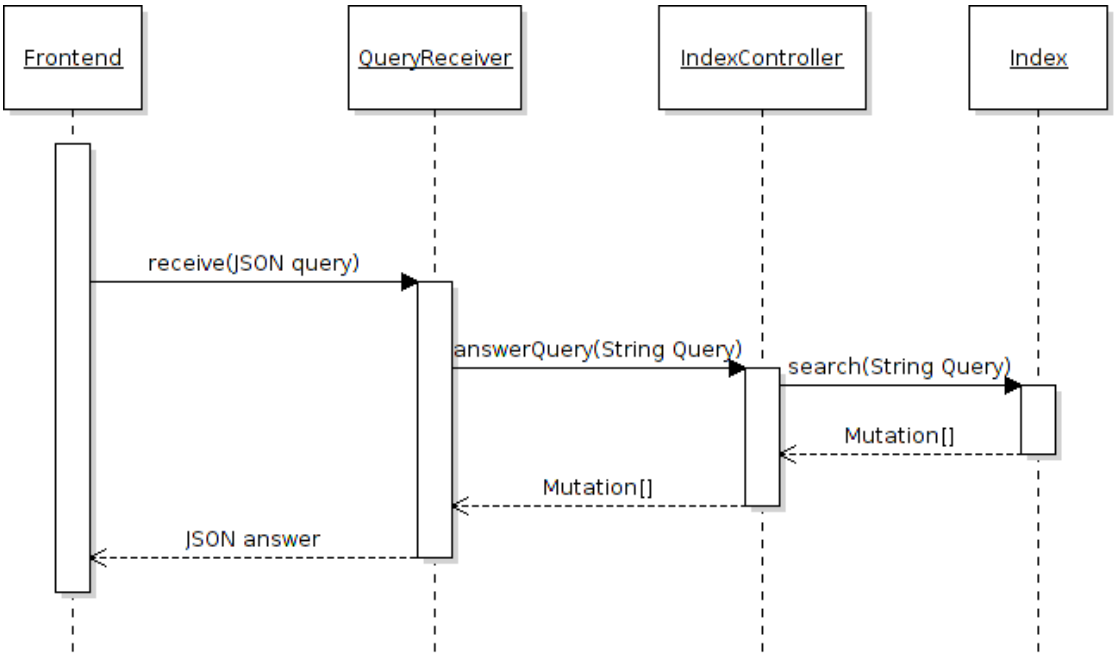
```
forall the elements in database do
    if element is already in index then
        skip element;
    end
    choose which sub index to insert into;
    index.addMutation();
end
return ïndex built";
```

2.3 Klassen-Diagramm

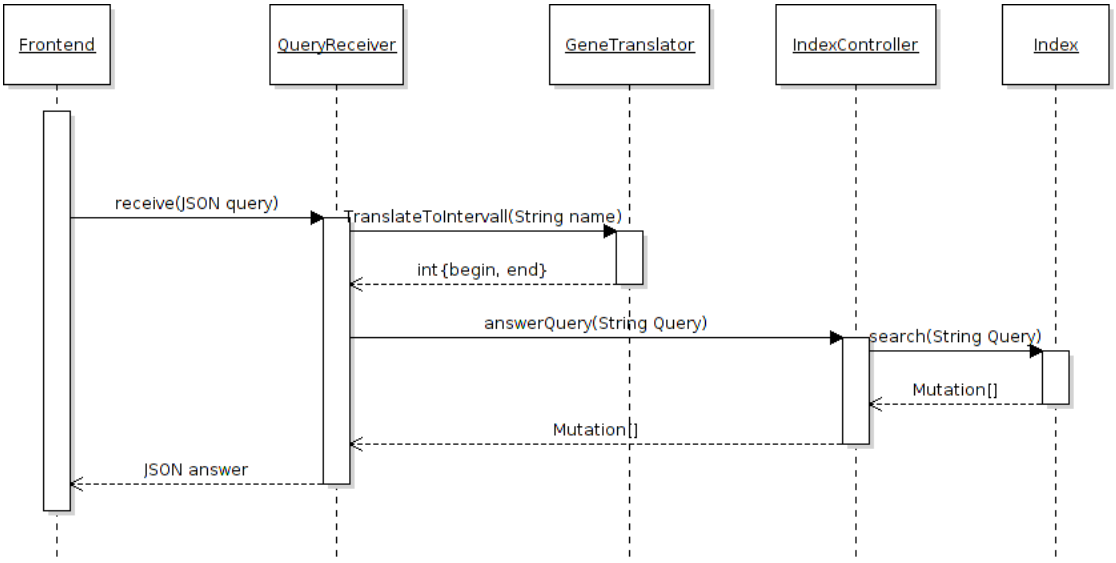


2.4 Sequenzdiagramme

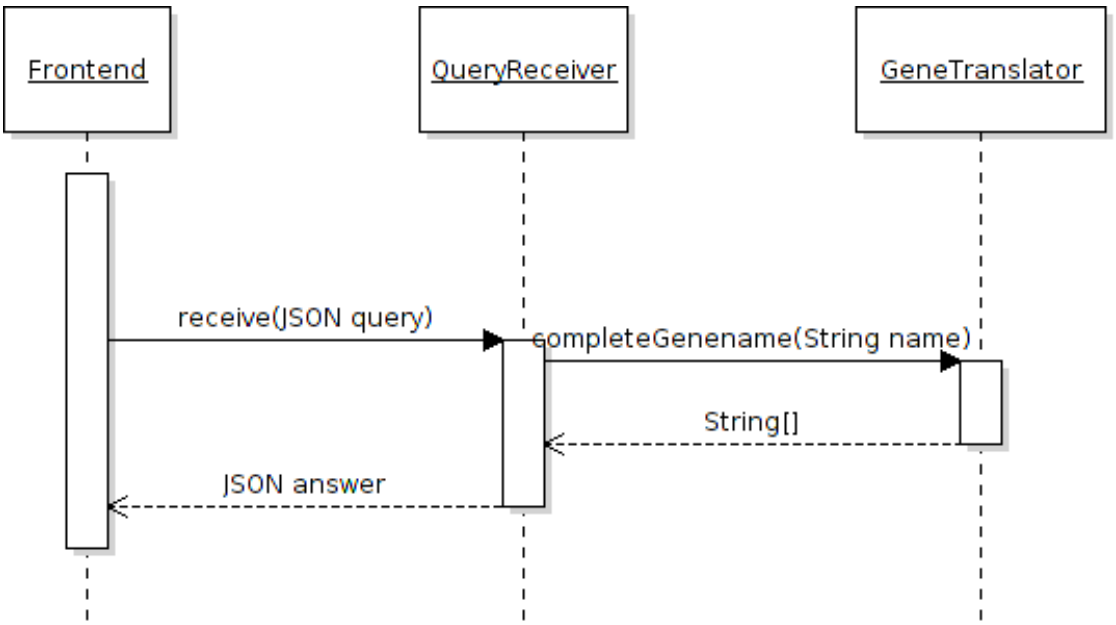
2.4.1 Intervall-Suche



2.4.2 Names-Suche



2.4.3 Suche nach Gennamen



2.5 Schnittstellenspezifikation: Middleware - Frontend

Zoomstufen (Intervallgrößen)

1. 200bp <– String
2. 1,000bp
3. 5,000bp
4. 10,000bp
5. 100,000bp
6. 1,000,000bp
7. 10,000,000bp

Subintervalle (Aufteilung der Intervallgrößen):

- 1,000/200 -> 5 Schritten
- 5,000/200 -> 25 Schritten
- 10,000/200 -> 50 Schritten
- 100,000/200 -> 500 Schritten
- 1,000,000/200 -> 5,000 Schritten
- 10,000,000/200 -> 50,000 Schritten

2.5.1 Use-Cases

Intervallsuche
I.
S:|270-330| = 60 < (1)
→ Zoomstufe (1)
von 270
bis [270+200]

II.
S:|220-530| = 310 > (1)
=> Zoomstufe (2)
von 220
bis [220+1,000]

III.
S:|23,578-57,654| = 36,000 > (4)
=> Zoomstufe (5)
von 23,578
bis [23,578+100,000]

Gensuche:
S:|FOXP2|
=> Namensuche an Backend mit der Antwort 'true' und dem Intervall von FOXP2
S:|13;2,700,000-3,800,000| = 1,100,000 > (6)
=> Zoomstufe (7)
von 2,700,000
bis [2,700,000+10,000,000]

Verschieben der Lanes
I.
Ausgangsstellung: P:[220+200] -> nach rechts um 100bp verschieben
=> Zoomstufe (1)
von [220+100]
bis [220+100+200]

II.
Ausgangsstellung: [220+1,000] -> nach rechts um 500bp verschieben
=> Zoomstufe (2)
von [220+500]
bis [220+500+1,000]

Zoomstufe verändern

(2) von 220
| bis [200+1,000]
v
(3) von 200 –> (1) von 220
| bis [220+5,000] bis [220+200]
v
(7) von 200
bis [220+10,000,000]

2.6 Kommunikation

→ (an Middleware)
Quellen; Chromosom; Position; Suche; Zoomstufe; Anzahl der Subintervalle; Detail-Flag(true | false)

← (von Middleware)
Wenn Detail-Flag = true:
RefSeq + Mutationen mit MetaDaten
Wenn Detail-Flag = false:
aggregierte Mutationen (durch Zoomstufe+Anzahl der Subintervalle)

Suchanfrage
GUI → Middleware (für alle Quellen)

```
{
  "source": ["src1", "src2", ...],
  "chromosome": int x,
  "search": String a
}
```

GUI ← Middleware (für alle Quellen, in dennen das gesuchte Gen vorkommt)

```
{
  "source": ["src1", "src2", ...],
  "chromosome": int x,
  "search": String a,
  "position": {"from": int x, "to": int y}
}
```

Intervallanfrage
GUI → Middleware (Je eine Nachricht pro Quelle)

```
{
  "source": String s,
  "chromosome": int x,
  "position": {"from": int x, "to": int y},
  "zoom": int y,
  "subindex": int z,
  "hasDetail": (true|false)
}
```

GUI ← Middleware (Je eine Nachricht pro Quelle)

```
{
  {
    "source:" String s,
    "chromosome": int x,
    "position": {"from": int x, "to": int y},
    "details": { "refseq": String b,
                 "mutations": [{ "name": String s,
                                 "position": {"from": int x, "to": int y},
                                 "metadata": {...}
                               },{...},...]},
    "graph": {
              { "subintervall": int x,
                "counts": int y
              }
            }
  }
}
```

Wenn das HasDetail-Flag „true“ ist, dann wird „detail“ befüllt und „graph“ bleibt leer.
Wenn das HasDetail-Flag „false“ ist, dann wird „graph“ befüllt und „detail“ bleibt leer.
Mit „subindex“ ist die Anzahl der Subintervalle gemeint.

2.7 Unit-Tests

2.7.1 QueryReceiver

Testfall 1 - erfolgreiche Intervallanfrage ohne Metadaten

```
  Eingabe
{
  "source": The Cancer Atlas,\usepackage[utf8]{inputenc}
  "chromosome": 3,
  "position": {"from": 100, "to": 200},
  "zoom": 1,
  "subindex": ,
  "hasDetail": (false)
}

  Ausgabe
{
  {
    "source:" The Cancer Atlas,
    "chromosome": 3,
    "position": {"from": 100, "to": 200},
    "details": { "refseq": ,
                "mutations": [{ "name": ,
                                "position": {"from": , "to": },
                                "metadata":
                              },]},
    "graph": {
              { "subintervall": Anzahl der Subintervalle,
                "counts": Anzahl der Ergebnisse
              }
            }
  }
}
```

Testfall 2 - erfolgreiche Intervallanfrage mit Metadaten

```
  Eingabe
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "position": {"from": 100, "to": 200},
  "zoom": 1,
  "subindex": ,
  "hasDetail": (true)
}
```

```
  Ausgabe
{
  {
    "source:" The Cancer Atlas,
    "chromosome": 3,
    "position": {"from": 100, "to": 200},
    "details": { "refseq": Referenzsequenz,
                 "mutations": [{Mutation1},{Mutation2},...]},
    "graph": {
              { "subintervall": ,
                "counts":
              }
            }
  }
}
```

Testfall 3 - erfolglose Intervallanfrage

```
  Eingabe
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "position": {"from": 100, "to": 200},
  "zoom": 1,
  "subindex": ,
  "hasDetail": (true)
}

  Ausgabe
{
  {
    "source:" The Cancer Atlas,
    "chromosome": 3,
    "position": {"from": 100, "to": 200},
    "details": { "refseq": ,
                 "mutations": ...},
    "graph": {
              { "subintervall": ,
                "counts":
              }
            }
  }
}
```

Testfall 4 - erfolgreiche Namensanfrage

Eingabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen im Cancer Atlas"
}
```

Ausgabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen im Cancer Atlas",
  "position": {"from": Startposition, "to": Endposition}
}
```

Testfall 5 - erfolglose Namensanfrage

Eingabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen, dass nicht im Cancer Atlas ist"
}
```

Ausgabe

```
{
  "source": The Cancer Atlas,
  "chromosome": 3,
  "search": "Gen im Cancer Atlas",
  "position": " "
}
```

Testfall 6 - fehlerhafte Anfrage

Eingabe

```
{
  "source": ""
}
```

Ausgabe

```
{
  "answer": "unknown format"
}
```

2.7.2 GeneTranslator

Der GeneTranslator hat 2 Aufgaben. Zum einen soll er während der Indexerstellung mit Inhalt (also Gennamen und zugeörigen Intervallen) befüllt werden, zum anderen soll eine Suche nach Gennamen in ihm möglich sein.

Testfall 1 - addGene - Einfügen in Datenstruktur
Eingabe: Testgen, 350, 500
Ausgabe: Dass Gen sollte in den Baum eingefügt sein und per searchForGene() findbar sein

Testfall 2 - addGene - Doppeltes Einfügen in Datenstruktur
Eingabe: Testgen, 350, 500
Testgen, 350, 500
Ausgabe: Dass Gen sollte nur einmal in die Datenstruktur eingefügt werden. Ausgabe, dass das Gen bereits in der Struktur vorhanden ist.

Testfall 3 - addGene - Aufruf ohne Parameter
Eingabe: [...]
Ausgabe: Fehlermeldung über parameterlosen Aufruf

Testfall 4 - tranlateToIntervall - erfolgreiche Suche
Eingabe: Testgen (befindet sich bereits in Datenstruktur)
Ausgabe: 350, 500

Testfall 5 - tranlateToIntervall - erfolglose Suche
Eingabe: Testgen2 (befindet sich nich in Datenstruktur)
Ausgabe: Fehlermeldung über erfolglose Suche

Testfall 6 - tranlateToIntervall - Aufruf ohne Parameter
Eingabe: [...]
Ausgabe: Fehlermeldung über Parameterlosen Aufruf

Testfall 7 - completeGeneName - erfolgreiche Suche mit einem Ergebnis
Eingabe: Test (Testgen1 befindet sich bereits in Datenstruktur)
Ausgabe: Testgen1

Testfall 8 - completeGeneName - erfolgreiche Suche mit mehreren Ergebnissen
Eingabe: Test (Testgen1 und Testgen2 befinden sich bereits in Datenstruktur)
Ausgabe: Testgen1, Testgen2

Testfall 9 - completeGeneName - erfolglose Suche
Eingabe: Test (Es befindet sich kein Gen mit Präfix Test in der Datenstruktur)
Ausgabe: Fehlermeldung über erfolglose Suche

2.7.3 IndexController

Testfall 1 - answerQuery - einfache Intervallanfrage
Eingabe: 1,3,100,200
Ausgabe: zum Intervall gehörende Mutationsobjekte

Testfall 2 - answerQuery - komplexere Intervallanfrage
Eingabe: 1,3,100,200 ; 1,3,150,350
Ausgabe: eine Mutationsliste mit den Ergebnissen beider Anfragen

Testfall 3 - answerQuery - unvollständige Intervallanfrage
Eingabe: 1,100,200
Ausgabe: Fehlermeldung über unvollständige Anfrage

Testfall 4 - answerQuery - leere Anfrage
Eingabe: [...]
Ausgabe: Fehlermeldung über leere Anfrage

Testfall 5 - answerQuery - überspezifizierte Intervallanfrage
Eingabe: 1,3,100,200,300,400
Ausgabe: Fehlermeldung über überspezifizierte Anfrage

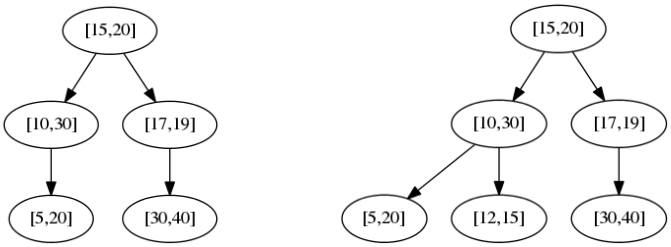
Testfall 6 - buildIndex - erfolgreicher Indexaufbau
Eingabe: [...] (Datenbank ist erreichbar)
Ausgabe: Erfolgreich gebauter Index

Testfall 7 - buildIndex - erfolgloser Indexaufbau
Eingabe: [...] (Datenbank ist nicht erreichbar)
Ausgabe: Fehlermeldung über erfolglosen Indexaufbau

2.7.4 Intervallbaum

1. Intervalle einfügen:
Das Intervall muss im Baum an der richtigen Stelle eingefügt werden und der Baum muss gegebenenfalls neu balanciert werden (z.B. wie ein AVL-Baum).

Bsp.: Einfügen des Intervalls [12,15]



2. Intervall mit Startpunkt > Endpunkt einfügen:
Wenn ein Intervall (S,E) mit S>E eingefügt wird, dann sollte unser Programm eine Fehlermeldung ausgeben und darauf hinweisen, dass die Grenzen für das Intervall nicht korrekt sind.

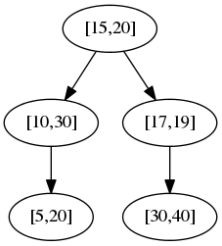
Bsp.: Einfügen des Intervalls [20,10] in einen beliebigen Baum.

3. Intervall mit Start- bzw. Endpunkt außerhalb des betrachteten Zahlenbereichs:
Wenn ein Intervall in dem Baum eingefügt werden soll, das teilweise oder vollständig außerhalb unseres Zahlenbereichs liegt (Länge des Genoms), dann muss es eine Fehlermeldung geben, die dem Nutzer mitteilt, dass der gültige Zahlenbereich überschritten wurde.

Bsp.: Einfügen des Intervalls [-5,7] in einen beliebigen Baum.

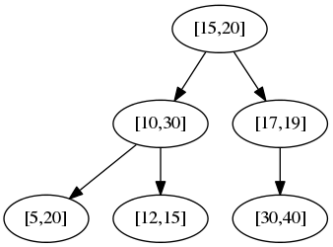
4. Schon vorhandenes Intervall einfügen:
Duplikate sollen von unserem Baum nicht gespeichert werden, d.h. es wird kein neuer Knoten hinzugefügt, sondern die Informationen (bei uns also Pointer auf Dateien) des neuen Knotens müssen im bereits vorhandenen Knoten mitgespeichert werden.

Bsp.: Einfügen des Intervalls [15,20] in den folgenden Baum



5. Suche nach vorhandenem Intervall:
Bei der Suche sollen alle Intervalle ausgegeben werden, die das gesuchte Intervall in irgendeinem Punkt überlappen.

Bsp.: Suche im folgenden Baum



Suche [4,5] ⇒ gib [5,20] aus
Suche [25,35] ⇒ gib [10,30] und [30,40] aus
Suche [20,20] ⇒ gib [15,20] und [5,20] aus

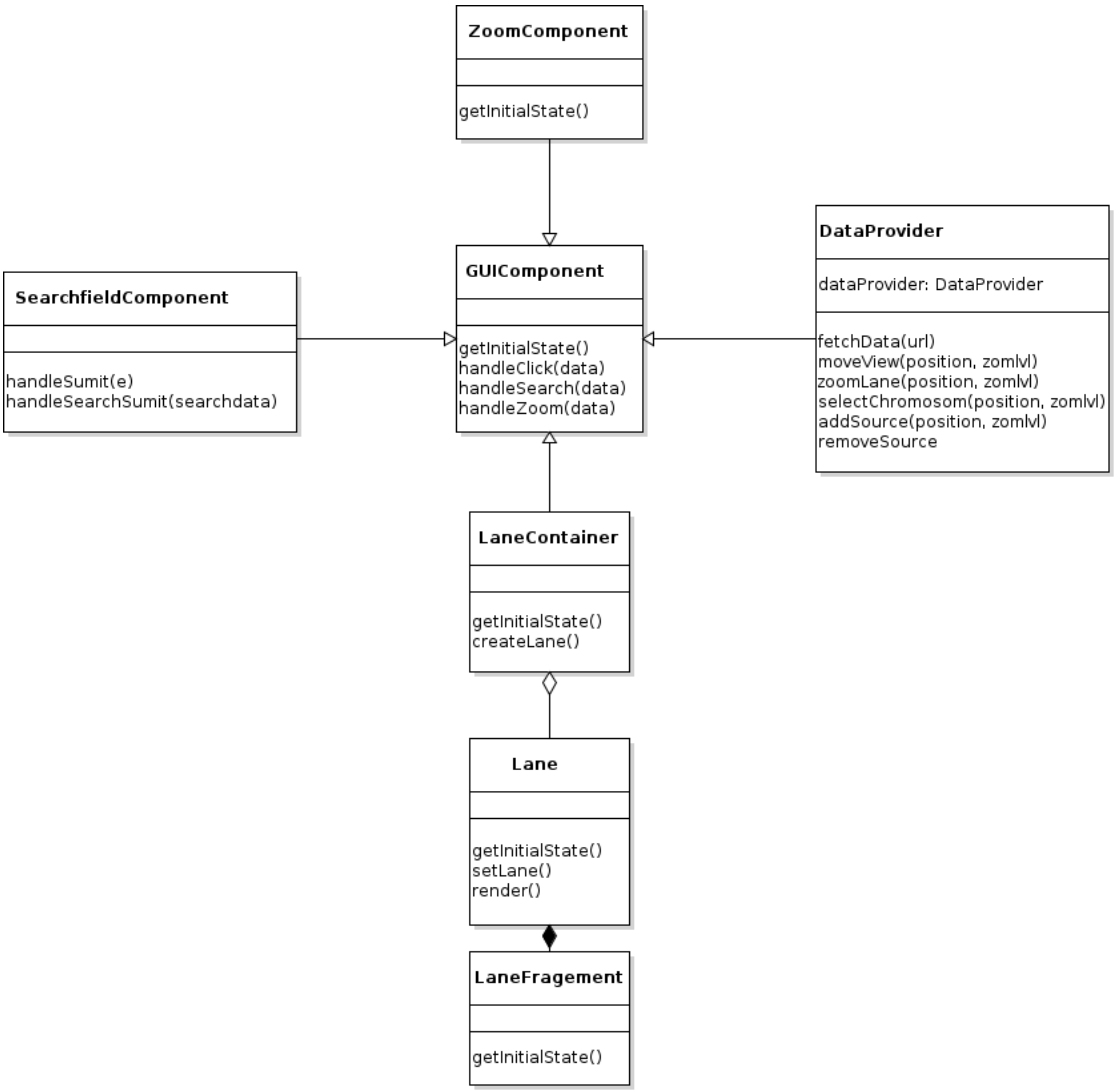
2.8 Stresstests

Der Stresstest hat zum Ziel herauszufinden, wie lange die durchschnittliche Query-Laufzeit ist.
Ziel des Systems ist es eine Laufzeit von unter einer Sekunde zu erreichen.
Um dies zu überprüfen werden mehrere Anfragen der gleichen Art (Intervallsuche, Gennamenssuche, Präfixsuche) sequentiell gestellt und die Antwortzeit gemessen. Alle Anfragen sollen in unter einer Sekunde eine Antwort erzielen.

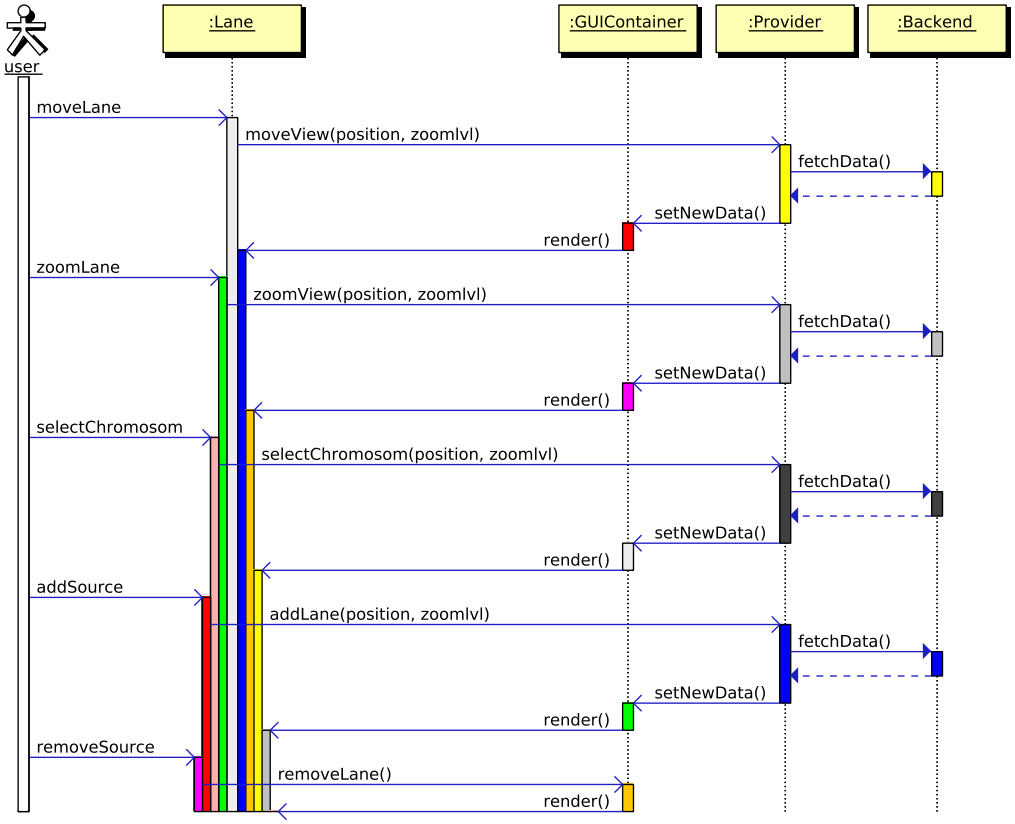
3 Frontend

3.1 Mock-Ups der Benutzerschnittstelle

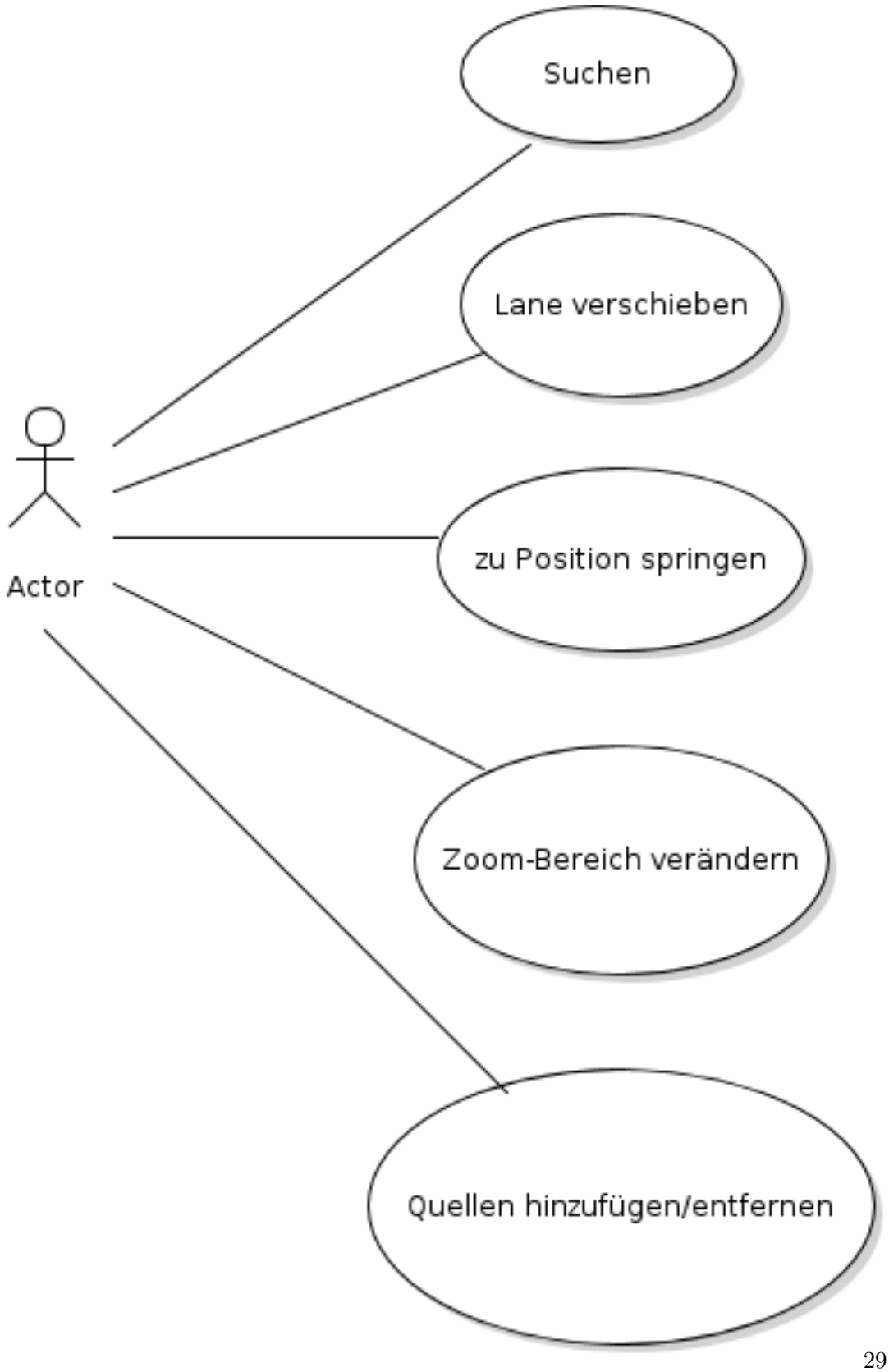
3.2 Klassen-Diagramm



3.3 Sequenzdiagramm



3.4 Use Cases


Suchen
Lane verschieben
zu Position springen
Actor
Zoom-Bereich verändern
Quellen hinzufügen/entfernen

3.5 Unit-Tests

3.5.1 Suchfunktion

1. Wenn ich als Nutzer eine leere Suche starte, dann möchte ich eine entsprechende Fehlermeldung angezeigt bekommen.
2. Wenn ich eine Suche mit falscher Eingabe starte, dann möchte ich eine entsprechende Fehlermeldung angezeigt bekommen.
3. Wenn ich als Nutzer nach einem gültigen Intervall suche, dann wird automatisch die Zoomstufe auf dieses Intervall angepasst.
4. Wenn ich als Nutzer nach einem vorhandenem Gene suche, dann wird automatisch die Zoomstufe auf dieses Intervall angepasst.

3.5.2 Quellen-Button

1. Wenn ich als Nutzer auf einen "QuellenButton drücke, dann wird mir die entsprechende Quelle zusätzlich zu den bereits dargestellten Quellen, angezeigt.
2. Wenn ich als Nutzer auf den "QuellenButton einer bereits angezeigten Quelle drücke, wird die entsprechende Quelle nicht mehr angezeigt.

3.5.3 Quellen-Scroller

1. Als Nutzer kann ich mich über horizontales Scrolling synchron durch die Quellen bewegen.

3.5.4 Zoom-slider

1. Wenn ich die Zoomstufe über den Slider ändere, dann werden die Quellen entsprechend der eingestellten Stufe dargestellt.
2. Wenn ich als Nutzer die feinste Zoomstufe einstelle, dann werden mir die Basenpaare angezeigt.
3. Wenn ich als Nutzer eine andere Zoomstufe einstelle, dann werden mir aggregierten Daten angezeigt.

3.5.5 Chromosom-Auswahl

1. Als Nutzer kann ich über ein Dropdown aus einer Vorauswahl von Chromosomen auswählen.
2. Wenn ich als Nutzer ein Chromosom auswähle, dann wird die Quellen-Anzeige automatisch entsprechend des ausgewählten Chromosoms aktualisiert.
3. Wenn ich als Nutzer das bereits ausgewählte Chromosom erneut auswählen, dann passiert nichts.

3.5.6 Allgemein

1. Wenn ich als Nutzer auf eine Anfrage warten muss, wird mir dies durch einen Loading-Spinner signalisiert.