Semesterprojekt - Formale Beschreibung der Middleware

December 20, 2015

1 Beschreibung der Middleware

Aufgabe der Middleware soll es sein die Daten, die in der integrierten Datenbank liegen effizient nach Quellen, Intervallen und weiteren Eigenschaften suchbar zu machen, also einen gezielten und schnellen Zugriff auf Mutationsdaten zu gewährleisten.
Hierfür muss sie sowohl in der Lage sein die Einträge aus den Datenbanken zu lesen und aus ihnen eine effizient suchbare Indexstruktur aufzubauen als auch eine Suchfunktionalität bereitstellen, auf die durch die Weboberfläche zugegriffen werden kann. Vom System unterstützte Suchformate sind:

- Intervallsuche
  - Input: Chromosom, Intervallgrenzen, Quellen und Suchfilter
  - Output: Alle Mutationen, die die vom Nutzer spezifizierten Einschränkungen erfüllen
- Gennamensuche
  - Input: Chromosom, Quellen, Genname
  - Output: alle Mutationen, die die vom Nutzer spezifizierten Einschränkungen erfüllen und die auf dem gesuchten Gen liegen
- Präfixsuche
  - Input: Präfix eines Gennamens
  - Output: alle Gene, die mit dem gesuchten Präfix beginnen

Es wird eine Suchstruktur erstellt, in der nach Intervallen gesucht wird und die Mutationsdaten enthält und eine weitere, in der nach Gennamen gesucht werden kann und die Intervallgrenzen für die jeweiligen Gene beinhaltet. Die Suchstruktur für Intervalle, in der die Mutationsdaten gespeichert werden wird verteilt aufgebaut, um eine effiziente Suche zu ermöglichen.
Als Austauschformat mit dem Frontend werden JSON-Nachrichten genutzt. Die

Kommunikation mit den Datenbanken erfolgt durch Datenbankzugriffe aus dem Programmcode heraus.
Dem Nutzer soll es weiterhin möglich sein die Ergebnisliste zu filtern. Unterstützte Filter sind die Quelldatenbanken in denen gesucht wird, das Chromosom auf dem gesucht wird, Geschlecht, bei der die Mutation auftritt, länderspezifische Herkunft der Mutationsträger und die relative Häufigkeit mit der die Mutation vorkommt.

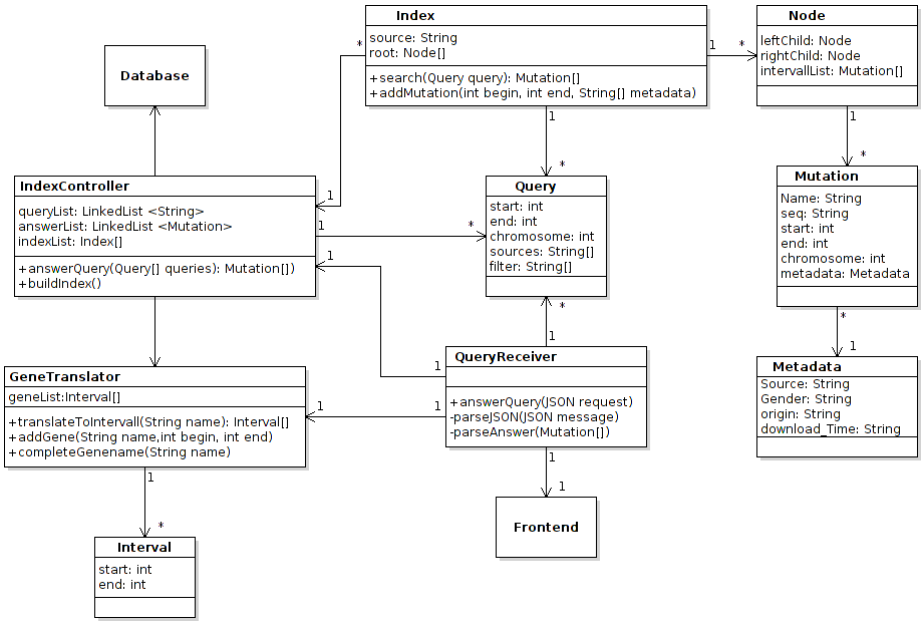

Figure 1: Schematischer Aufbau des Systems. Die Kommunikation mit dem Frontend findet über den QueryReceiver statt. Die Kommunikation mit der Datenbank übernimmt der IndexController. Die interne Kommunikation zwischen dem QueryReceiver, dem IndexController und den Indizes finden über Query-Objekte statt, die alle Informationen beinhalten, die für eine Suche benötigt wird. Der IndexController baut sowohl den Mutationsindex, als auch den GeneTranslator auf.

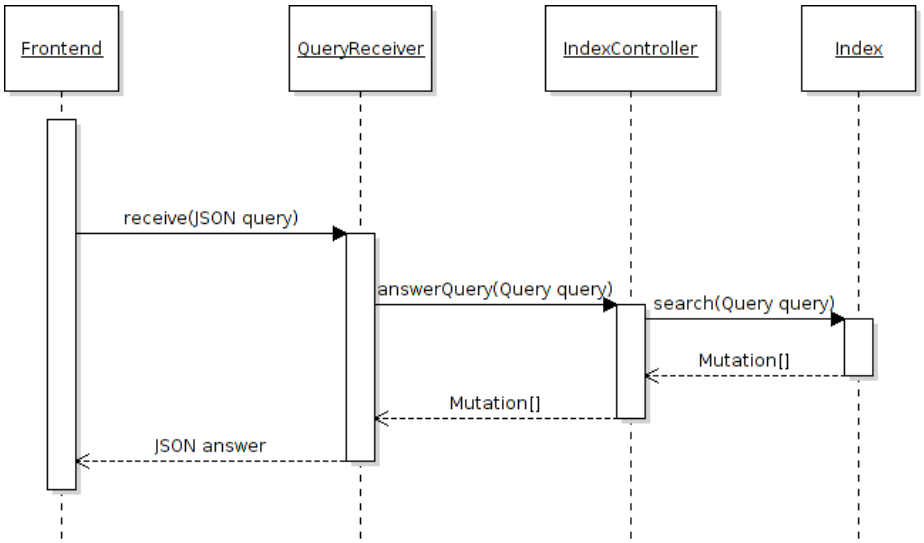

Figure 2: Schematischer Ablauf einer Intervallanfrage. Der QueryReceiver erhält eine Anfrage, interpretiert diese und sendet eine Anfrage mit allen nötigen Informationen an den QueryHandler. Dieser ruft mit der Anfrage die Suchfunktionen der Teilindizes auf. Das Ergebnis wird zurückgereicht, ins Antwortformat gepackt und an das Frontend gesendet

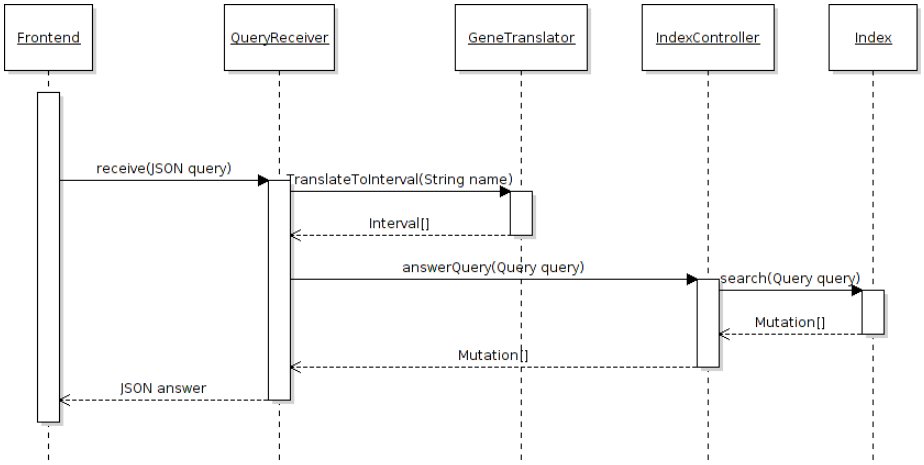

Figure 3: Schematischer Ablauf einer Gennamenanfrage. Nachdem der QueryReceiver die Nachricht erhalten hat schickt er den Gennamen an den GeneTranslator und erhält alle Intervalle, die dem Gen entsprechen. Mit dieser Information wird nun eine Query an den IndexController gestellt. Der weitere Ablauf ist identisch zu dem einer Intervallanfrage.

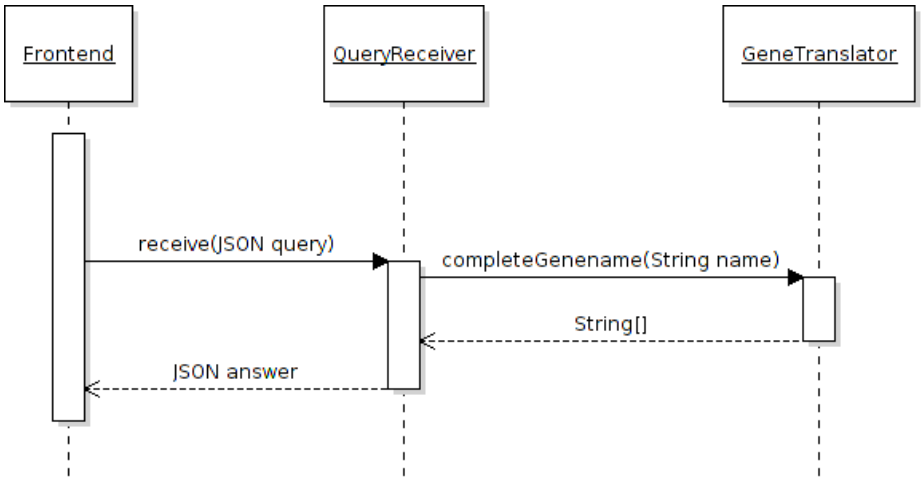

Figure 4: Schematischer Ablauf einer Präfixsuche. Der QueryReceiver erhält eine Anfrage und leitet den Genpräfix in der Anfrage an den GeneTranslator weiter. Dieser gibt alle Gene zurück, die mit diesem Präfix beginnen. Die Antwort wird vom QueryReceiver in das Anfrageformat gepackt und an das Frontend gesendet.

2 IndexController

2.1 Anforderungen

Der IndexController ist für den Zugriff auf den Gesamtindex verantwortlich. Er nimmt Intervallanfragen entgegen und leitet diese an die Teilindizes weiter. Die Teilergebnisse, die von den Indizes zurückgegeben werden, werden zusammengeführt und zurückgegeben. Außerdem baut er zu Programmstart aus der bestehenden Datenbank heraus den Index auf.

2.2 Funktionen

- answerQuery(Query[] queries) returns Mutation[] answers
  Die Funktion erhält eine Liste an Anfragen und gibt eine Liste aller Mutationen zurück, die den Anfrageforderungen entsprechen
- buildIndex() returns void
  Die Funktion baut den Index aus der Datenbank auf

2.2.1 answerQuery()

Die Funktion erhält eine Liste an Query-Objekten. Ein Query-Objekt beinhaltet alle nötigen Information zu einer gegebenen Anfrage (gesuchtes Intervall, Quelle, Chromosom, Filterangaben)
Die Anfragen werden nebenläufig an die 4 Teilindizes weitergeleitet. Die einzelnen Teilergebnisse werden zusammengefasst und zurückgegeben.

```
answerQuery()
Data: Query[] queries
Result: List of all matching Mutations for all queries
Query[] queries;
Index[] indices;
Mutation[] answers=null;
forall the q ∈ queries do
    forall the i ∈ indices do
        answers.add(i.search(q));
    end
end
return answers;
```

2.2.2 buildIndex()

Die Funktion wird bei Programmstart ausgeführt und baut auf Basis der Datenbank die 4 Teilindizes auf. Der Zugriff auf die Datenbank erfolgt mithilfe des

"JDBC41"-Treibers für "PostgreSQL". Es werden Anfragen der Form "SELECT * FROM Metadaten" und "SELECT * FROM Mutation" an die Datenbank gestellt, um mit den Ergebnissen die Indexstruktur zu befüllen. Wahrscheinlich werden die Anfragen noch geeignet aufgeteilt, da die Zwischenergebnisse sonst unter Umständen zu groß für den Hauptspeicher wären.
Für jede Mutation wird zufällig entschieden in welchen Index sie eingefügt wird. Nachdem der Gesamtindex aufgebaut wird, wird der GeneTranslator aus der Datenbank "RefSeq" aufgebaut.

```
buildIndex()
Data: –
Result: A built index
connect to mutation_database;
ResultSet rs = executeQuery(query1);
GeneTranslator j;
forall the elements e ∈ rs do
    choose sub index i to insert into;
    i.addMutation();
end
connect to genename_database;
rs = executeQuery(query2);
forall the elements e ∈ rs do
    j.insert(e);
end
return;
```

3 Index

3.1 Anforderungen

Der Index soll eine effiziente Suche nach Mutationen ermöglichen. Dabei gibt es die Möglichkeiten direkt nach einem Intervall zu suchen oder aber nach allen Mutationen zu suchen, die auf einem bestimmten Gen liegen. 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 sich in irgendeinem Punkt mit dem gesuchten Intervall überschneiden.
Der Index wird verteilt aufgebaut und teilt sich auf 4 virtuellen Maschinen auf. Die für den Index spezifizierten Funktionen sprechen immer einen der 4 Teilindizes an. Der Aufruf der Funktionen wird über den IndexController erfolgen, der immer alle 4 Teilindizes ansprechen wird.

3.2 Datenstruktur für den Index

Die dem Index zugrundeliegende Datenstruktur ist ein Intervallbaum. Dieser ermöglicht effizientes Suchen nach und Einfügen von Intervallen.
Hierfür wird die frei zugängliche Bibliothek "IntervalST.java" der Universität Princeton als Grundgerüst verwendet. Die vorgenommenen Änderungen belaufen sich hauptsächlich auf eine Anpassung der verwendeten Datenstrukturen, eine Erweiterung der Suchfunktion um Filter und einen geeigneten Umgang mit Duplikaten (mehrere Mutationen auf demselben Intervall). Außerdem können einige für uns unnötige Funktionen entfernt werden. Die Funktionen addMutation() und search() dienen als Interface zu den in der Bibliothek definierten Funktionen, die Elemente einfügen und suchen
Das Suchergebnis wird nach den bei der Anfrage spezifizierten Filtern gefiltert. Bei n Intervallen und einer Such-Ergebnisliste der Größe m ergibt sich ein Komplexität von $\mathcal{O}(\log n + m)$

3.3 Funktionen

- addMutation(Mutation mutation)returns void
  Die Funktion erhält eine Mutation, die eingefügt werden soll
- search(Query query)returns Mutation[]
  Die Funktion erhält eine Anfrage und gibt alle Mutation zurück, die den Anfrageanforderungen entsprechen

3.3.1 addMutation()

Die Funktion erhält eine Mutation als Parameter und fügt diese in Indexstruktur ein.

addMutation()
Data: Mutation input
Result: input added to index
IntervalTree i;
insert input into i;
return;

3.3.2 search()

Die Funktion erhält eine Anfrage nach dessen Kriterien nach Mutationen gesucht werden soll und einen Knoten des Baumes (zu Beginn der Wurzelknoten). Es werden alle Mutationen gesucht, die komplett oder teilweise innerhalb der Intervallgrenzen liegen. Für alle Mutationen auf die dies zutrifft wird geprüft, ob sie den Filtern entsprechen nach denen gesucht wird. Alle Elemente, die die Filtereinschränkung erfüllen werden zurückgegeben.

search()
Data: Query input, Node current
Result: List of all matching mutations
IntervalTree i;
Mutation[] answer_list;
if *current node matches searched interval* **then**
  add mutation to answer_list if it corresponds to specified filters;
else
  continue search in subtrees;
end
return answer_list;

4 GeneTranslator

4.1 Anforderungen

Der GeneTranslator wird aufgerufen, falls es sich bei der Anfrage um eine Präfixsuche oder eine Gennamensuche handelt. Die Komponente beinhaltet eine Datenstruktur, die alle im System vorhandenen Gennamen und ihre Intervallbereiche beinhaltet.
Die Komponente ermöglicht eine effiziente Suche nach Gen-Präfixen und nach Gennamen um die zugehörigen Intervalle zu erfragen.
Als interne Datenstruktur wird ein PatriciaTrie genutzt. Dieser ist in der frei zugänglichen Bibliothek "Apache Commons" vorhanden und wird für diese Implementation genutzt.

4.2 Funktionen

- translateToInterval(String name) returns Intervall[]
  Die Funktion erhält einen Gennamen, nach dem gesucht wird und gibt eine Liste von Intervallobjekten zurück, die den Start-und den Endpunkt aller Vorkommen des Gens beinhalten

- addGene(String name, int begin, int end) returns void
  Die Funktion erhält einen Gennamen und die Intervallgrenzen des Gens und fügt diese in die zur Suche benötigten Datenstruktur ein

- completeGenename(String name) returns String[]
  Die Funktion erhält einen Präfix eines Gennamens und gibt alle Gennamen zurück, die mit diesem Präfix beginnen

4.2.1 translateToInterval()

Diese Funktion wird aufgerufen, falls eine Gensuche angefragt wird. Sie überstetzt das gesuchte Gen in die zugehörigen Intervalle um eine Intervallsuche aus der Anfrage zu machen, die vom Index bearbeitet werden kann.

4.2.2 addGene()

Die Funktion fügt der internen Datenstruktur ein Gen und die zugehörigen Intervalle hinzu. Sollte sich das Gen bereits in der Struktur befinden, so wird überprüft, ob eines der vorhandenen Intervalle für diesen Gen komplett im neu hinzugefügten befindet. Ist dies der Fall, so wird das alte Intervall durch das neue ersetzt. Andernfalls wird das neue Intervall an die Liste der Intervalle für dieses Gen zugefügt. Sollten die Intervalle eines Gens komplett denen eines anderen Gens entsprechen, so kann davon ausgegangen werden, dass diese synonyme Bezeichnungen für das gleiche Gen sind. Da die Intervalle bereits gleich sind werden Synonyme implizit abgefangen.

4.2.3 completeGenename()

Die Funktion gibt alle Gennamen zurück, die mit dem gesuchten Präfix beginnen

5 QueryReceiver

5.1 Anforderungen

Der QueryReceiver bildet die direkte Schnittstelle zum Frontend. Alle Anfragen, die der Nutzer eingibt werden als JSON-File an ihn gesendet und Antworten werden von ihm an das Frontend zurückgegeben. Die Komponente überprüft für alle eingehenden Anfragen um was für eine Anfrage-Art es sich handelt und leitet sie an die zuständigen Teilkomponenten weiter.
Für die einzelnen JSON-Formate siehe Abschnitt Schnittstelle GUI-Middleware

5.2 Funktionen

- answerQuery(JSON request) returns JSON answer
  Die Funktion erhält eine Anfrage im JSON-Format und gibt eine Antwort im JSON-Format zurück

5.2.1 answerQuery()

Alle Nutzer-Anfragen werden an diese Funktion gesendet. Sie überprüft anhand des Aufbaus der JSON-Anfrage, um was für eine Anfrageart es sich handelt.
Bei der Intervallsuche wird die JSON-Anfrage in ein Query-Objekt umgewandelt und an den IndexController weitergeleitet. Bei der Gennamensuche wird zuerst im GeneTranslator nach dem zum Gen gehörigen Intervall gesucht und mit dieser Information und den Filterangaben des Nutzers ein Query-Objekt erstellt, dass dem IndexController übergeben wird. Handelt es sich um eine Präfixsuche wird im Genetranslator nach Gennamen gesucht, die mit dem Präfix beginnen. Die Antworten der jeweiligen Anfragen werden in das zum Anfragetyp gehörende JSON-Antwortformat verpackt und an das Frontend zurückgesendet.

6 Stresstest

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 zufällig je 50 korrekte Anfragen der gleichen Art (Intervallsuche, Gennamenssuche, Präfixsuche) an das System gestellt und die Antwortzeiten gemessen. Diese dürfen alle nicht länger als eine Sekunde zur Antwort brauchen.
Hierfür wird ein Testprogramm geschrieben, dass Anfragen im korrekten Format stellt und die Antwortzeiten automatisch auswertet. Die Anfragen werden zufällig generiert.