Diplomarbeit

Splicy:
Die automatisierte Analyse alternativer Spleißmuster

Verfasser:

Fabian Weiß

Betreuer:

Prof. Dr. Bernhard Haubold
Dr. Sascha Röhrig

Eidesstattliche Erklärung

Gemäß §23 Abs. 6 der Prüfungsordnung

Ich erkläre hiermit an Eides statt, dass die vorliegende Arbeit von mir selbst und ohne fremde Hilfe verfasst und noch nicht anderweitig für Prüfungszwecke vorgelegt wurde.

Es wurden keine als die angegebenen Quellen oder Hilfsmittel benutzt. Wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

Fürstenfeldbruck, 27. Februar 2004

Fabian Weiß

Danksagung

Ich möchte diese Möglichkeiten nutzen um einigen Personen zu danken.
Ich danke Herrn Prof. Dr. Bernhard Haubold für die Betreuung und tatkräftige Unterstützung bei dieser Arbeit.
Ein spezieller Dank gilt dem Abteilungsleiter der Bioinformatik bei der Firma Xantos Biomedicine AG, Herrn Dr. Sascha Röhrig, für die Betreuung und Unterstützung bei dieser Arbeit.
Bedanken möchte ich mich bei der Firma Xantos Biomedicine AG für die Möglichkeit der Durchführung dieser Arbeit und Herrn Dr. Ulrich Brinkmann für die Bereitstellung des Themas dieser Arbeit.
Des Weiteren bedanke ich mich für die Unterstützung in bioinformatischen, programmiertechnischen, mathematischen und administrativen Fragen bei meinen Kollegen, Frau Dr. Bettina Ehring, Herrn Alexander Felber, Frau Dr. Beate Gawin, Herrn Dr. Björn Kesper, Herrn René Korn, Herrn Dr. Reinhold Köckerbauer, Herrn Robert Sachse, Herrn Jürgen Schiefeneder und Herrn Dr. Alexander Spychaj.
Besonderer Dank gilt meiner Familie. Seiner Mutter hat man stets am meisten zu danken und so ist das auch bei mir der Fall. Um dies alles anzusprechen ist hier jedoch leider nicht der Platz gegeben. Deshalb danke ich meiner Mutter und meinem Bruder an dieser Stelle für die Unterstützung, die Geduld und die Ermöglichung des Studiums.
Besonderer Dank gilt meiner Freundin und baldigen Frau Andrea Kötting.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungen

G	Guanin
C	Cytosin
A	Adenin
T	Thymin
U	Uracil
EST	Expressed Sequence Tag
DNA	Deoxyribonucleic Acid
cDNA	Copy DNA
mRNA	Messenger Ribonucleic acid
snRNP	small nuclear Ribonucleoprotein
bp	Basenpaare
CDS	Coding Sequence
ORF	Open Reading Frame
HEK293	Human embryonic kidney cells
CGAP	Cancer Genome Anatomy Project
ASD	Alternative Splicing Database
BLAST	Basic Local Alignment Search Tool
EMBOSS	European Molecular Biology Open Software Suite
NCBI	National Center for Biotechnology Information
NCI	National Cancer Institute
TAP	Transcript Assembly Programm
WUBlastN	Washington University BLAST Nucleotide
EU	Europäische Union
BNF	Backus-Naur-Form
EBNF	Erweiterte Backus-Naur-Form
UML	Unified Modelling Language
RMI	Remote Method Invocation
JDK	Java Development Kit
XML	Extensible Markup Language
SQL	Structured Query Language
JVM	Java Virtual Machine
ERD	Entity Relationship Modell
CLOB	Character Large Object
javaCC	Java Compiler Compiler
JDBC	Java Database Connectivity
AWT	Abstract Windowing Toolkit
JFC	Java Foundation Classes

1. Zusammenfassung

Das Ziel dieser Arbeit lag in der Implementierung eines Programms zur automatisierten Analyse alternativer Spleißvarianten. Als zugrunde liegende Daten wurde die dbEST, die *Expressed Sequence Tags* Datenbank des *National Centers for Biotechnology Information* (NCBI) benutzt. Das Ergebnis dieser Arbeit ist das Programm *Splicy*.

Zu Beginn dieser Arbeit wurde eine manuelle Auswertung bestimmter Genloci auf alternative Spleißmuster durchgeführt. Darauf aufbauend wurde ein effizienter Algorithmus entwickelt, der diese Aufgabe automatisierte.

Für die Implementierung wurde sowohl ein *Entity-Relationship Modell* (ERD) für die Datenbankstruktur, als auch ein UML Diagramm für die benötigten Klassen erstellt.

Zur Visualisierung der Analysenergebnisse wurde anschließend eine grafische Oberfläche implementiert. Diese stellt die Daten ansprechend und übersichtlich dar und stellt unter Anderem folgende zwei Funktionalitäten zur Verfügung. Erstens ermöglicht die Analyse und Visualisierung möglicher Verschiebungen des *Open Reading Frames* (ORF) eines Transkriptes gegenüber der entsprechenden *Reference Sequence* (*RefSeq*) eine schnelle Beurteilung der Auswirkung von Spleißvarianten auf das entstehende Protein. Zweitens wird eine Expressionsdatenanalyse einzelner ESTs durchgeführt und basierend auf den Exons visualisiert. Anhand letzterem lassen sich Aussagen über die Verteilung alternativer Spleißvarianten im Organismus und eventuell auch Rückschlüsse über deren Funktion machen.

Abschluss dieser Arbeit bildet eine erste Anwendung *Splicy*s. Dabei wurden die Gene von 144 Clustern (firmeneigene Definition einer Sammlung an cDNA-Klonen, deren Sequenzen den besten *RefSeq* Treffer bei einer BLAST-Analyse gemeinsam haben), die auf Apoptose Induzierung getestet wurden, auf alternative Spleißvarianten geprüft.

2. Einleitung

Die Entschlüsselung des menschlichen Genoms durch das *International Human Genome Sequencing Consortium* war eine der bedeutendsten wissenschaftlichen Leistungen des 20. Jahrhunderts. Die Entschlüsselung ermöglichte erstmals eine Vorhersage der Anzahl der menschlichen Gene basierend auf der fast vollständigen genomischen Sequenz. Überraschenderweise erwies sich dabei die Zahl der Gene mit 30.0000 – 40.000 weitaus geringer als bisher angenommen [1]. Bis dato schätzte man die Anzahl der Gene anhand der Zahl unterschiedlicher Transkripte auf ca. 150.000 [2]. Somit besitzt der Mensch nur ca. doppelt soviel Gene wie *Caenorhabditis elegans* oder *Drosophila melanogaster* [2]. Die höhere biologische Komplexität des Menschen lässt sich daher nicht allein aufgrund der Genzahl erklären. Einen weitaus höheren Beitrag zur gesteigerten Komplexität kann ein zellulärer Mechanismus leisten, durch den aus einem einzigen Gen eine Vielzahl an Proteinen entstehen kann, das so genannte **alternative Spleißen**. Alternatives Spleißen wurde bereits 1978 von Gilbert et al. [3] beschrieben. Zu dieser Zeit wurde klar, dass eukaryotische und prokaryotische Gene nicht dieselbe Struktur besitzen.

Nach der Transkription, dem Abschreiben der DNA zur mRNA, werden die für das später entstehende Protein kodierenden Bereiche (Exons) von den Nichtkodierenden (Introns) getrennt. Dabei werden aus der prä-mRNA, der ersten Abschrift der DNA, die Introns ausgeschnitten und die Exons zu einem funktionellen Botenmolekül (der mRNA) zusammengesetzt. Diesen Vorgang bezeichnet man als Spleißen. Die Exons können in unterschiedlicher Weise (alternativ) wieder zusammengesetzt werden. Dadurch können aus einem Genlocus mehrere verschiedene mRNAs entstehen. Dies wird als alternatives Spleißen bezeichnet. Bei der Translation, der Übersetzung der in der mRNA kodierten Information in die Aminosäuresequenz des Proteins, können schließlich verschiedene Proteine entstehen.

Über die Anzahl alternativ gespleißter Gene im humanen Genom gibt es noch keine genauen Zahlen. Anhand von Vergleichen mit *Expressed Sequence Tags* (ESTs) kamen Brett et al. darauf, dass ca. 38% der humanen mRNA mögliche

alternative Spleißmuster enthalten [4], jedoch bestätigen die bisher sequenzierten ESTs nur einen Teil der Gesamtheit aller Gene des menschlichen Organismus.

2.1. Mechanismus des alternativen Spleißens

Das alternative Spleißen findet nach der Transkription statt. Dabei können aus einem Genlocus mehrere mRNAs gebildet werden.
Die zunächst bei der Transkription gebildete prä-mRNA enthält noch die vollständige genetische Information, also alle Introns und Exons. Beim Spleißen werden nun die Exons aus der prä-mRNA herausgeschnitten und zur reifen mRNA zusammengesetzt.
Als Erkennungsmerkmal an welchen Stellen die prä-mRNA geschnitten werden muss, fungieren die Spleiß-Donor- und Spleiß-Akzeptorstellen. In Abb.1 wird ersichtlich, dass ein Intron meistens mit der Sequenzfolge „GU“ startet und mit „AG“ endet. Dabei wird „GU“ als der Spleiß-Donor und „AG“ als der Spleiß-Akzeptor bezeichnet.

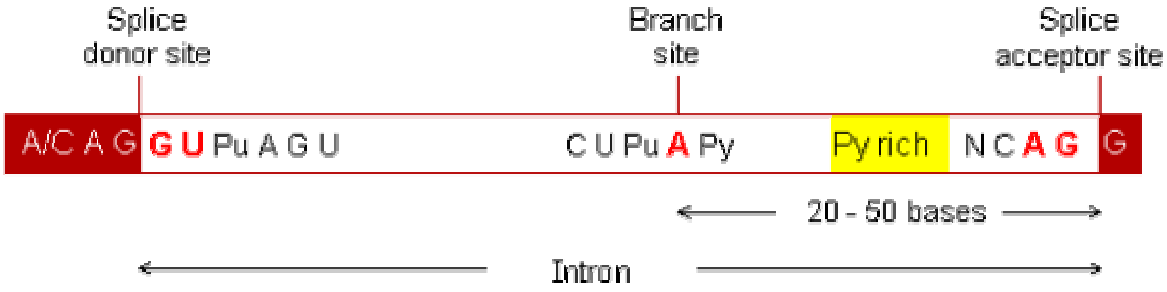

Abb.1: Erkennung der Spleißstellen (entnommen aus [5])

Ein weiteres, wichtiges Muster für die Erkennung der Spleißstellen befindet sich 20-50 Basen vom Spleiß-Akzeptor entfernt, die sog. *branch site*. Dieser Bereich enthält eine Purinbase (Adenin oder Guanin) gefolgt von einem Adenin und einer Pyrimidinbase (Cytosin oder Uracil). Dabei ist das Adenin, welches sich zwischen der Purin- und der Pyrimidinbase befindet, in allen Genen konserviert. In 60% aller Fälle ist die Exonsequenz am Splice-Donor Adenin oder Cytosin gefolgt von Adenin und Guanin (siehe Abb.1). Am Splice-Donor ist in diesem Fall die

Exonsequenz ein Guanin [5]. Katalysator für die enzymatische Spleißreaktion ist das sog. Spleißosom. Dieser 60S-Komplex besteht aus fünf *small nuclear Ribonucleicprotein* (snRNPs), dem U1, U2, U5, U4/U6 (siehe Abb.2). Die Reaktion läuft in zwei Schritten ab. Im ersten Schritt spielt die o.g. *branch site* eine große Rolle. Das Adenin der *branch site* attackiert das 3'-Ende des ersten Exons und schneidet an dieser Stelle die prä-mRNA. Dabei wird dort eine freie Hydroxylgruppe erzeugt. Das nun freie 5'-Ende des Introns bindet kovalent an das Adenin der *branch site*.

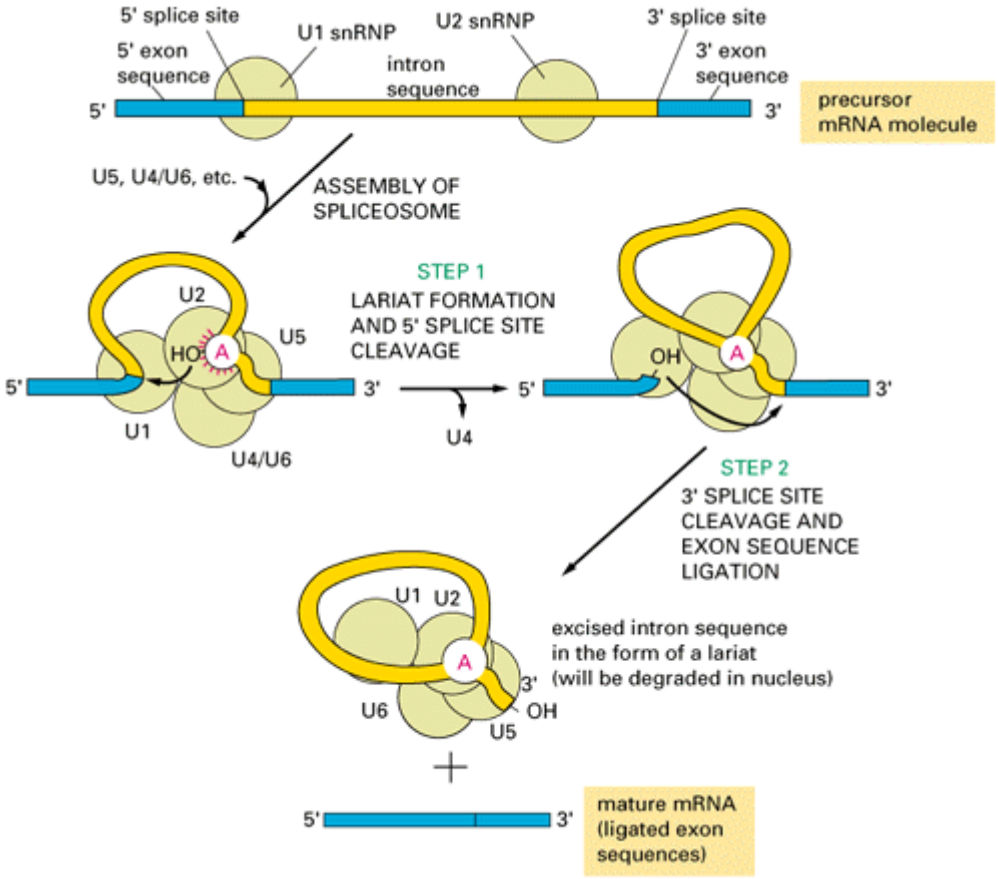

Abb.2: Spleißmechanismus (entnommen aus [6])

Im zweiten Schritt bindet das 3'-Ende des ersten Exons an das 5'-Ende des zweiten Exons, wobei das Intron abgeschnitten wird. Wenn dies für alle Introns und Exons durchgeführt wird, liegt eine mRNA ohne Introns vor.

Beim alternativen Spleißen variiert die Zusammensetzung der Exons. Es kann z.B. ein Exon ausgelassen werden, am 3'- oder 5'-Ende eines hinzugefügt werden. Die Art und die Häufigkeit des alternativen Spleißens variiert unter den Genen. Von

einigen Genen weiß man, dass durch alternatives Spleißen tausende Proteinvarianten exprimiert werden können. Als Beispiel wäre das *Down syndrome cell adhesion molecule* (DSCAM) bei Drosophila zu nennen. Dort können durch den Mechanismus des alternativen Spleissens 38.016 unterschiedliche mRNAs entstehen [7]. Diese Proteine können ein breit gefächertes Wirkungsspektrum besitzen. Ihre Eigenschaften können ähnlich sein, jedoch auch weit voneinander abweichen, bis hin zum antagonistischen Verhalten [8].

2.2. Alternatives Spleißen und seine Auswirkungen

Der Mechanismus des alternativen Spleißens spielt nicht nur im gesunden Organismus eine entscheidende Rolle, sondern auch bei Krankheiten. Wenn alternatives Spleißen am falschen Ort oder zur falschen Zeit ausgelöst wird, kann es weitreichende Folgen für den Organismus haben und zu Krankheiten wie Krebs führen.

Caceras und Kornblihtt entdeckten 2002, dass 15% der Mutationen, die eine Krankheit auslösen, das Spleißen betreffen [9]. Daher wird dem alternativen Spleißen mittlerweile eine große Bedeutung in der Diagnostika- und Medikamentenentwicklung zugesprochen.

Die EU hat deshalb 2003 das *Alternative Splicing Database* Projekt (ASD) ins Leben gerufen. Dabei soll ein Konsortium aus sieben Forschungsteams eine Datenbank alternativ gespleißter Gene und deren Expressionsmuster erstellen [10]. Man denkt daran Medikamente zu entwickeln, die nur gegen die krankheitsauslösende Spleißvariante wirksam sind. Somit würden die anderen Spleißvarianten vom Medikament unberührt bleiben und eventuelle Nebenwirkungen könnten verhindert werden [11].

2.3. Informationsvielfalt der *Expressed Sequence Tags* (ESTs)

Um möglichst viele Transkripte eines Organismus in kürzester Zeit kennen zu lernen, werden EST-Sequenzierungsprojekte durchgeführt.

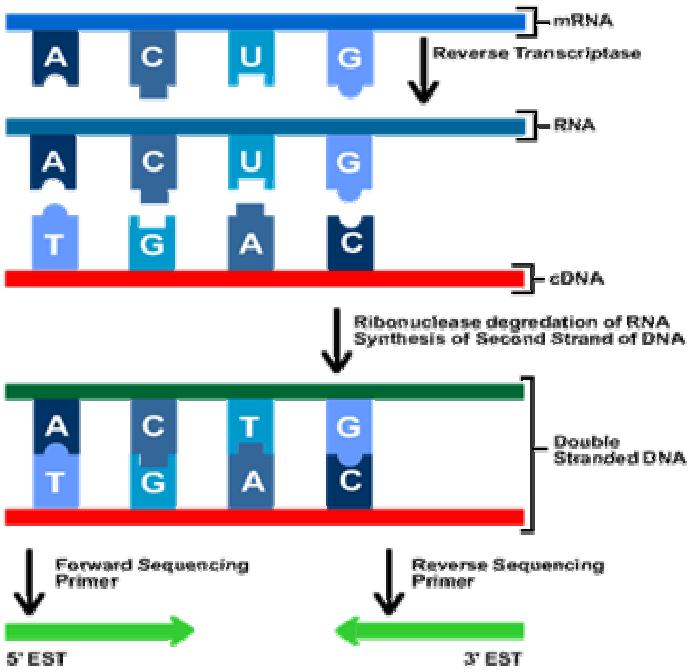

Abb. 3: EST Entstehung (entnommen aus [12])

ESTs sind kurze Sequenzen von 300 – 500 bp Länge. Zu ihrer Gewinnung wird mRNA, die einer bestimmten Zelle bzw. Gewebe zu einer bestimmten Zeit entnommen wurde, in cDNA transformiert [12]. Die Isolierung der mRNA gestaltet sich einfach, da nur nach dem Poly-A-Schwanz gefischt werden muss. Durch Sequenzierung eines der beiden Enden (5'- oder 3'-read, siehe Abb.3) erhält man dann die 5'- bzw. 3'- Teilsequenzen exprimierter Gene.
ESTs geben damit Aufschluss über Gene, die in einer Zelle oder einem Gewebe, in einem bestimmten Entwicklungsstadium, tatsächlich exprimiert sind.

Die enorme Anzahl an verfügbaren humanen ESTs birgt ein großes Potential an Information. Durch die hohe Anzahl an ESTs kann damit ein gezieltes *Clustering* von alternativen Spleißformen einzelner Gene durchgeführt werden. Als *Clustering* wird in diesem Zusammenhang die Gruppierung der einzelnen ESTs bezeichnet. Alle übereinstimmenden Spleißvarianten werden zu einer, der Längsten, zusammengefasst um am Ende nur die längsten Repräsentanten der alternativen Spleißmuster vorzufinden. Für „echte" Spleißvarianten, also Varianten, die in der Natur wirklich vorkommen, wird es mit hoher Wahrscheinlichkeit mehrere

bestätigende ESTs geben. Diese Tatsache kann man sich zunutze machen und versuchen mit dieser Methode neue Spleißvarianten zu bestimmten Genen zu finden.
Eine zusätzliche interessante Informationsquelle sind die Gewebeinformationen der ESTs.

AU120023. AU120023 HEMBA1 H...[gi:10935258]

IDENTIFIERS

dbEST Id: 6511496
EST name: AU120023
GenBank Acc: AU120023
GenBank gi: 10935258

CLONE INFO
Clone Id: HEMBA1007177 (5')
DNA type: cDNA

PRIMERS
PolyA Tail: Unknown

SEQUENCE
CTTTTTCGCAACGGGTTTGCCGCCAGAACACAGGTGTCGTGAAAACTACCCCTAAAAGCCAAAATGGGAAAGGAAAAG
ACTCATATCAACATTGTCGTCATTGGACACGTAGATTCGGGCAAGTCCACCACTACTGGCCATCTGATCTATAAATGCG
GTGGCATCGACAAAAGAACCATTGAAAAATTTGAGAAGGAGGCTGCTGAGATGGGAAAGGGCTCCTTCAAGTATGCCT
GGGTCTTGGATAAACTGAAAGCTGAGCGTGAACGTGGTATCACCATTGATATCTCCTTGTGGAAATTTGAGACCAGCAA
GTACTATGTGACTATCATTGATGCCCCAGGACACAGAGACTTTATCAAAAACATGATTACAGGGACATCTCAGGCTGAC
TGTGCTGTCCTGATTGTTGCTGCTGGTGTTGGTGAATTTGAAGCTGGTATCTCCAAGAATGGGCAGACCCGAGAGCAT
GCCCTTCTGGCTTACACACTGGGTGTGAAACAACTAATTGTCGGTGTTAACAAAATGGATTCCACTGAGCCACCCTACA
GCCAGAAGAGATATGAGGAAATTGTTAAGGAAGTCAGCACTTACATTAAGAAAATTGGCTACAACCCCGACACAGTAGC
ATTTGTGCCAATTTCTGGTTGGAATGGTGACAACATGCTGGAGCCAAGTGCTAACATGCCTTGGTTCAAGGGATGGAAA
GTCACCCGTAAGGATGGCAATGCCAGTGGAACCACGCTGCTTGANGCTCTGGACTGNATTCTACCACCNACT

Entry Created: Oct 19 2000
Last Updated: Aug 1 2002

COMMENTS
HRI human cDNA project; 5'- & 3'-end one pass sequencing:
Helix Research Institute; cDNA library construction:
Department of Virology, Institute of Medical Science,
University of Tokyo, and Helix Research Institute.

LIBRARY
Lib Name: HEMBA1
Organism: Homo sapiens
Tissue type: whole embryo, mainly head
Develop. stage: embryo, 10 weeks
Vector: pME18SFL3

Abb.4: Ausschnitt eines EST Eintrags in der dbEST Datenbank

Zu jedem EST wird neben seinen verschiedenen Identifikationsschlüsseln (*Accessions / Identifiers*), der Sequenz und einigen weiteren, der Abb.4 zu entnehmenden, Informationen auch die Daten der entsprechenden cDNA-Bibliothek (*Library*) angegeben. Zur *Library* gehört ein eindeutiger Name, sowie Angaben über Entwicklungsstadium und Art des Gewebes, aus dem die

vorliegende Sequenz generiert wurde. Im Falle des ESTs aus Abb. 4 wäre der Name „HEMBA1“.
Zusätzliche Informationen über diese *Library* sind in Abb. 5 zusammengefasst.

Title	Tissue	Histology	Type	Protocol	Keywords
HEMBA1	head and neck	Normal	bulk	uncharacterized treatment	normal, bulk, embryo, head, EST

Abb.5: Zusätzliche Informationen zu einer Library

Daraus lässt sich ableiten, dass die mRNA, die dieses EST beschreibt, in Gewebe Kopf und Nacken vorkommt. Außerdem besagt die Histologie, dass es in normalem Gewebe vorkommt. ESTs werden prinzipiell in zwei Gruppen eingeteilt. Zum einen sind dies ESTs, die in gesundem Gewebe vorkommen, zum anderen ESTs aus malignem Gewebe.
Um sich die Daten zunutze zu machen, wurden die Daten der dbEST aus der flachen Struktur der Textdatei in eine relationale Datenbank transformiert. dbEST ist die *Mirrordatenbank* des NCBI (*National Center for Biotechnology Information* http://www.ncbi.nih.gov). Damit liegen die Daten relational vor.
In dbEST befinden sich momentan 5.471.624 (Stand: 11. Februar 2004) Einträge für *Homo Sapiens*.

2.4. Erarbeitung der Vorgehensweise

Ziel dieser Arbeit war die Implementierung eines Programms zum automatisierten Auffinden und Auswerten alternativer Spleißvarianten.
Zuerst wurde der Schwerpunkt auf die Methodenentwicklung durch manuelle Auswertungen von Genloci gelegt.
Dabei wurden grundsätzliche Vorgehensweisen zu diesem Thema erarbeitet.
Dafür wurde auf bereits öffentlich verfügbare Informationen zurückgegriffen. Eine Informationsquelle war der *evidence view* des NCBI. Dort sind bereits Daten zu Genloci verzeichnet.

Es werden u.a. bekannte Gene, die auf ihren genomischen Abschnitt kartiert wurden, im gespleißten Zustand visualisiert (siehe Abb.6).

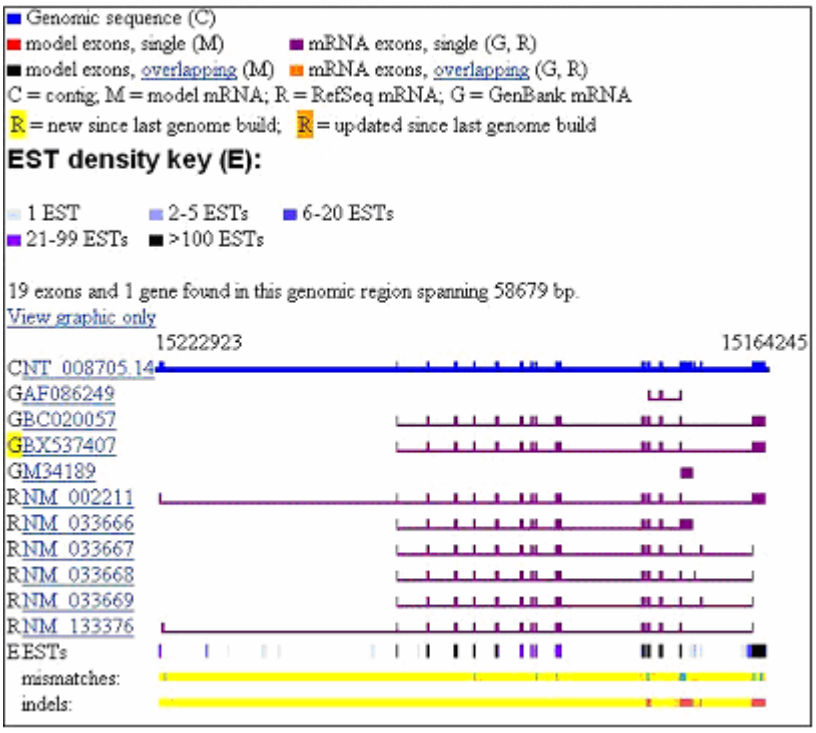

Abb.6: evidence view zu RefSeq Klon NM_002211

Für eine Analyse von Genen auf alternative Spleißvarianten hin ist es nötig, die genomische Sequenz und die Koordinaten des Genlocus zu kennen. Für eine Untersuchung wurde ein Sequenzvergleich mittels BLAST mit der aus dem *evidence view* entnommenen, genomischen Sequenz mit der EST-Datenbank vollzogen. Als Ergebnis erhielt man eine große Anzahl von ESTs, die dem entsprechenden Genlocus zugeordnet werden konnten. Anschließend wurden die kartierten (zugeordneten) ESTs erneut mit der genomischen Region abgeglichen, um auf ihre alternativen Spleißstellen überprüft zu werden. Dies geschah mit Hilfe des Programms *Spidey* [13].

Anschließend wurden jeweils ca. 50 ESTs mit der genomischen Region abgeglichen und auf alternative Spleißstellen und Qualitätsmerkmale überprüft. Mehr als 50 ESTs erwiesen sich als zu viel, da unter diesen Umständen sowohl die Rechenzeit zu lang wurde, als auch die Vergleiche auf alternative Spleißmuster visuell nicht mehr wahrgenommen werden konnten. Zur Qualitätssicherung der Analyse wurden die gefundenen alternativen

Spleißvarianten auf Vorhandensein von Spleiß-Donor- und Spleiß-Akzeptorstellen und einer Identität mit der genomischen Region von über 90% kontrolliert.

2.5. Aufgabenstellung

Alternatives Spleißen hat mit der Entschlüsselung des humanen Genoms enorm an Bedeutung gewonnen. Um die möglichen Auswirkungen von alternativem Spleißen analysieren zu können, ist es unumgänglich, möglichst umfassend Spleißvarianten zu finden.
Ziel dieser Arbeit war es daher, ein Programm zu entwickeln, welches schnell, einfach und zuverlässig alternative Spleißmuster zu frei wählbaren Genloci auffindet.

Aus 2.4. wurde deutlich, dass für Spleißanalysen eine Automatisierung der dort beschriebenen Vorgehensweise wünschenswert wäre. Eine manuelle Analyse hat Nachteile insbesondere in zwei wichtigen Punkten:

1.) Bei der Genauigkeit der Analyse. Bei den enormen Datenmengen, die der Auswertung zu Grunde liegen, ist es nur bedingt möglich, alle Parameter manuell zu prüfen und somit alle Spleißmuster zu finden.

2.) Bei der Dauer einer Analyse. Bei manueller Auswertung ist ein Mitarbeiter an einer Analyse mehrere Stunden beschäftigt. Eine Automatisierung bringt somit neben der höheren Geschwindigkeit auch den Vorteil, dass sich der Benutzer in der Zwischenzeit mit anderen Dingen beschäftigen kann.

Wie auch im Fall der manuellen Auswertung, sollte für die automatisierte Analyse die dbEST Datenbank zugrunde gelegt werden. Bereits andere Forschungsgruppen nutzten die Vielfalt der ESTs und derer Annotationen für Ihre Vorhaben. So erkannten z.B. Brett et al. [4] durch EST-Vergleiche, dass von 7867 nicht redundanten mRNAs 3011 alternative gespleißte Formen haben. Das

bedeutet, es konnte anhand von EST-Vergleichen bei 38% der untersuchten mRNAs alternatives Spleißen festgestellt werden. Weiterhin benutzten Hanquing Xie et al. [14] die Gewebeinformationen der ESTs um alternativ gespleißte Gene und Gensegmente zu finden, die in einzelnen Krebsarten hoch exprimiert vorkommen.

Anhand dieser Beispiele wird klar, welche Informationsvielfalt in den ESTs steckt. Darüber hinaus kommen täglich viele ESTs zur dbEST hinzu, womit ein Wachstum dieses enormen Potentials gesichert ist.

Aus o.g. Gründen wurde für diese Arbeit die dbEST Datenbank als Ausgangspunkt gewählt. Der Schritt von einer enormen Menge an ungeordneten ESTs hin zum Auffinden alternativer Spleißvarianten bestimmter Genloci war die Hauptaufgabe dieser Arbeit.

3. Materialien

3.1. Biologische Datenbanken

Für diese Arbeit wurden mehrere biologische Datenbanken verwendet. Im Folgenden werden diese beschrieben.

3.1.1. *Expressed Sequence Tags* Datenbank

Wie bereits in 2.5. beschrieben, wurde dbEST als wichtigste Datenbank für *Splicy* benutzt. dbEST ist eine Datenbank des NCBIs, die mittlerweile über 5 Mio. humane ESTs beinhaltet. Verfügbar sind die Reports unter ftp://ftp.ncbi.nih.gov/repository/dbEST/ und die FASTA Dateien unter ftp://ftp.ncbi.nih.gov/blast/db/FASTA/est_human.gz .

3.1.2. Kontig

Ein Sequenzkontig ist definiert als eine Gruppe von einzelnen Sequenzen, die aufgrund von Sequenzüberlappungen zu einem größeren Ganzen zusammengefügt werden. Sequenzkontigs des NCBI stellen überlappende Sequenzen dar, die von mehreren Klonen stammen. Sie können einen Entwurfsstatus oder vollkommen sequenzierte Klone enthalten. Möglich ist auch, dass Sequenzlücken (*Gaps*) enthalten sein. Diese können von *Gaps* in Klonen oder wenn sie zwischen zwei Klonen liegen, von einem noch nicht sequenzierten Klon herrühren.
Die Kontigs von *Homo sapiens* (und andere) sind auf dem NCBI Server nach Chromosomen sortiert zu finden (ftp://ftp.ncbi.nih.gov/genomes/H_Sapiens).
Für *Splicy* ist dies neben dbEST die wichtigste Datenbank. Sie enthält das (fast) komplette humane Genom. *Splicy* benutzt die einem Genlocus zugehörige genomische Sequenz, um mittels BLAST einen Sequenzvergleich mit der EST_Datenbank durchzuführen. Da die Kontigs alle möglichen Introns und Exons

einer Genregion enthalten, bekommt man anhand einer BLAST-Analyse mit einem Kontigbereich und der EST-Datenbank alle möglichen Spleißvarianten.
Die genauen Koordinaten des gesuchten Genes auf den bestimmten Kontigs sind in der *LocusLink* Datenbank enthalten.

3.1.3. *LocusLink* Datenbank

LocusLink ist eine Datenbank des NCBI, die verschiedene Informationen über einzelne Genloci enthält (http://www.ncbi.nih.gov/*LocusLink*/index.html). Die wichtigsten Informationen, die *Splicy* von *LocusLink* benötigt, ist die Verlinkung der *RefSeq*-Klone mit den Genloci auf den genomischen Regionen.
Mit Hilfe dieser Informationen ist es *Splicy* möglich, den entsprechenden genomischen Bereich, den es benötigt, zu extrahieren.

3.1.4. *RefSeq* Datenbank

Die *Reference Sequence* (*RefSeq*) Datenbank ist eine nicht redundante Sammlung von DNA, RNA und Proteinsequenzen [15].

3.2. Bioinformatik- und Informatikwerkzeuge

3.2.1. Entwicklungsumgebung

Implementierungssprache:
Für die Implementierung *Splicy*s wurde die objektorientierte Sprache Java™ benutzt. Dabei wurde zur Entwicklung das JDK 1.4.1 von *Sun Microsystems* benutzt. Da im weiteren Verlauf der Arbeit eine Portierung des bestehenden Systems auf einen unter LINUX betriebenen Webserver angedacht wurde, bot sich Java™ neben anderen Argumenten v.a. durch die vorhandene Plattformunabhängigkeit an.

Betriebssystem:
Für die Visualisierung der Ergebnisse wurde *Microsoft Windows XP* benutzt. Die leistungskritische Analyse von Genen auf alternative Spleißmuster wurde auf einem LINUX Server implementiert.

Entwicklungsumgebung
Als Entwicklungsumgebung wurde das unter www.netbeans.org frei verfügbare *NetBeans 3.5* gewählt.

Zugrunde liegende Datenbank:
Um die Ergebnisse der Analysen relational sichern zu können, wurde eine *Oracle 8i* Datenbank benutzt.

Parsergenerator:
Für die Implementierung von Parsern, die für die Übernahme biologischer Datenbanken in die lokale Datenbankstruktur benötigt werden, wurde der Parsergenerator *javaCC* benutzt.
JavaCC ist ein *open-source-Compiler-Compiler* der unter https://*javaCC*.dev.java.net/ frei verfügbar ist.

Ein Parsergenerator ist ein Programm, welches eine Grammatik einliest und daraus einen Parser für diese Grammatik erstellt. Die Grammatik beschreibt dabei, was der Parser als korrekt erkennt. Die Grammatik kann in diesem Fall selbst erstellt und auf die Bedürfnisse der einzelnen Aufgaben angepasst werden. Die Grammatik von *javaCC* ist an die *erweiterte Backus Naur Form* (EBNF) angelehnt. Diese beschreibt eine Metasyntax, um kontextfreie Grammatik zu erstellen. Für weitere Informationen zur *Backus Naur Form* sei aufgrund der Komplexität dieses Themas auf [16] verwiesen.

Ein mit *javaCC* erstellter Parser kann einen Eingabestrom anhand der vorher definierten Grammatik analysieren. Somit lassen sich biologische Datenbanken, die häufig in Form von *Flatfiles*, d.h. Textdateien vorliegen, in die interne Datenbankstruktur übernehmen. Hierfür wird eine Grammatik beschrieben, die auf das einzelne *Flatfile* angepasst ist. Durch Kompilierung mit *javaCC* wird ein JAVA Quelltext erzeugt, der wiederum nach erfolgreicher Kompilierung einen ausführbaren Parser darstellt. Mit dessen Hilfe ist es schließlich möglich die Daten zu übernehmen.

Weitere Komponenten:
Splicy soll neben der Analyse alternativer Spleißmuster auch die Expressionsdaten der gefundenen ESTs anzeigen. Für die Visualisierung dieser Daten wurde eine *JTreeTable* benutzt. Eine *JTreeTable* ist eine Kombination eines *JTrees* mit einer *JTable*. Diese Komponente ist kein Standard in *Swing* ist aber unter http://java.sun.com/products/jfc/tsc/articles/treetable1/ frei verfügbar.

3.2.2. BLAST

Basic Local Alignment Search Tool (BLAST) hat sich als eines der bedeutendsten bioinformatischen Tools etabliert [17]. BLAST ist eine heuristische Methode des Sequenzvergleichs, die für den Vorteil, schnell Sequenzübereinstimmungen zu bekommen, die sehr geringe Gefahr zulässt, evtl. eine besser passende Sequenz zu übersehen.

Die Signifikanz einer gefundenen Sequenzübereinstimmung wird durch den E-Wert beschrieben. Er gibt die Wahrscheinlichkeit an, mit der es sich bei dem Treffer um einen zufälligen Treffer handelt.
Von BLAST ist sowohl die Einzelplatz- als auch die Netzversion unter http://www.ncbi.nih.gov/BLAST/ frei erhältlich.
In dieser Arbeit wurde das Programm *blastn* des Einzelplatz-BLAST-Paketes benutzt, um Sequenzvergleiche auf Nukleotidebene durchzuführen.
.

3.2.3. Spidey

Spidey wurde in dieser Arbeit für den Abgleich von mRNA Teil- oder Vollsequenzen mit einem genomischen Sequenzkontig benutzt. Dabei hat *Spidey* optimierte Algorithmen um Spleißstellen des entsprechenden Gens zu finden. Im Gegensatz zur einfachen BLAST-Analyse zweier Sequenzen sucht *Spidey* nach übereinstimmenden und die ganze mRNA abdeckenden Sequenzübereinstimmungen. Anschließend wird versucht, die noch nicht abgeglichenen Teilstücke der mRNA mit der genomischen Region abzugleichen. Als letztes überprüft *Spidey* die Exongrenzen darauf hin, dass sie nicht überlappen und dass sie Spleiß-Donor und Spleiß-Akzeptor besitzen. Nach erfolgter Analyse liefert *Spidey* die genauen Positionen der Introns und Exons auf der genomischen Region. Zusätzlich dazu erhält man Informationen über Donor- und Akzeptorstellen, an welchem Strang der DNA die Übereinstimmung gefunden wurde und weitere Qualitätsmerkmale, wie die prozentuale Übereinstimmung der verglichenen Sequenzen.
Die Netz- sowie die für *Splicy* benutzte Einzelplatzversion sind unter http://www.ncbi.nlm.nih.gov/IEB/Research/Ostell/Spidey/ frei erhältlich.

3.2.4. EMBOSS

EMBOSS (European Molecular Biology Open Software Suite Internet: http://www.hgmp.mrc.ac.uk/Software/EMBOSS/) ist ein frei verfügbares Softwarepaket für Sequenzanalysen.
Splicy benutzt zweie Programme des EMBOSS-Pakets:

- *Extractseq*: Mit Hilfe dieses Programms ist es möglich, bestimmte Abschnitte einer genomischen Region zu extrahieren. *Splicy* benutzt diese Funktionalität, um aus der genomischen Region den Teil zu isolieren, der für den *RefSeq* Klon kodiert.
- *Merger*: Diesem Programm können zwei Sequenzen übergeben werden, die anschließend zusammengefügt werden. Dabei vergleicht *Merger* die beiden Sequenzen und bestimmt dann die Konsensussequenz der beiden. *Splicy* bedient sich dieser Funktionalität, indem es dem Benutzer ermöglicht 3' und 5' read ESTs verschiedener Transkripte mit Hilfe *Mergers* zusammenzufügen.

3.2.5. Client-/Server-Architektur

Splicy ist für eine Client-/Server-Architektur entwickelt worden. Die für diese Arbeit benutzte Architektur ist der Abb.7 zu entnehmen. Die grafische Oberfläche des entwickelten Programms wird auf den Clients ausgeführt. Dem Benutzer wird über die grafische Oberfläche angeboten, eine Analyse zu starten. Diese wird per RMI aus dem Server ausgeführt.
Remote Method Invokation (RMI) bedeutet übersetzt soviel wie „entfernter Methodenaufruf". Normalerweise können in Java™ Methoden immer nur innerhalb einer *Java Virtual Machine* (JVM) aufgerufen werden. Mit Hilfe von RMI ist es möglich, dass ein Objekt aus einer JVM ein Objekt in einer anderen JVM aufruft. Da diese Methode auch über Rechnergrenzen hinweg funktioniert, kann sie für ein Client-/Servermodell angewendet werden.

Der Server erhält über JDBC (Java Database Connectivity) Zugriff auf den Datenbankserver und kann damit die Ergebnisse der Analyse auf der Datenbank sichern.

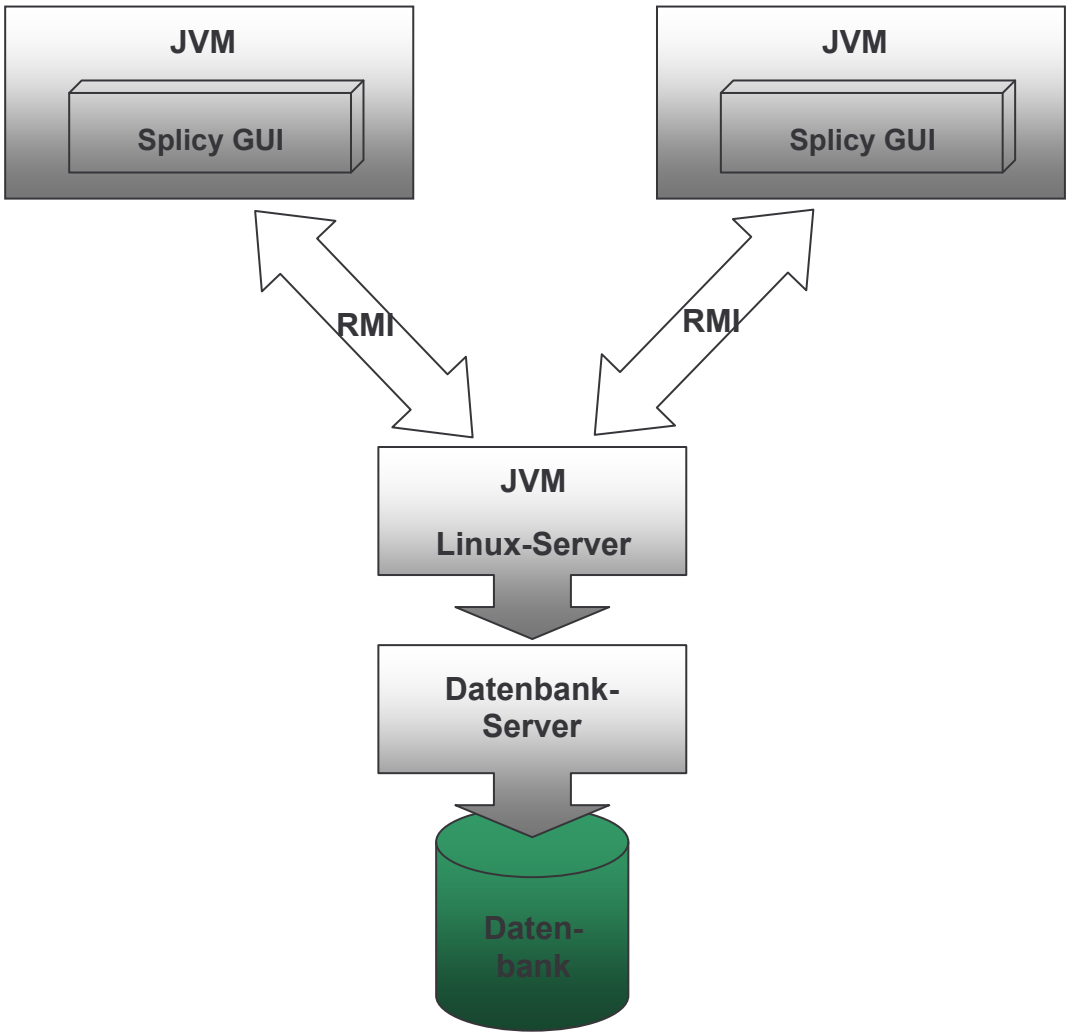

Abb.7: Client-/Server-Architektur die von Splicy verwendet wird

Durch diese Architektur ergeben sich folgende Vorteile:

- Die Analyse kann auf dem ressourcenreichen Server ausgeführt werden und belastet nicht den Client.

- Die benötigten Programme, BLAST und *Spidey*, sowie die für den BLAST biologischen Datenbanken müssen nicht auf jedem einzelnen Client verfügbar sein.
- Nach dem Starten einer Analyse kann *Splicy* beendet oder auch der Client heruntergefahren werden.
- Die Analysenergebnisse sind von jedem Client aus abrufbar.

4. Ergebnisse

Ziel dieser Arbeit war es, die Auffindung alternativer Spleißmuster zu bestimmten Genloci zu automatisieren. Dafür bedurfte es einer Reihe von Vorüberlegungen, die durchgeführt werden mussten, bevor mit der Implementierung begonnen werden konnte.

Der erste wichtige Punkt nach erfolgter Einarbeitung war, bereits vorhandene Programme zu finden, die *Splicy* sich zunutze machen könnte. Dabei fiel das Hauptaugenmerk auf ein Programm, *Transcript Assembly Program (TAP)*, das von Wissenschaftlern der *Washington University* erstellt wurde [18]. TAP sollte in der Lage sein, neben einer Genvorhersage auch alternative Spleißvarianten zu bestimmten Genloci aufzufinden. Dabei bedient sich dieses Programm *sim4,* um die ESTs mit der zugehörigen genomischen Region abzugleichen. Sim4 ist, ähnlich wie *Spidey*, ein Programm für die Auffindung der Spleißstellen einer mRNA [13].

TAP identifiziert alternative Spleißmuster eines Gens, indem es vorhergesagte Spleißpaare mit der bekannten Genstruktur vergleicht. Aus ESTs gefolgerte Spleißpaare, welche nicht in den bekannten Genstrukturen gefunden wurden, werden als „alternativ“ erachtet.

Nach erfolgreicher Installation des Programms und all seiner benötigten Komponenten (sim4 und WUBlastN (BLAST-Programm der Washington University)) wurde es getestet. Mit Daten bereits zuvor manuell analysierter Genloci fand TAP keine alternativen Spleißmuster. Bei der manuellen Ausarbeitung hingegen wurden je nach Genlocus mehrere alternative Spleißvarianten gefunden.

Ein Installationsfehler konnte ausgeschlossen werden, da zusätzlich die Netzversion TAPs getestet wurde und auch dort keine zufrieden stellenden Ergebnisse geliefert wurden. Infolgedessen war TAP für die Implementierung *Splicy*s keine geeignete Grundlage.

Da ansonsten kein Programm dieser Art gefunden werden konnte, wurde an dem Algorithmus der in 2.4. beschriebenen Einarbeitung festgehalten.

In der Designphase gab es drei Punkte abzuhandeln:

- Es war wichtig, den in 2.4. beschriebenen Algorithmus auf die Bedürfnisse einer automatisierten Analyse anzupassen
- Für die Effizienz des Programms musste eine performante Datenbankstruktur erarbeitet werden. Dazu wurde ein *Entity-Relationship-Modell* erstellt.
- Für eine objektorientierte Implementierung wurde ein UML-Klassendiagramm erstellt. Dies sollte die Klassenstruktur der einzelnen Javaklassen visualisieren.

Die folgenden Abschnitte werden die Vorgehensweisen bei den genannten Punkten verdeutlichen.

4.1. Algorithmus des Programms

Aufgrund der großen Datenmengen kommt dem im Rahmen dieser Arbeit entwickeltem Algorithmus eine große Bedeutung zu. Die Art des Algorithmus bestimmt wesentlich die Dauer und die Genauigkeit der Analyse. Veranschaulichen lässt sich dieser Algorithmus in den Flussdiagrammen der folgenden Abbildungen.
Abb.8 zeigt einen Überblick über den gesamten Ablauf einer *Splicy*-Analyse. Wert wurde bei der Entwicklung des Algorithmus hauptsächlich auf die Genauigkeit der Analyse gelegt. Somit ließ sich die Dauer einer Analyse zwar optimieren, jedoch wird es nicht möglich sein, aufgrund einiger nicht weiter zu optimierender Schritte, die Analyse zur Laufzeit durchzuführen.
Einigen Zeitgewinn ließ sich durch die Verwendung bereits optimierter bioinformatischer Werkzeuge, BLAST und *Spidey*, erreichen. Zusätzlich dazu benutzt *Splicy* die in 3.2.5. beschriebene Client-/Server Architektur, mithilfe derer eine Analyse auf einem Client gestartet werden kann, welche jedoch auf dem ressourcenreicheren Server durchgeführt wird.

Ferner ist es möglich, die Analyse direkt auf dem Server zu starten, was vor allem für eine größere Anzahl von Analysen genutzt werden kann.

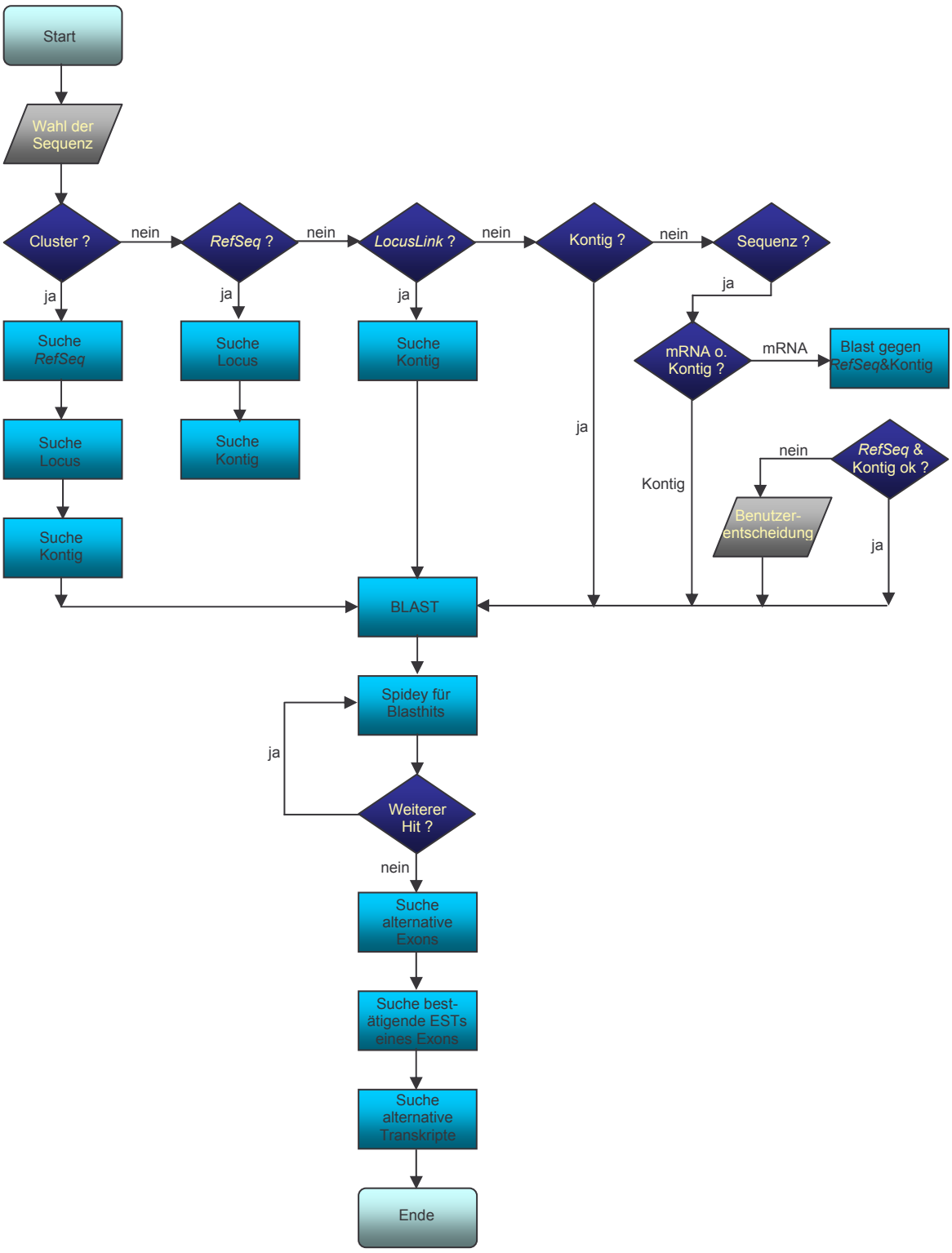

Abb.8: Übersicht über den Algorithmus Splicys

Splicy wurde so konzipiert, dass dem Benutzer fünf verschiedene Möglichkeiten eines Analysenstarts geboten werden. Diese sollen alle dazu führen, die für die weitere Analyse benötigte genomische Region des zu untersuchenden Genlocus zu finden.
Folgende Wege werden angeboten:

a) Cluster: Ein Cluster ist eine firmeneigene Definition einer Sammlung an cDNA-Klonen, deren Sequenzen den besten *RefSeq* Treffer bei einer BLAST-Analyse gemeinsam haben.
In diesem Fall wird *Splicy* den entsprechenden *RefSeq* Hit in der Datenbank suchen. Mit Hilfe von *LocusLink* gibt es nun die Möglichkeit, den Genlocus auf der Kontigsequenz zu lokalisieren.

b) *RefSeq*: Mit Hilfe der *RefSeq Accession* ist es *Splicy* möglich, analog der in a) beschriebenen Methode die benötigte genomische Sequenz zu erhalten.

c) *LocusLink*: Wiederum analog a) erhält *Splicy* die genomische Region.

d) Kontig: Da hier bereits die genomische Region vorgegeben wird, benötigt *Splicy* vom Benutzer nur noch die genaue Start- und Stoppposition des Gens auf dieser Kontigsequenz.

e) Sequenz: Bei dieser Möglichkeit wird es dem Benutzer ermöglicht, eine Sequenz an *Splicy* zu übergeben. Um die zugehörige genomische Region zu finden, führt *Splicy* zur Laufzeit sowohl eine BLAST-Analyse gegen die *RefSeq* Datenbank, als auch gegen die Kontig Datenbank durch. Anschließend wird überprüft, ob der Genlocus des *RefSeq* Treffers mit dem des Kontig Treffers übereinstimmt. Dies wird wieder mit Hilfe von *LocusLink* durchgeführt. Bei positivem Ergebnis wird die Analyse mit der gefundenen genomischen Region gestartet. Im Falle einer Nichtübereinstimmung wird dem Benutzer die Möglichkeit eröffnet, sich zwischen der *RefSeq*- und der Kontigsequenz zu entscheiden.

Wie in Abb.8 zu ersehen ist, laufen alle fünf Startmöglichkeiten in einem Punkt zusammen, dem BLAST. An diesem Punkt beginnt der eigentliche Algorithmus *Splicy*s.

Um die einem Genlocus zugehörigen ESTs zu finden, wird mit dessen Sequenz eine BLAST-Analyse gegen dbEST ausgeführt. Diese kann, je nach Länge des Locus, zwischen einigen Sekunden bis zu mehreren Stunden dauern. Somit wird sich dieser Schritt geschwindigkeitslimitierend auswirken. Als Ergebnis werden die zu einem Locus gehörigen ESTs identifiziert.

Da eine BLAST-Analyse über die gesamte genomische Kontigsequenz nicht nur lange Zeit in Anspruch nehmen würde, sondern auch das Ergebnis durch zufällig gefundene Sequenzübereinstimmungen verfälschen könnte, wurde für die BLAST-Analyse nur der Teil der Kontigsequenz benutzt, der für den entsprechenden *RefSeq*-Klon kodiert. Diese Koordinaten sind der *LocusLink* Datenbank zu entnehmen.

Bei der BLAST-Analyse werden aus noch unerklärlichen Gründen auch ESTs gefunden, die zu beiden Strängen der genomischen Region komplementär sind. Dies könnte auf sogenannten *inverted repeats* der DNA beruhen, welche Regionen darstellen, bei denen auf komplementären Strängen der DNA identische Sequenzen auftreten. Eine mRNA eines Locus wird jedoch immer nur von einem Strang der DNA abgelesen. Daher musste für das weitere Vorgehen der kodierende Strang der DNA bestimmt werden. Möglich wurde dies durch eine erneute BLAST-Analyse der Sequenz des entsprechenden *RefSeq* Klons gegen die genomische Region. Da die Sequenz des *RefSeq* Klons vom NCBI bestätigt ist, sollte die zugehörige Orientierung der DNA die Korrekte sein. Für die weitere Analyse werden somit nur die dem kodierenden Strang zugehörigen ESTs einbezogen.

Um eine Analyse nach alternativen Spleißvarianten durchführen zu können, wird jedoch nicht nur das EST selbst, sondern auch dessen Spleißstellen benötigt. Ermöglicht wird dies durch eine *Spidey* Analyse für jedes einzelne EST. Diese werden auf ihre Spleißstellen analysiert und auch Aussagen über die Qualität dieser Analyse gemacht.

Bei einigen durch die BLAST-Analyse gefundenen ESTs ist es unter Umständen möglich, dass der Sequenzvergleich 3’ oder 5’ über die genomische Region hinausginge. Um bei der Spleißstellenfindung die ESTs über die Gesamtlänge vergleichen zu können, wird *Spidey* mit der gesamten Kontigsequenz ausgeführt. Aus diesem Grund kann sich, je nach Länge der genomischen Sequenz, auch dieser Schritt geschwindigkeitslimitierend auswirken. Da jedoch aus den oben genannten Gründen auf eine Analyse zur Laufzeit verzichtet wurde, kann auch dieser Schritt zugunsten der Genauigkeit auf diese Weise vollzogen werden.
Nach erfolgter *Spidey* Analyse eines ESTs wird dessen Ausgabe von *Splicy* geparst und alle relevanten Daten in der Datenbank abgelegt.
Im Anschluss daran erfolgt die eigentliche Suche nach alternativen Spleißmustern. Dieser Vorgang wird in den folgenden drei Schritten ausgeführt:

- Suche nach alternativen Exons
- Suche nach bestätigenden ESTs für einzelne Exons
- Suche nach alternativen Transkripten

4.1.1. Suche nach alternativen Exons

In diesem Schritt untersucht *Splicy* die Daten der *Spidey* Analyse auf alternative Exons. Es werden dabei die Exons aller im BLAST gefundenen ESTs geprüft und nur diejenigen in der Datenbank gesichert, die als alternativ erachtet wurden. Ein Exon gilt dabei als alternativ wenn es folgende Kriterien erfüllt:

- Das Alignment des Exons muss eine Übereinstimmung mit der genomischen Region von über 90% besitzen.
- Falls sich das Exon in der Mitte eines ESTs befindet, also kein 3’ oder 5’ Exon, müssen ein Spleiß-Donor und ein Spleiß-Akzeptor vorhanden sein.
- Bei 3’ Exons genügt das Vorkommen eines Spleiß-Akzeptors, da das 3’ Ende durch Sequenzierabbrüche o.ä. verkürzt sein kann.
- Bei 5’ Exons muss ein Spleiß-Donor vorhanden sein.

Im Falle von *RefSeq* Klonen wurde eine Ausnahme von den oben beschriebenen Vorgaben gemacht. Im Rahmen mehrerer Analysen kam es bei zwei *RefSeq* Klonen dazu, dass bei einzelnen Exons kein Spleiß-Akzeptor und Spleiß-Donor vorgefunden wurde. Da der *RefSeq*-Klon einer Analyse für einige Funktionalitäten, wie der Rasteranalyse, die in 4.4.3.5. besprochen wird, benötigt wird, wurde diese Ausnahme mit in *Splicy*s Algorithmus übernommen.

Für die Überprüfung aller Exons auf Alternativität durchläuft *Splicy* die in den nächsten beiden Abbildungen visualisierten Algorithmen.

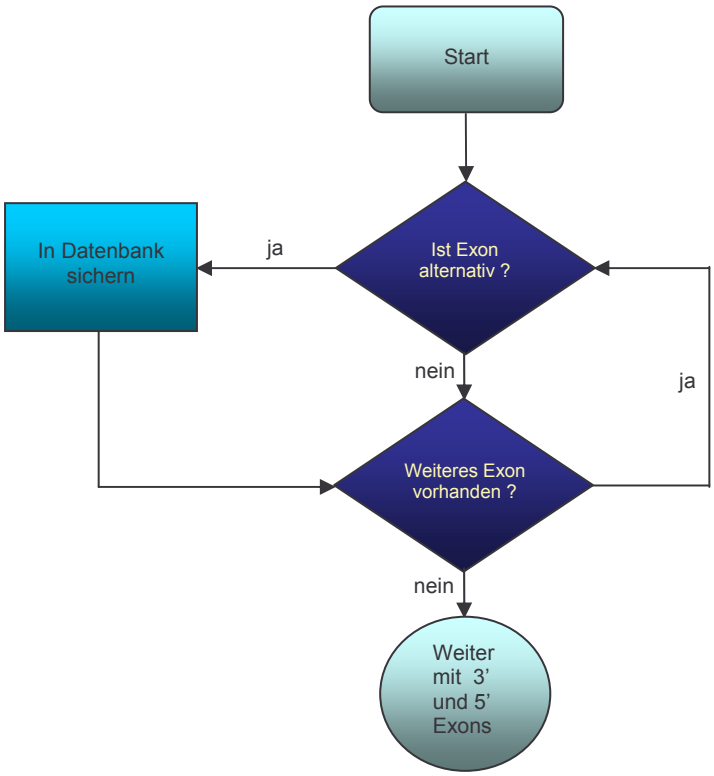

Abb.9: Algorithmus zum Auffinden alternativer Mittelexons

Für diese Überprüfung muss zwischen Exons, die sich am Ende eines ESTs befinden (5'- und 3'- Exons), und den Mittleren unterschieden werden. Im Gegensatz zu den Exons an den Enden, wird bei einem mittleren Exon jeder Unterschied in den genomischen Koordinaten als alternatives Spleißmuster gewertet. Bei Exons am 3' oder 5' Ende könnte es sich bei einer Abweichung lediglich um eine Verkürzung anderer mittlerer Exons handeln.

Um ein mittleres Exon als alternativ zu klassifizieren führt *Spidey* den in Abb.9 gezeigten Algorithmus aus. Dabei wird jedes mittlere Exon, das in der *Spidey*-Analyse gefunden wurde, mit jedem anderen mittleren Exon verglichen. Wenn ein Exon in seinen genomischen Koordinaten nicht mit denen anderer übereinstimmt, es zusätzlich noch Spleiß-Donor und Spleiß-Akzeptor besitzt und der Vergleich mit der genomischen Region eine Übereinstimmung über 90% zeigt, wird es als alternativ befunden und in der Datenbank abgelegt. Falls es mehrere Exons dieser Art gibt, wird nur eins von Ihnen in der Datenbank gesichert.
Dieser Vergleich wird bei *Splicy* in einem SQL-Statement direkt auf der Datenbank ausgeführt. Durch die außerordentliche Optimierung, die Datenbanken aufweisen, kann somit dieser Vergleich in kürzester Zeit vollzogen werden.
Im gleichen SQL-Statement wird auch die Suche nach alternativen Exons mit den 3'- und 5'-Exons durchgeführt.

Für 3'- und 5'-Exons ist der Algorithmus etwas komplizierter. Da Exons am Ende eines ESTs höchstwahrscheinlich durch Sequenzierabbrüche o.ä. verkürzt sind, dürfen nicht alle, in den genomischen Koordinaten unterschiedliche, Exons als alternativ erachtet werden. Wenn ein Exon am 3' Ende die gleiche genomische Startkoordinate wie ein Mittelexon besitzt, die Stoppposition aber im Vergleich zum Mittelexon verkürzt ist, darf es nicht als alternativ angesehen werden. Vielmehr kann es als bestätigendes Exon für das entsprechende Mittelexon gelten.
Genau dies wird in Abb.10 veranschaulicht. Zuerst erfolgt eine Überprüfung auf gleiche genomische Startkoordinaten bei allen 3' Exons. Bei Übereinstimmung wird das Exon gesucht, welches die größte genomische Stoppkoordinate besitzt und somit das längste der übereinstimmenden Exons ist. Zusätzlich muss es, wie bereits erwähnt, mit den Mittelexons verglichen werden. Fall es daraufhin keine Verkürzung eines Mittelexons darstellt, wird es als Alternatives in die Datenbank aufgenommen.
Zu Beginn der 3'-Exon-Überprüfung muss der Strang, mit dem abgeglichen wurde, abgefragt werden, da im Falle des Minusstranges sich die Start- und Stoppkoordinaten vertauschen würden.

Analog zu dem in Abb. dargestellten Algorithmus erfolgt letztendlich auch die Überprüfung auf alternative 5’ Exons.

Am Ende dieses Schrittes liegen nun alle, in dieser Analyse möglichen, alternativen Exons vor, mit deren Hilfe alternative Transkripte gefunden werden können.

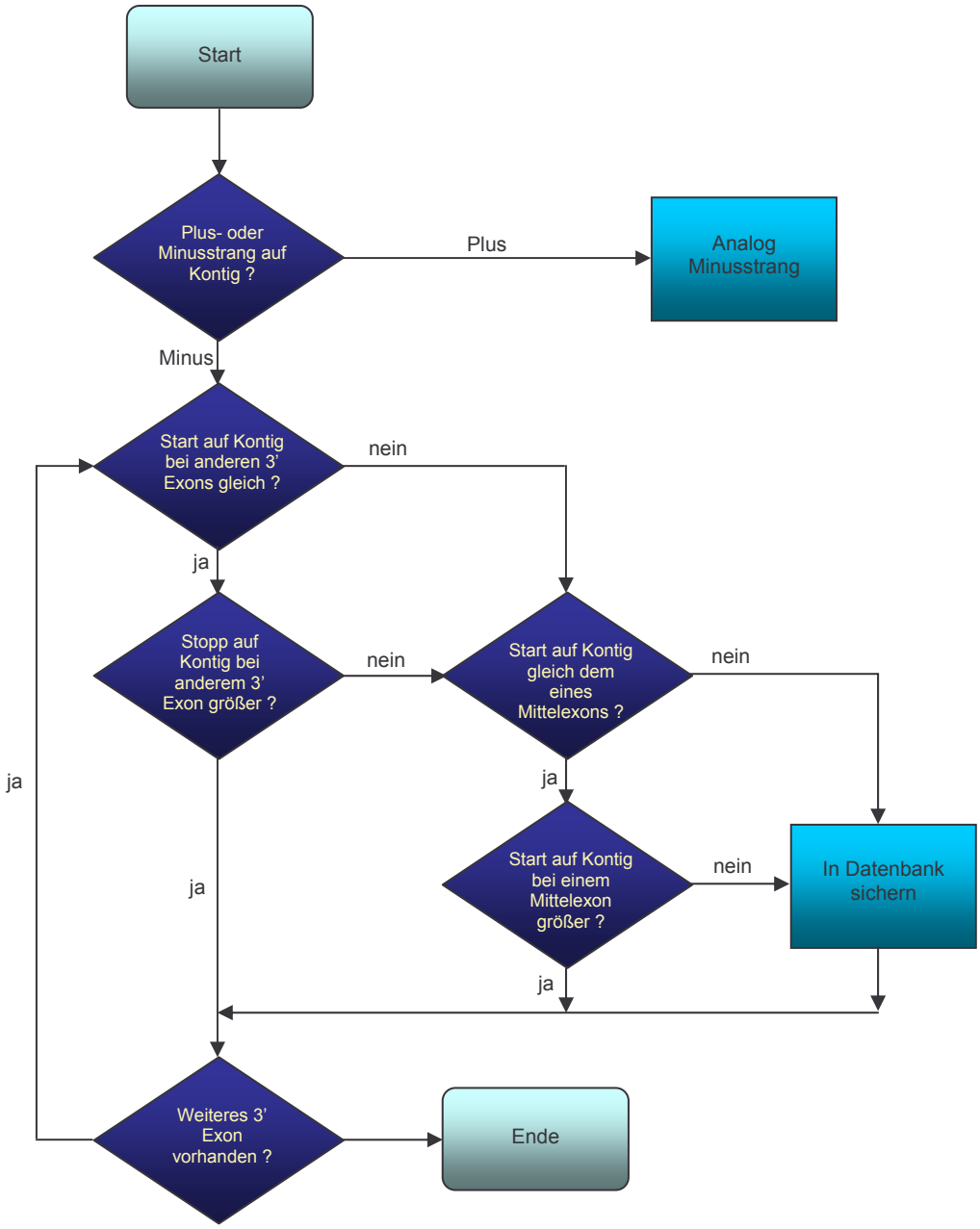

Abb.10: Algorithmus zum Auffinden alternativer 3’ Exons

4.1.2. Suche nach bestätigenden ESTs für einzelne Exons

Um dem Benutzer Auskunft über die Häufigkeit des Vorkommens eines alternativen Exons zu geben, werden in diesem Schritt bestätigende ESTs für die alternativen Exons gesucht.
Ein weiterer wichtiger Punkt, für den dieser Schritt bedeutungsvoll ist, ist die Expressionsanalyse. Dabei werden zu einem alternativ gespleißten EST die Gewebe angezeigt, in denen es exprimiert ist. Das Interesse liegt dabei besonders bei den alternativen Exons. Wenn ein EST, das alternative Exons besitzt, in malignem Gewebe vorkommt, ist dies vor allem für die Medikamentenentwicklung von Interesse. Um eine größere und damit sicherere Datenmenge zu bekommen, werden die Expressionsdaten aller ESTs, die dieses eine Exon bestätigen, mit aufgenommen. Damit wird klar, dass es nötig ist, alle bestätigenden ESTs eines Exons zu sichern.

Die Durchführung dieses Schrittes erfolgt wieder in einem SQL-Statement. Es werden die als alternativ erachteten Exons mit den Alignments der *Spidey*-Analyse verglichen. Bei Übereinstimmung und Erfüllung aller Qualitätskriterien kann das dem Alignment zugehörige EST als Bestätigung für das Exon gesichert werden.
Bei den 3'- und 5'-Exons muss dabei wieder auf die Verkürzungen geachtet werden.

4.1.3. Suche nach alternativen Transkripten

Der finale Schritt *Splicy*s ist die Suche nach alternativen Transkripten. Dafür führt *Splicy* den in Abb.11 vereinfachten Algorithmus aus. Mit Hilfe eines größeren SQL-Querys wird es ermöglicht, die ESTs geordnet nach der Anzahl ihrer Exons zu erhalten. Beginnend mit dem EST, das die meisten der in 4.3.1. beschriebenen Exons bestätigt, startet *Splicy* die Suche nach Transkripten. Dieses EST wird als erstes Transkript sowohl in der Datenbank gesichert als auch in einer Liste. In dieser Liste werden alle ESTs, die als Transkript erachtet werden, gespeichert.

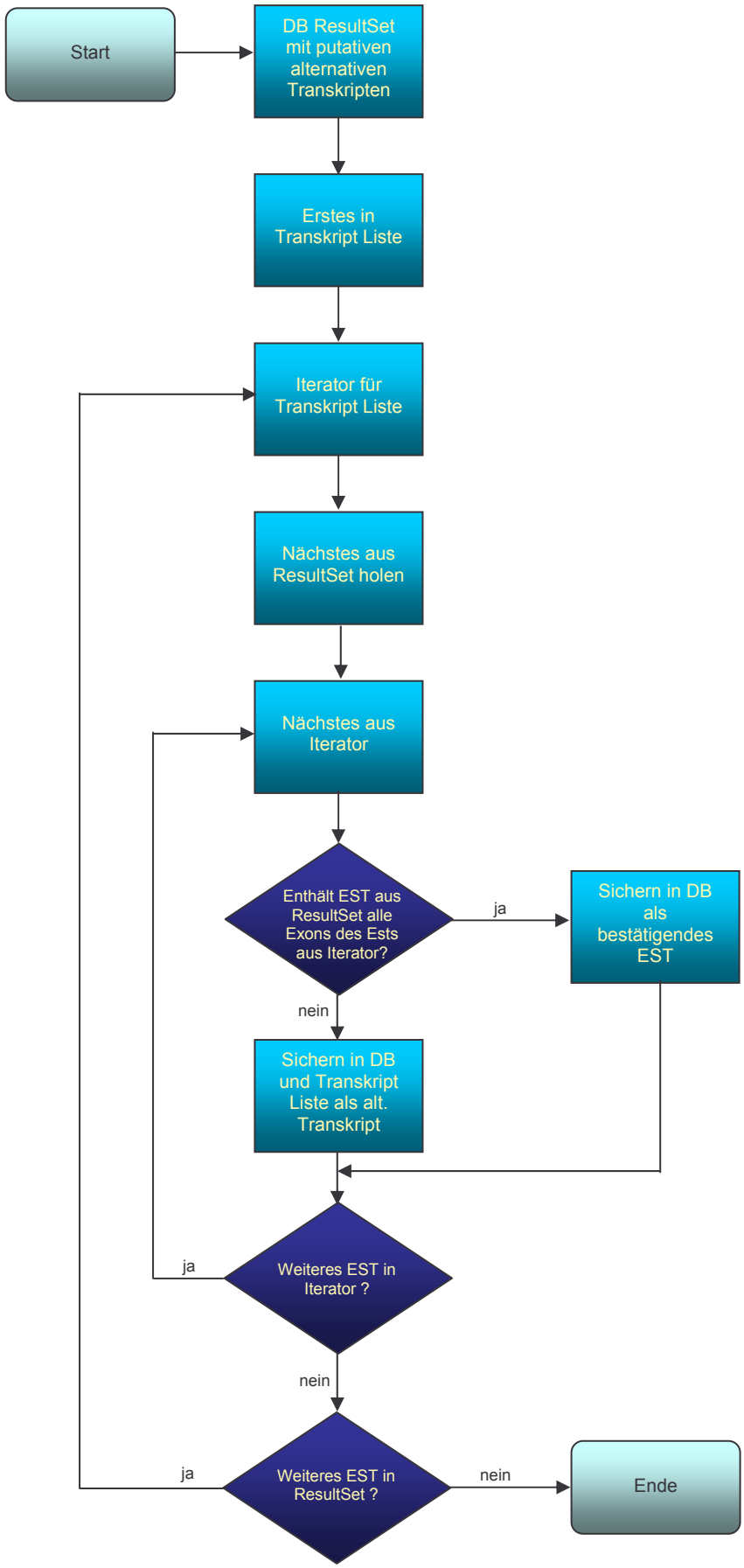

Abb.11: Algorithmus zum Auffinden alternativer Transkripte

Anschließend wird das nächste EST aus dem Datenbank-Query auf Exonebene mit dem ersten verglichen. Sobald eine Abweichung zwischen den Exons dieses ESTs und denen des als Transkript befundenen ESTs entdeckt wird, wird es als alternatives Transkript beurteilt und als solches in der Datenbank abgelegt. Falls keine Abweichung gefunden wurde, stellt dieses EST eine Verkürzung des Transkripts dar und wird als bestätigendes EST gespeichert. Wichtig bei dem Vergleich auf Exonebene ist es wieder, die 3' und 5' Exons getrennt davon zu beurteilen.

Am Ende dieses Schrittes und damit der gesamten Analyse sind alle als alternative erachteten Transkripte und deren bestätigende EST in der Datenbank gesichert.

4.2. Entwicklung eines ERD-Modells

Ein wichtiger Punkt, der vor Beginn der Implementierung abgehandelt werden musste, war der Entwurf der Datenbankstruktur. Eine performante Datenbankstruktur ist unerlässlich für jedes Programm. Da *Splicy* u.a. einige leistungskritische Algorithmen beinhaltet, war es von Bedeutung, eine sinnvolle und performante Datenbankstruktur aufzubauen.

Die Entwicklung erfolgte über ein *Entity-Relationsship Modell* (ERD). Ein ERD ist ein semantisches Datenmodell, mit dem sich die Informationsstrukturen der Wirklichkeit übersichtlich in das konzeptionelle Schema der Datenbank übertragen lassen.
Das erste in Abb. 12 ersichtliche ERD-Modell veranschaulicht die zu Beginn dieser Arbeit bei der Fa. *Xantos biomedicine AG* bereits bestehende Datenbankstruktur für biologische Datenbanken.

Um die Zusammenhänge zu verdeutlichen, erfolgt eine Beschreibung der Tabellen:

- *Any Biological Database:* Veranschaulicht eine beliebige biologische Datenbank. Jeder Eintrag bekommt eine ID und eine REFERENCEID.
- *Reference:* Beinhaltet Referenzen aller gespeicherten Datensätze verschiedener biologischer Datenbanken.
- *Referencetype:* Gibt jeder biologischen Datenbank eine eindeutige ID.
- *Entry:* Speichert Daten zu jedem Eintrag einer biologischen Datenbank.
- *Entryreference:* Enthält weitere Referenzen zu einem Eintrag in *Entry* (z.B. Referenz zu einem zugehörigen *PubMed* Eintrag).
- *Sequenceentry:* Relation zwischen *Entry* und *Sequence*.
- *Sequence:* Enthält den Verweis auf ein *Character Large OBject* (CLOB; analog *text* in Standard SQL). Sequenzen werden in der Datenbank als CLOB gespeichert, da diese mehr als die für einen VARCHAR maximalen 4000 Zeichen sichern können. Weiterhin gibt es einen Verweis auf die Taxonomie der zu speichernden Sequenz
- *Clob:* Enthält die Sequenzen als CLOB.
- *Taxonomy:* Enthält die *Taxonomy Datenbank* des NCBI.
- *Taxonomydivision:* Relation für die *Taxonomy Datenbank*

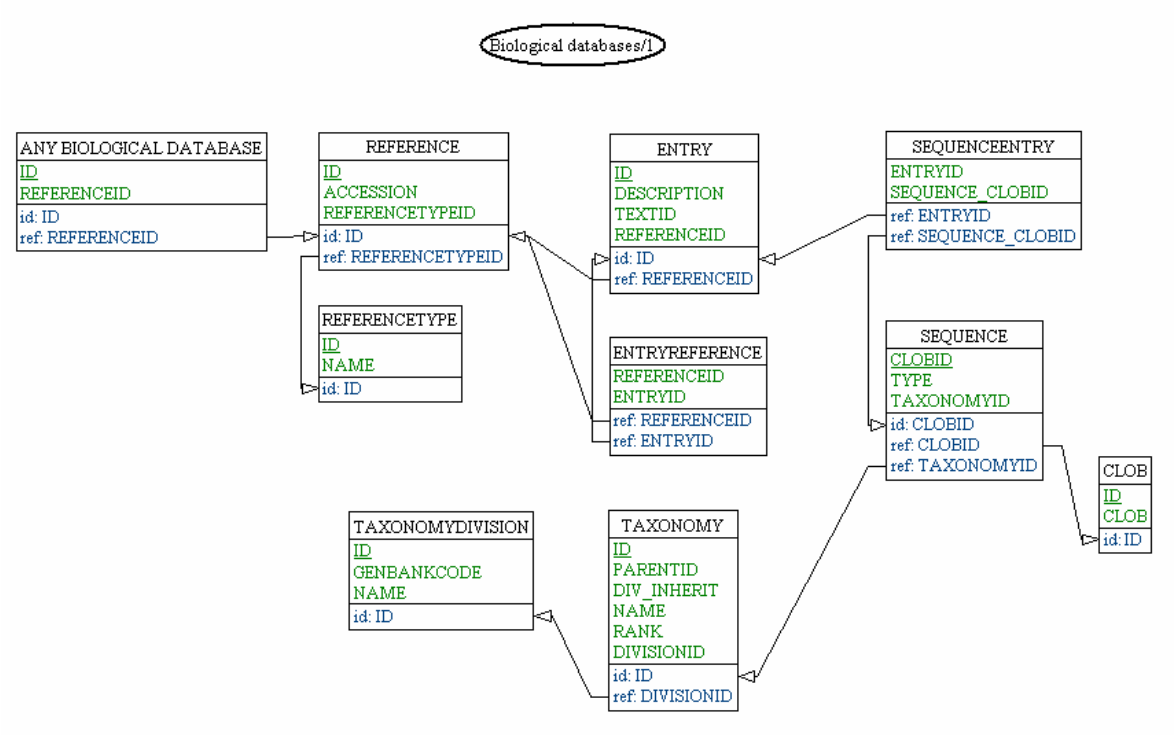

Abb.12: ERD-Modell zur relationalen Speicherung der benötigten biologischen Datenbanken

Innerhalb dieser Speicherstrukturen konnten die für *Splicy* benötigten biologischen Datenbanken gesichert werden.
Um die Daten der dbEST- und Kontig-Datenbank zu übernehmen, mussten Parser implementiert werden, die die Daten in die entsprechenden Datenbankentitäten übernehmen. Für die Implementierung wurde, wie bereits in 3.2.1. beschrieben, *javaCC* benutzt.
Die Tabellen für die Datenbanken *LocusLink* und *RefSeq* lagen zu Beginn dieser Arbeit bereits relational vor. Für *LocusLink* mussten jedoch noch einige Änderungen am Parser vorgenommen werden, um alle für *Splicy* benötigten Daten lokal vorliegen zu haben.

Die beiden folgenden Abbildungen (Abb.13 & Abb.14) repräsentieren die Datenbankstrukturen, die *Splicy* benutzt. Dabei werden in den Tabellen in Abb.13 die Daten, die *Splicy* durch die Ausgabe *Spideys* bekommt, gesichert. Die Tabellen in Abb.14 hingegen, enthalten das eigentliche Ergebnis der *Splicy*-Analyse.
In Abb. 13 sind die folgenden drei Tabellen veranschaulicht:

- *Splicy*: Enthält Informationen zu einer *Splicy*analyse. Jede Analyse bekommt eine eigene eindeutige ID zugewiesen, anhand derer sie jederzeit auffindbar ist.
- *Splicy_Est*: Diese Tabelle speichert alle Informationen zu jeweils einem EST einer Analyse. Die Informationen dazu bekommt *Splicy* hauptsächlich durch eine *Spidey*-Analyse.
- *Est_Alignment:* Dort werden alle Informationen zu jedem einzelnen Alignment eines *Splicy_Ests* gespeichert. Die Daten dazu bekommt *Splicy* wieder von *Spidey*.

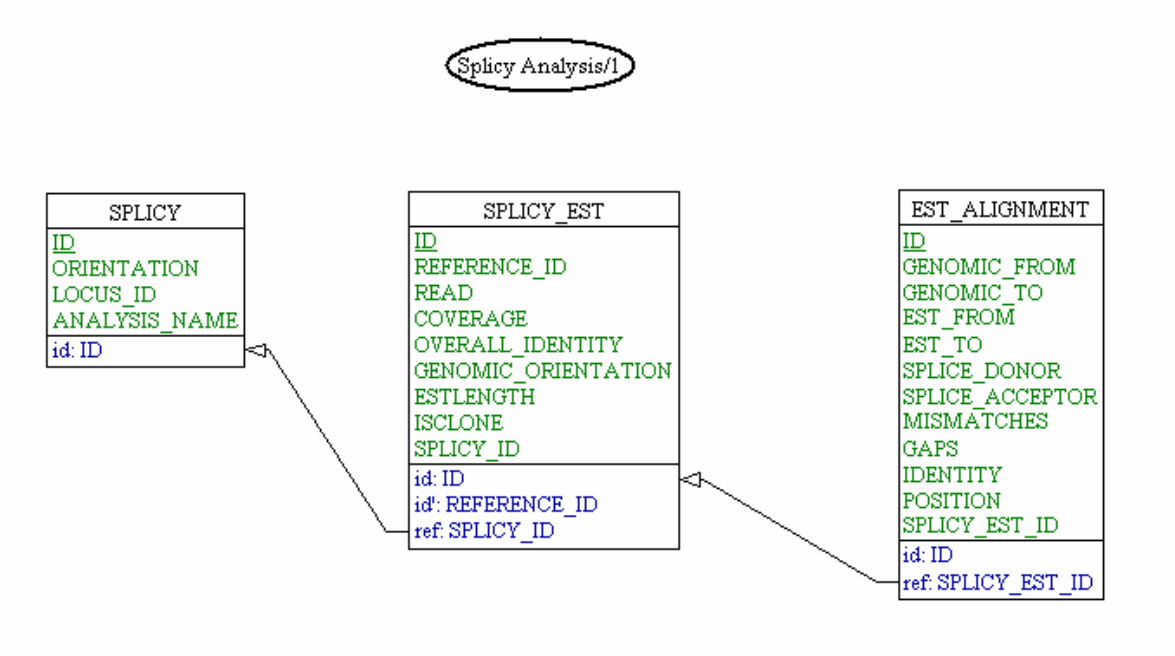

Abb.13: ERD-Modell zur Sicherung des *Spidey*outputs

Abb. 14 enthält folgende drei Tabellen:

- *Exon*: Hier werden Daten zu einem Exon gespeichert. Ein Exon ist in diesem Fall ein *EST_Alignment*, das von *Splicy* als wirkliches Exon eines als alternativen gespleißten ESTs erachtet wird.
- *Est2Exon:* Ist eine Relation zwischen den beiden Tabellen *Splicy_Est* und Exon. Dabei werden zu einem Exon diejenigen *Splicy_Ests* gespeichert, die dieses Exon bestätigen.

- *Transcript*: Hier werden die von *Splicy* als Transkript erachteten ESTs gespeichert. Dabei kann jedes Transkript mehrere bestätigende ESTs enthalten.

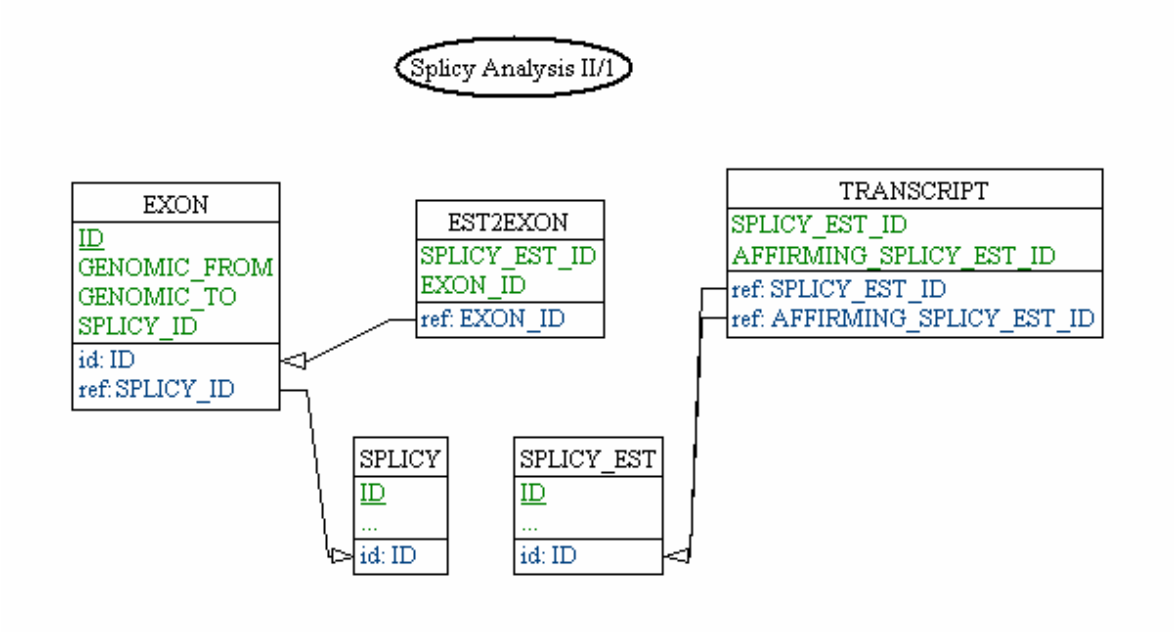

Abb.14: ERD-Modell zur Speicherung des *Splicy*-Ergebnisses

4.3. Entwicklung eines Klassenmodells

In diesem Kapitel wird das Klassenmodell beschrieben, mit dessen Hilfe *Splicy* entwickelt wurde. Dabei werden in den UML-Diagrammen nur die wichtigsten Parameter und Methoden angegeben, die zur Beschreibung des Ablaufs notwendig sind. Für genauere Informationen zu Klassen und Methoden wird auf den Javadoc und das Klassenmodell auf der beiliegenden CD verwiesen.

Die Klassen des Programms lassen sich grob in drei Teilgebiete gliedern:

- Klassen zur Steuerung des Analysenablaufes
- Klassen für die Sicherung der Daten in der Datenbank
- Klassen zur Steuerung und Erzeugung der grafischen Oberfläche

4.3.1. Klassen zur Steuerung des Analysenablaufes

Die Suche nach alternativen Spleißvarianten wird hauptsächlich von einer Klasse, *SplicingControl*, gesteuert. Sie bietet für die in 4.2. beschriebenen Startmöglichkeiten verschiedene Konstruktoren an (siehe Abb.15)

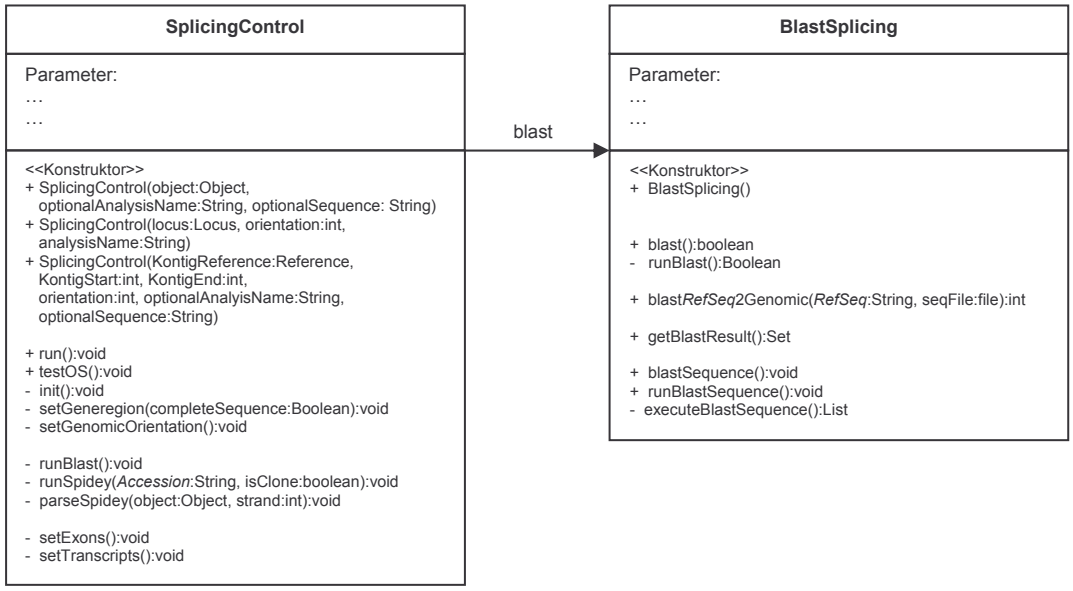

Abb.15: Klassen zur Steuerung des Analysenablaufes

Diese Klasse wurde als eigener *Thread* implementiert und enthält somit eine *run()-Methode*. Außerdem wird die Klasse per RMI auf dem Server ausgeführt. Zuständig hierfür ist die Methode *testOS()*, die eine Verbindung mit dem Server über RMI aufbaut. Vorteil dieser Implementierung ist es unabhängig vom Benutzer zu sein. Sobald dieser eine Analyse gestartet hat, hat er die Möglichkeit, andere Analysen anzusehen oder auch die Applikation zu beenden, wobei die gestartete Analyse in diesem Fall aber nicht abgebrochen wird.

Zusätzlich enthält *SplicingControl* alle Methoden, um die gesamte Analyse durchführen zu können.

Für den BLAST zu Beginn der Analyse wird in *SplicingControl* ein Objekt der Klasse *BlastSplicing* erzeugt. Dieses Objekt kann innerhalb des Programms folgende drei BLAST-Analysen ausführen:

- BLAST gegen die EST-Datenbank, um die ESTs auf den Genlocus zu mappen.
- BLAST zum Auffinden des kodierenden Stranges der DNA.
- BLAST gegen die Kontig- und *RefSeq*-Datenbank zum Auffinden der richtigen genomischen Region bei Analysenstart mit einer Sequenz.

4.3.2. Klassen für die Sicherung der Daten in der Datenbank

Für die Sicherung der Daten in der Datenbank wurden Klassen für alle wichtigen Datenbankentitäten erstellt. Diese leiten sich alle von einer Oberklasse *DatabaseObject* ab, die bereits implementiert vorlag. Diese Klasse regelt hauptsächlich die Zugriffe auf die Datenbank, sowie die Vergabe eines eindeutigen *Identifiers* für jedes Objekt.
Insgesamt wurden die vier in Abb.16 zu erkennenden Klassen implementiert.

- *Splicy*: Die Klasse *Splicy* ist für die Sicherung der zu einer Analyse gehörigen Daten verantwortlich. Dazu gehören der kodierende Strang der DNA, der Locus und der optionale Analysenname. Zusätzlich bietet *Splicy* noch die Möglichkeit, mit Hilfe der Methode *delete()* eine Analyse aus allen zugehörigen Tabellen zu löschen.

- *SplicyEST*: Sie ist die der gleichnamigen Tabelle entsprechende Klasse und sichert alle Informationen zu einem von *Spidey* analysierten EST. Weiterhin bietet diese Klasse nützliche Methoden für die Arbeit mit den gespleißten ESTs an.
  Für zwei wichtige Teilbereiche *Splicy*s hält diese Klasse Methoden bereit.
  Die beiden Methoden *getCdsGenomicStart(RefSeqGenomicStart)* und *getCdsGenomicStop(RefSeqGenomicStop)* spielen in der später beschriebenen Frame-Analyse eine wichtige Rolle. Sie bieten die Möglichkeit, die *Coding Sequence* (CDS), also die Sequenz die bei der Transkription für das entstehende Protein verantwortlich ist, des *RefSeq* Klons auf das EST abzubilden.

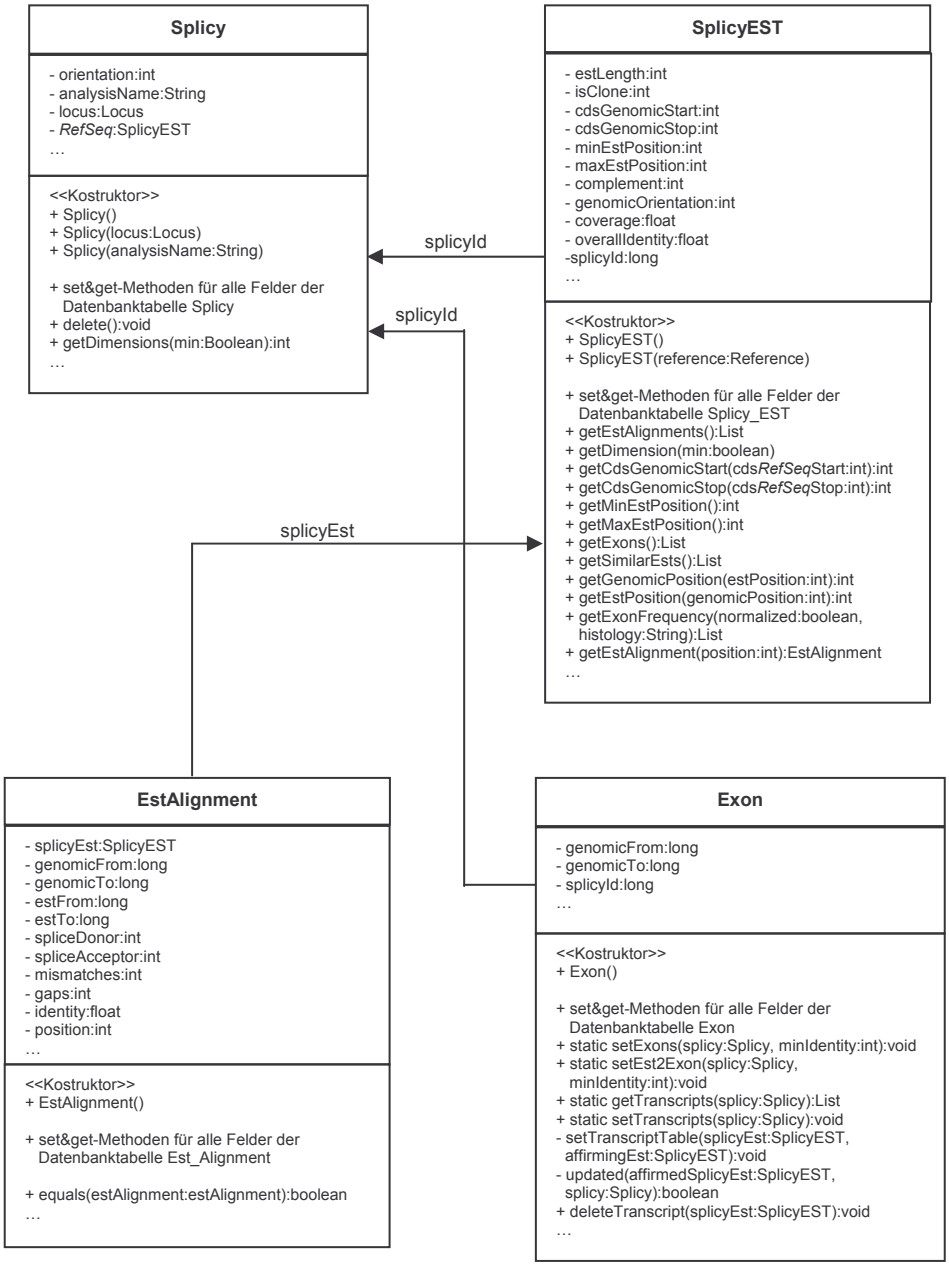

Abb.16: Klassen für die Sicherung der Daten in der Datenbank

Als weitere wichtige Methode ist noch *getExonFrequency(normalized:boolean, histology:String)* zu nennen. Dort berechnet *Splicy* die Expressionsdaten der ESTs, die das gesuchte Exon bestätigen. Dies wird in der Expressionsdatenanalyse genauer beschrieben.

- *EstAlignment*: Diese Klasse sichert die Daten aller von *Spidey* berechneten Vergleiche in der Datenbankentität *Est_Alignment*. Zusätzlich zu allen *get()- & set()-Methoden* der Parameter, bietet die Klasse noch eine Methode *equals().* Diese überschreibt die gleichnamige Methode der Oberklasse *Object* und vergleicht zwei *EstAlignments* miteinander. Wichtig dabei ist, dass sie auch 5' und 3' Alignments berücksichtigt. Damit gilt auch ein verkürzter Abgleich am Ende eines ESTs als gleich mit einem anderen aus der Mitte.

- *Exon*: Sie enthält zusätzlich zu den *get()- & set()-Methoden* die Methoden, die die Vergleichsoperationen *Splicy*s steuern und somit nach erfolgten BLAST- und *Spidey*-Analysen die Suche nach alternativen Transkripten übernehmen.
  Zu nennen wären in diesem Zusammenhang die Methoden *setExons(Splicy:Splicy, minIdentity:int)* und *setEst2Exon(Splicy:Splicy, minIdentity:int), setTranscripts(Splicy:Splicy).*
  SetExons(Splicy:Splicy, minIdentity:int) übernimmt, wie der Name bereits sagt, die Kontrolle über die Auffindung alternativer Exons. Dabei wird ein SQL-Statement ausgeführt, das die Daten berechnet und anschließend in der Tabelle *Exon* ablegt.
  Den Rest der Suche nach alternativen Spleißvarianten wie in 4.2. beschrieben übernehmen die anderen beiden Methoden.

4.3.3. Klassen zur Steuerung und Erzeugung der grafischen Oberfläche

Die komplette Steuerung der grafischen Oberfläche übernimmt die Klasse *SplicyPanel* (siehe Abb.17). Sie ist eine Unterklasse der Klasse *JFrame* und somit für sich ausführbar.
Sie bietet dem Benutzer sowohl die grafische Oberfläche zum Ausführen einer Analyse, sowie alle Funktionen zur Visualisierung der Analysenergebnisse.

Die wichtigsten Methoden, neben den Funktionalitäten, die in 4.5. besprochen werden, sind *setView(fromNavigation:boolean)* und *setTree(selectedEstAlignment:EstAlignment).*

Diese beiden Methoden steuern den Aufbau des Baumes, in dem die alternativ gespleißten ESTs angezeigt werden. Jedes EST ist dabei ein Objekt der Klasse *TranscriptPanel*, die wiederum eine Unterklasse der Klasse *JPanel* ist. Die ESTs werden in einem *JTree* angezeigt, dessen *Nodes* als *TranscriptPanels* gerendert werden. Das Rendering übernimmt dabei die Klasse *SplicyTreeRenderer*, die Unterklasse des *DefaultTreeCellRenderer* ist. Sie überschreibt die Methode *getTreeCellRendererComponent(tree:JTree, value:Object, sel:boolean, expanded:boolean, leaf:boolean, row:int, hasFocus:boolean)* und somit ist es möglich, die ESTs als *TranscriptPanels* im *JTree* zu visualisieren.

Wichtigste Methode bei der Visualisierung ist *paintComponent(g:Graphics)* der Klasse *TranscriptPanel*.

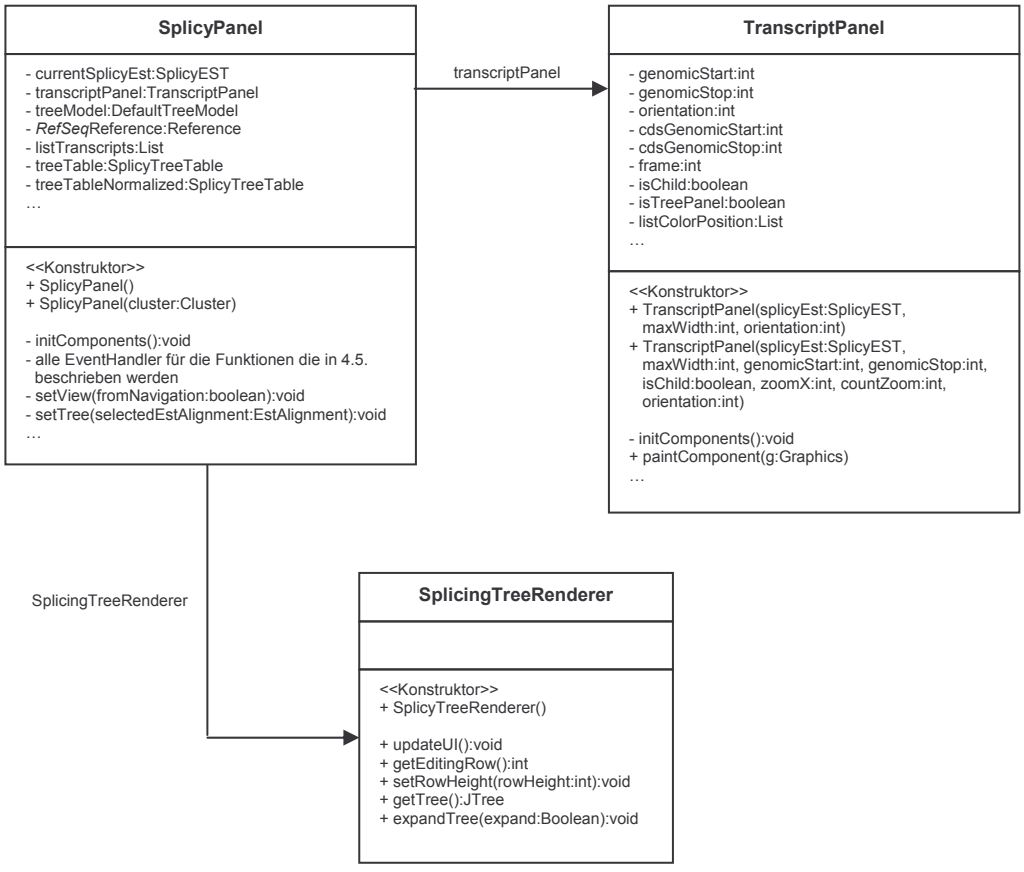

Abb.17: Klassen zur Steuerung und Erzeugung der grafischen Oberfläche

In *paintComponent(g:Graphics*) findet die komplette Berechnung statt, um ein EST mit seinen Spleißstellen auf ein Panel zeichnen zu können. Damit diese Berechnung nicht bei jedem Neuzeichnen des Panels ausgeführt wird, wird das Ergebnis der Berechnung in einer Liste, der *listColorPosition*, gespeichert. Somit ist es möglich, das Panel ohne erneute Berechnung zu zeichnen.

Eine weitere Eigenschaft des *SplicyPanels* ist es, die Expressionsdaten der ESTs in einer geeigneten Form anzuzeigen. Dafür wurde eine für *Splicy* modifizierte *JTreeTable* verwendet, die *SplicyTreeTable* (siehe Abb.18). Dieser wird ein *SplicyTreeTableModel* übergeben. Modelle übernehmen grundsätzlich die Aufgabe, die Daten zu verwalten um sie unabgängig von der Anzeige zu machen (siehe Model-View-Controller Muster [19]). Dies ist auch hier der Fall. Das *SplicyTreeTableModel* lädt die Daten aus der Datenbank in seine interne Speicherstruktur und *SplicyTreeTable* greift bei der Visualisierung der Daten darauf zurück.
Für die Speicherstruktur wurde eine eigene Klasse, die *SplicyTreeTableNode*, implementiert. Eine Instanz dieser Klasse entspricht einer *Node* des Baumes der *SplicyTreeTable*. Diese *Node* enthält alle Daten, die dann für einen Eintrag in dem Baum und der Tabelle gemacht werden sollen.

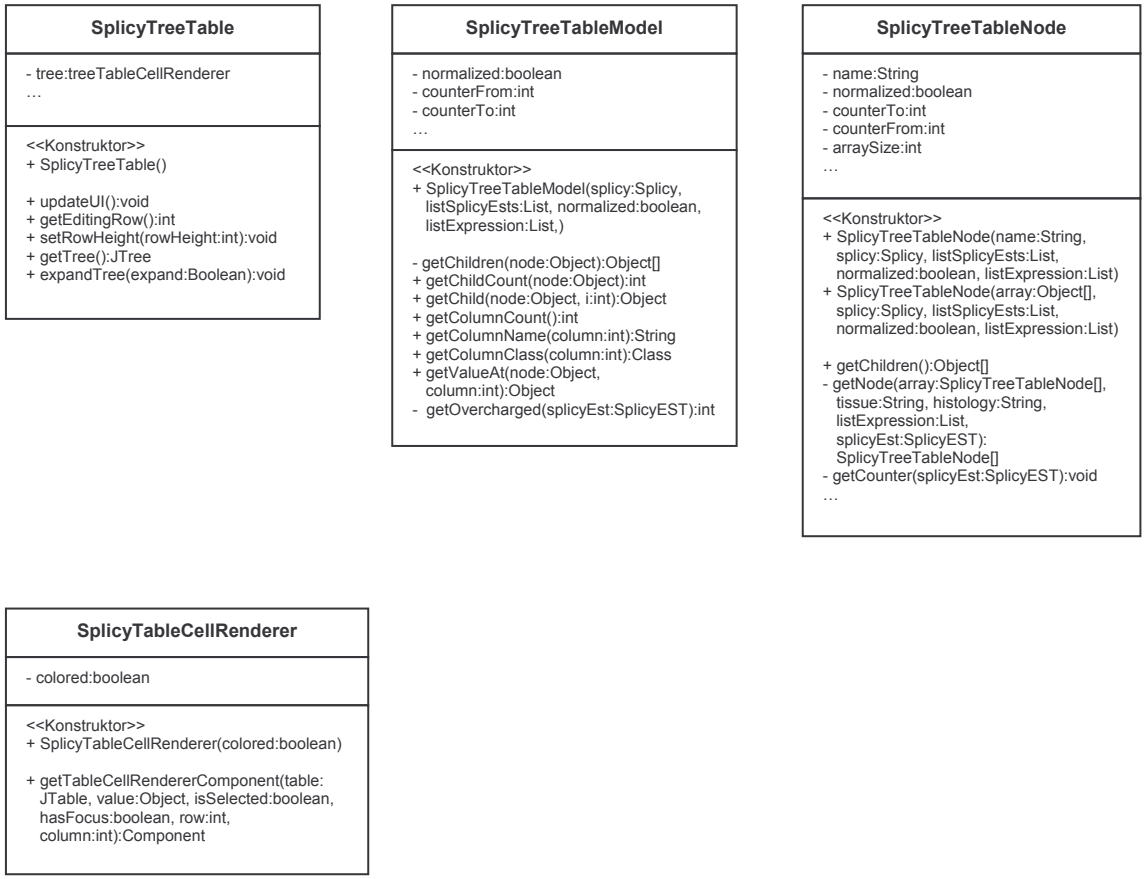

Abb.18: Klassen zur Erzeugung einer *JTreeTable*

Dafür werden in den Methoden *getChildren()* und *getNode(array:SplicyTreeTableNode[], tissue:String, histology:String, listExpression:List, SplicyEst:SplicyEST)* die Daten für jede einzelne Zeile aus der Datenbank geholt.

SplicyTableCellRenderer ist eine Unterklasse des *DefaultTableCellRenderers*. Dort wird die Methode *getTableCellRendererComponent(table:JTable, value:Object, isSelected:boolean, hasFocus:boolean, row:int, column:int)* überschrieben, um die Tabelle der *SplicyTreeTable* modifiziert zu rendern. Es ist somit möglich, die einzelnen Tabellenzellen farblich zu markieren. Dies wird in der Expressionsdatenanzeige benötigt.

4.4. Erstellung einer grafischen Oberfläche

Nachdem *Splicy*s Algorithmus zum Auffinden alternativer Spleißvarianten fertig gestellt wurde, war es nun notwendig, die Daten sinnvoll und für den Benutzer gut überschaubar darzustellen. Dazu wurde das *SplicyPanel* entwickelt.
Für die grafische Oberfläche wurde *Swing* benutzt. *Swing* ist Bestandteil der *Java Foundation Class* (JFC) und stellt eine umfangreiche Klassenbibliothek zur Erstellung grafischer Oberflächen bereit.
Um die Analyse von der Visualisierung des Ergebnisses zu trennen, wurde eine *JTabbedPane* mit zwei Reitern benutzt. Die beiden Reiter wurden folgendermaßen aufgeteilt:

- *Splicy*-Analyse Panel zum Starten einer Analyse
- Grafische Darstellung der Daten

4.4.1. *Splicy*-Analyse Panel

Damit der in 4.1. besprochene Algorithmus der *Splicy* Analyse nicht nur per Kommandozeile auf dem Server, sondern auch über eine grafische Oberfläche vom Benutzer durchgeführt werden kann, wurde ein entsprechendes Panel entwickelt.
Bereits bei der Beschreibung des Algorithmus in 4.1. wurde angesprochen, dass dem Benutzer fünf Möglichkeiten des Analysenstarts geboten werden. Der Benutzer kann zwischen *Cluster, RefSeq-Klon, LocusLink Accession, Kontig Accession* und einer beliebigen Sequenz wählen (siehe Abb.19).

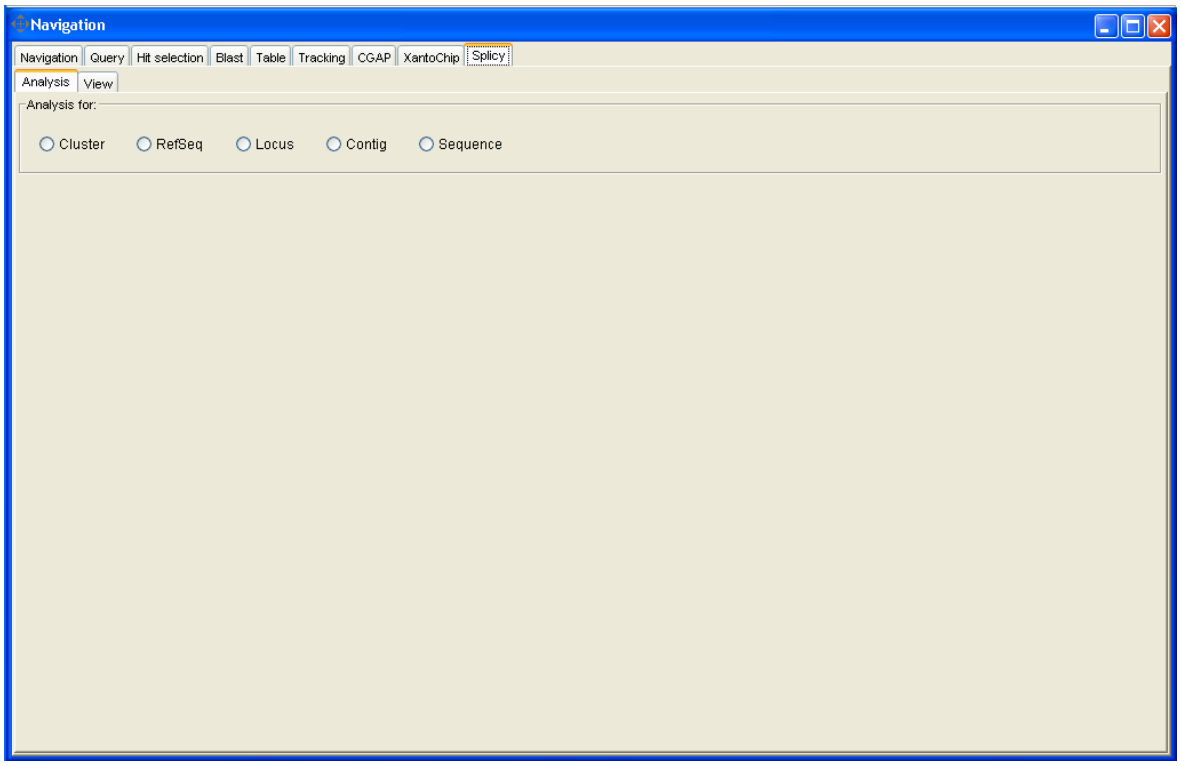

Abb.19: Auswahl des Ausgangspunktes der Analyse

Nach Auswahl des entsprechenden Ausgangspunkts wird der Benutzer nun aufgefordert, die seiner Auswahl entsprechenden Daten einzugeben (siehe Abb. 20).

Im Falle von *RefSeq* oder Cluster sind das nur die entsprechenden *Accessions* und ein optionaler Name. Dieser Name wurde eingeführt, um den Benutzern von *Splicy* das Wiederfinden Ihrer Analyse zu erleichtern. Wenn kein Name angegeben wird, kann man die Analyse anhand der eindeutigen *Accession* wieder finden.

Wenn Locus ausgewählt wurde, ist es zusätzlich zu *Accession* und optionalem Namen nötig, den kodierenden Strang der DNA anzugeben. Diese Angabe ist nötig, da *Splicy* normalerweise den kodierenden Strang anhand einer BLAST-Analyse des *RefSeq* Klons gegen die Kontigsequenz ermittelt. Der *RefSeq* Klon wird jedoch bei einer Suche nach dem Locus nicht spezifiziert und somit kann die des kodierenden Stranges nicht automatisch erfolgen.

Bei der Wahl des Kontigs als Ausgangspunkt ist es zusätzlich nötig, den Bereich auf dem Kontig anzugeben, der analysiert werden soll. Da Kontigs teilweise mehrere Millionen Basen lang sind, wäre es nicht sinnvoll, eine BLAST-Analyse

mit der kompletten Sequenz durchzuführen, da eine solche Analyse überaus zeitintensiv wäre.

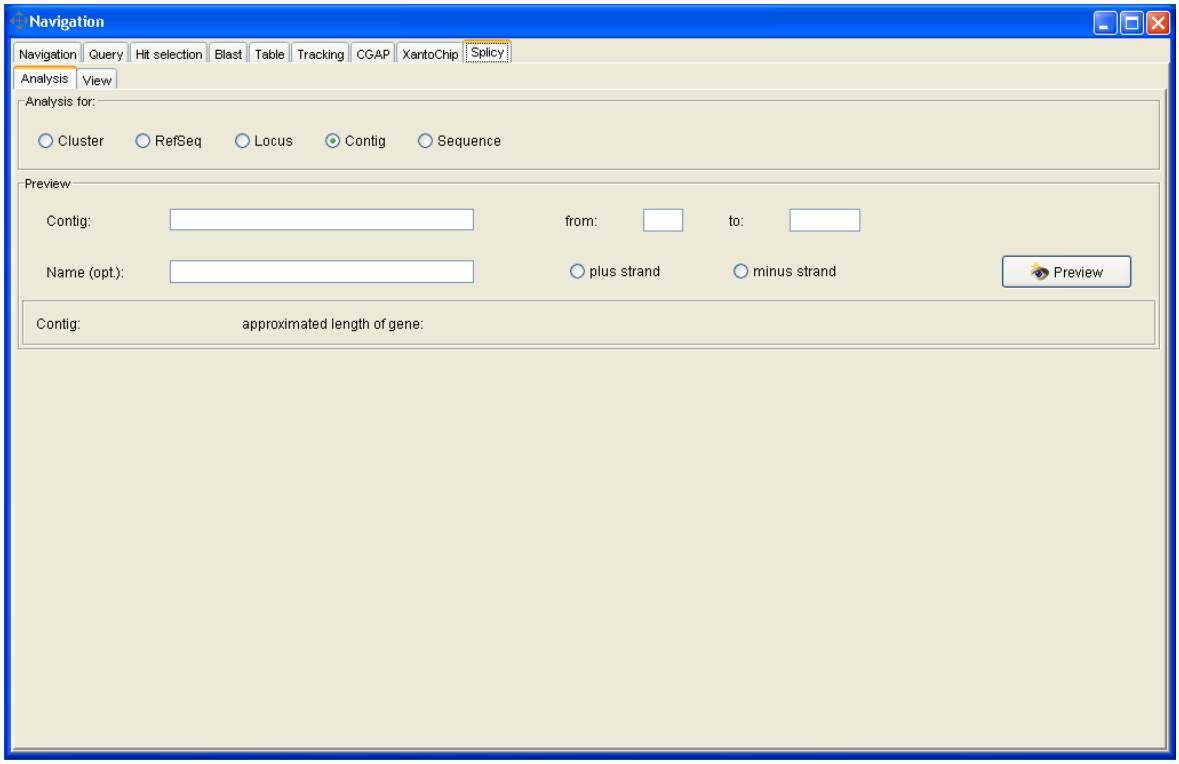

Abb.20: Eingabe der Daten zur Durchführung der Analyse

Nach erfolgter Dateneingabe und Klick auf den *Preview Button* werden die eingegebenen Daten zur Bestätigung nochmals angezeigt. Außerdem wird bei einer Länge der genomischen Sequenz von über 10.000 Nukleotiden eine Warnung ausgegeben. Diese informiert den Benutzer über die voraussichtlich lange Dauer der Analyse und empfiehlt ihm sich per E-Mail Benachrichtigung über den Abschluss der Analyse zu informieren. Diese Option ist durch eine *JCheckBox* gegeben, die nach Betätigung des *Preview Buttons* angezeigt wird (siehe Abb.21). Weiterhin werden noch vier Möglichkeiten zur Feineinstellungen der Analyse angeboten:

- Einstellung des E-Wertes: Da der E-Wert die erwartete Anzahl an Zufallsalignments repräsentiert, muss er möglichst klein gewählt werden. Um bei der Analyse mit möglichst guten Treffern arbeiten zu können, ist der Standardwert auf 10^{-50} eingestellt.
- Anzahl der Sequenzen: Mit Hilfe dieses Wertes kann der Benutzer angeben, wie viel Treffer die BLAST-Analyse maximal zurückliefern soll.

Eine größere Zahl bedeutet, dass auch die BLAST-Analyse länger dauert und mehr Sequenzen mit Hilfe *Spideys* verglichen werden müssen. Somit hat der Benutzer hier die Möglichkeit, zwischen Vollständigkeit und Geschwindigkeit zu wählen.

- Einstellung der Abdeckung (*coverage*): Die Abdeckung gibt die Anzahl der Sequenzunterschiede (*Mismatches*) addiert mit den *Gaps* im Verhältnis zu der Anzahl der Sequenzübereinstimmungen (*Matches*) in Prozent an. Sie ist also ein Maß für die Genauigkeit des Treffers.
- Einstellung der minimalen Länge der ESTs: Hier kann der Benutzer die Mindestlänge wählen, die ein EST haben muss, um in die Analyse mit aufgenommen zu werden.

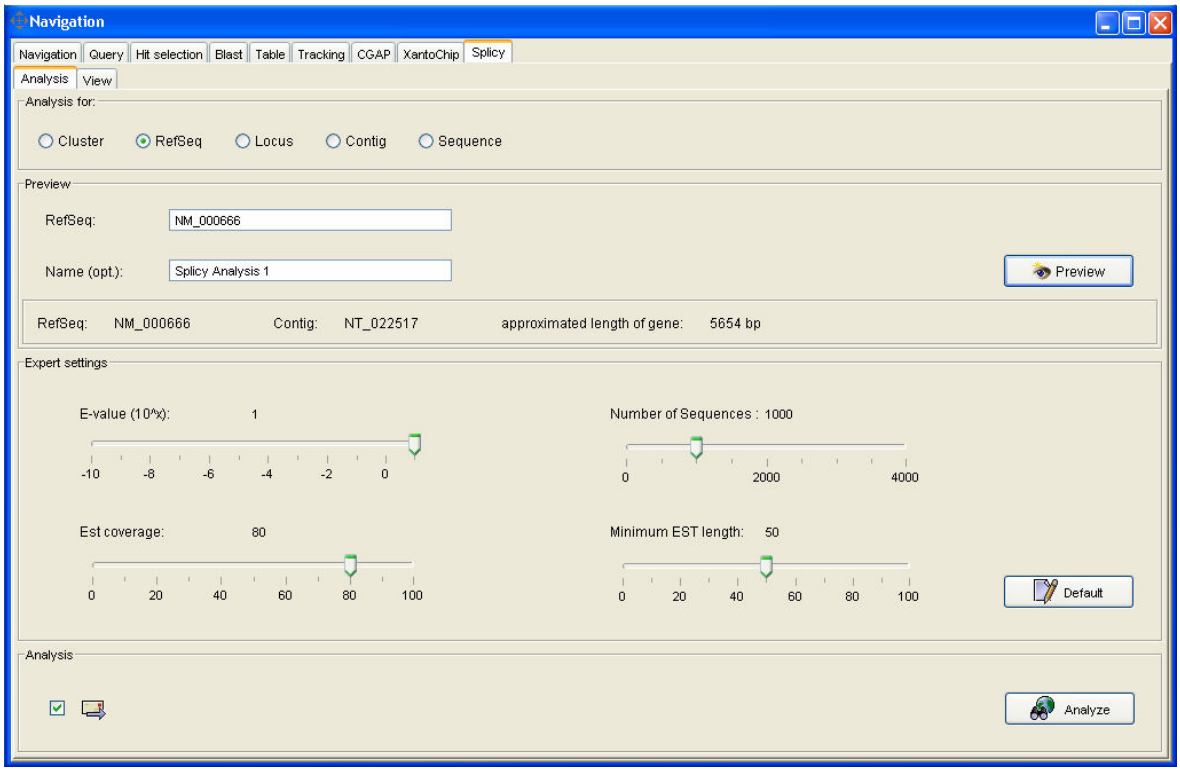

Abb.21: Feineinstellungen und Start der Analyse

Durch Klick auf den *Analyze Button* wird die Analyse in einem eigenen *Thread*, der per RMI auf dem Server läuft, gestartet. Nach dem Start wird der in 4.1. beschriebene Algorithmus auf dem Server ausgeführt.

Durch die Implementierung eines eigenen *Threads* für die Analyse wird es dem Benutzer ermöglicht, weiter mit *Splicy* zu arbeiten, ohne auf das Ende der gerade gestarteten Analyse warten zu müssen.

Für die Benachrichtigung über eine abgeschlossene Analyse bietet *Splicy* zwei Funktionalitäten an:

- E-Mail Benachrichtigung: Der Benutzer bekommt, sofern er diese Option gewählt hat, am Ende einer Analyse automatisch eine E-Mail mit den wichtigsten Daten zugeschickt.
- Bei weiterhin geöffnetem Programm wird der Benutzer durch eine Nachricht über das Ende der Analyse informiert. Um dies zu ermöglichen, wurde ein weiterer *Thread* implementiert, der jeweils in einem bestimmten Zeitabschnitt den Analyse-*Thread* auf dessen Abschluss überprüft.

4.4.2. Grafische Darstellung der Daten

Um mit den aus der Analyse verfügbaren Daten arbeiten zu können, bedarf es einer sinnvollen und ansprechenden Darstellung. Diese wird im Folgenden beschrieben.
Das Panel für die Anzeige wurde in vier Bereiche unterteilt (siehe Abb.4):

1) Suchbereich
2) Menübereich
3) Anzeige des Ergebnisses
4) Informationen zu EST

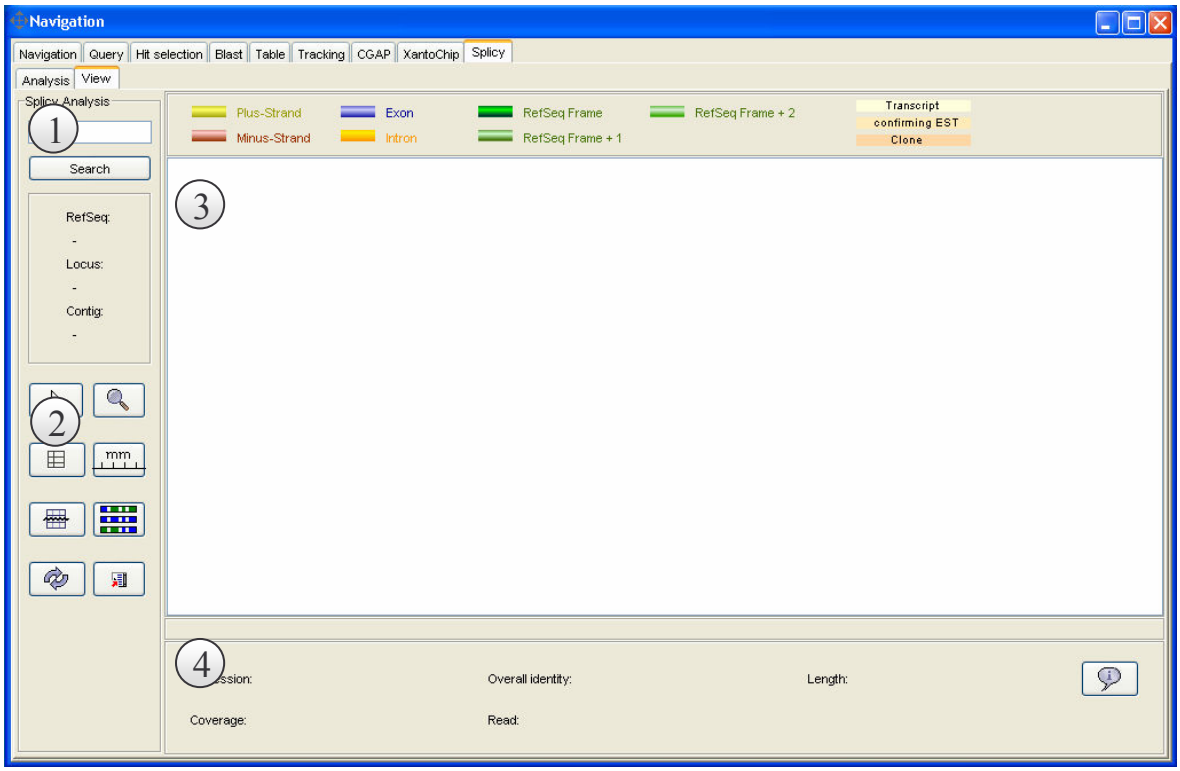

Abb.22: Übersicht über das Panel für die Darstellung mit Suchbereich (1), Menübereich (2), Anzeige des Ergebnisses (3) und Informationen zu ESTs (4)

Suchbereich

Der in Abb.22 mit 1 markierte Bereich ist der Suchbereich. Dort kann der Benutzer nach bereits beendeten Analysen suchen. Falls die Analyse gefunden wurde, werden im Feld darunter relevante Daten, wie zugehöriger *RefSeq* Klon, *LocusLink Accession* und *Kontig Accession*, angezeigt.

Menübereich

Dieser Bereich wurde in Abb.22 mit 2 markiert. Dort werden einige Buttons zur Verfügung gestellt, deren Funktionalitäten ab 4.4.3. beschrieben werden.

Anzeige des Ergebnisses

Im mit 3 bezeichneten Bereich befindet sich der Kern der grafischen Oberfläche, die Anzeige aller gefundenen alternativen Spleißmuster. Als ideale Komponente für die Anzeige wurde ein *JTree* gewählt, da alle bestätigenden ESTs als „Kinder“ der Transkripte angezeigt werden können.

Informationen zu ESTs

Im vierten Bereich werden weitere Informationen zu ausgewählten ESTs angezeigt.

Bei Auswahl einer Analyse in Bereich 1 wird das Ergebnis der Analyse in Bereich 2 visualisiert.
Splicy zeigt unabhängig von der Orientierung der genomischen Region alle Transkripte immer von 5’ nach 3’ an, d.h. das 5’ Ende eines Transkripts befindet sich stets auf der linken Seite. Als oberstes Transkript wird immer der *RefSeq*-Klon angezeigt, gefolgt von den Klonen eines *Clusters*. Diese sind auch zur Unterscheidung von den ESTs dunkler hinterlegt. Unterhalb der Klone befinden sich die ESTs, die sich als putative alternative Spleißvarianten aus der vorangegangenen Analyse finden ließen. Über den Transkripten befindet sich die genomische Region. Ein gelber Balken kündigt den Plusstrang als Kodierenden an, ein Roter den Minusstrang.
Um diese Art der Darstellung zu ermöglichen, war es notwendig, den *JTree* neu zu rendern. Dafür wurde ein eigener *JTreeRenderer* implementiert, der ein *JPanel* zurückgibt. Somit war es möglich, im *JTree* Panels anzuzeigen.

Für diese Panels wurde die in 4.3.3. beschriebene Klasse *TranscriptPanel* implementiert. In dieser Klasse finden alle Berechnungen, die für die Anzeige eines Transkripts notwendig sind, statt.
Um alle Transkripte im *JTree* vollständig anzeigen zu können, muss zu Beginn die maximale Ausdehnung der Transkripte festgestellt werden. Dazu wird das längste Exon am 5’ und am 3’ Ende ermittelt. Dies geschieht in der ebenfalls in 4.3.3. besprochenen Klasse *SplicyPanel*, die dann die Parameter an jedes *TranscriptPanel* übergibt. Über diese so gefundenen maximalen genomischen Abmessungen kann anhand der Fensterbreite ein Koeffizient errechnet werden, mit dem die Pixelkoordinaten aller Transkripte bestimmt werden können.
Zur Zeichnung jedes einzelnen Transkripts wird eine Schleife durchlaufen, in der alle Exons eines Transkripts abgehandelt werden. Jedes Exon wird anhand des zuvor berechneten Koeffizienten in Pixelkoordinaten umgerechnet und anschließend als ein *Rectangle2D* gezeichnet. *Rectangle2D* ist eine Klasse des

Abstract Windowing Toolkit (AWT), mit deren Hilfe zweidimensionale Rechtecke gezeichnet werden können.
Um die Anzeige aufzuwerten, wird anstatt einer einfachen blauen Farbfüllung des Rechtecks ein *GradientPaint* benutzt. Mit dieser Klasse des AWTs können Farbverläufe gezeichnet werden, mit denen eine dreidimensionale Darstellung erreicht wird. Nach einem Exon wird ein Intron, zur Unterscheidung etwas dünner und mit gelbem Farbverlauf gezeichnet (siehe Abb.23).

Gelegentlich kommt es vor, dass die Abdeckung (*coverage*) eines Transkripts unter 100% liegt, d.h. Teile des Transkripts konnten nicht mit der vorliegenden Kontigsequenz aligniert werden. Zur Klärung dieser Eigenheit wurde die Länge des verbleibenden Fragments überprüft. Dabei konnte festgestellt werden, dass diese unter der benötigten Wortlänge für den BLAST liegt und somit kein Alignment möglich ist. Um dem Benutzer ein solches Vorkommen zu signalisieren, wird das entsprechende Ende mit Hilfe von Pfeilen markiert (siehe Abb.23).
Um dies zu ermöglichen, wird untersucht, ob am 5‘ Ende das erste Exon mit der Position 1 im Transkript startet. Falls dem so ist, liegt dort keine Verkürzung vor. Am 3’-Ende wird die Position des letzten Nukleotids im Transkript mit der Länge des gesamten Transkripts verglichen. Im Falle einer Abweichung handelt es sich um eine Verkürzung am 3’-Ende, welche mit einem Pfeil in die entsprechende Richtung gezeichnet wird.
Um die Exons verschiedener Transkripte auf Basis genomischer Koordinaten schnell miteinander vergleichen zu können, wurde ein *Mouseover* implementiert, der die Positionen des Exons auf der genomischen Region und auf dem EST selbst darstellt (siehe Abb.23).
Dieser Event wurde im *TranscriptPanel* implementiert. Anhand der Mausposition und dem bereits berechneten Koeffizienten kann auf die genomischen Koordinaten rückgeschlossen werden. Durch einen Vergleich der Exondaten mit denen der Mausposition können schließlich die Koordinaten des gesuchten Exons als *ToolTipText* angezeigt werden.

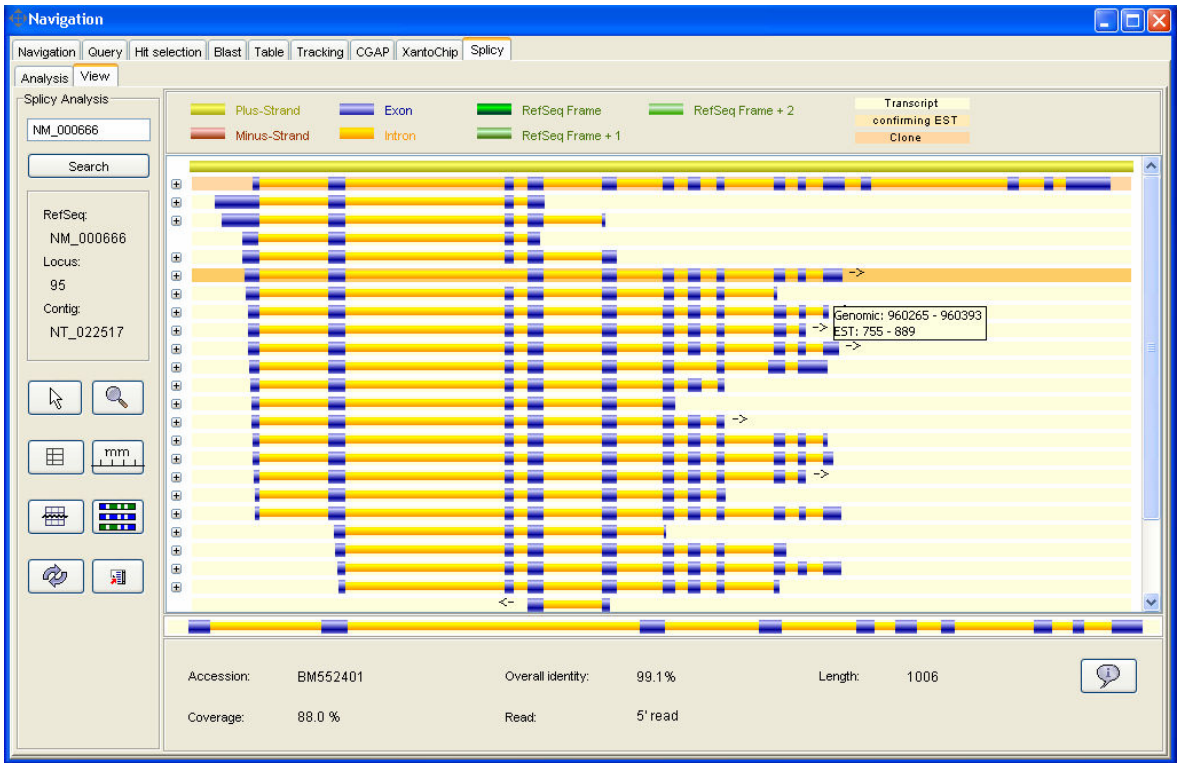

Abb.23: Darstellung der alternativen Spleißmuster

Um genauere Informationen zu einem Transkript zu erhalten, muss dieses im Baum ausgewählt werden. Daraufhin bekommt der Benutzer diese im dritten Bereich des Panels angezeigt (siehe Abb.22 bzw. 23).

Angezeigt werden die *Accession*, die prozentuale Sequenzübereinstimmung über die gesamte Länge, die Länge des ESTs und die Abdeckung. Alle diese Angaben wurden direkt aus der Ausgabe von *Spidey* übernommen. Das Feld *Read* zeigt an, auf welche Art das EST sequenziert wurde, ob es ein 5'- oder 3'-Read war.

Zusätzlich wird in dem Bereich nochmals das *TranscriptPanel* angezeigt. Dort ist es aber nicht wie im Baum an den maximalen genomischen Koordinaten bemessen, sondern es bekommt die maximale Fenstergröße für sich selbst zugewiesen.

Weitere Daten zu einem EST bekommt man bei Klick auf den Informationsbutton (siehe Abb.24).

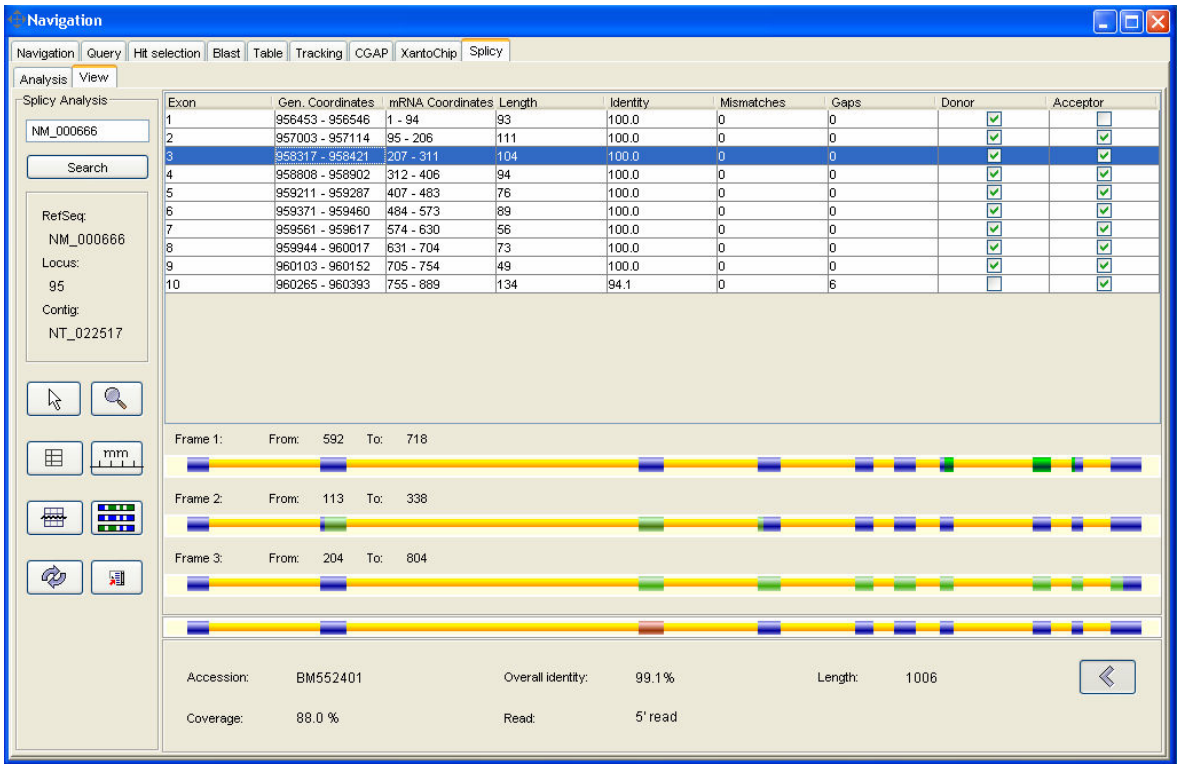

Abb.24: Detailinformationen zu einem Transkript

Hier werden alle aus *Spideys* Ausgabe erhältlichen Daten in einer *JTable* zusammengefasst. Wenn in der Tabelle ein Exon ausgewählt wird, wird dies im *TranscriptPanel* farblich markiert (siehe Abb.24).

Dazu wird im *TranscriptPanel* ein Wert gesetzt, der die Nummer des gewählten Exons enthält. Falls der Wert gesetzt ist, wird für das entsprechende Exon bei der Zeichnung des Panels die Farbe auf rot gesetzt. Weiterhin kann bei Rechtsklick auf ein Exon das entsprechende Alignment angezeigt werden. Die Implementierung dieser Funktionalität wird in 4.4.3.7. noch ausführlicher behandelt.

4.4.3. Buttons im Menübereich

4.4.3.1. Mauszeiger und Lupen-Button

Abb.25: Zooming Funktion (Zoom Button)

Damit die einzelnen Exons besser verglichen werden können, vor allem wenn die Transkripte auf einen großen genomischen Bereich verteilt sind, wurde eine *Zooming*-Funktion eingerichtet (siehe Abb.25). Die beiden Buttons *Mauszeiger* und *Lupe* repräsentieren diese Funktion. Bei Klick auf den Lupenbutton wird ein *Event* ausgelöst. Dieser ändert dabei den Mauszeiger in eine Lupe. In diesem Modus wird bei Klick in den Baum der gewählte Ausschnitt um das Doppelte vergrößert. Ein Klick auf die rechte Maustaste hat das Gegenteil zur Folge und verkleinert den Ausschnitt wieder.

Bei Klick auf den Mauszeigerbutton wird die *Zooming*-Funktion ausgeschaltet und der Mauszeiger wieder in den Default-Mauszeiger umgewandelt. Um die *Zooming*-Funktion zu implementieren, wurden einige Änderungen im *SplicyPanel* und *TranscriptPanel* vorgenommen. Die Vorgabe besagt, dass bei Klick auf eine bestimmte Stelle im Baum, genau dieser Ausschnitt vergrößert werden soll.

Um dies zu ermöglichen, wird beim Klick eine Variable, die die Anzahl der bereits getätigten Vergrößerungen enthält, inkrementiert und eine zweite Variable mit der x-Koordinate der momentanen Mausposition belegt. Anschließend wird der Baum neu gezeichnet.
Beim Zeichnen des Baumes wird jedem *TranscriptPanel* die Größe des Anzeigefensters übergeben und die Größe des *TranscriptPanels* auf diese Größe gesetzt. Beim *Zooming* wird diese Größe verdoppelt. Damit werden die Positionen der Exons und Introns neu berechnet und das *TranscriptPanel* doppelt so groß wie vorher dargestellt.
Um nun noch an die vergrößerte Stelle zu gelangen, wird die *JScrollPane* an die Koordinaten der Position des Mauszeigers beim Klick bewegt.
Bei einer weiteren Vergrößerung wird oben genannte Variable weiter inkrementiert und die Größe des *TranscriptPanels* somit vervierfacht. Bei einem Rechtsklick wird obige Prozedur umgekehrt, die Variable um eins erniedrigt und somit die vorige Vergrößerung wieder angezeigt.

4.4.3.2. Koordinaten Button

Bei Klick auf den Koordinaten-Button wird ein Lineal mit den genomischen Koordinaten angezeigt (siehe Abb.26).
Dazu wurde die *paint(Graphics g)-Methode* des Panels, auf dem der *JTree* gezeichnet wird, überschrieben. In dieser Methode werden ähnlich wie im *TranscriptPanel* die maximalen genomischen Koordinaten abgefragt. Diese werden anschließend in neun gleiche Teile zerlegt und somit können mit Hilfe der AWT-Klasse *Line2D* 10 vertikale Linien gezeichnet werden. Zusätzlich wird am oberen Rand ein Lineal mit den Koordinaten gezeichnet.
Damit beim Scrollen über das Panel das Lineal mit den Koordinaten nicht verschwindet, wurde auf die *JScrollPane* ein Event gelegt, der bei jedem scrollen die *repaint()-Methode* des Panels aufruft und somit die Koordinaten neu zeichnet.

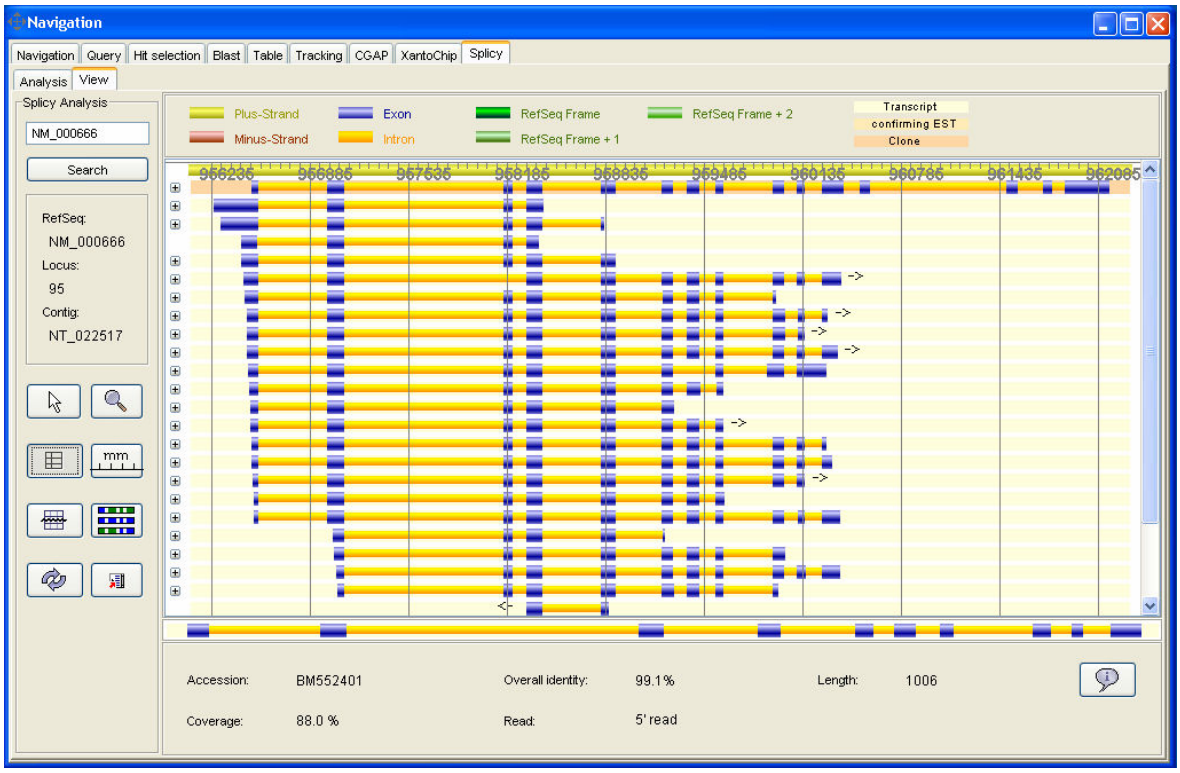

Abb.26: Anzeige der genomischen Koordinaten (Koordinaten Button)

4.4.3.3. Positions-Button

Diese Funktion ist ähnlich der in 4.4.3.2. beschriebenen Koordinaten-Funktion. Bei Auswahl der Funktion wird auf dem Panel an der momentanen Mausposition eine vertikale Linie gezeichnet. An dieser Linie wird die Position des Mauszeigers auf genomischer Ebene angezeigt (siehe Abb.27).

Um dies zu ermöglichen, wurde ein *MouseMoved-Event* für den Baum implementiert. Dort wird ständig die aktuelle Position des Mauszeigers neu gesetzt und anschließend die *repaint()-Methode* des darunter liegenden Panels aufgerufen. Aus den maximalen genomischen Koordinaten und der Fensterbreite wird wieder ein Koeffizient berechnet, mit Hilfe dessen und der gerade aktuellen X-Position die genaue genomische Koordinate berechnet werden kann. An Position des Mauszeigers wird dann eine vertikale Linie gezeichnet und die berechnete genomische Koordinate angezeigt.

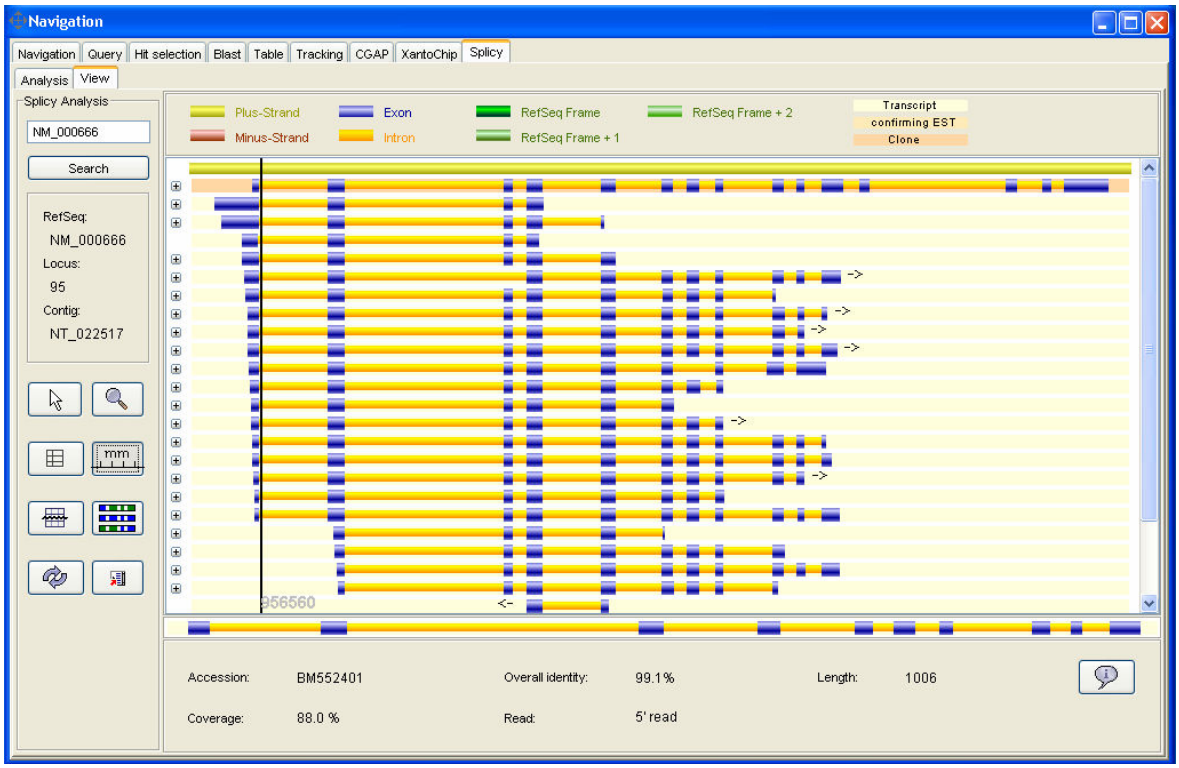

Abb.27: Positionsbestimmung (Positions Button)

4.4.3.4. Der „no-leafs“ Button

Da beim Auffinden einer alternativen Spleißvariante, die durch kein anderes EST bestätigt werden kann, die Wahrscheinlichkeit eines Sequenzier- oder anderen Fehlers recht hoch ist, wurde eine Funktion implementiert, die solche Treffer ausblendet (siehe Abb. 28).

Diese Implementierung ist recht unkompliziert: Es wird durch alle alternativen Transkripte iteriert und diejenigen aus dem Baum gelöscht, die keine weiteren „Kinder“ enthalten.

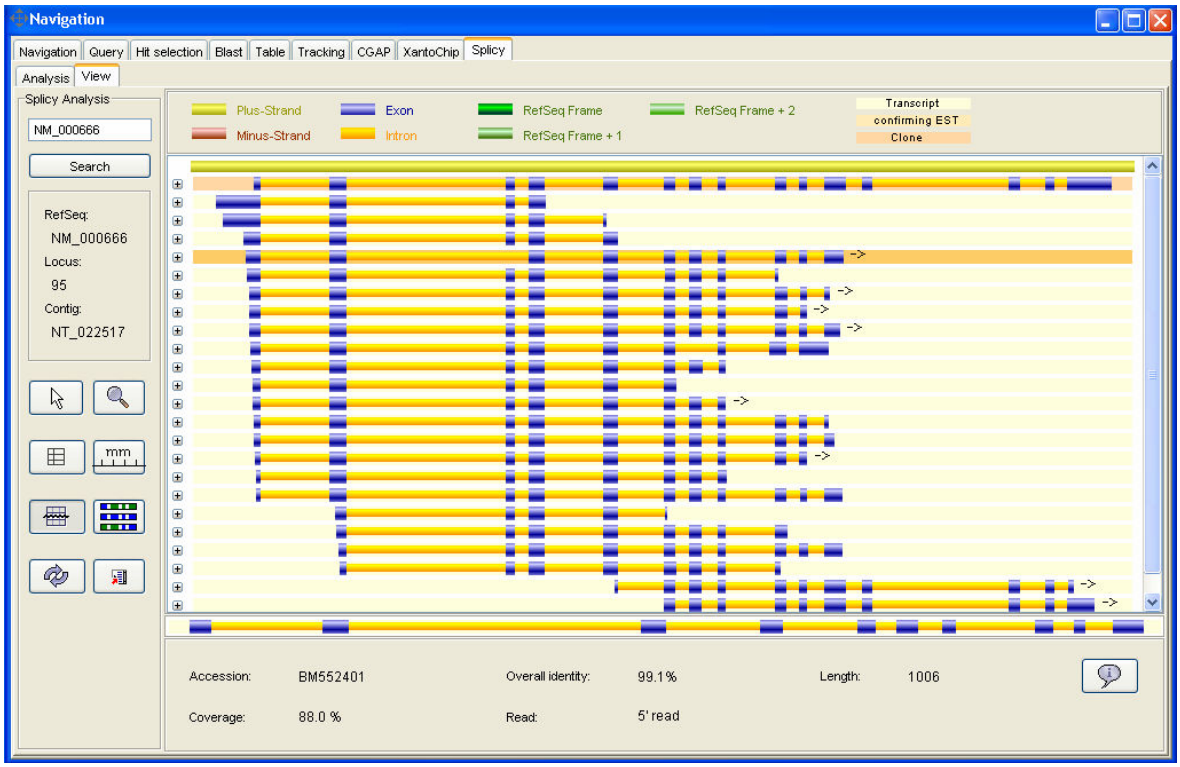

Abb.28: Anzeige nur bestätigter Transkripte (no-leafs Button)

4.4.3.5. Der Frame-Analyse Button

Eine der wichtigsten und auch in der Implementierung aufwendigsten Funktionen war die Leseraster / -rahmen-Analyse.

Aufgabe dieser Funktionalität ist es, die Auswirkungen der Spleißvariabilität auf das translatierte Protein zu bestimmen. Der Bezugspunkt für diese Analyse ist der *RefSeq*-Klon. Alle folgenden Transkripte werden im Bezug auf das translatierte Protein mit diesem Klon verglichen. Deshalb kann diese Funktionalität, bei Analysen, die anhand einer *LocusLink Accession* oder *Kontig Accession* gestartet wurde, nicht genützt werden. Da die *Coding Sequence* (CDS) des *RefSeq*-Klons als einzige einen bestätigten Offenen Leserahmen (*Open Reading Frame*; ORF) besitzt, war dies der einzig mögliche Bezugspunkt.

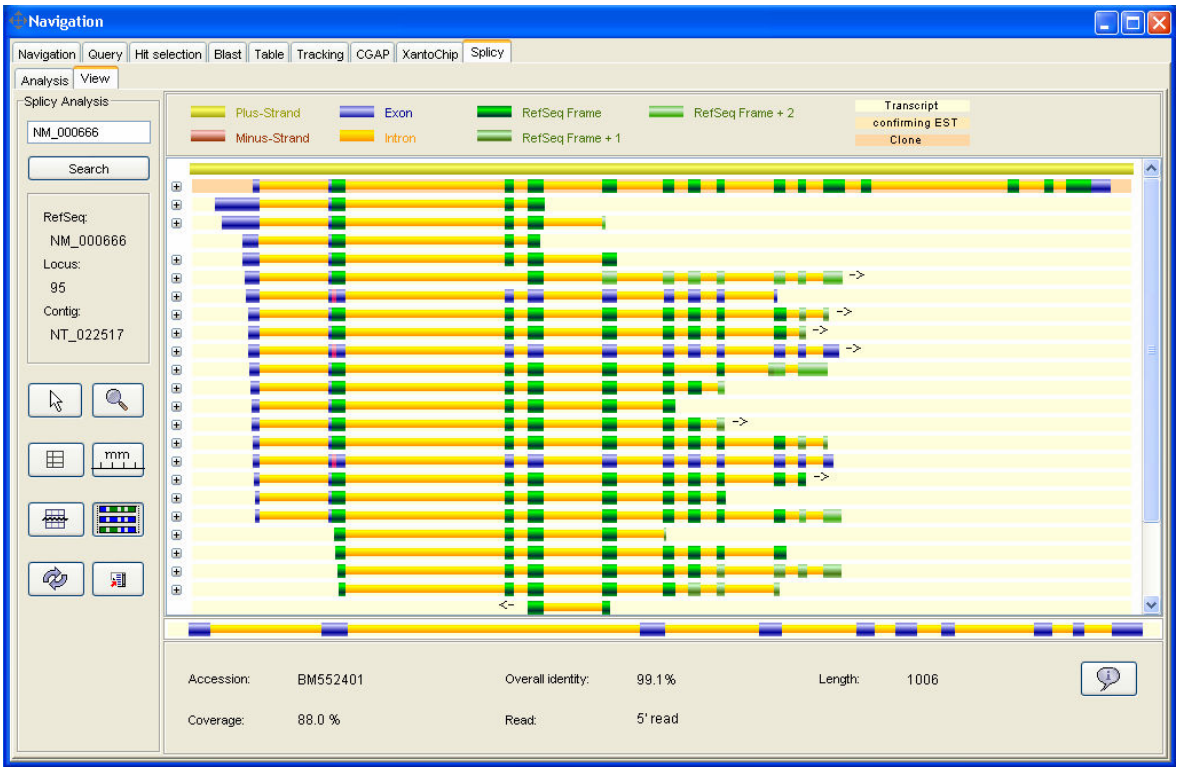

Abb.29: *Frame*-Analyse (Frame-Analyse Button)

Bei der *ORF*-Analyse gibt es vier Punkte zu berücksichtigen:

a) Ist an der Startposition des *RefSeq* Klons auch bei den anderen Transkripten ein Startcodon enthalten?
b) Trifft a) zu, sind bei alternativen Spleißmustern Leseraster-Verschiebungen anzuzeigen.
c) Trifft a) nicht zu, ist es interessant, ob weiter vorne im Transkript im gleichen Leseraster zum *RefSeq* Klon noch ein Startcodon enthalten ist.
d) Gibt es das Exon bei a) im Transkript überhaupt nicht, wird im Falle von weiteren Exons 3' UND 5' keine *ORF*-Analyse gemacht, da dann der Bezug zum *RefSeq*-Klon nicht hergestellt werden kann. Im Falle von Exons nur in Richtung 3' wird eine normale *ORF*-Analyse wie bei b) gemacht.

Implementiert wurden oben genannte Punkte in der *paintComponent(Graphics g)-Methode* des *TranscriptPanels* und zwar so, dass die Analyse nur einmal durchgeführt und anschließend in einem Array gespeichert wird, aus dem schließlich das Transkript gezeichnet wird. Ohne dieses Array würde bei jeder

Neuzeichnung des Panels immerzu die Analyse durchgeführt, was den Ablauf des Programms erheblich beeinträchtigen würde.

Für Punkt a) wird zum ersten Mal die eigentliche Sequenz des Transkripts benötigt. Die Sequenzen aller ESTs werden im *SplicyPanel* aus der Datenbank geladen und anschließend an jedes einzelne *TranscriptPanel* übergeben. Dies geschieht in einem eigenen *Thread*, nachdem der Baum das erste Mal angezeigt wurde. Somit werden die Sequenzen im Hintergrund geladen und damit die Performance erhöht.
Bei Aufruf der *ORF*-Analyse wird zu jedem Transkript die Position auf der genomischen Region im Bezug auf den Startpunkt der CDS des *RefSeq* Klons berechnet. Anschließend wird überprüft, ob sich an dieser Stelle ein Startcodon befindet. Im positiven Fall wird Punkt b) abgearbeitet, im anderen Fall ist an dieser Stelle ein rotes Rechteck zu zeichnen, um dem Benutzer zu signalisieren, dass an dieser Stelle das Startcodon fehlt. Anschließend wird Punkt c) abgearbeitet.

Um Punkt b) zu realisieren wurde eine Funktion *setFrame(int frame)* implementiert.
Diese simuliert einen Wert, der nur drei Werte, nämlich null, eins und zwei annehmen kann. Auf die Zwei folgt wieder die Null.
Nach Vergleich eines Exons des Transkripts mit dem entsprechenden des *RefSeq* Klons wird diese Funktion mit der entsprechenden Rasterverschiebung aufgerufen. Angenommen das Exon des Transkripts ist 5’ gegenüber dem des *RefSeq* Klons verlängert, dann wird die Differenz modulo 3 genommen (um Tripletts zu simulieren) und der Rest der *setFrame(int frame)* Methode übergeben. Somit wird der aktuelle Leserahmen gesetzt. Für die anderen Fälle, Exon ist 5’ verkürzt, 3’ verlängert oder 3’ verkürzt, gilt dies analog, wobei bei Verkürzungen gegenüber dem *RefSeq*-Exon der Rest negiert gesetzt wird.
Wenn sich nun ein alternatives Spleißmuster bei einem Exon als Leserasterverschiebung herausstellt, werden alle nachfolgenden Exons bis zur nächsten Verschiebung in einer anderen Farbe gezeichnet. Dunkelgrün bedeutet dabei Raster null, d.h. im gleichen Raster wie der Leserahmen des *RefSeq* Klons, helleres grün Raster eins und hellgrün Raster zwei (siehe Legende in Abb.29).

Bei Zutreffen von Punkt c) wird durch die Sequenz in Dreierschritten nach vorne iteriert und nach einem weiteren Startcodon gesucht. Dabei wird das weitest 5' im gleichen Leseraster liegende Startcodon gesucht, bei dem zwischendurch kein Stopcodon auftrtitt.

4.4.3.6. Merger-Button

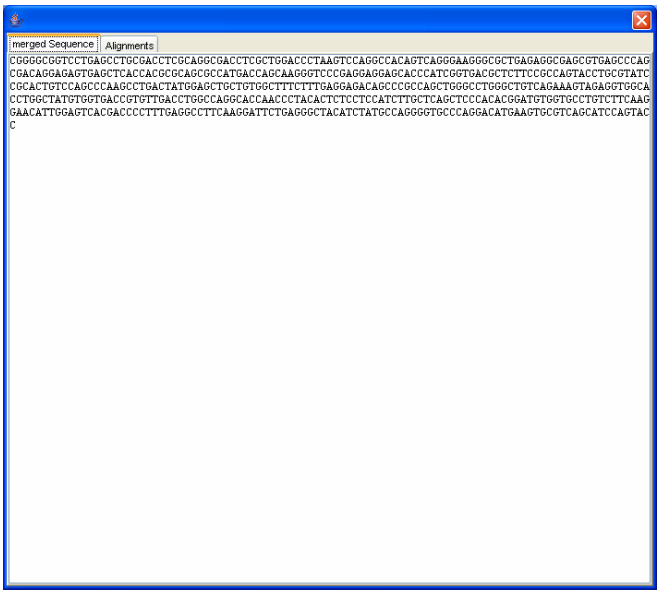

Abb.30: resultierende Sequenz (Merger Button)

Mit dieser Funktion soll es dem Benutzer ermöglicht werden, mehrere ESTs zu einem Transkript zusammenzufügen und die resultierende Sequenz zu erhalten.
Dazu wird das Programm *Merger* von EMBOSS benutzt.
Das Programm läuft über RMI auf dem LINUX Server. Um die resultierende Sequenz zu bekommen, werden *Merger* zwei Sequenzen übergeben und anschließend das Programm ausgeführt.
Als Ergebnis bekommt man die Sequenz, sowie die entsprechenden Sequenzabgleiche der beiden Sequenzen, die dann in einem *JDialog* angezeigt werden (siehe Abb.30 und 31).

Abb.31: Sequenzabgleich nach Zusammenfügen der Sequenzen (Merger Button)

4.4.3.7. Anzeige der Sequenzabgleiche

Dem Benutzer soll es ermöglicht werden, die Sequenzabgleiche der Exons eines Transkripts mit der genomischen Region betrachten zu können.

Diese Abgleiche erhält *Splicy* bereits bei der *Spidey*-Analyse. Um jedoch keinen unnötigen Platz in der Datenbank zu belegen, wurde eine Klasse *Spidey* implementiert, die über RMI auf dem Server läuft. Wenn ein Benutzer auf einem Transkript im Baum die rechte Maustaste drückt, wird *Spidey* auf dem Server ausgeführt und er kann sich den Abgleich somit dynamisch anzeigen lassen, ohne dass es vorher in der Datenbank abgelegt werden musste.

Die *Spidey*-Analyse wird in einem eigenen *Thread* ausgeführt und bei Abschluss das Ergebnis in einem *JDialog* angezeigt.

Dieselbe Funktion wurde auch für die in 4.4.2. *JTable* implementiert, wobei hier bei Rechtsklick auf ein Exon in der Tabelle nur das Alignment für das einzelne Exon angezeigt wird.

4.4.3.8. Darstellung der Expressionsdaten

Diese Funktionalität ist mit der *ORF*-Analyse eine der wichtigsten Punkte der grafischen Oberfläche.

Hierbei soll die Auswirkung des alternativen Spleißens auf den Organismus verdeutlicht werden. Wie bereits in 2.3. angesprochen, werden in der EST-Datenbank die *Libraries* verzeichnet, in denen die ESTs vorkommen. Damit erhält man Aufschluss über die Gewebe und die Zeit des Entwicklungsstadiums, in denen die ESTs exprimiert sind. Außerdem bekommt man Aussagen über die Histologie, ob das EST in gesundem oder malignem Gewebe gefunden wurde. Wenn ein EST als Spleißvariante zu einem anderen erkannt wird, kann damit der Unterschied in der Expression festgestellt werden, was vor allem für die Medikamentenentwicklung von Bedeutung ist.

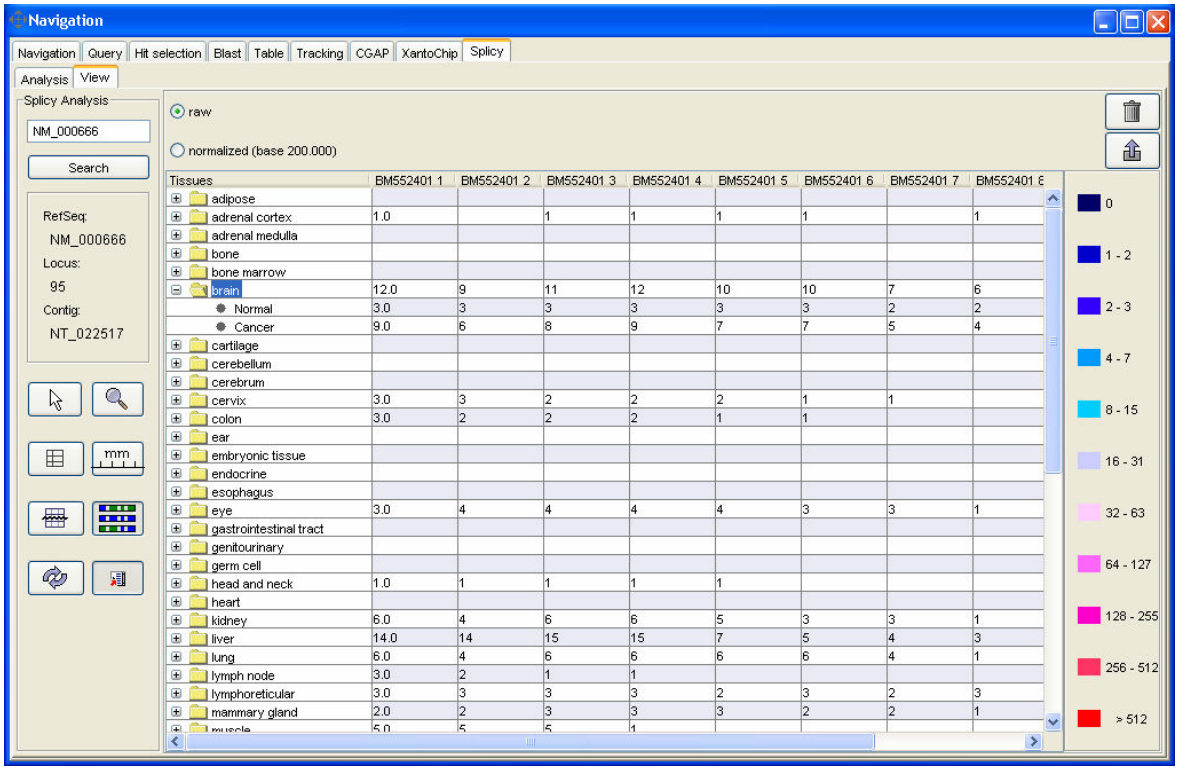

Abb.32: Anzeige der Expressionsdaten (Rohdaten)

Um bei Auswahl einer alternativen Spleißvariante eine größere und damit sicherere Datenmenge zu bekommen, werden auch, wie bereits in 4.1.2. beschrieben, die Expressionsdaten aller ESTs, die ein alternatives Exon bestätigen mit aufgenommen.

Um die Daten angemessen anzeigen zu können, wurde dafür die bereits in 3.2.1. beschriebene *JTreeTable* benutzt.

In der *JTreeTable* stellt jede Spalte ein Exon eines Transkripts dar, dessen Häufigkeit seines Vorkommens und des seiner bestätigenden ESTs in den unterschiedlichen Geweben angezeigt wird.

Diese Daten sind jedoch nur Rohdaten und eignen sich somit nicht für einen Vergleich, da sie keinen Bezugspunkt haben. Deshalb wurde eine zweite *JTreeTable* implementiert, die normalisierte Daten anzeigt.

Für eine Normalisierung der Daten auf eine Basis von 200.000 muss die Anzahl der tatsächlich vorhandenen ESTs durch die Zahl der gesamt vorkommenden ESTs eines Gewebes dividiert und mit 200.000 multipliziert werden. Die entsprechenden Daten können anschließend farblich markiert werden, um eine schnelle Übersicht zu gewährleisten. Dies ist in Abb.33 zu sehen.

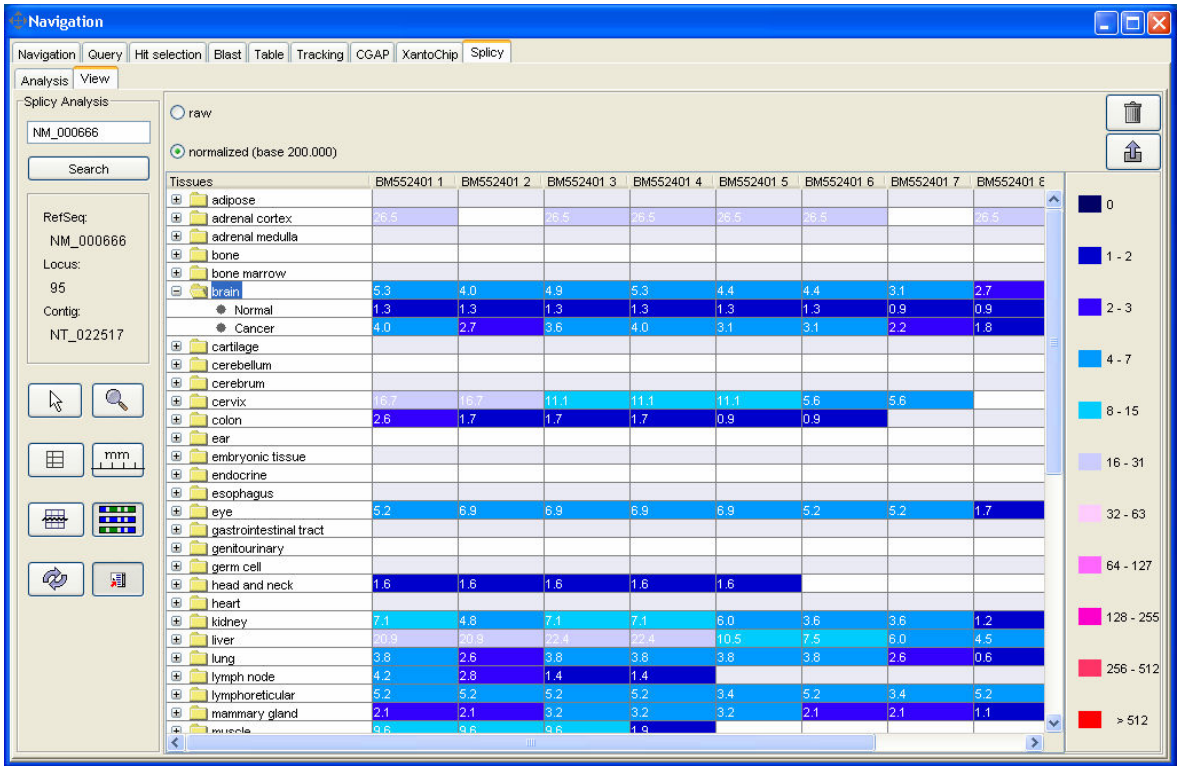

Abb.33: Anzeige der Expressionsdaten (normalisierte Werte)

Da die Bestimmung der Gesamtzahl aller ESTs in den verschiedenen Geweben für die dynamische Auswertung zuviel Zeit in Anspruch nimmt, wurde diese Aufgabe durch einen *materialized view* (Schnappschuss der Datenbank) realisiert. Ein *materialized view* ist eine Tabelle, die durch ein einziges *Select-Statement*

erstellt wurde. Somit kann in Zukunft immer direkt auf diese Tabelle zugegriffen werden, was einen erheblichen Zeitvorteil bietet. Außerdem werden *materialized Views* (Eine Oracle-spezfische Implementierung eines Standard SQL *Views*) in regelmäßigen Abständen automatisch aktualisiert. Somit kann es auch zu keiner Verfälschung der Daten kommen.

Es wurde auch ein weiterer, interessanter Punkt im Sinne der Expressionsdatenanalyse implementiert. Damit ist es möglich herauszufinden, wie sich zwei alternative Spleißvarianten bzgl. ihrer Expression unterscheiden. Dazu kann der Benutzer zwei oder mehrere Spleißvarianten markieren und sich ihre Expression anzeigen lassen. Daraufhin wird wieder eine *JTreeTable* erzeugt, in der sich die Expressionsdaten beider Transkripte nebeneinander angeordnet befinden. Um nun die expliziten Unterschiede zwischen Beiden herauszufinden, wurde eine Funktionalität implementiert, mit deren Hilfe es möglich ist, nur die Expressionsdaten der Exons anzuzeigen, die sich bei beiden Transkripten unterscheiden.

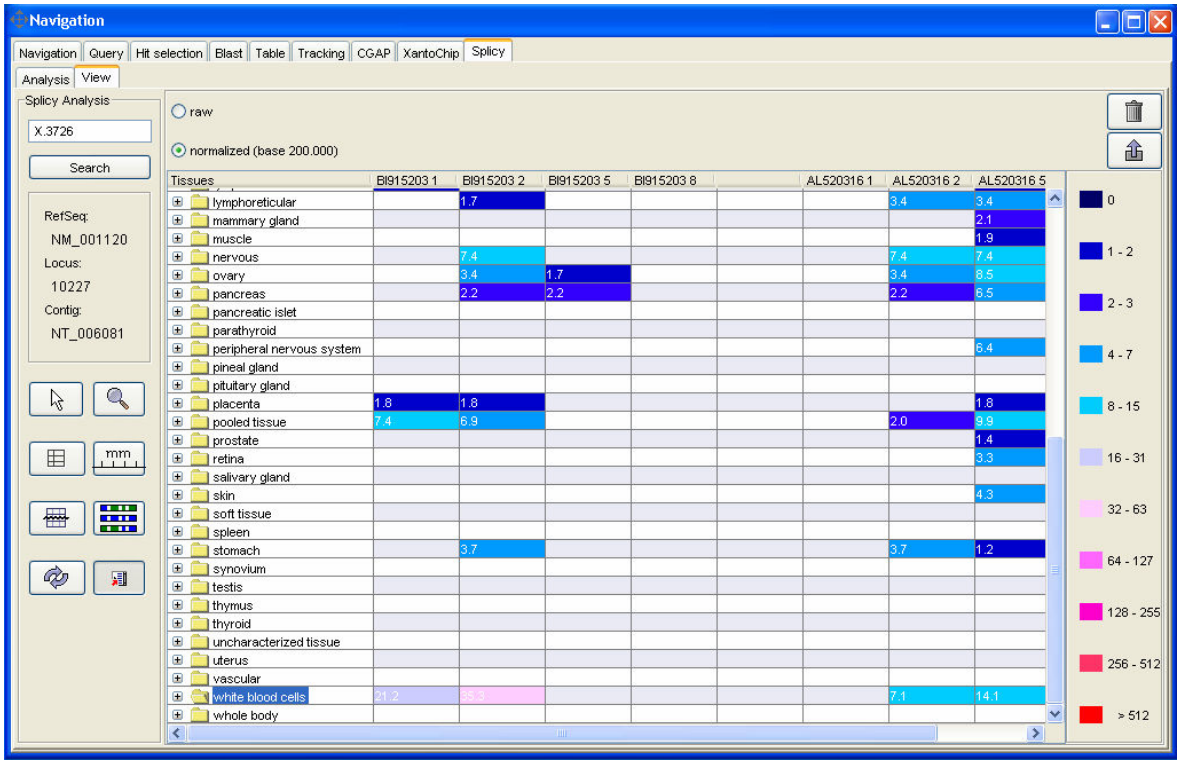

Abb.34: Unterschiede in den Expressionsdaten zweier ESTs

Dies ist z.B. in Abb.34 ersichtlich. Dort unterscheiden sich die Exons 1,2,5 und 8 der beiden ausgewählten Transkripte. Durch die Expressionsdaten lässt sich erkennen, dass die erste Spleißvariante im Gegensatz zur Zweiten vermehrt in den Leukozyten exprimiert ist.

4.5. Analyse der auf Apoptose getesteten Gene

Die Apoptose ist die häufigste Form des natürlichen Zelltods im Organismus. Sie dient der Beseitigung von Zellen, die ihre Aufgabe erfüllt haben, im Laufe der Embryonalentwicklung überflüssig geworden sind, oder Fehler im Genmaterial aufweisen. Apoptose spielt auch eine entscheidende Rolle bei Krankheiten. Der als Schutzmechanismus gedachte Vorgang kann dabei über- oder unterreguliert sein. Als zwei Beispiele sind vor allem das verringerte Auftreten von Apoptose und vermehrter Zellproliferation bei Krebs sowie das vermehrte Auftreten von Apoptose bei AIDS oder degenerativen Krankheiten wie Morbus Alzheimer oder Parkinson zu nennen [20].
Um Apoptose induzierende Gene zu finden, testeten Wissenschaftler der *Xantos biomedicine AG* Klone einer humanen embryonalen cDNA Bank auf apoptotische Vorgänge. Das Testen (*Screening*) wurde dabei in drei Schritten durchgeführt.
Im ersten Schritt wurde ein Lebend-/Tod *Assay*, der auf Robotern automatisiert werden konnte, durchgeführt. Dabei wurden alle Gene der cDNA Bank in *HEK293*-Zellen (*Human embryonic kidney cells*) transfiziert und anschließend die Zellen auf Leben oder Tod kontrolliert. Die in diesem *Assay* positiv bewerteten Gene wurden im zweiten Schritt weiter getestet. Die transfizierten Zellen wurden auf ein Hauptmerkmal der Apoptose, die DNA-Fragmentierung, getestet. Die Gene, die auch in diesem Test mit positivem Ergebnis abgeschnitten hatten, wurden anschließend im Labor auf weitere apoptotische Merkmale getestet:

- *Annexin V*: Ein Oberflächenmarker, der bei apoptotischen Vorgängen vermehrt auf der Zelloberfläche präsentiert wird.
- *Lamin*: Bestandteil der Membran, die bei der Apoptose gespalten wird.

- *Aktive Kaspasen*: Bei der Apoptose werden *Prokaspasen* in *Kaspasen* umgewandelt, die somit ein Merkmal für apoptotische Vorgänge sind.
- Messung des Mitochondrienpotentials. Der elektrochemische Gradient an der Mitochondrienmembran bricht während der Apoptose zusammen. Dieser kann gemessen und so auf Apoptose rückgeschlossen werden.
- *Hoechst-Farbstoff*: Dieser Stoff wird von apoptotischen Zellen eingelagert.

Von 144 Clustern, deren Gene auf Apoptose Induzierung getestet wurden, fand *Splicy* bei 33 verschiedenen Clustern Gene, die alternative Spleißmuster, verglichen mit dem RefSeq Klon aufwiesen. In diesen 33 Clustern befinden sich 184 Klone, von denen 38 als alternativ gespleißt festgestellt werden konnten. Von diesen 38 alternativen Spleißvarianten wurden durch EST-Vergleiche und Vergleiche mit bekannten Datenbanken, wie GenBank, 24 gefunden, die restlichen 14 wurden interessanterweise als alternative Spleißvarianten identifiziert, die noch nicht beschrieben waren.
Von den 38 Genen wurde im Apoptose-Screen festgestellt, dass 15 davon Apoptose induzieren, 20 nicht und 3 wurden bisher noch nicht auf apoptotische Vorgänge getestet.
Dieses Ergebnis wurde anschließend noch weiter untersucht. Es sollte festgestellt werden, ob die gefundenen alternativen Spleißvarianten sich in ihrem apoptotischen Verhalten von dem der entsprechenden RefSeq Klone unterscheiden. Dabei konnte beobachtet werden, dass von den 38 Genen, 15 im Gegensatz zum RefSeq Klon keine Apoptose induzieren, 2 lösen Apoptose aus und die entsprechenden RefSeq Klone nicht, bei 5 Klonen induzieren weder RefSeq Klon noch die alternativen Spleißvarianten Apoptose, bei 6 ist das Gegenteil der Fall, beide induzieren Apoptose und bei 11 Klonen lässt sich keine Aussage machen. Letzteres ist darauf zurückzuführen, dass die Gene entweder im Labor nicht auf Apoptose getestet wurden, oder kein dem RefSeq-Klon entsprechendes Gen gefunden werden konnte. Dies ist wichtig, da nur die Klone im Cluster und nicht der RefSeq Klon selbst auf Apoptose getestet wurde.
Als interessantestes Ergebnis dieser Analyse lässt sich festhalten, dass 14 noch nicht beschriebene Spleißvarianten gefunden werden konnten. Von diesen 14 Genen ließ sich wiederum feststellen, dass 3 von ihnen im Gegensatz zum

RefSeq Klon Apoptose auslösten, 1 nicht, bei 2 Genen lösten beide, RefSeq und alternative Spleißvariante Apoptose aus und bei ebenfalls zweien war das Gegenteil der Fall. Bei insgesamt 7 der 14 Gene konnte aus bereits o.g. Gründen keine Aussage gemacht werden.

5. Diskussion

Splicy ist ein Programm zum Auffinden alternativer Spleißmuster. Für diese Aufgabe wurde durch intensive Vorbereitung ein Algorithmus entwickelt, der diese Aufgabe sehr zuverlässig erledigt. *Splicy* erfüllt die der Aufgabenstellung zu entnehmenden Anforderungen.

Durch Tests des Programms wurden Probleme bei genomischen Region über 10.000 bp Länge festgestellt. Es gab dort zwei Fälle, bei denen die durch die BLAST-Analyse gefundenen ESTs nicht dem entsprechenden Genlocus zugeordnet werden konnten. Durch Fehlersuche wurde klar, dass das Problem an dem verwendeten Programm *Spidey* liegt. Für die BLAST-Analyse wird, wie in 4.1. beschrieben, nur der Teil der genomischen Region benutzt, der für den entsprechenden *RefSeq*-Klon kodiert. Bei den genannten Fällen kam es dabei zu einer Übereinstimmung von 95-100% der Sequenzen bei der BLAST-Analyse. Bei der anschließenden *Spidey*-Analyse, die mit der gesamten Kontigsequenz durchgeführt wird, wurden von *Spidey* Übereinstimmungen auf einem anderen Bereich der genomischen Region gefunden. Dieser Bereich gehörte nicht zu dem des Genlocus. Um diese Eigenart aufzuklären, wurde für die *Spidey*-Analyse nur der dem Genlocus zugehörige genomische Abschnitt, wie auch beim BLAST, benutzt. Daraufhin wurden von *Spidey* keine Alignments mehr gefunden. Da dies anhand des BLAST Ergebnisses definitiv nicht möglich ist, wurde vorliegende Analyse beim NCBI eingereicht, wobei bis heute leider keine Antwort kam.

Falls dieses falsche Ergebnis in Zukunft bei mehr als den bisher zwei Analysen erneut eintreten sollte, wäre zu überlegen, ein Alternative zu Spidey zu benutzen. In die engere Auswahl würde sim4 kommen, das von Wissenschaftlern der *Pennsylvania State University* und des *Drosophila Genome Centers* implementiert wurde [21]. Bereits zu Beginn dieser Arbeit kam es in Betracht, jedoch wurde *Spidey* sim4 vorgezogen, da dies bereits in der manuellen Analyse benutzt wurde. Um *Splicy* dementsprechend umzustellen, wäre es nur nötig, sim4 auf dem Server zu installieren und den Parser der *Spidey*ausgabe sim4 anzupassen.

Literaturverzeichnis

[1]. International Human Genome Sequencing Consortium: Initial sequencing and analysis of the human genome, Nature Vol. 409 (2001), S. 861-921

[2]. Barmak Modrek, Christopher Lee: A genomic view of alternative splicing, Nature genetics Vol. 30 (2002), S. 13-19

[3]. Gilbert W.: Why genes in pieces? Nature 271, S. 501

[4]. Brett, D. et al. EST comparison indicates 38% of human mRNAs contain possible alternative splice forms, FEBS Lett. 474 (2000), S. 83-86

[5]. RNA Splicing; http://www.web-books.com/MoBio/Free/Ch5A4.htm

[6]. Molecular Biology of the Cell, 3rd edn., Part II. Molecular Genetics, Chapter 8. The Cell Nucleus, RNA Synthesis and RNA Processing http://www.ncbi.nlm.nih.gov/books/bv.fcgi?call=bv.View..ShowSection&rid=cell.figgrp.1709

[7]. Celotto AM, Graveley BR: Alternative Splicing of the Drosophila Dscam pre mRNA is both temporally and spatially regulated, Genetics 159 (2001), S.599-608

[8]. Linde Peters: Postgenomik Teil II; http://home.t-online.de/home/linde.peters/postgen1.htm

[9]. J.F. Caceras u. A.R. Kornblihtt, Alternative splicing: multiple control mechanisms and involvement in human disease, Trends in Genetics Vol. 18 (2002), S. 186 – 192

[10]. European Bioinformatics Institute: ASD Alternative Splicing Database, http://www.ebi.ac.uk/asd/asd-ec/index.html

[11]. Website der Fa. Compugen:

http://www.labonweb.com/sitehtml/solutions/splice_variants.html

[12]. National Center for Biotechnology Information (NCBI): ESTs Factsheet; http://www.ncbi.nlm.nih.gov/About/primer/est.html

[13]. Sarah J. Wheelan, Deanna M. Church, James M. Ostell: Spidey: A Tool for mRNA-to-Genomic Alignments, Genome Research 11 (2001), S. 1952-1957

[14]. Hanquing Xie, Wei-young Zhu, Alon Wassermann, Vladimir Grebinskiy, Andrew Olson, Liat Mintz: Computational Analysis of Alternative Splicing Using EST Tissue Information, Genomics Vol.80 (2002), S. 326-330)

[15]. Kim Pruitt, Tatiana Tatusova, James Ostell: The Reference Sequence (RefSeq) Project, NCBI Handbook, (2002), S.18-1 – 18-20

[16]. International Organization for Standardization: Extended BNF, ISO 14977

[17]. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: Basic local alignment search tool, J. Mol. Biol. 215 (1990), S. 403-410

[18]. Zhengyan Kan et al.: Gene Structure Prediction and Alternative Splicing Analysis Using Genomically Aligned ESTs, Genome Research 11 (2001), S.889-900

[19]. Gamma E., Helm R., Johnson R., Vlissides J.: Entwurfsmuster, Addison Wesley Verlag (1996) 5. korrigierter Nachdruck, S.5ff

[20]. Ruhr-Unniversität Bochum: Forschungsgebiet Apoptose, http://www.ruhr-uni bochum.de/anat1/Forschungsschwerpunkte/Forschung_Apoptose.htm#Apoptose%20und%20AIDS

[21]. Florea L., Hartzell G., Zhang Z., Rubin G.M., Miller W.: A computer program for aligning a cDNA sequence with a genomic DNA sequence, Genome Research 8 (1998), S.967-974

Anhang A

Inhalt der beiliegenden CD

- Diplomarbeit im PDF-Format
- Javadoc aller in Splicy enthaltenen Klassen
- Source-Code aller im Rahmen dieser Arbeit erstellten Klassen
- UML-Klassendiagramm
- Beschreibung des UML-Klassendiagramms