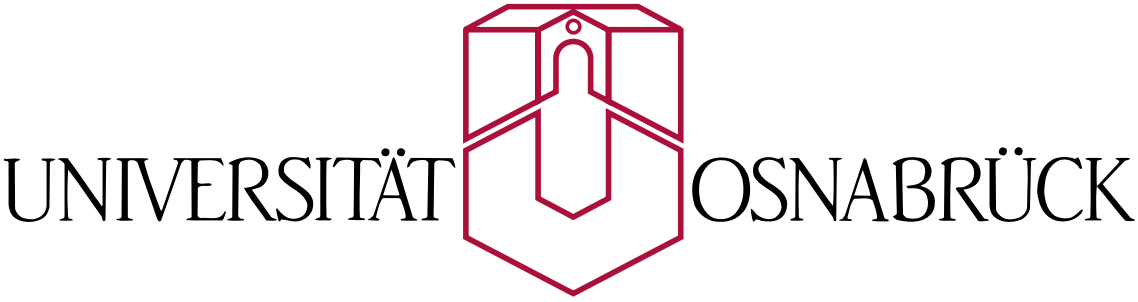


Fachbereich Mathematik/Informatik

SVG und Java-Applets als Web-Interfaces für interaktive Notengrafiken

Diplomarbeit

Sascha Wegener

Gutachter:

Prof. Dr. Oliver Vornberger, Universität Osnabrück
Dr. Tillman Weyde, City University London

Lübbecke, September 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Einleitung

Die vorliegende Arbeit entstand auf der Grundlage des MUSITECH-Projektes der Forschungsstelle Musik- und Medientechnologie an der Universität Osnabrück.

Kern von MUSITECH ist ein flexibles Objektmodell, das es erlaubt, komplexe musikalische Strukturen zu erfassen und auch verschiedene Medientypen einzubinden. Für viele dieser Daten gibt es Module, die diese - zum Teil auch auf verschiedene Arten - visualisieren und hörbar machen können. Das Modul für die Darstellung von Noten basiert auf Arbeiten von Martin Gieseking[2]. Es ermöglicht die dynamische Generierung eines Notenbildes aus dem Objektmodell.

Was dem Modul noch fehlt, ist die Möglichkeit, zum Beispiel über Mausoperationen das Notenbild zu veränderen und diese Veränderungen in das zugrundeliegende Objektmodell zu übertragen. Dies ist jedoch eine essentiell notwendige Eigenschaft, will man das Modul zum Beispiel in Lehr- und besonders Lernkontexten effektiv einsetzen.

Wenn Noten im Internet dargestellt werden sollen, werden häufig noch statische Rasterbilder bereitgestellt. Diese haben oft eine schlechte Auflösung und sind eben statisch. Einen Schritt weiter gehen Browser-Plugins wie zum Beispiel Sibelius Scorch[3]. Jedoch lassen sich mit diesem noch nicht einmal beliebige Sibelius-Dateien anzeigen, sondern nur über einen kostenpflichtigen Sibelius-Service veröffentlichtes Notenmaterial (Lehrmaterialen können auch kostenlos über die Webseite `http://www.sibeliuseducation.com` ausgestauscht werden). Die Interaktivität beschränkt sich auf das Abspielen und Ausdrucken der Noten.

Dabei ständen mit Java-Applets, SVG, Flash, usw. genug Technologien zur Verfügung, die die Entwicklung von interaktiven Musik-Webanwendungen möglich machten. Deshalb ist es jetzt die Aufgabe, diese Techniken zu nutzen und die Möglichkeiten interaktiver Notengrafik im Internet zu untersuchen.

In dieser Arbeit soll ein erster Schritt in diese Richtung angeboten werden. Die Möglichkeiten von interaktiver Notengrafik im Internet sollen anhand zweier verschiedener Ansätze untersucht werden. Als technologische Basis soll in beiden Fällen das MUSITECH-Framework dienen, da es mit dem Objektmodell eine Möglichkeit für flexible interne Repräsentation der Daten bietet und zusätzlich schon über ein Modul für die automatische Generierung von hochwertigen Notengrafiken verfügt. Als Web-Interfaces soll ein Client-basiertes, auf Java-Applets aufbauendes Interface mit einem Client-Server-gestützten Interface auf SVG-HTML-AJAX-Basis verglichen werden.

[2] vgl. auch [Gieseking, 2001]

[3] zu beziehen unter `http://www.sibelius.com/products/scorch/index.html`

Gliederung der Arbeit

In Kapitel 2 soll eine allgemeine Einführung in die Hauptanforderungen an interaktive Notengrafik gegeben werden. In Kapitel 3 erläutere ich das MUSITECH-Projekt und die für diese Arbeit relevanten Module. Kapitel 4 wird einen Überblick über die in dieser Arbeit verwendeten Kerntechnologien geben. Danach werden in Kapitel 5 und 6 die verschiedenen Umsetzungen beschrieben. In Kapitel 7 werde ich Vor- und Nachteile anhand von zwei Beispiellösungen diskutieren. In Kapitel 8 soll das Fazit gezogen und einige Optionen für die Zukunft betrachtet werden.

2 Interaktive Notengrafik

2.1 Computergestützter Notensatz

Für die Eingabe der musikalischen Information in den Computer haben sich zwei grundsätzlich verschiedene Verfahren als besonders effektiv herausgestellt. Zum einen gibt es Skriptsysteme, die den Code in Form von alphanumerischen Zeichen einlesen. Ein Parser übersetzt diesen Code dann in eine interne Datenstruktur, aus welcher das Notenbild errechnet wird. Beispiel dafür ist unter anderem das Notensatzsystem Lilypond[4], das für das Eingabeformat eine LaTeX-ähnliche Syntax benutzt.

```
\version "2.10.25"
\header{
  title = "Invention Nr. 8"
}
\new PianoStaff <<
    \new Staff {
        \relative c'
            r8 f a f c f,
        }
    }
    \new Staff {
        \relative c' {
            s1
        }
    }
>>
```

Quellcode 2.1: Lilypond-Beispieldatei

Das errechnete Notenbild lässt sich in einer Postscript- oder PDF-Datei oder auch z.B. im SVG-Format speichern.

Die Eingabe auf diese Weise erfordert von dem Benutzer nicht nur genaue Kenntnis vom Notensatz, sondern auch von der programmspezifischen Eingabesprache. Eingabefehler lassen sich erst nach dem Übersetzen in das Ausgabeformat im Notenbild erkennen. Danach muss die Stelle in der Eingabedatei lokalisiert und der Fehler dort behoben werden. Lilypond zum Beispiel bietet durch die Point-and-Click-Funktion die Möglichkeit, durch Klicken auf eine Note in der erzeugten PDF-Datei durch einen Hyperlink an die entsprechende Stelle im Quelltext zu kommen. Dieses Verfahren erleichtert zwar die Fehlerkorrektur erheblich, ist aber vom Komfort her nicht mit dem zweiten Verfahren, den interaktiven Systemen, vergleichbar.

[4] `http://www.lilypond.org/`

Nicht nur die effiziente Fehlerkorrektur verlangt nach interaktiven Systemen. Diese zweite Gruppe von Notensatzsystemen ermöglicht zusätzlich auch das interaktive Setzen von Noten. Man startet mit einem oder mehreren leeren Systemen. Die Eingabe der Noten geschieht dann entweder mit der Maus durch Klicken an die entsprechende Stelle im Notentext oder mit der Tastatur durch Drücken der entsprechenden Buchstaben, die den Notennamen entsprechen. Manche Programme ermöglichen auch das Einspielen Step-by-Step (also Note für Note) oder in Echtzeit per MIDI. Als Eingabegerät kann dann jedes MIDI-fähige Instrument dienen (meist ein Keyboard, es gibt auch midifizierte Gitarren, Geigen oder Wind-Controller, die Blasinstrumente simulieren).

Wichtig hierbei ist immer die Kontrolle am Bildschirm. Der Notentext ist nicht nur eine Aneinanderreihung von Noten, sondern muss mehrdimensional gesehen werden. Die Komplexität der Darstellung ist nicht mit der verhältnismäßig einfachen Darstellung von Text vergleichbar. Als Beispiel kann man hierfür das von Donald Knuth in den 80er Jahren an der Stanford University entwickelte Textsatzsystem TEX betrachten. Dieses basiert auf sogenannten symbolumgebenen Boxen. Sie können zwar beliebig ineinander verschachtelt sein, aber sich nicht überlappen.[5] Diese Einschränkung ist bei Notengrafik nicht möglich. Im einfachsten Fall liegen nur die Notenköpfe und -hälse über dem Notensystem mit den Grundlinien. Gegebenenfalls müssen noch zusätzliche Hilfslinien gezeichnet werden. Hierzu können dann aber noch Elemente wie Überbalkungen, Binde- und Legatobögen oder Punktierungen kommen, deren Boxen sich immer überlappen.

Eine besonderere Anforderung ist, dass der Notentext nicht nur durch die syntaktischen Vorgaben beeinflusst wird, sondern auch durch die darinliegende Semantik. Je nachdem, wie ein Komponist eine bestimmte Stelle ausgeführt haben möchte, fällt die Darstellung anders aus.

[5] vgl. [Knuth, 1987] S. 63ff.

2.2 Realisation notengrafischer Interaktion

2.2.1 Auswahl von Objekten

Eine Notengrafik ist ein rechteckiger Bereich mit vielen verschiedenfarbigen Pixeln. Ein Klick mit der Maus markiert zunächst nur einen Punkt mit den Koordinaten (x, y). Das Programm muss nun herausfinden, zu welchem Objekt dieses Koordinatenpaar gehört. Im Wesentlichen beruht diese Zuordnung auf dem Vergleich mit den Positions- und Größenangaben, die in der internen Repräsentation abgelegt sind.

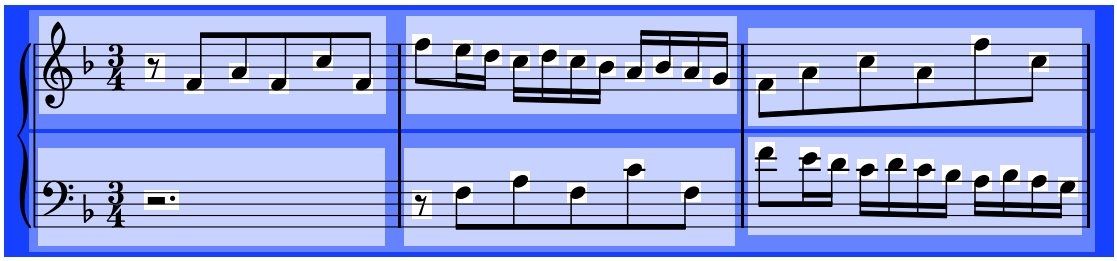


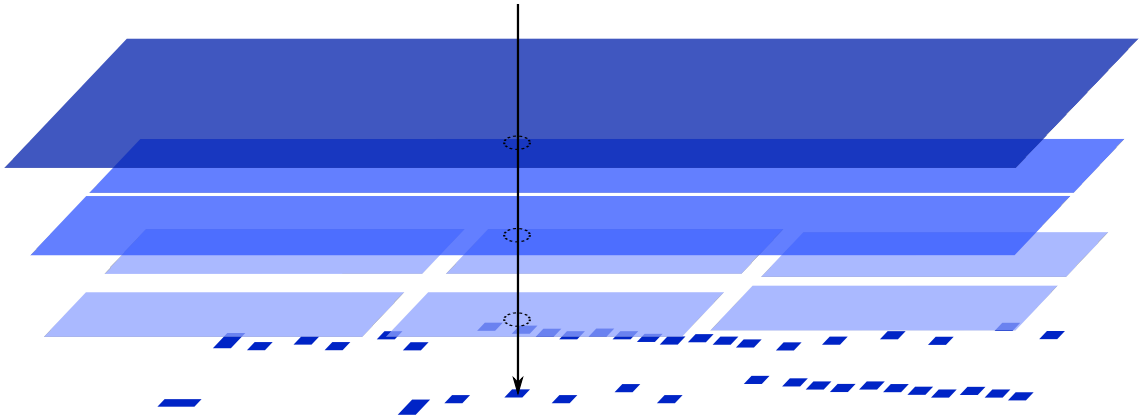
Abbildung 2.1: Notensysteme mit Boundingboxen

Allerdings treten Schwierigkeiten in der Bestimmung auf. Durch die vorher beschriebene Verschachtelung können mehrere Objekte zu den Koordinaten passen: Soll ein ganzer Takt oder nur eine einzelne Note ausgewählt werden? Man kann sich die Notengrafik hierarchisch aufgebaut vorstellen. An oberster Stelle steht die Akkolade. Diese kann mehrere Notensysteme umfassen. Die Notensysteme wiederum bestehen aus Takten. Erst diese Takte enthalten die eigentlich interessanten Elemente, wie die Noten und Pausen.

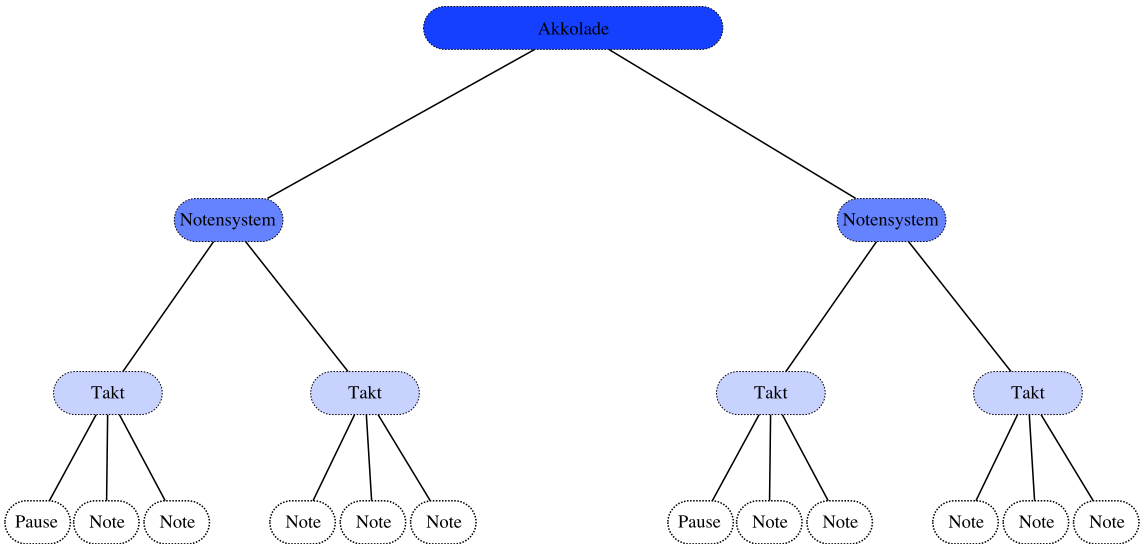
Empfängt das Programm einen Mausklick, steigt es in dieser Hierarchie hinab, bis es das angeklickte Objekt gefunden hat. Um das Problem der Mehrdeutigkeit bezüglich dieser Objektselektion zu lösen, sind verschiedene Modi für die Auswahl denkbar. Es könnte einen Modus geben, bei dem nur Notensysteme ausgewählt werden, einen für einzelne Takte und dann auf unterster Ebene einen Modus für Noten, Pausen und andere Elemente auf Taktebene. Bei manchen Objekten, wie zum Beispiel Vorzeichen, ist es auch nicht sinnvoll, sie auswählen zu können. Sie können dann einfach ignoriert werden.

Anders verhält es sich bei der Auswahl von mehreren Objekten mit Hilfe eines Auswahlrechteckes. Alle relevanten Objekte werden innerhalb eines rechteckigen Bereiches gesucht. Da das Auswahlrechteck nicht unbedingt komplett in dem den auszuwählenden Objekten übergeordneten Containern liegen muss, ist es nicht wie beim Mausklick möglich, in dem Baum hinabzusteigen. Bei kleineren musikalischen Kontexten ist es sinnvoll, die relevanten Objekte in einer Liste zu speichern und dann der Reihe nach deren Lage zu untersuchen. Bei größeren Kontexten ist die Organisation in einer mehrdimensionalen Suchstruktur wie zum Beispiel dem k-d-Baum[6] möglich.

[6] vgl. [Vornberger, 2005] S. 54



(a) Absteigen in der Boxenhierarchie



(b) Baummodell

Abbildung 2.2: Hierarchie der Boundingboxen

2.2.2 Editierung der Objekte

Wenn der Benutzer ein oder mehrere Notationselemente ausgewählt hat, soll es auch die Möglichkeit geben, diese Elemente zu verändern; zum Beispiel soll die Tonhöhe einer Note verändert werden oder gar eine oder mehrere Noten gelöscht werden. Im Vergleich zum Auswählen ist diese Bearbeitung mit erheblich mehr Aufwand verbunden. Das Auswählen hat im Gegensatz zum Verschieben keinen Einfluss auf das zugrundeliegende Notenmaterial. Falls markierte Notationselemente zum Beispiel eingefärbt werden sollen, müssen die Grafikelemente geändert werden, nämlich ein anderes Farbattribut bekommen. Dies hat aber keinen Einfluss auf das Layout des Notenbildes und das Notenmaterial.

Möchte man jetzt zum Beispiel eine Note um ein bestimmtes Intervall vertikal verschieben, wäre es kein Problem, einfach die y-Koordinate des Grafikobjekts der

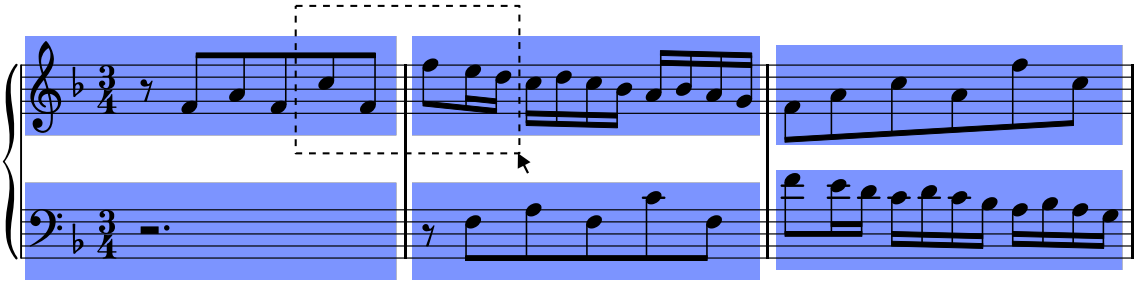


Abbildung 2.3: Mögliche Überlappung bei der Rechteckauswahl

Note um einen gewissen Wert zu verändern und die Grafik neu zu zeichnen. Dadurch ergeben sich aber weitere Probleme. Zum einen hat man dadurch nur die aus dem Notenmaterial resultierende Grafik geändert und nicht das Notenmaterial selbst. Zum anderen muss die Darstellung aus notensatztechnischer Sicht geprüft werden. Verschiebt man zum Beispiel eine einzelne Note über die mittlere Notenlinie, muss normalerweise der Notenhals umgekehrt werden. Dies gilt aber nicht unbedingt, wenn die Note Teil eines Akkordes ist und sich den Hals mit anderen Noten auf der gleichen Zählzeit teilt. Aber selbst das Ändern der Halsrichtung kann im Extremfall das Neuberechnen der Taktverteilung im ganzen System notwendig machen. Daher muss die komplette Darstellung neu berechnet und aufgebaut werden. Die Änderung von sehr wenigen lokalen Parametern ist nur selten möglich.[7]

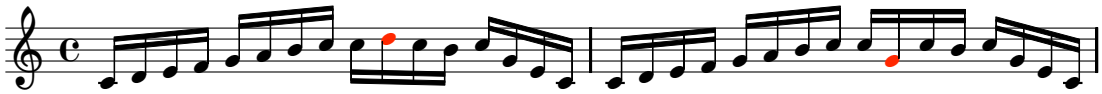


Abbildung 2.4: Die Änderung der Tonhöhe einer Note zieht die Umkehrung der Hälse einer ganzen Balkengrupe nach sich.

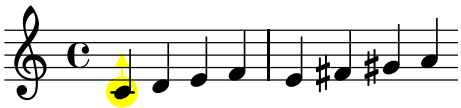
Daher muss ein anderer, geeigneterer Ansatz gewählt werden. Durch das Verschieben einer Note mit der Maus auf dem Bildschirm muss das Notenmaterial entsprechend geändert werden. Aus diesem Notenmaterial kann dann die neue korrekte Darstellung berechnet werden. Das Verschieben der Note sollte mithilfe eines Maus-Drags geschehen, da man so auch in anderen Programmen Objekte auf dem Bildschirm verschiebt. Hier könnte man die Differenz dy der y-Koordinaten der Punkte, an dem die Maustaste gedrückt und wieder losgelassen wurde, bestimmen. Da der Abstand d der Notenlinien zwei Tonschritten entspricht, kann man die Anzahl n der Tonschritte, um die der Ton verschoben werden soll, durch folgende Formel bestimmen:

$$n = \left\lceil \frac{dy}{\frac{1}{2}d} \right\rceil$$

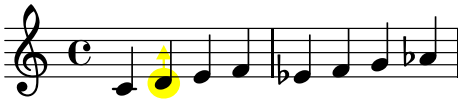
Mit dieser Formel wird aber nur angegeben, um wie viele Tonschritte die Note auf dem Bildschirm verschoben werden soll. Es wird noch nichts darüber ausgesagt, wie die Note in der Notationsbasis geändert werden muss. Dazu muss nämlich noch unter

[7] vgl. [Gieseking, 2001] S. 81

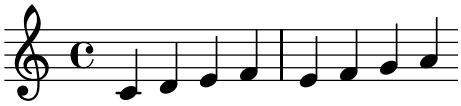
anderem die Tonart und damit die Lage der Halbtonschritte einbezogen werden. Ein weiteres Problem tritt auf, wenn mehrere Noten markiert und verschoben werden. Werden alle Töne um die gleiche Anzahl von Ganz- und Halbtonschritten verschoben, kann es passieren, dass manche Töne, die vorher noch aus dem Tonmaterial der Tonart stammten, nach dem Verschieben nicht mehr in die Tonart gehören; ganz davon abgesehen, dass auch nicht klar ist, welcher von den ausgewählten Tönen die Anzahl der Ganz- und Halbtöne, um die verschoben werden soll, festlegt. Sinnvoller erscheint es in diesem Fall, dann jeden Ton einzeln zu betrachten.



(a) Verschieben um eine große Terz (Referenzton c)



(b) Verschieben um eine kleine Terz (Referenzton d)



(c) Verschieben um eine Terz abhängig vom jeweiligen Ton

Abbildung 2.5: Verschiedene Verschiebemethoden

3 MUSITECH

Das Projekt lief vom 1. März 2001 bis zum 31. September 2004 und wurde von der Deutschen Forschungsgemeinschaft (DFG) gefördert. Thema des Projektes war die

> *"Entwicklung, Konzeption und Implementation von Navigationselementen und Kommunikationsmodellen zur interaktiven Handhabung akustischer Informationen in virtuellen Wissensräumen (Internet-Verlagen, Datenbanken und vernetzten Lehr/Lernsystemen)[8]."*

3.1 Ausgangssituation

Im Projekt MUSITECH wurde eine neue Konzeption und Infrastruktur für musikalische Wissensräume entwickelt, die die Information und Kommunikation auf mehreren musikalischen Ebenen unterstützt. Untersuchungen zu Beginn des Projektes hatten ergeben, dass für so komplexe Interaktion und Navigation noch kein technischer Rahmen zur Verfügung stand[9]. Zwar existierte eine Vielzahl von geplanten Softwareumgebungen, Datenmodellen und -formaten, diese stellten sich aber als zu eingeschränkt und unflexibel heraus.

Auf der Basis von Java und XML sollten nun ein Datenmodell, eine Persistenzschicht und Softwarekomponenten entwickelt werden, die es ermöglichen, interaktive, musikalische Anwendungen zu entwickeln.

3.2 Repräsentation von musikalischen Informationen in MUSITECH

Der Repräsentation liegen zwei Prinzipien zugrunde: Zum einen das Prinzip der integrierten Repräsentation, das bedeutet, dass verschiedene Arten und Stufen von Informationen kombiniert werden können, zum Beispiel verschiedene Medientypen; zum anderen das Prinzip der Objektunabhängigkeit, das bedeutet, dass die einzelnen Objekte unabhängig von ihrem Kontext sein sollen. Also muss zum Beispiel eine Note nicht verändert werden, nur weil die Tonart geändert wird.

Kern der MUSITECH-Infrastruktur ist ein Objektmodell zur Beschreibung musikalischer Informationen und Strukturen. Dieses Modell ist möglichst flexibel und erweiterbar gehalten, um alle musikalisch relevanten Aspekte erfassen zu können.

[8] [Enders (e.a.), 2004], S. 1
[9] [Weyde, 2002], S. 2f

3.2.1 Physikalische und metrische Zeit

In MUSITECH gibt es zwei Arten von Zeit: In der metrischen Zeit werden Zeitpunkte und die Dauer von Ereignissen im Verhältnis zu einem Grundschlag, dem Metrum, mit Hilfe von Brüchen angegeben. Der Nenner ist gewöhnlich eine Potenz von 2. Demgegenüber steht die physikalische, absolute Zeit, die in Mikrosekunden vom Beginn des Stückes an gemessen wird.

Objekte können das Interface `Metrical` oder das Interface `Timed` implemtentieren. Über das Interface `Metrical` können sie eine metrische Zeit referenzieren, über das Interface `Timed` eine physikalische.

Mit Hilfe der `MetricalTimeLine` geschieht die Abbildung der metrischen Objekte auf die absolute Zeit. Sie enthält unter anderem `TimedMetrical`-Objekte, die eine metrische und eine absolute Zeit haben. Um ein Tempo festzulegen, muss eine `MetricalTimeLine` mindestens zwei `TimedMetrical`-Objekte besitzen.

Beispiel

Abbildung 3.1 zeigt eine `MetricalTimeLine` mit drei `TimeSignatureMarker`n. Der erste `TimeSignatureMarker` besitzt die absolute Zeit 0s und die metrische Zeit $\frac{0}{4}$ und der zweite die absolute Zeit 30s und die metrische Zeit $\frac{60}{4}$. Dies bedeutet, dass auf 30 Sekunden 60 Viertelnoten kommen und hierdurch implizit ein Tempo von 120 bpm festgelegt wird. Der dritte `TimeSignatureMarker` besitzt die absolute Zeit 60s und die metrische Zeit $\frac{100}{4}$. Hier kommen auf 30s genau 40 Viertelnoten, was einem Tempo von 80 bpm entspricht.

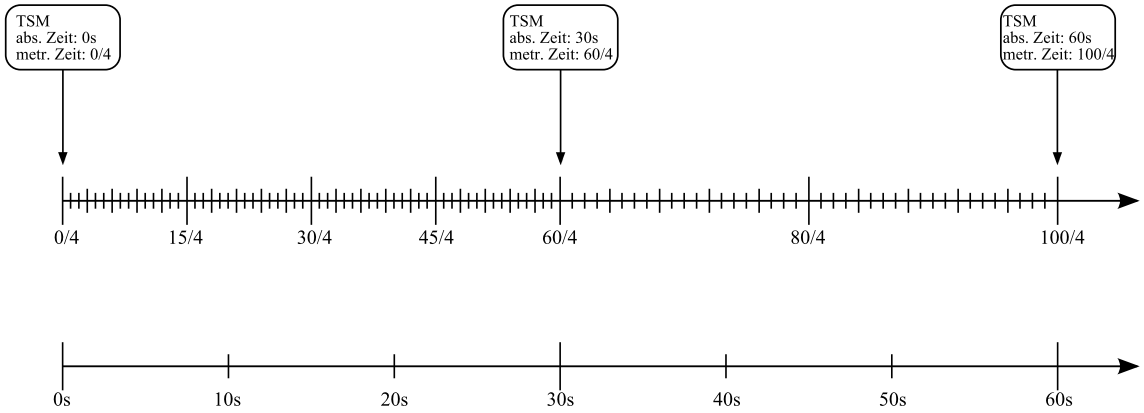


Abbildung 3.1: Schematisches Beispiel einer MetricalTimeLine

3.2.2 Noten

Noten werden in MUSITECH durch zwei verschiedene Klassen repräsentiert. Es gibt auf der einen Seite die `ScoreNote`, die für die Darstellung der Note wichtig ist. Auf der anderen Seite gibt es noch die `PerformanceNote`, die für die akustische Wiedergabe zuständig ist. Diese Aufteilung ist sinnvoll, da zum einen die absolute Position

einer Note nicht mit der metrischen Position übereinstimmen und zum anderen eine notierte Note nicht unbedingt einer gespielten Note entsprechen muss. Beispiele sind Triller, bei denen auf eine `ScoreNote` mehrere `PerformanceNote`s kommen, oder angebundene Noten, die als mehrere Noten notiert werden, aber nur als eine Note gespielt werden. Bei Arpeggien werden mehrere Noten auf einer Zählzeit notiert, aber leicht versetzt gespielt.

Die unterschiedlichen Klassen enthalten natürlich auch unterschiedliche Informationen über die Note. Während für die `ScoreNote` die metrische Einsatzzeit und Dauer wichtig ist, ist für die `PerformanceNote` der absolute Anfang und das Ende entscheidend. Für die `ScoreNote` wird die diatonische Tonhöhe, das Versetzungszeichen gespeichert, dagegen reicht der `PerformanceNote` die absolute Tonhöhe codiert als ganzzahliger Wert (MIDI-Pitch).

Zusammengefasst werden `ScoreNote` und `PerformanceNote` in der Klasse `Note`.

3.2.3 Strukturen

Musikstücke weisen unterschiedliche Arten von Strukturierungen auf. Zum einen gibt es eine Aufteilung in verschiedene Systeme und Stimmen. Dann kann es aber auch eine Aufteilung in Formteile geben, wie zum Beispiel bei der Sonatenhauptsatzform, der dreiteiligen Liedform oder dem Rondo.

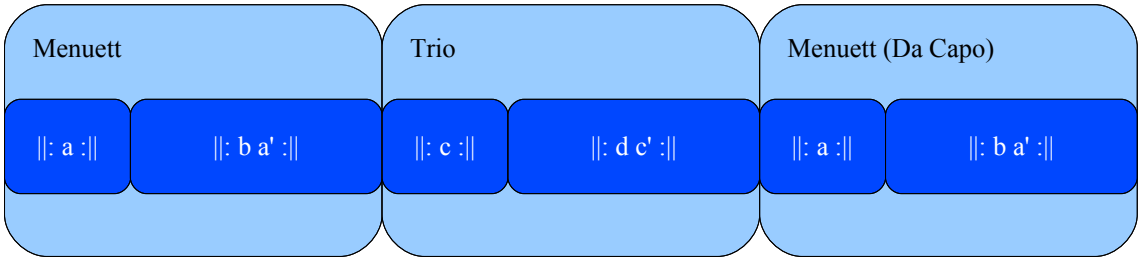


Abbildung 3.2: Schematischer Ablauf eines Menuetts mit Trio[10]

Bestehende Formate bieten meist nicht genug Flexibilität bei der Strukturierung. Zum Beispiel erlaubt MIDI nur eine Aufteilung in verschiedene Stimmen. MUSITECH hingegen ermöglicht eine beliebige Strukturierung, sowohl hierarchisch als auch linear. Die wichtigsten Strukturelemente können Tabelle 3.1 entnommen werden.

Hierarchische Strukturierung	Lineare Strukturierung
NotationSystem	Form
NotationStaff	Motif
NotationVoice	Movement
NotationChord	NoteGroup
	Phrase
	Section

Tabelle 3.1: Einige Elemente für die hierarchische und lineare Strukturierung

[10] vgl. [Kühn, 2001], S. 68ff.

3.3 Weitere wichtige Funktionen des MUSITECH-Frameworks

3.3.1 DataChangeManager

Mit Hilfe des `DataChangeManager`s können Veränderungen an Objekten überwacht werden. Hier kommt das Observer-Pattern[11] zum Einsatz: Jedes Objekt einer Klasse, die das `DataChangeListener`-Interface implementiert, kann sich beim `DataChangeManager` für ein zu überwachendes Objekt anmelden. Werden nun Änderungen an diesem zu überwachenden Objekt vorgenommen, müssen diese dem `DataChangeManager` gemeldet werden. Dieser ruft dann bei den angemeldeten Listenern die `dataChanged()`-Methode auf.

3.3.2 SelectionManager

Der `SelectionManager` funktioniert ähnlich wie der `DataChangeManager`. Es können sich Objekte, die das Interface `SelectionListener` implementieren, anmelden und sich über Selektionsänderungen informieren lassen. Dies ermöglicht zum Beispiel, dass Noten, die in einem `PianoRollDisplay` markiert werden, gleichzeitig auch in einem `NotationDisplay` als markiert angezeigt werden. Da die markierten Objekte zentral gespeichert werden, kann man sich auch leicht Zugriff auf diese Objekte beschaffen, etwa um sie zu verändern. Zum Beispiel kann der `ActionListener` eines Buttons so auf die markierten Elemente zugreifen und diese löschen (Entfernen-Button).

3.3.3 PlayTimer

Der PlayTimer ist der zentrale Zeitgeber. Alle Objekte, die zeitlich synchronisiert werden sollen, müssen sich bei ihm anmelden. Für die Synchronisation gibt es zwei verschiedene Methoden:

- Zeitkritische Objekte - wie zum Beispiel der `ObjectPlayer` - können im Pull-Modus die aktuelle Zeit beim PlayTimer abfragen.
- Für den weniger präzisen Push-Modus können sich Objekte, die das Interface `Timeable` oder `MetricTimeable` implementieren, als Listener anmelden. Bei ihnen wird dann immer wieder die Methode `setTimePosition()` bzw. `setMetricTime()` aufgerufen.

Über die Methoden `start()`, `stop()`, `reset()` wird der PlayTimer gesteuert.

3.3.4 ObjectPlayer

Der `ObjectPlayer` ist die zentrale Klasse für die MIDI-basierte Aufnahme und Wiedergabe. Er ermöglicht es, Objekte, die das Interface `Container` implementieren (`NotationSystem`, `NotationStaff`, `NotationVoice`, `Motif`, `Phrase`, usw.), per MIDI auszugeben.

[11] vgl. [Ullenboom, 2007], S. 497

4 Grundlegende Technologien

4.1 Extensible Markup Language

Die Extensible Markup Language (XML) ist eine Auszeichnungssprache und dient der Strukturierung von Texten und Daten. Bereits Mitte der Achtzigerjahre wurde die Standard Generalized Markup Language (SGML) als ISO-Standard definiert. Da sich SGML aber als zu komplex für einfache Anwendungen heraustellte, wurde ab 1996 von einer Arbeitsgruppe des World Wide Web Consortium eine Markup-Sprache auf Basis von SGML entwickelt, die auf der einen Seite so flexibel wie SGML sein, auf der anderen Seite aber so einfach zu nutzen und implementieren sein sollte, wie z.B. HTML. Das Ergebnis war XML.[12]

4.1.1 Eigenschaften von XML

XML ist eine Metamarkup-Sprache. Das bedeutet: Es gibt keine festgelegte Menge von Tags und Elementen. Sie können für jede Anwendung die Elemente so definiert werden wie sie benötigt werden.

Obwohl XML einerseits sehr flexibel ist, bestehen andererseits strikte Regeln, zum Beispiel, wie ein Element auszusehen hat oder wie Attribute definiert werden müssen. Dokumente, die diesen Regeln entsprechen, heißen *wohlgeformt*. Dies ermöglicht die Entwicklung von Parsern, die es erlauben, jedes beliebige XML-Dokument zu verarbeiten.

Das Markup beschreibt die Struktur eines Dokumentes; es sagt im Allgemeinen aber nichts über das Aussehen aus. XML ist in erster Linie eine strukturelle, semantische Markup-Sprache und legt nicht fest, wie das Dokument angezeigt werden soll. Dennoch gibt es auch einige Anwendungen, wie SVG oder XML-FO, bei denen mittels XML das Aussehen und Layout beschrieben wird.

XML bietet die Möglichkeit, Daten plattformunabhängig zu speichern oder zu übertragen. Die Dateien können mit jedem Texteditor geöffnet und bearbeitet werden.[13]

4.1.2 Aufbau eines XML-Dokuments

Die erste Zeile in einen XML-Dokument besteht immer aus der XML-Deklaration, in der die XML-Version und eventuell die Codierung der Datei angeben wird. Der Rest

[12] vgl. [Eisenberg, 2002] S. 277ff.
[13] vgl. [Harrold/Means, 2002] S. 3ff.

des Dokumentes besteht aus strukturierten Elementen, die hierarchisch geschachtelt sind. Die Elemente können entweder andere Elemente, Text oder auch beides enthalten. Zusätzlich kann ein Element noch mit Attributen versehen werden.

```
<?xml version="1.0" encoding="UTF-8">
<adressbuch>
    <eintrag>
        <name>
            <nachname>Wacker</nachname>
            <vorname>Willi</vorname>
        </name>
    </eintrag>
</adressbuch>
```

Quellcode 4.1: XML-Beispieldatei

Elemente

Mit Hilfe von Elementen werden XML-Dateien strukturiert. Elemente bestehen aus einem öffnenden und einem schließenden Tag. Der öffnende Tag beginnt mit `<`, der schließende mit `</`. Danach folgt der Elementname und dann `>`. Innerhalb der Tags können sich weitere Elemente und/oder Text befinden.

Ebenfalls möglich sind leere Elemente. Sie können entweder aus einem öffnenden und schließenden Tag ohne Inhalt bestehen oder aus einem einzigen Tag, der mit `<` beginnt und mit `/>` schließt.[14]

Attribute

Jedes XML-Element kann zusätzlich noch Attribute besitzen. Attribute sind Schlüssel-Wert-Paare, die innerhalb des öffnenden oder des leeren Tags stehen. Die Schlüssel werden von den Werten mit einem Gleichheitszeichen getrennt. Die Werte stehen zusätzlich in einfachen oder doppelten Anführungszeichen.

```
<student matrikelnummer="123456" fachbereich="6">
    <name>Willi Wacker</name>
</student>
```

Quellcode 4.2: XML-Attribute

Für die Namen der Elemente und Attribute können alle alphanumerischen Zeichen und zusätzlich noch der Unterstrich, der Bindestrich und der Punkt verwendet werden. Sie dürfen nur nicht mit einer Zahl oder dem Unterstrich beginnen.

[14] vgl. [Harrold/Means, 2002] S. 13f.

Namensräume, Namespaces

Mithilfe von Namensräumen oder Namespaces ist es möglich, verschiedene XML-Anwendungen in einem Dokument zu benutzen. Beispiele hierfür sind etwa XHTML-Dateien, die zusätzlich SVG oder XSL-Stylesheets, die XSL-Anweisungen und Elemente aus dem Zielvokabular enthalten.

Namensräume trennen Elemente und Attribute verschiedener Anwendungen durch Zuweisung von URIs. Diese Zuweisung geschieht im Wurzelelement. So können auch Elemente oder Attribute verschiedener Anwendungen mit gleichem Namen unterschieden werden. Da URIs Sonderzeichen, wie /,% und ∼ enthalten können, sind sie in Elementnamen nicht zugelassen. Deshalb werden ihnen Abkürzungen zugeordnet, die mit einem Doppelpunkt abgetrennt vor dem eigenlichen Element- oder Attributnamen stehen. Alles vor dem Doppelpunkt Stehende heißt Prefix, alles danach Local Part. Der komplette Elementname wird dann als Qualified Name bezeichnet.[15]

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:svg="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <head>
        <title>XML Namespaces</title>
    </head>
    <body>
        <h1>Hello World</h1>
        <svg:svg width="400" height="200">
            <svg:rect x="50" y="50" width="100" height="100"
                style="fill:yellow; stroke:black;
                       stroke-width:0.1cm;" />
            <svg:rect x="250" y="50" width="100" height="100"
                style="fill:red; stroke:black;
                       stroke-width:0.1cm;" />
            <svg:circle cx="200" cy="100" r="75"
                style="fill:blue; stroke:green;
                stroke-width:0.5cm; opacity:0.4;" />
        </svg:svg>
    </body>
</html>
```

Quellcode 4.3: Beispiel für die Verwendung verschiedener Namespaces

[15] vgl. [Harrold/Means, 2002] S. 60ff.

4.1.3 Document Object Model (DOM)

Im Jahre 1997 begann das W3C, ein Objektmodell für strukturierte Dokumente zu entwickeln. Das Ziel war es, ein programmiersprachenunabhängiges API für den Zugriff und die Manipulation von XML-Dokumenten zu entwerfen. Die verschiedenen Versionen werden als *Level* bezeichnet.

DOM Level 1 war die erste Spezifikation des W3C und auf die Arbeit mit HTML- und XML-Dokumenten im Browser ausgerichtet. Da davon ausgegangen wurde, dass die Dokumente schon im Browser verfügbar waren, enthält die Level 1-Spezifikation nur das Objektmodell und Methoden zum Manipulieren dieses Modells.

In DOM Level 2 wurde die Spezifikation modularisiert und erweitert. Die wichtigsten Module sind:

DOM Core Das Core-Modul enthält die Methoden zum Zugriff auf die Elemente des Objektmodells.

DOM Style und DOM CSS Das Modul ermöglicht das Ändern der Formatierung und des Layouts über Stylesheets oder Cascading Stylesheets.

DOM Event Dieses Modul standardisiert die Verarbeitung von Events. Es gibt verschiedene Arten von Events. Userinterface-Events sind vom Benutzer zum Beispiel durch Tastendruck oder Mausklicks ausgelöste Events. Mutation-Events sind Events, die beim Ändern der Struktur des Dokuments ausgelöst werden.

Durch diese Modularisierung muss eine Anwendung nur die benötigten Module und nicht die komplette Spezifikation implementieren. Zusätzlich existieren noch Erweiterungen für verschiedene Anwendungsbereiche, wie etwa SVG, MathML oder SMIL.

Da das DOM programmiersprachenunabhängig sein soll, werden die Interfaces in der Interface Description Language (IDL) definiert. Diese Interfaces können dann in der jeweiligen Programmiersprache nach den Richtlinien der OMG, die die IDL spezifiziert hat, implementiert werden.

```
interface NodeList {
    Node                item(in unsigned long index);
    readonly attribute unsigned long    length;
};
```

Quellcode 4.4: Das IDL-Interface für eine NodeList[16]

Das Document Object Model stellt ein XML-Dokument als eine Hierarchie von Knoten (Nodes) dar. Es gibt generische und spezielle Interfaces. Das Basis-Interface für alle anderen Interfaces ist das generische Interface `Node`. Es stellt die Basisfunktionalität zum Navigieren im XML-Baum und zum Zugriff auf die elementaren Eigenschaften des Knotens bereit (vgl Tabelle 4.1).

Zusätzlich zu den generischen Interfaces, über die schon ein Zugriff auf alle Daten möglich ist, gibt es noch die speziellen Interfaces, die vereinfachten Zugriff auf die Knoten bieten. Über das `Node`-Attribut `type` kann der Knotentyp

[16] aus: `http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/idl-definitions.html`

Name	Beschreibung
Attribute	
`nodeName`	Name des Knotens
`nodeValue`	Wert des Knotens zurück
`parentNode`	übergeordneter Knoten
`childNode`	Kindknoten als `NodeList`
`firstChild`	erster Kindknoten
`lastChild`	letzter Kindknoten
`previousSibling`	Vorgängerknoten
`nextSibling`	Nachfolgeknoten
Methoden	
`insertBefore(newChild, refChild)`	Fügt den Knoten `newChild` vor `refChild` ein.
`replaceChild(newChild, oldChild)`	Ersetzt den Knoten `oldChild` durch `newChild` ein.
`appendChild(newChild)`	Fügt den Knoten `newChild` als letzten Knoten an
`replaceChild(newChild, oldChild)`	Ersetzt den Knoten `oldChild` durch `newChild` ein.

Tabelle 4.1: Die wichtigsten Attribute und Methoden des `Node`-Interfaces

herausgefunden und der Knoten zu einem spezielleren Typ gecastet werden. Die wichtigsten speziellen Interfaces sind `Document`, `Element`, `Attr` und `Text`.

Document Der `Document`-Knoten ist der oberste Knoten im DOM. Er enthält Information über das XML-Dokument und auch das Wurzelelement. Über diverse `create...()`-Methoden können verschiedene Knotentypen erzeugt werden.

Element Dies ist der meist gebrauchte Knotentyp für ein XML-Dokument. Er entspricht den XML-Elementen.

Attr Ein `Attr`-Knoten entspricht einem Attribut eines XML-Elementes. Als Eigenschaften hat er Name und Wert des Attributes.

Text Falls ein XML-Element Text enthält, wird er in einem `Text`-Knoten gespeichert.

Ein gravierender Nachteil vom DOM ist sein hoher Speicherverbrauch. Bevor auf einzelne Knoten zugegriffen werden kann, muss erst das ganze Dokument eingelesen, geparst und der komplette Objektbaum im Speicher aufgebaut werden. Deshalb existieren zum Beispiel mit der Simple API for XML (SAX) auch Ansätze, Dokumente sequentiell zu verarbeiten.

4.2 Scalable Vector Graphics

4.2.1 Rastergrafik versus Vektorgrafik

Rastergrafik

In einer Rastergrafik wird jedes Bild durch eine bestimmte Anzahl von Pixeln dargestellt. Jedes Pixel enthält dabei entweder einen RGB-Farbwert oder wird durch einen Verweis in eine Farbtabelle repräsentiert.[17]

Rastergrafiken werden (bzw. sollten) immer dann eingesetzt (werden), wenn das Bild nicht durch geometrische Primitive beschrieben werden kann - zum Beispiel bei digitalen Fotos - oder wenn die geometrische Struktur nicht bekannt ist (zum Beispiel beim Einscannen von Strichzeichnungen).

Wegen der großen Datenmenge, die bei einem Rasterbild durch Speichern der einzelnen Pixel entsteht, gibt es viele Dateiformate, bei denen der Datenstrom komprimiert wird (z.B. PNG, GIF, JPEG). Aufgabe eines Bilddarstellungsprogramms ist es nun, die Daten wieder zu dekomprimieren und die einzelnen Pixel zum Anzeigen an die Grafikkarte zu senden.

Vektorgrafik

In einer Vektorgrafik wird das Bild durch eine Anzahl von geometrischen Primitiven, wie Linien, Rechtecke, Kurven, Kreise, usw. beschrieben. Statt z.B. bei einer Linie jedes Pixel zwischen Anfangs- und Endpunkt zu speichern, braucht nur der Anfangs- und Endpunkt der Linie gespeichert zu werden. Die Datenmenge ist dadurch also viel kleiner.

Ein entscheidener Vorteil von Vektorgrafiken ist, dass sie unabhängig vom Zoomfaktor sind, da das anzuzeigende Bild für jede Zoomstufe neu berechnet wird. Ebenso können die Bilder in beliebig hoher Auflösung dargestellt werden, was z.B. beim Erstellen von hochwertigen Ausdrucken wichtig ist.

4.2.2 Die Entwicklung von SVG

Im Jahre 1998 wurde vom W3C, dem World Wide Web Consortium, eine Arbeitsgruppe mit der Aufgabe, eine Beschreibungssprache für Vektorgrafiken auf XML-Basis zu entwickeln, gegründet. Das Ergebnis war das SVG-Format (Scalable Vector Graphics).

Die Empfehlung des W3C für die Version 1.0 erschien am 4. September 2001. Die aktuelle Empfehlung ist Version 1.1 vom 14. Januar 2003[18]. Parallel dazu existieren die sogenannten SVG Mobile Profiles, die mit den Profilen SVG Tiny und SVG Basic speziell für die Verwendung auf mobilen Geräten mit beschränkter Leistung (z. B. Handys) ausgelegt sind.

[17] vgl. [Eisenberg, 2002] S. 2
[18] vgl. [W3C, 2003]

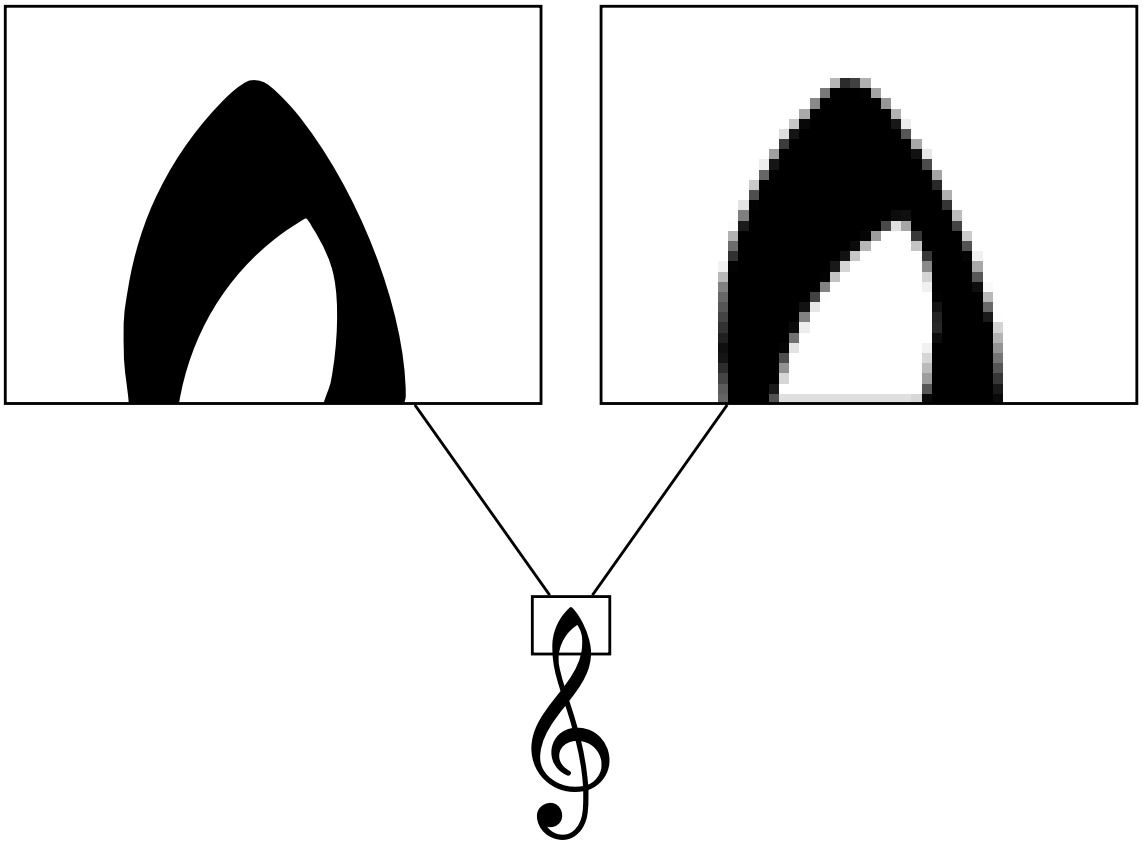


Abbildung 4.1: Vektorgrafik im Vergleich mit Rastergrafik

Die nächste Version 1.2 liegt als Entwurf vom 13. April 2005 (Mobile 10. August 2006) vor. Daneben existiert auch ein Entwurf für eine SVG Print Spezifikation.

4.2.3 Vor- und Nachteile von SVG gegenüber anderen Vektorgrafikformaten

Bei SVG handelt es sich um einen offenen Standard des W3C. Die Daten liegen im XML-Format, einem Text-Format, vor. Damit lassen sich SVG-Dateien mit jedem beliebigen Texteditor erstellen. Weiterhin existieren beinahe für jede Programmiersprache XML-Parser, die so das Verarbeiten von SVG-Daten möglich machen.

Ein Nachteil von SVG ist im Moment sicherlich noch die teilweise schlechte Unterstützung in Webbrowsern. An der Unterstützung wird jedoch gearbeitet. Mozilla Firefox, Opera und die Entwicklerversion Webkit des Mac OS X Standardbrowser Safari bieten schon native Unterstützung. Diese ist jedoch noch nicht vollständig und nicht einheitlich. Weiterhin existiert als Browser-Plugin auch noch der Adobe SVG Viewer für Windows, Linux und Mac, allerdings wird der Support dieses Plugins am 1. Januar 2008 von Adobe eingestellt.

Ein weiteres im Internet sehr verbreitetes Vektorgrafikformat ist das Flash-Format von Adobe (früher Macromedia). Es handelt sich um ein proprietäres, binäres Format. Sein Vorteil ist die hohe Verbreitung. Adobe gibt an, dass Flash-Inhalte auf

99.3%[19] der Browser in den USA, Kanada, England, Japan und Deutschland wiedergegeben werden können.

Mit Silverlight[20] versucht auch Microsoft ein flash-ähnliches proprietäres Format auf den Markt zu bringen. Die Schwerpunkte liegen auf der Erstellung von Rich-Internet-Applications und der Integration von Video und Sound. Jedoch bietet Silverlight auch Unterstützung für Vektorgrafik.

4.2.4 Aufbau einer SVG-Datei

Eine SVG-Datei hat als Wurzelelement immer das Element `<svg>`. Darin kann als erstes durch das Element `defs` ein Bereich für allgemeine Definitionen abgesteckt werden. Dieser Bereich kann zum Beispiel JavaScript oder CSS-Angaben enthalten. Weiterhin können wiederverwendbare Objekte oder Schriften definiert werden. Nach dem `<def>`-Element folgt der eigenliche Inhalt.

4.2.5 Grundformen und -Elemente

Linien Das `line`-Element repräsentiert eine einzelne Linie. Anfangs- und Endpunkt werden über die Attribute `x1`, `y1` bzw. `x2`, `y2` festgelegt. Mit dem Attribut (oder der CSS-Eigenschaft) `stroke` muss der Linie eine Farbe zugeordnet werden.

Rechtecke Mit Hilfe des `rect`-Elementes lassen sich Rechtecke darstellen. Mit den Attributen `x`, `y` wird der Punkt der oberen linken Ecke festgelegt. Die Attribute `width` und `height` geben Breite und Höhe an. Die Linienfarbe wird wieder mit dem Attribut `stroke` festgelegt; die Füllfarbe mir dem Attribut `fill`.

Kreise Das `circle`-Element erwartet mit den Attributen `cx` und `cy` die Koordinaten des Kreismittelpunktes. Der Radius wird mit dem Attribut `r` festgelegt.

Ellipsen Anders als beim Kreis kann der Radius des `ellipse`-Elementes für x- und y-Achse mit den Attributen `rx` und `ry` getrennt festgelegt werden.

Polylinien Das `polyline`-Element erzeugt eine aus mehreren einzelnen Linien zusammengesetzte Linie. Die einzelnen Koordinaten werden nacheinander, durch Komma oder Leerzeichen getrennt, im Attribut `points` angegeben.

Polygone Polygone werden durch das `polygon`-Element erzeugt. Sie verhalten sich genauso wie Polylinien mit dem Unterschied, dass der letzte angegebene Punkt automatisch mit dem ersten verbunden wird und der Polygonzug so geschlossen wird.

Pfade Über diese Grundformen hinaus gibt es noch das `path`-Element. Mit diesem lassen sich beliebige Formen erzeugen. Der Verlauf des Pfades wird als Serie von Anweisungen über das Attribut `d` angegeben. Folgende Anweisungen sind möglich:

[19] Daten vom Juni 2007 `http://www.adobe.com/products/player_census/flashplayer`
[20] `http://www.microsoft.com/silverlight/`

- **moveto (M, m):** Mit dieser Anweisung wird der Stift an die den nachfolgenden Koordinaten entsprechende Stelle bewegt (M200, 300). Große Buchstaben bedeuten immer absolute Positionsangaben, kleine Buchstaben relative Angaben zum vorigen Punkt.
- **lineto (L, l):** Mit dieser Anweisung wird eine Linie vom aktuellen Punkt zum nachfolgenden Punkt gezeichnet (L30, 40).
- **curveto (Q, q; C, c):** Mit dieser Anweisung lassen sich quadratische und kubische Bezierkurven zeichnen. Quadratische Bezierkurven benötigen einen Startpunkt, einen Endpunkt und einen Stützpunkt, kubische Bezierkurven benötigen zwei Stützpunkte. Als Startpunkt wird der aktuelle Punkt genommen. Es folgt die Angabe der Stützpunkte und dann des Endpunktes (C10,10 30,30 50,40).
- **arc (A, a):** Um eine elliptische Bogenkurve zu zeichnen, werden nach der Anweisung A (oder a) sieben Werte erwartet:
  1. Radius in Richtung der x-Achse
  2. Radius in Richtung der y-Achse
  3. Rotation der x-Achse der Ellipse in Grad
  4. large-arc-flag, das angibt, ob der kurze (0) oder der lange (1) Weg über die Ellipse gewählt werden soll.
  5. sweep-flag , das angibt, ob gegen (0) oder im Uhrzeigersinn (1) gezeichnet werden soll
  6. x-Koordinate des Endpunktes
  7. y-Koordinate des Endpunktes
- **closepath (Z, z):** Mit Hilfe der closepath-Anweisung wird der Pfad mit einer Linie geschlossen.

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg">
    <line x1="20" y1="30" x2="140" y2="40" stroke="black"/>
    <line x1="50" y1="40" x2="145" y2="60" stroke="blue" />
    <rect x="10" y="80" width="100" height="90"
        stroke="black" fill="none"/>
    <rect x="150" y="70" width="120" height="80"
        stroke="red" fill="yellow" />
    <circle cx="70" cy="250" r="60" stroke="#ff0000"
        fill="#00ff00"/>
    <ellipse cx="200" cy="240" rx="100" ry="40"
        stroke="#ff00ff" fill="none"/>
    <polyline points="10,350 300,310 360,250 290,40"
        stroke="black" fill="none" />
    <polygon points="20,370 100,350 250,400"
        stroke="black" fill="none" />
    <path d="M30,400 180,40 l-50,60 L20,450"
        stroke="black" fill="none" />
    <path d="M100,400 Q200,520 400,350"
        stroke="black" fill="none" />
    <path d="M100,480 c50,50 90,-50 250,-20"
        stroke="black" fill="none" />
    <path d="M20,490 A200,100,10,1,0,500,500 Z"
        stroke="black" fill="none" />
</svg>
```

Quellcode 4.5: SVG-Datei mit Beispielen für die Grundformen

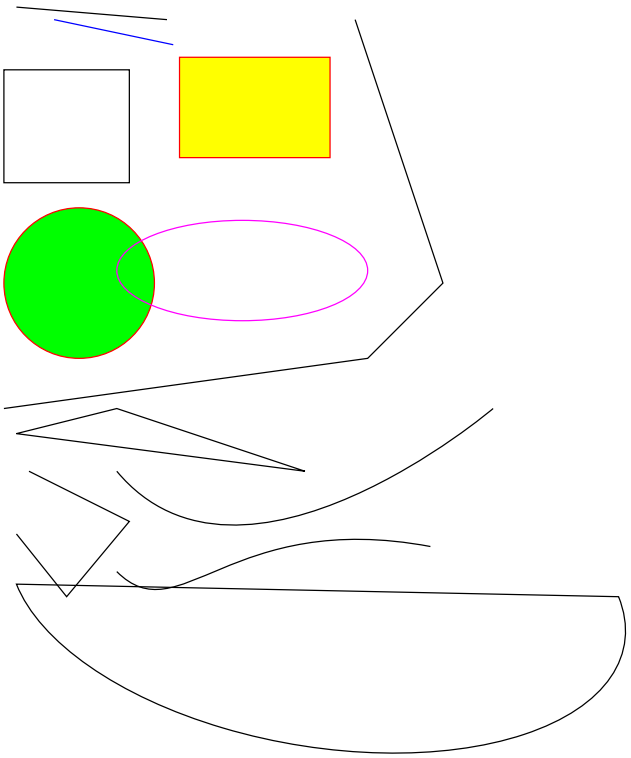


Abbildung 4.2: Gerenderte SVG-Datei (vgl. Quellcode 4.5)

4.2.6 Scripting

Mit Hilfe von JavaScript (vgl. Abschnitt 4.3) kann SVG-Applikationen Interaktion hinzugefügt werden. Zum Beispiel können Objekte über so genannte Event-Handler auf vom Benutzer ausgelöste Ereignisse (Mausklick, Tastaturanschlag oder ähnliches) reagieren. Ein einfaches Beispiel findet man in Quellcode 4.6

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg">
    <defs>
        <script type="text/javascript"><![CDATA[
            function resize(evt){
                var circle = evt.target
                var r = parseInt(circle.getAttribute('r'));
                if(r < 100){
                    r = r + 10;
                } else {
                    r = 10;
                }
                circle.setAttribute('r',r);
            }
        ]]></script>
    </defs>
    <rect x="0" y="0" width="400" height="200" fill="#cccccc"/>
    <circle cx="200" cy="100" r="10" stroke="black"
        fill="yellow" onclick="resize(evt)" />
</svg>
```

Quellcode 4.6: SVG-Datei mit Beispielen für die Grundformen

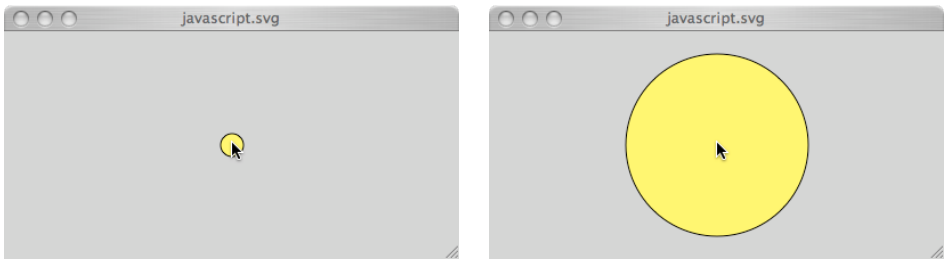


Abbildung 4.3: Gerenderte SVG-Datei (Listing 4.6 vor und nach ein paar mal Klicken)

4.3 JavaScript

JavaScript ist eine objektorientierte Skriptsprache, die 1995 von Netscape eingeführt wurde. Später wurde sie von der ECMA (European Computer Manufacturers Association) unter dem Namen ECMAScript standardisiert. Wurde sie damals entwickelt, um HTML-Seiten im Browser etwas dynamischer zu gestalten, wird sie heute vielfältiger für verschiedenste Zwecke in Programmen eingesetzt. Zum Beispiel können PDF-Dokumente mit JavaScript versehen oder OpenOffice Makros in JavaScript programmiert werden. Java besitzt seit der Version 6.0 den JavaScript-Interpreter Rhino[21], der es ermöglicht, JavaScript mit Java auszuwerten[22].

Obwohl das DOM programmiersprachenunabhängig ist, wird wegen der sehr guten JavaScript-Unterstützung in den Browsern meist JavaScript als Sprache zum Zugriff auf das DOM verwendet. Auch das in Abschnitt 4.6 beschriebene Google Web Toolkit macht ausgiebig von JavaScript Gebrauch.

JavaScript ist mit eine Kerntechnologie dieser Arbeit, da die ganze Webanwendung auf JavaScript basiert. Trotzdem wird es wegen des Google Web Toolkits nur an wenigen Stellen direkt benutzt.

Ein Beispiel für JavaScript findet sich schon in Abschnitt 4.2.6 in Quellcode 4.3.

[21] `http://www.mozilla.org/rhino/`
[22] vgl. [Ullenboom, 2007] S. 506f

4.4 AJAX

AJAX steht für *Asynchronous Javascript And Xml*. Der Begriff tauchte das erste Mal im Februar 2005 in einem Aufsatz von Jesse James Garrett auf[23], in dem er über einen neuen Ansatz für Internetanwendungen schreibt, der sich mehr an der Funktionsweise von Desktopanwendungen orientiert.

4.4.1 Das Konzept

Angenommen man hat eine Internetseite in dem häufig verwendeten Layout mit einer Titelzeile oben und einer Navigationsleiste links. Klickt man nun auf einen Link in der Navigationsleiste, wird die neue Seite vom Webserver in den Browser geladen. Die Titelzeile und die Navigationsleiste sind wahrscheinlich wieder identisch, werden aber trotzdem auch neu geladen.

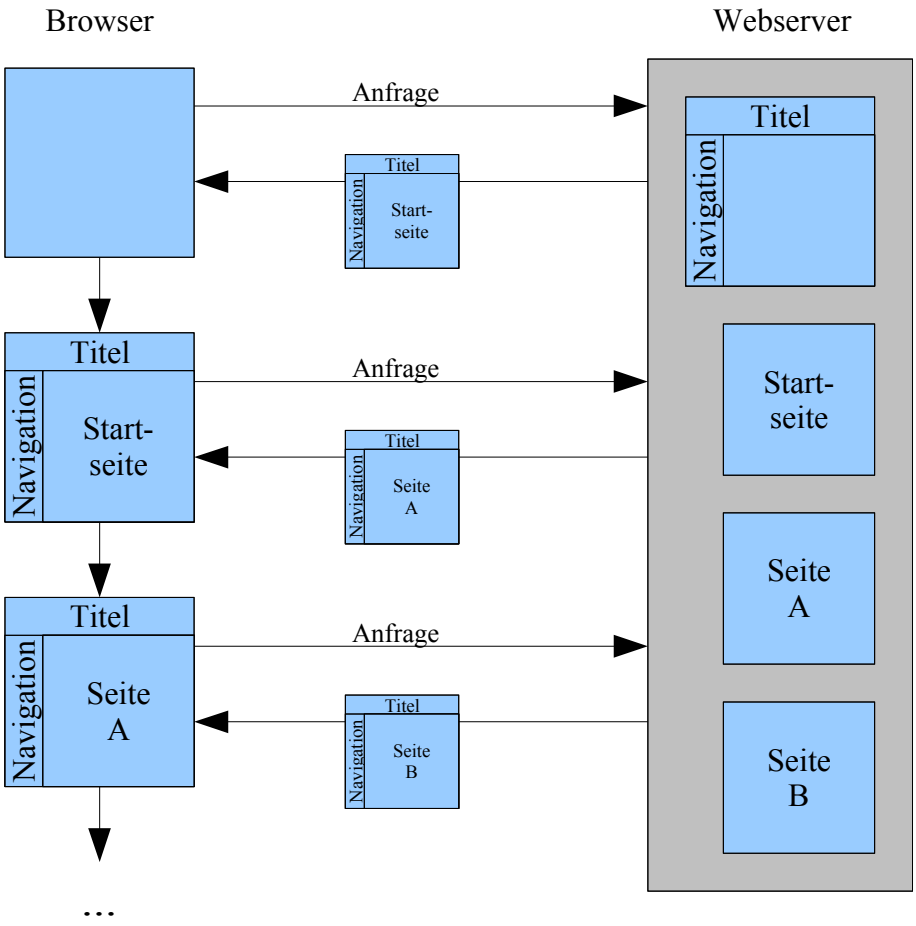


Abbildung 4.4: Kommunikation mit dem Webserver ohne AJAX: Es werden immer komplette Webseiten übertragen.

Bei einer Desktopanwendung ist dies anders. Hier werden immer nur die Teile der Anwendung aktualisiert, die auch von Änderungen betroffen sind. AJAX

[23] siehe [Garrett, 2005]

versucht dieses Verhalten so weit wie möglich auf Webanwendungen zu übertragen und immer nur die veränderten Inhalte neu zu laden. Diese werden dann ohne einen kompletten Neuaufbau der Seite eingefügt oder gegen andere Inhalte ausgetauscht.

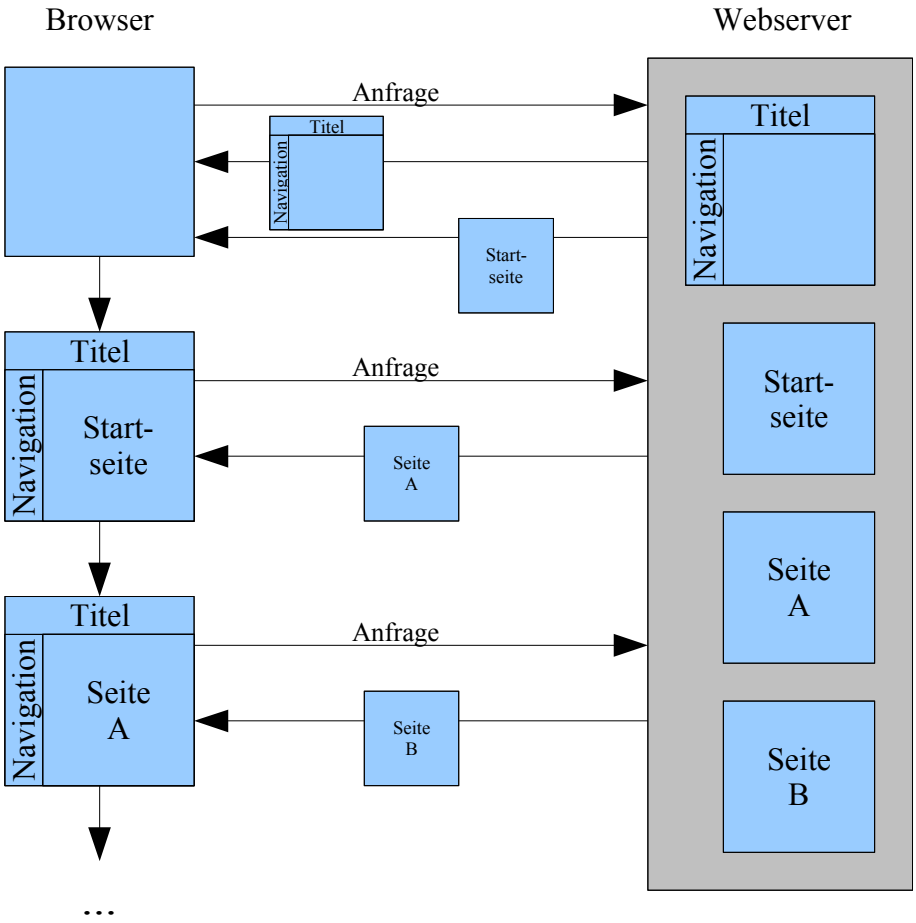


Abbildung 4.5: Kommunikation mit dem Webserver mit AJAX: Es werden nur die benötigten Teile der Seite übertragen.

4.4.2 Technische Grundlagen für AJAX

AJAX bezeichnet nicht eine neue Technologie, sondern das Zusammenspiel von schon vorhandenen Technologien auf eine neue Weise. Im Wesentlichen sind dies folgende[24]:

- HTML bzw. XHTML und CSS für die Anzeige im Browser
- DOM zur Manipulation der Anzeige
- XML als Format zu Datenaustausch
- XMLHttpRequest zum asynchronen Datenaustausch mit dem Server
- JavaScript zum Verbinden der genannten Technologien

Den Grundstein für AJAX legte Microsoft im Jahre 1998. Damals wurde in den Internet Explorer 5 eine neue Technologie namens XMLHTTP integriert. Microsoft

[24] vgl. [Garrett, 2005]

wollte seinen Benutzern das eMail- und Organizer-Programm Outlook auch im Internet verfügbar machen. Daher versuchte man, das Programm komplett im Browser nachzubilden. Das Nachbilden der Oberfläche war kein Problem, jedoch störte das ständige Neuladen der kompletten Oberfläche, wenn ein Datenaustausch mit dem Server nötig war. Hier kam jetzt die neue, für diesen Zweck entwickelte Technologie zum Einsatz, um das ständige Neuladen zu vermeiden.

War es in den Anfangszeiten noch so, dass XMLHTTP nur im Internet Explorer zur Verfügung stand, implementieren mittlerweile fast alle modernen Browser eine sehr ähnliche Technik namens XMLHttpRequest.

4.4.3 Synchron vs. Asynchron

Die herkömmliche Kommunikation zwischen Client und Server ist durch das Laden kompletter Seiten gekennzeichnet. Die Verbindung zum Server wird vom Client nur dann aufgenommen, wenn eine neue Seite angefordert wird. Der Benutzer stößt diese Verbindung z.B. durch das Klicken auf einen Link oder das Absenden eines Formulars an. Wird eine neue Seite angefordert, ist das meist mit einer Wartezeit verbunden, da alle Daten auf einmal geladen werden. Diese Art der Kommunikation wird *synchron* genannt.

Bei der *asynchronen* Kommunikation wird die Verbindung zum Server zwar häufig auch aufgrund eines Klicks auf einen Button oder auf einen Link hergestellt, dies muss aber nicht so sein. Der Client kann auch völlig unabhängig vom Benutzer Kontakt zum Server aufnehmen. Viel wichtiger ist aber, dass die Seite hierfür nicht komplett neu geladen werden muss, sondern einzelne Bereiche dynamisch geändert werden können. Da die Kommunikation technisch gesehen ähnlich abläuft, also auch per HTTP über eine GET- oder POST-Anfrage, können auch hier Wartezeiten entstehen (Latenz). Jedoch muss der Benutzer nicht auf den Aufbau einer neuen Seite warten, da die vom Server empfangenen Daten dynamisch in die Seite eingebunden werden, wenn sie verfügbar sind. Die Seite ist in der Zwischenzeit nicht blockiert. Ein einfaches Beispiel für das asynchrone Nachladen ist Google Suggest[25]. Immer wenn ein Buchstabe eingetippt wird, wird das bisherige Suchwort an den Server übermittelt. Dieser erstellt eine Liste mit Vorschlägen für das komplette Suchwort und liefert auch noch die Anzahl der Suchtreffer für dieses Suchwort mit. Die Liste wird dann im Browser angezeigt.

[25] http://labs.google.com/suggest

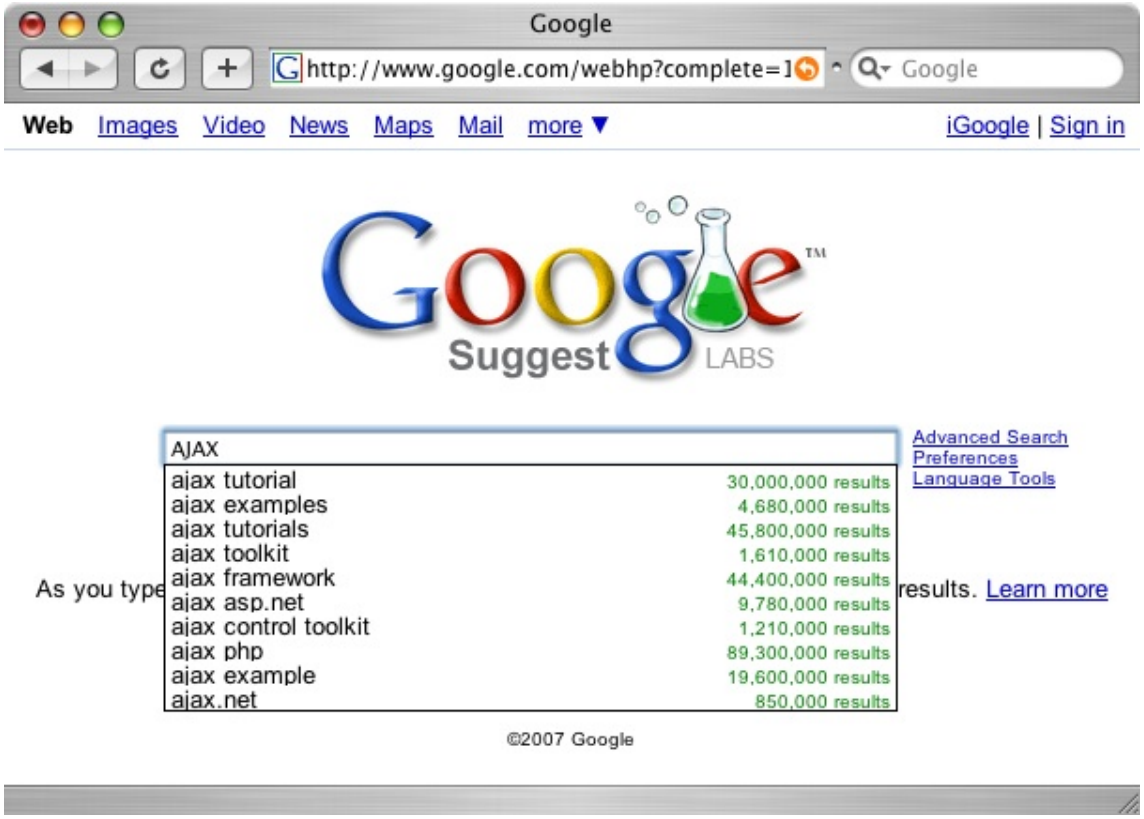


Abbildung 4.6: Google Suggest

4.5 Java

Java ist eine objektorientierte Programmiersprache, die sich durch einige zentrale Eigenschaften auszeichnet.

Java verfolgt die Philosopie: "write once, run anywhere"[26]. Ein in Java geschriebener Quelltext wird vom Java-Compiler nicht in Maschinencode, sondern in plattformunabhängigen Bytecode übersetzt. Damit dieser Bytecode ausgeführt werden kann, wird eine virtuelle Maschine benötigt, die diesen Bytecode interpretiert. Da das reine Interpretieren zu Geschwindigkeitsproblemen führt, enthält die virtuelle Maschine einen Just-In-Time-Compiler, der zur Laufzeit des Programms Anweisungen in Maschinencode übersetzt. Mit Hilfe dieser Technik wird die Geschwindigkeit erhöht, so dass sie die anderer, direkt in Maschinencode compilierende Sprachen erreicht.

4.5.1 Java-Applets

Diese Eigenschaft ist besonders für Applets wichtig. Applets sind Java-Programme, die auf Webseiten eingebettet werden. Sie brauchen für die Ausführung einen Webbrowser und die virtuelle Maschine. Die wichtigsten Unterschiede zu normalen Anwendungen sollen hier kurz beschrieben werden:

- Ein Applet wird nicht über eine `main()`-Methode gestartet. Die Klasse des Hauptprogrammes erbt entweder von der Klasse `Applet` oder `JApplet`. Für Initialisierungen kann die Klasse die Methode `init()` überschreiben, die einmal ausgeführt wird, wenn die Seite vom Browser geladen wird. Nach der Initialisierung folgen immer Aufrufe der überschreibbaren Methoden `start()` bzw. `stop()`, je nachdem, ob das Applet gerade sichtbar wird oder wieder aus dem sichtbaren Bereich verschwindet. Beim Verlassen der Seite wird die Methode `destroy()` aufgerufen. Falls diese Methode überschrieben wird, können hier zum Beispiel Ressourcen freigegeben werden.
- Applets haben nicht die gleichen Rechte wie normale Applikationen. Sie laufen in einer sogenannten Sandbox. Die Sandbox besteht aus einem besonderen Klassenlader und dem `SecurityManager`. Der Klassenlader kontrolliert, dass das Applet keine Klassen von einem fremden Server lädt. Der `SecurityManager` kontrolliert den Zugriff auf das System. Er unterbindet zum Beispiel den Zugriff auf lokale Dateien. Ebenso sorgt er dafür, dass das Applet keine Netzwerkverbindungen zu Fremdservern aufbaut. Verbindungen sind nur zu dem Server erlaubt, von dem das Applet geladen wurde. Applets dürfen auch keine Programme vom lokalen Rechner ausführen. Da alle Applets gemeinsam in einer virtuellen Maschine laufen, muss sichergestellt werden, dass das Applet nur auf eigene Threads zugreift. Auch der Zugriff auf Systemeigenschaften (über `System.getProperty(String)`) ist eingeschränkt.
- Um einem Applet doch zusätzliche Rechte zu geben, ist es möglich, es zu signieren. Mit Hilfe eines Zertifikates kann die Echtheit des Applets überprüft werden, und der Benutzer kann dem Applet dann die benötigten Rechte gewähren.

[26] [Ullenboom, 2007] S. 63

4.5.2 Java Webapplikationen

Java Webapplikationen bieten eine Möglichkeit, mit Java dynamische Webseiten zu generieren. Es gibt von Hause aus zwei Möglichkeiten, die Inhalte zu erzeugen.

Servlets Servlets sind Klassen, die von einer Servlet-Basisklasse wie `GenericServlet` oder `HTTPServlet` erben. Sie überschreiben eine der Methoden `doGet()`, `doPost()` oder `service()`. Die Methoden werden dann, je nachdem, ob es sich um eine GET- oder POST-Anfrage handelt, aufgerufen. Wird die Methode `service()` überschrieben, wird diese bei jeder Art von Anfrage aufgerufen. Alle diese Methoden werden mit einem `ServletRequest`- und `ServletResponse`-Objekt als Argumente aufgerufen. Darüber kann man sich Zugriff auf übergebene Parameter, die Benutzersession oder den OutputStream, in den die zu sendenden Daten geschrieben werden, verschaffen.

Java Server Pages Einen anderen Ansatz verfolgen Java Server Pages (JSP). Möchte man mit Servlets eine HTML-Seite generieren, muss der gesamte HTML-Quelltext mühsam mittels `print()`-Anweisungen ausgegeben werden. Dies stellt oft zum Beispiel für Webdesigner ein Problem dar, da sie sich auf das Layout der Seite konzentrieren und mitunter gar kein Java beherrschen. Java Server Pages gehen jetzt den umgekehrten Weg. Es sind HTML-Dateien, in die Java-Befehle eingestreut werden können. Jede normale HTML-Seite stellt schon eine Java Server Page dar. Hierauf soll im weiteren nicht näher eingegangen werden, da Java Server Pages in dieser Arbeit nicht verwendet werden.

```
package server;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MyServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req,
            HttpServletResponse res)
                throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Quellcode 4.7: Beispiel für ein Servlet, dass eine kurze HTML-Seite generiert

Um Servlets ausführen zu können, benötigt man einen Servlet-Container. Dieser nimmt die Anfragen der Browser an und leitet sie an die entsprechenden Servlets weiter. Er übergibt auch die Ein- und Ausgabeströme und übernimmt die Verwaltung der Sessions.

Eine Webapplikation hat eine genau definierte Struktur. Im Wurzelverzeichnis liegen alle Dateien, die von außen erreichbar sind, wie zum Beispiel statische Seiten, JSP oder Grafiken. Es können natürlich Unterordner erstellt werden, die dann auch von außen erreichbar sind. Zusätzlich gibt es noch ein Verzeichnis namens *WEB-INF*, das nicht nach außen freigegeben wird. In diesem Verzeichnis befindet sich unter anderem der Deployment Descriptor.

Der Deployment-Descriptor ist eine XML-Datei namens `web.xml`. Hier kann das Servlet-Mapping festgelegt werden, d.h. den Servlets werden URL zugewiesen, mit denen sie vom Browser aus erreichbar sind.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <display-name>Webapplication</display-name>
    <servlet>
        <description></description>
        <display-name>MyServlet</display-name>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>server.MyServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

Quellcode 4.8: Deployment-Descriptor für das Beispiel-Servet (Listing 4.7)

Zusätzlich kann das *WEB-INF*-Verzeichnis noch die Unterverzeichnisse *classes* und *lib* besitzen. Das Verzeichnis *classes* kann Klassen-Dateien und das Verzeichnis *lib* jar-Archive enthalten. Diese werden dann beim Start mit in den Classpath der Anwendung aufgenommen.

4.6 Google Web Toolkit

Das Google Web Toolkit ist eine Java-Entwicklungsumgebung für Ajax-Webanwendungen. Im Gegensatz zu anderen AJAX-Toolkits programmiert man beim Google Web Toolkit in Java und nicht in JavaScript. Ein Compiler übersetzt dann die Java-Quelltexte nach JavaScript.

Die aktuell verwendete Version ist Version 1.4 vom 28. August 2007. Seit der Version 1.3 steht das komplette Google Web Toolkit unter der Apache 2.0 Lizenz und ist Open Source.

4.6.1 Aufbau des Google Web Toolkit

Das Google Web Toolkit besteht aus vier Hauptkomponenten. Es gibt zwei Klassenbibliotheken und einen GWT Hosted Browser und den Java-nach-JavaScript Compiler.

JRE Emulation Library Die JRE Emulation Library enthält JavaScript-Implementationen der wichtigsten Java-Klassen. Dazu gehören die meisten Klassen aus den Paketen `java.lang` und `java.util`. Eigene Klassen müssen von diesen Klassen abgeleitet werden.

GWT Web UI Library Diese Bibliothek enthält alle Klassen, die für die Programmierung der Weboberfläche nötig sind, wie zum Beispiel Buttons, Textboxen und verschiedene Panels. Weiterhin sind Klassen zum Zugriff auf den Browser und die Datenübertragung zwischen Browser und Server enthalten.

GWT Compiler Der GWT Compiler bildet den Kern des Toolkits. Mit ihm lassen sich die Java-Quelltexte nach JavaScript übersetzen. Es wird danach zum Ausführen der Anwendung kein Browser-Plugin benötigt, da es sich nur um HTML und JavaScript handelt. Für Anwendungen ohne eine Form von Remote Procedure Calling ist auch keine spezielle Servertechnologie nötig.

GWT Hosted Browser Der GWT Hosted Browser ermöglicht es, die Anwendung als Java-Bytecode ohne Übersetzung nach JavaScript auszuführen. Hierdurch hat man die Möglichkeit, die Webanwendung wie eine normale Java-Anwendung zu debuggen. Die Ausführung geschieht mittels eines modifizierten Browser-Widgets aus der SWT-Bibliothek[27]. Das modifizierte Widget setzt auf den jeweiligen Systembrowser (Safari unter Mac OSX, Mozilla unter Linux und Internet Explorer unter Windows) und benötigt plattformabhängigen Code. Deshalb ist es im Moment nur für die verbreitetesten Plattformen Mac OS X, Linux mit GTK und Windows XP/2000 erhältlich. Alle anderen Bestandteile sind vollständig in Java programmiert und daher plattformunabhängig.

4.6.2 Aufbau einer GWT-Anwendung

Eine Standard-GWT-Anwendung besteht aus einem Wurzelverzeichnis mit einer Konfigurationsdatei (Modul), einem Paket `client`, das die Java-Klassen enthält,

[27] `http://www.eclipse.org/swt/`

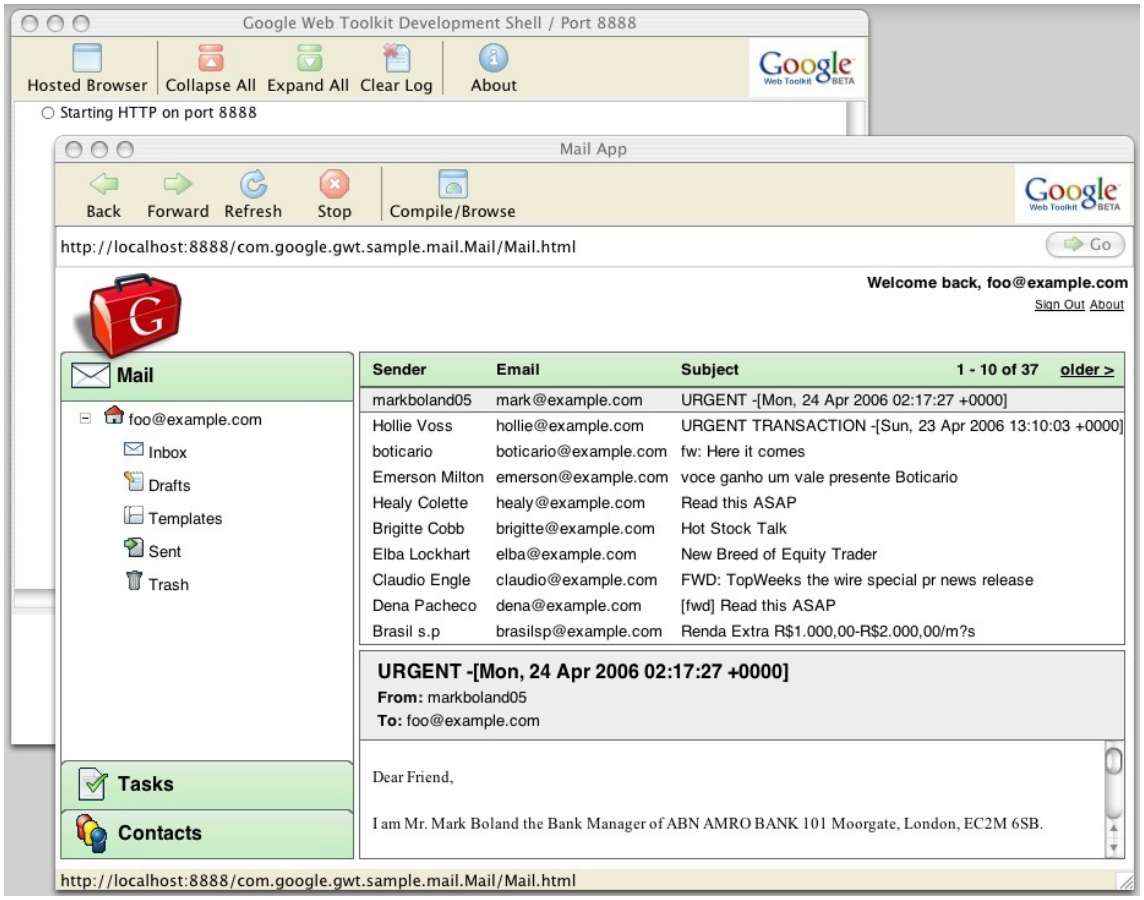


Abbildung 4.7: GWT Hosted Browser mit Debugging-Konsole

die später nach JavaScript übersetzt werden, eventuell einem Paket `server` mit serverseitigen Klassen und einem Verzeichnis `public`, das Dateien enthält, die später mit veröffentlicht werden sollen, so zum Beispiel die HTML-Datei, in die die Anwendung eingebettet werden soll oder Bilder. Dem GWT liegt ein Skript bei, das automatisch die benötigten Verzeichnisse und Dateien erstellt.

Module

Über Module wird die GWT-Anwendung konfiguriert. Es sind XML-Dateien mit der Endung `.gwt.xml`. Folgende Einstellungen können zum Beispiel vorgenommen werden:

- Über das Element `<entry-point>` wird der Einstiegspunkt der Anwendung festgelegt.
- Mit Hilfe von `<source>`-Elementen können Pakete für die Übersetzung nach JavaScript vorgesehen werden. Wird kein `<source>`-Element angegeben, so wird standardmäßig das Paket `client` mit seinen Unterpaketen übersetzt.
- Über `<public>`-Elemente können Verzeichnisse festgelegt werden, dessen Inhalte mit in den öffentlichen Ordner der Webanwendung kopiert werden. Wird kein `<public>`-Element definiert, ist dies automatisch der Ordner `public`.

- Mit dem Element `<inherits>` können andere Module eingebunden werden. Alle im eingebundenen Modul gesetzten Einstellungen werden dann übernommen. Zum Beispiel werden über `<inherits name="com.google.gwt.user.User"/>` üblicherweise die Grundklassen des Google Web Toolkits eingebunden.
- Für den Hosted Mode können Servlets mit Hilfe des `<servlet>`-Elementes eingebunden werden. Der Klassenname und der Name des Pfades, unter dem das Servlet ereichbar sein soll, muss angegeben werden.

```
<module>
    <!-- Inherit the core Web Toolkit stuff.                    -->
    <inherits name='com.google.gwt.user.User'/>
    <!-- Specify the app entry point class.                     -->
    <entry-point class='client.HelloWorld'/>
</module>
```

Quellcode 4.9: Minimale Modul-Datei für ein Projekt mit Standard-Struktur

HTML-Einstiegsdatei

Das Google Web Toolkit benötigt eine HTML-Rumpfdatei, in die es eingebettet werden kann. Diese Datei kann schon Inhalte enthalten, kann aber auch einen leeren Body haben. Falls die Datei schon Inhalt hat, benötigt das GWT ein HTML-Element als Einstiegspunkt. Dies kann zum Beispiel ein leeres `<span>`- oder `<div>`-Element sein. Dem Einstiegselement muss eine Id zugewiesen werden, die dann aus der Web Toolkit-Anwendung identifiziert wird.

```
<html>
    <head>
        <title>Wrapper HTML for HelloWorld</title>
        <script language='javascript'
                src='HelloWorld.nocache.js'></script>
    </head>
    <body>
        <!--OPTIONAL: include this if you want history support-->
        <iframe src="javascript:''" id="__gwt_historyFrame"
                    style="width:0;height:0;border:0"></iframe>
        <h1>HelloWorld</h1>
        <p>
            This is an example of a host page for the HelloWorld
            application. You can attach a Web Toolkit module
            to any HTML page you like,  making it easy to add
            bits of AJAX functionality to existing pages
            without starting from scratch.
        </p>
        <table align=center>
            <tr>
                <td id="slot1"></td><td id="slot2"></td>
            </tr>
        </table>
    </body>
</html>
```

Quellcode 4.10: HTML-Rumpf-Datei

Java-Einstiegsklasse

Das Google Web Toolkit benötigt eine Klasse, die das Interface `EntryPoint` implementiert. Dieses Interface schreibt eine Methode `onModuleLoad()` vor. Sie wird beim Laden des Moduls, das diese Klasse als Einstiegspunkt festgelegt hat, automatisch aufgerufen.

```
package client;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.ClickListener;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;
/**
 * Entry point classes define <code>onModuleLoad()</code>.
 */
public class HelloWorld implements EntryPoint {
  /**
   * This is the entry point method.
   */
  public void onModuleLoad() {
    final Button button = new Button("Click me");
    final Label label = new Label();
    button.addClickListener(new ClickListener() {
      public void onClick(Widget sender) {
        if (label.getText().equals(""))
          label.setText("Hello World!");
        else
          label.setText("");
      }
    });
    RootPanel.get("slot1").add(button);
    RootPanel.get("slot2").add(label);
  }
}
```

Quellcode 4.11: Einstiegsklasse mit `onModuleLoad()`

4.6.3 Übersicht über die wichtigsten Klassen und Funktionen des GWT

Widgets und Panels

Dem Google Web Toolkit liegt mit der User-Interface Library eine umfangreiche Sammlung an Widgets und Panels bei. Die Widgets basieren teilweise auf den vorhandenen HTML-Elementen, zusätzlich gibt es aber auch noch komplizierte Widgets, wie zum Beispiel ein `MenuBar`-Widget zum Erzeugen eines Menüs oder ein `Tree`-Widget für die Darstellung von Baumstrukturen. Die Widget-Eigenschaften lassen sich ähnlich wie bei SWING-Komponenten mit get- und set-Methoden auslesen und setzen.

Für das Layout der Seite stehen verschiedene Panels zur Verfügung. Durch die Möglichkeit, die Panels ineinander zu verschachteln, lassen sich beliebig umfangreiche Layouts erstellen. Widgets und Panels können mithilfe einer `add()`-Methode und gegebenenfalls zusätzlichen Parametern hinzugefügt werden. Intern werden die einfachen Panels meist über HTML-Tabellen realisiert.

Zusätzlich zu den normalen Panels gibt es auch noch Popup- und Dialog-Boxen, die den normalen Seiteninhalt überlagern. Wird das entsprechende Event zum Anzeigen des Popups oder Dialogs zum Beispiel durch Klicken auf einen Button oder beim Überfahren eines Elementes mit der Maus ausgelöst, werden die Inhalte durch Einfügen eines `<div>`-Elementes in das DOM per JavaScript angezeigt. Das `<div>`-Element wird mittels CSS positioniert. Zum Schließen des Popups oder Dialogs wird das Element wieder aus dem DOM entfernt.

Eine weitere Besonderheit stellt das `RootPanel` dar. Es steht an der Spitze der Panel-Hierarchie und stellt die Schnittstelle zur HTML-Seite dar. Das Standard-RootPanel, das man mit der Methode `RootPanel.get()` erhält, kapselt das `<body>`-Element des HTML-Dokumentes. Möchte man ein RootPanel eines anderen Elements, so bekommt man dies mit der Methode `RootPanel.get(id)`, wobei mit `id` die Id des HTML-Elementes als String übergeben wird. Dadurch ist es möglich, die Anwendung in ein bestehendes Seitenlayout einzufügen.

Eine Übersicht über die wichtigsten verfügbaren Panels und Widget findet man in der Widget-Galerie[28].

Die Formatierung der Elemente geschieht üblicherweise über Cascading Style Sheets (CSS). Jedem Widget kann über die Methode `setStyleName()` eine Style Sheet Klasse zugewiesen werden. Es ist aber auch möglich, durch eine Regel `.gwt-Widgetname` alle Widgets eines Typs zu formatieren. Die CSS-Datei kann im Kopf der HTML-Datei wie üblich eingebunden werden.

Kommunikation

Für die Kommunikation mit dem Server stehen zwei verschiedene Möglichkeiten zur Verfügung. Zum einen gibt es im Paket `com.google.gwt.user.client` eine Klasse `HTTPRequest`, mit der es möglich ist, asynchrone Anfragen an den Server nach Art des JavaScript HTTPRequests zu machen. Die Klasse besitzt die Methoden `asyncGet(String, ResponseTextHandler)` und `asyncPost(String, String, ResponseTextHandler)`, mit denen entweder eine GET- oder eine POST-Anfrage abgesetzt wird. Als Argumente wird die URL als String und ein ResponseTextHandler als Callback-Objekt erwartet, beim POST-Request werden als zweites Argument noch die POST-Daten als String übergeben. Das Interface `ResponseTextHandler` schreibt nur eine Methode `onCompletion(String)` vor, der bei erfolgreichem Request das Ergebnis des Servers übergeben wird. Diese Methode bietet die Möglichkeit, mit jeder Art von Web-Server zu kommunizieren.

Die zweite Möglichkeit ist wesentlich komfortabler, setzt jedoch einen Java-

[28] `http://code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.UserInterface.WidgetGallery.html`

Servlet-Container voraus. Die Methode ermöglicht Remote Procedure Calls (RPC) und ähnelt der Remote Method Invocation (RMI) von Java. Es können direkt Methoden, die der Server bereit stellt, aufgerufen werden. Im Gegensatz zu RMI sind diese Aufrufe aber asynchron.

Um Methoden entfernt verfügbar zu machen, benötigt man auf Client-Seite ein Interface, das von `RemoteService` erbt und die Methoden deklariert. Auf der Server-Seite muss nun eine Klasse, die von `RemoteServiceServlet` erbt, dieses Interface implementieren.

```
package client;
import com.google.gwt.user.client.rpc.RemoteService;
public interface RemoteProcedureCall extends RemoteService{
    public int add(int x, int y);
}
```

Quellcode 4.12: Beispiel-Interface

```
package server;
import client.RemoteProcedureCall;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
public class RemoteProcedureCallImpl extends
        RemoteServiceServlet implements RemoteProcedureCall {
    public int add(int x, int y) {
        return x + y;
    }
}
```

Quellcode 4.13: Beispiel-Servlet

Die Klasse `RemoteServiceServlet` erbt von `HTTPServlet`, und die Kommunikation läuft über POST-Requests. Falls der Client Komprimierung unterstützt, wird der Datenstrom mittels GZip[29] komprimiert, um das zu übertragende Datenvolumen zu verringern. Das Servlet muss über den Servlet-Container unter einem frei wählbaren URL verfügbar gemacht werden.

Auf der Client-Seite muss zusätzlich zu dem synchronen Interface noch ein asynchrones Interface erstellt werden. Das Interface muss den Namen *SynchronesInterfaceAsync* tragen und im gleichen Paket liegen. Für jede Methode im synchronen Interface muss es eine Methode im asynchronen Interface mit gleichem Namen geben. Im Unterschied zum synchronen Interface hat diese Methode jedoch keinen Rückgabewert und bekommt als zusätzlichen Parameter ein Objekt, das das Interface `AsyncCallback` implementiert übergeben. Das Interface `AsyncCallback` deklariert die beiden Methoden `onSuccess(Object result)` und `onFailure(Throwable caught)`, die bei Erfolg oder Fehlschlagen des Methodenaufrufs ausgeführt werden. Das Objekt `result` enthält den eigentlichen Rückgabewert, `caught` Informationen über den Fehlschlag des Aufrufes.

[29] `http://www.gzip.org/`

```
package client;
import com.google.gwt.user.client.rpc.AsyncCallback;
public interface RemoteProcedureCallAsync {
    public void add(int x, int y, AsyncCallback ac);
}
```

Quellcode 4.14: Beispiel-Servlet

Das Aufrufen einer entfernten Methode läuft in 4 Schritten ab:

1. Die statische Methode `GWT.create()` liefert ein Objekt vom Typ des asynchronen Interfaces. Als Argument muss der Methode das Klassenobjekt des synchronen Interfaces übergeben werden.
2. Die URL für das Servlet wird angeben.
3. Das Callback-Objekt, das benachrichtigt werden soll, wird erstellt.
4. Mit dem Callback-Objekt kann die asynchrone Methode aufgerufen werden.

```
RemoteProcedureCallAsync rpc = (RemoteProcedureCallAsync)
                GWT.create(RemoteProcedureCall.class);
ServiceDefTarget sdf = (ServiceDefTarget) rpc;
sdf.setServiceEntryPoint(GWT.getModuleBaseURL() + SERVLET_NAME);
rpc.add(1, 2, new AsyncCallback() {
    public void onSuccess(Object result) {
        Integer sum = (Integer) result;
        Window.alert(sum.toString());
    }
    public void onFailure(Throwable caught) {
    }
});
```

Quellcode 4.15: Beispiel-Aufruf

5 Umsetzung als Java-Applet

Der naheliegende Ansatz, ein Web-Interface für interaktive Notengrafik zu entwickeln, basiert wegen der vorhandenen Komponenten des MUSITECH-Projektes auf Java-Applets. Dieser Ansatz soll im folgenden näher erläutert werden.

5.1 Vorausetzungen

Bei der Entwicklung eines Web-Interfaces auf Applet-Basis kann auf die vorhandenen Komponenten des MUSITECH-Projektes zurückgegriffen werden. Dies sind im einzelnen außer dem Objektmodell, das als Grundlage zur Speicherung der musikalischen Information dienen soll, das Paket `de.uos.fmt.musitech.score.gui` rund um die Klasse ScorePanel, die für die Anzeige der Noten zuständig ist, und die Module zum Abspielen der Daten per MIDI.

MUSITECH unterstützte bisher nur die Anzeige von Notengrafik. Deshalb musste zunächst ein Panel entwickelt werden, dass auch Manipulationen am Notenbild zulässt. Da im Paket `de.uos.fmt.musitech.score.gui` einige Änderungen notwendig waren, wurde ein Branch in das Paket `score.elements` gemacht.

5.2 Anzeige von Notengrafik in MUSITECH

Die Klasse `ScorePanel` dient der Anzeige von Notengrafik. Sie erbt von `JLayeredPane` und ist damit eine SWING-Komponente, die in SWING-Anwendungen verwendet werden kann. Mit der Methode `setNotationSystem(NotationSystem)` kann der Klasse ein `NotationSystem` übergeben werden. Aus diesem wird dann eine grafische Darstellung berechnet und angezeigt.

Für die Darstellung der Noten wird ein Objektbaum aufgebaut, der die Partitur hierarchisch darstellt. Basisklasse aller anzeigbaren Partiturelemente ist das `ScoreObject`. Alle anderen Objekte bauen darauf auf. Die Klasse `ScoreContainer`, die ebenfalls auf `ScoreObject` aufbaut, ist Basis für alle Klassen, die anderen `ScoreObject`s aufnehmen können und stellt hierfür Funktionalität zur Verfügung.

Mit Hilfe der Klasse `ScoreMapper` wird ein `NotationSystem` auf dieses Objektmodell abgebildet. Zur Ausrichtung der Elemente wird die Methode `arrange(int)` (eventuell in mehreren Durchgängen, `int` gibt dann die Nummer des Durchgangs an) aufgerufen. Diese Methode, die schon in `ScoreObject` vorhanden ist, wird von jeder Klasse entsprechend implementiert und der Aufruf gegebenenfalls an Kindobjekte weitergeben.

Ähnlich ist der Ablauf beim Zeichnen der Partitur. Alle Klassen implementieren ebenfalls die Methode `paint(Graphics)`. Diese Methode wird in der `paint()`-Methode von `ScorePanel` beim Wurzel-Element des Objektbaumes aufgerufen. Die jeweiligen Objekte zeichnen sich selbst und geben den Aufruf an die Kinder weiter.

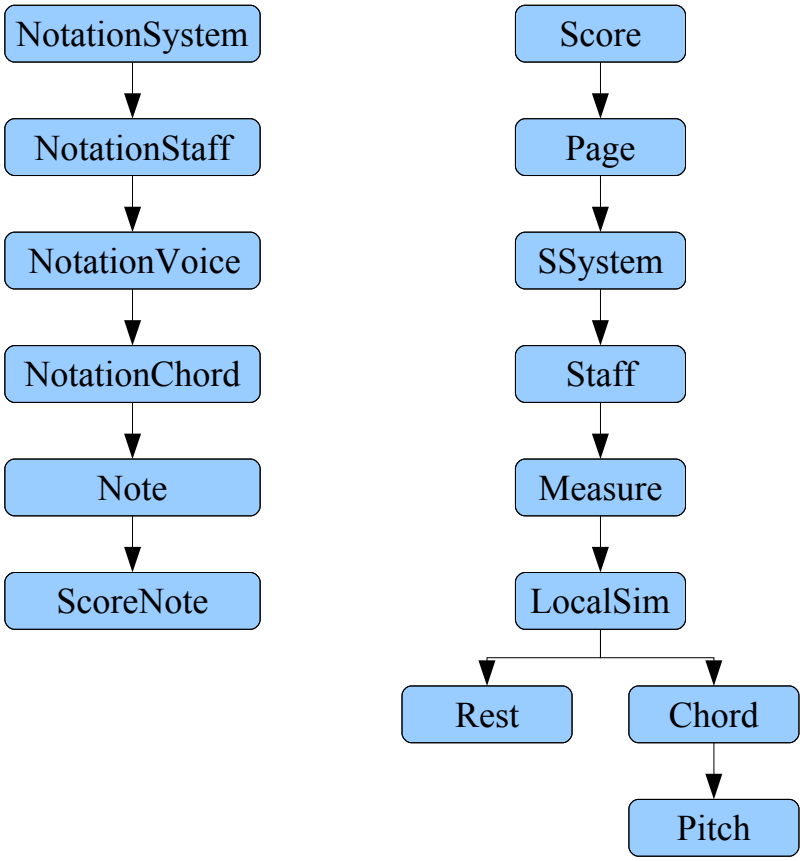


Abbildung 5.1: Die Objekthierarchien im Vergleich

5.3 Interaktion

In diesem Abschnitt soll kurz dargestellt werden, welche Möglichkeiten zur Interaktion und Manipulation des Stückes oder Notenmaterials zur Verfügung gestellt werden:

- Takt und Tonart eines Stückes können geändert werden.
- Grundlage für alle weiteren Operationen ist das Auswählen bzw. Markieren von einzelnen oder mehreren Noten.
- Ist eine Note markiert, kann sie gelöscht werden. Anstatt der Note soll dann eine Pause erscheinen.
- Markierte Noten können mit Vorzeichen versehen oder vorhandene Vorzeichen geändert werden.
- Markierte Noten sollen mit der Maus verschoben werden können (vertikal und horizontal).

5.4 Die Klasse ScoreEditor

Die Klasse `ScoreEditor` erbt von der Klasse `ScorePanel` und stellt die Erweiterungen für die Interaktion zur Verfügung.

Der `ScoreEditor` besitzt 3 verschiedene Modi, die über die Methode `setModus()` mit einer `enum`-Konstanten gesetzt werden können.

1. Im `VIEW`-Modus sind keine Veränderungen möglich. Das Panel reagiert nicht auf Maus-Ereignisse.
2. Im `SELECT_AND_EDIT`-Modus können Noten ausgewählt und dann horizontal und vertikal verschoben werden. Über entsprechende Buttons können Noten auch gelöscht oder mit Vorzeichen und Akzenten versehen werden.
3. Im `INSERT`-Modus können Noten durch Klicken mit der Maus hinzugefügt werden.

Zum Verschieben und Einfügen stellt die Klasse ein horizontales Raster zur Verfügung, an dem die Noten ausgerichtet werden. Die MouseListener und MouseMotionListener werden über die Klasse `ScoreEditorMouseAdapter` verwaltet. Funktionen zur Manipulation des Notenmaterials stellt die Klasse `NotationManipulator` zur Verfügung.

5.5 Ausgewählte Funktionen und Algorithmen der Klasse ScoreEditor

Die Klasse `ScoreEditor` stellt allgemeine Funktionen bereit, die häufig von anderen Methoden benötigt werden.

5.5.1 Raster

> *"Noten unterliegen in ihren Abständen einem Abstandsverhältnis, das die Beziehung ihrer rhythmischen Zeitwerte verdeutlicht: so hat etwa die Halbe-Note mehr Hinterfleisch als die Viertel-Note"*[30]

Diese Abstände sind jedoch nicht proportional zu den Notenwerten[31], was dazu führt, dass das Notenraster nicht regelmäßig ist und die Takte alle unterschiedliche Länge haben können.

Die Klasse `ScoreEditor` stellt ein Raster bereit, dass nicht nur der visuellen Orientation des Benutzers dient, sondern auch zur Bestimmung der zugehörigen metrischen Zeit zu einem gegebenen x-Wert. Mit der Methode `setGrid(Rational gridSize)` wird die Rastergröße festgelegt. Das alte Raster wird gelöscht, und die Positionen der neuen Rasterlinien werden mit dem in Abschnitt 5.5.2 angegebenen Algorithmus berechnet und in einer Liste mit Integer-Werten gespeichert.

[30] Wanske in [Gieseking, 2001], S. 155

[31] Zur Problematik der Bestimmung von Notenabständen siehe [Gieseking, 2001] Abschnitt 5.3

```
public void setGrid(Rational gridSize) {
    this.gridSize = gridSize;
    grid.clear();
    if (gridSize == null) {
        repaint();
        return;
    }
    Rational end = getNotationSystem().getEndTime();
    for (Rational pos = Rational.ZERO; pos.isLess(end); pos = pos
            .add(gridSize)) {
        grid.add(getPosition(pos));
    }
    repaint();
}
```

Quellcode 5.1: Methode `setGrid()` aus der Klasse `ScoreEditor`

Die Methode `getPosition(Rational)` wird im nächsten Abschnitt beschrieben.

5.5.2 Bestimmung der Bildschirm-Position eines metrischen Zeitpunktes

Mit Hilfe der Methode `getPosition(Rational)` kann zu einem gegebenen metrischen Zeitpunkt die Position auf dem Bildschirm bestimmt werden.

Alle `LocalSim`s des Systems werden betrachtet. Dies sind Objekte, die alle Events eines Zeitpunktes enthalten. Wenn die metrische Zeit eines `LocalSim`s genau der übergebenen Zeit entspricht, kann die Position des `LocalSim`s zurückgegeben werden. Ansonsten wird das Verhältnis von metrischer Distanz zum vorigen `LocalSim` zu der metrischen Distanz vom vorigen zum nachfolgenden `LocalSim` berechnet. In diesem Verhältnis wird dann auch die Entfernung der `LocalSim`s auf dem Bildschirm geteilt und so die zugehörige Position bestimmt.

Beispiel Es ist ein Takt wie in Abb. 5.2 gegeben. Über diesen Takt soll ein Achtel-Raster gelegt werden. Sei R der metrische Zeitpunkt der Rasterlinie, deren x-Koordinate bestimmt werden soll. LS_V und LS_N seien die metrischen Einsatzzeiten des vorigen und nachfolgenden `LocalSim`s. Dann wird das Verhältnis r durch die Formel

$$r = \frac{R - LS_V}{LS_N - LS_V}$$

bestimmt. Wenn jetzt LS_V^X und LS_N^X die x-Koordinaten der `LocalSim`s sind, lässt sich die x-Koordinate R^X der gesuchten Rasterlinie durch die Formel

$$R^X = LS_V^X + r \cdot (LS_N^X - LS_V^X)$$

bestimmen. Die Ergebnisse findet man in Tabelle 5.1.

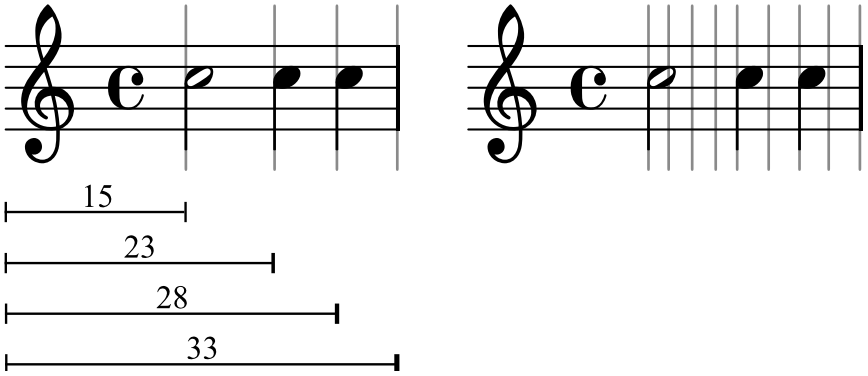
LS	$\frac{0}{4}$	$\frac{2}{4}$	$\frac{3}{4}$	$\frac{4}{4}$
LS^X	15	23	28	33

(a) Vorgegebene Positionen

R	$\frac{1}{8}$	$\frac{2}{8}$	$\frac{3}{8}$	$\frac{5}{8}$	$\frac{7}{8}$
R^X	17	19	21	25.5	30.5

(b) Neu berechnete Rasterpositionen

Tabelle 5.1: Berechnung eines Achtel-Rasters für einen Takt



(a) Notenbeispiel mit Positionen

(b) Notenbeispiel mit berechnetem Raster

Abbildung 5.2: Beispiel für die Berechnung eines Rasters

5.5.3 Bestimmung der metrischen Zeit von einer gegebenen Position

Die Bestimmung der metrischen Zeit von einer gegebenen Position könnte analog wie in Abschnitt 5.5.2 beliebig genau geschehen. Nur die Rollen von den metrischen Positionen und den Bildschirm-Positionen müssten vertauscht werden. Dies ist hier aber nicht gewünscht, da man nur die durch das Raster vorgegebene Genauigkeit erreichen möchte. Daher wird in der Methode `getTime(int posX)` zu dem gegebenen Wert `posX` die nächstliegende Rasterlinie gesucht und deren metrische Zeit zurückgegeben.

5.5.4 Bestimmung der Tonhöhe einer Position bzgl. eines Notensystems

Die Methode `pitchForY(int y)` gibt die diatonische Tonhöhe zu einem gegebenen y-Wert zurück. Dies geschieht mit Hilfe der Methode `pixelToHs()` des gewählten `Staff`s. Diese Methode konvertiert den Pixelwert in hs-Einheiten (*halfspace* - halber Abstand der Notenlinien) gemessen von der mittleren Linie. Zu diesen Wert wird noch der Wert `clefline` addiert, der widerspiegelt, um wie viele Schritte das c von der Mittellinie nach unten verschoben ist. Er kann über den Notenschlüssel des Systems besorgt werden. Rechnet man diese Summe modulo sieben und addiert

```
public Rational getTime(int posX) {
    if (gridSize == null)
        return null;
    int dx = Integer.MAX_VALUE;
    int index = 0;
    for (int i = 0; i < grid.size(); i++) {
        int d = Math.abs(grid.get(i) - posX);
        if (d < dx) {
            dx = d;
            index = i;
        }
    }
    return gridSize.mul(index);
}
```

Quellcode 5.2: Methode `getTime()` zur Bestimmung des metrischen Zeitpunktes zu einem Punkt an Hand des Rasters

noch einmal sieben, falls der Wert kleiner null ist, um den zugehörigen positiven Repräsentanten zu bekommen, so erhält man den Index, mit dem man die diatonische Tonhöhe bestimmen kann (vgl. Quellcode 5.3).

```
private char pitchForY(int y) {
    Page p = (Page) getScore().child(getScore().getActivePage());
    SSystem sys = (SSystem) p.child(selectedSystem);
    Staff s = (Staff) sys.child(selectedStaff);
    char[] pitches = { 'c', 'd', 'e', 'f', 'g', 'a', 'b' };
    int clefline = -1;
    if (((NotationStaff) getNotationSystem().get(selectedSystem))
            .getClefTrack() != null
            && ((NotationStaff) getNotationSystem()
                    .get(selectedSystem))
                        .getClefTrack().size() > 0)
        clefline = ((Clef) (((NotationStaff) getNotationSystem()
            .get(selectedSystem)).getClefTrack().get(0)))
                .getClefLine();
    int hs = s.pixelToHs(y) + clefline;
    int pitchIndex = hs % 7;
    if (pitchIndex < 0)
        pitchIndex += 7;
    return pitches[pitchIndex];
}
```

Quellcode 5.3: Methode zur Bestimmung der Tonhöhe zu einem gegebenen Punkt

Auf ähnliche Weise wird so auch noch die Oktave, in der die Note liegt, bestimmt.

```
private int octForY(int y) {
    Page p = (Page) getScore().child(getScore().getActivePage());
    SSystem sys = (SSystem) p.child(selectedSystem);
    Staff s = (Staff) sys.child(selectedStaff);
    Clef c = null;
    if (((NotationStaff) getNotationSystem().get(selectedSystem))
            .getClefTrack() != null
            && ((NotationStaff) getNotationSystem()
                    .get(selectedSystem))
                        .getClefTrack().size() > 0) {
        c = ((Clef) (((NotationStaff) getNotationSystem().get(
                selectedSystem)).getClefTrack().get(0)));
    }
    int clefline = -1;
    if (c != null)
        clefline = c.getClefLine();
    int hs = s.pixelToHs(y) + clefline;
    int oct = 0;
    if (hs < 0) {
        hs++;
    } else {
        oct++;
    }
    oct += (hs / 7);
    if (hs >= 0) {
    }
    if (c != null)
        if (c.getClefType() == 'c')
            oct -= 1;
        else if (c.getClefType() == 'f')
            oct -= 2;
    return oct;
}
```

Quellcode 5.4: Methode zur Bestimmung der Oktave zu einem gegebenen Punkt

Zusätzlich muss betrachtet werden, ob aufgrund der Tonart noch ein Vorzeichen hinzugefügt werden muss.

5.6 Auswahl von Noten

MUSITECH stellt einen `SelectionManager`, der die Selektion verwaltet. Über diesen Manager können Elemente zu der Selektion hinzugefügt und aus der Selektion gelöscht werden. Beim `SelectionManager` angemeldete Listener, die das Interface `SelectionListener` implementieren, werden dann bei Änderungen informiert. Zum Beispiel implementiert das `ScorePanel` das SelectionListener-Interface. Selektierte Noten werden so automatisch farblich unterlegt.

5.6.1 Auswahl einzelner Noten durch Klicken

Bei einem einzelnen Klick auf den `ScoreEditor` wird die Selektion gelöscht. Wenn sich an der angeklickten Stelle eine Note befindet, wird diese ausgewählt. Die neu selektierte Note bekommt man über die `ScoreEditor`-Methode `getNoteAt(Point)`. Diese gibt die Note an dem übergebenen Punkt oder `null` zurück.

Die Methode nutzt die Methode `catchScoreObject(int x,int y,Class c)` von `ScoreContainer`, die rekursiv im Objektbaum nach einem `ScoreObjekt` vom Typ `c` an der Stelle (`x`,`y`) sucht. In diesem Fall ist `c` vom Typ `Pitch`. Über das `Pitch`-Objekt bekommt man eine Referenz auf die damit zusammenhängende Note.

5.6.2 Auswahl mehrere Noten mit Auswahl-Rechteck

Durch Ziehen der Maus bei gedrückter linker Maustaste wird ein Auswahl-Rechteck erstellt. Wird die linke Maustaste dann losgelassen, wird für alle Noten überprüft, ob sie im Auswahlrechteck liegen und falls ja zur Liste der ausgewählten Noten hinzugefügt. Das `ScoreMapper`-Objekt unterhält die HashMaps `notationToGraphical` und `graphicalToNotation`, mit deren Hilfe die musikalischen Objekte den zugehörigen grafischen Objekten und umgekehrt zugeordnet werden können. Diese werden dazu benutzt, die Koordinaten der Noten herauszubekommen.

5.7 Verschiebung in Richtung der y-Achse

Sind erst einmal Noten ausgewählt, können diese zum Beispiel nach oben oder unten verschoben werden. Das Problem, das dabei auftaucht, ist vergleichbar mit dem Problem der realen und tonalen Sequenz[32] und soll an einem einfachen Beispiel erläutert werden.

Beispiel In der Tonart C-Dur wird ein C-Dur-Akkord mit den Tönen c, e und g (Abb. 5.3, Takt 1) ausgewählt und soll um einen "Schritt" nach oben verschoben werden. Es sind drei unterschiedliche Lösungen möglich:

1. Man verschiebt alle Noten innerhalb der Tonart C-Dur einen Schritt nach oben und erhält die Töne d, f und g (Abb. 5.3, Takt 2). Hierbei wurde der Abstand der Töne zueinander verändert. Hatte man vorher zwischen c und e eine große und zwischen e und g eine kleine Terz, hat man jetzt zunächst eine kleine Terz und dann eine große. Aus dem Dur-Dreiklang ist ein Moll-Dreiklang geworden.
2. Man verschiebt die unterste Note um einen Schritt und behält die Intervalle bei. Das Ergebnis ist ein D-Dur-Akkord, der nicht in der Tonart C-Dur vorhanden ist (Abb. 5.3, Takt 3). Das gleiche Resultat erhält man in diesem Fall auch, wenn man die oberste Note um einen Schritt nach oben verschiebt und die Intervalle beibehält.
3. Wählt man den mittleren Ton als Referenz, werden alle Noten nur um einen Halbtonschritt verschoben. Das Ergebnis ist ein Des-Dur-Akkord, der ebenfalls nicht in der Tonart C-Dur enthalten ist (Abb. 5.3, Takt 4).

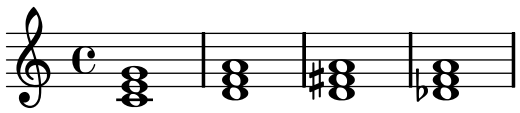


Abbildung 5.3: Drei Möglichkeiten zum Verschieben des Akkordes

Im Zusammenhang von Sequenzen würde man im ersten Fall von einer tonalen Sequenz sprechen, beim zweiten und dritten Fall von einer realen Sequenz. Für diese Arbeit wurde der erste Fall implementiert, da er am intuitivsten erscheint. Für den zweiten und dritten Fall ist intuitiv nicht klar, welcher Ton aus einer beliebig ausgewählten Menge als Bezugspunkt genommen werden soll.

Bei der ersten Variante muss jeder Ton einzeln betrachtet werden. In der Klasse `ScoreManipulator` gibt es eine Methode `transpose(Note n, int interval)`, die eine Note um das angegebene Intervall (Sekunde 1, Terz 2, Quarte 3, ...) transponiert. Sei *dy* die Strecke, um die die Note auf dem Bildschirm verschoben wurde (nach oben pos., nach unten neg.) und *ld* der Abstand zwischen den Notenlinien.

[32] vgl. [Hempel, 1997], S. 195

Dann kann das Intervall I mit folgender Formel bestimmt werden:

$$I = \left\lceil 2 \cdot \frac{dy}{ld} + 0.5 \right\rceil$$

Das eigentliche Transponieren erfolgt dann mit folgendem Algorithmus:

```
public void transpose(Note n, int interval) {
    char[] pitches = { 'c', 'd', 'e', 'f', 'g', 'a', 'b' };
    int index = 0;
    while (n.getScoreNote().getDiatonic() != pitches[index]
            && index < pitches.length)
        index++;
    int newIndex = (index + interval) % 7;
    int oct = (index + interval) / 7;
    if (newIndex < 0) {
        newIndex += 7;
        oct -= 1;
    }
    n.getScoreNote().setDiatonic(pitches[newIndex]);
    n.getScoreNote().setOctave((byte) (n.getScoreNote()
            .getOctave() + oct));
```

Quellcode 5.5: Algorithmus zum Transponieren einer Note (Teil 1)

Auf den alten Index `index` der diatonischen Tonhöhe wird das Intervall `interval` addiert und dann modulo 7 gerechnet, falls es Oktavüberschreitungen gegeben hat. Die Anzahl der Oktavüberschreitungen wird in der Variablen `oct` gespeichert. Der neue Index `newIndex` kann noch kleiner als 0 sein. In diesem Fall wird auf `newIndex` noch einmal 7 addiert und `oct` angepasst. Die neue wird gesetzt und die Oktave korrigiert. Es fehlt noch die Anpassung des Vorzeichens. Das Vorzeichen wird bei der `ScoreNote` über die Methode `setAlteration(byte b)` gesetzt. -1 bedeutet ein Be, 0 kein Vorzeichen und 1 ein Kreuz. Um das Vorzeichen korrekt zu setzen, benötigt man ein zweidimensionales `byte`-Array (vlg. Quellcode 5.6), in dem für jede Tonart und für jede Tonstufe gespeichert ist, ob ein oder welches Vorzeichen gesetzt werden muss. Mithilfe eines `KeyMarker`s aus dem `harmonyTrack` des `Piece` kann man die aktuelle Tonart feststellen und mit dem `byte`-Array das für die Tonart entsprechende Vorzeichen setzen.

```
private static final byte[][] KEY_SIGNATURES = {
    { -1, -1, -1,  0, -1, -1, -1 },
    {  0, -1, -1,  0, -1, -1, -1 },
    {  0, -1, -1,  0,  0, -1, -1 },
    {  0,  0, -1,  0,  0, -1, -1 },
    {  0,  0, -1,  0,  0,  0, -1 },
    {  0,  0,  0,  0,  0,  0, -1 },
    {  0,  0,  0,  0,  0,  0,  0 },
    {  0,  0,  0,  1,  0,  0,  0 },
    {  1,  0,  0,  1,  0,  0,  0 },
    {  1,  0,  0,  1,  1,  0,  0 },
    {  1,  1,  0,  1,  1,  0,  0 },
    {  1,  1,  0,  1,  1,  1,  0 },
    {  1,  1,  1,  1,  1,  1,  0 },
};
```

Quellcode 5.6: `byte`-Array zur Bestimmung der Vorzeichen

```
    KeyMarker km = null;
    for (Iterator i = getNotationSystem().getContext().getPiece()
            .getHarmonyTrack().iterator(); i.hasNext();) {
        Object o = (Object) i.next();
        if (o instanceof KeyMarker) {
            KeyMarker k = (KeyMarker) o;
            if (k.getMetricTime().isLessOrEqual(
                    n.getMetricTime()))
                km = k;
        }
    }
    if (km != null)
        n.getScoreNote().setAlteration(
                KEY_SIGNATURES[km.getAccidentalNum() + 6]
                    [newIndex]);
    else
        n.getScoreNote().setAlteration((byte) 0);
}
```

Quellcode 5.7: Algorithmus zum Transponieren einer Note (Teil 2)

5.8 Verschiebung in Richtung der x-Achse

5.8.1 Verschiebung von einer Note in Richtung der x-Achse

Beim Verschieben einer Note, wird diese aus der Stimme entfernt und an anderer Stelle wieder eingefügt. Die metrische Position, an der die Note eingefügt werden soll, kann mit der in Abschnitt 5.5.3 vorgestellten Methode berechnet werden. Beim Einfügen können jetzt folgende Situationen auftreten:

An der Stelle, an der die Note eingefügt werden soll, befindet sich...

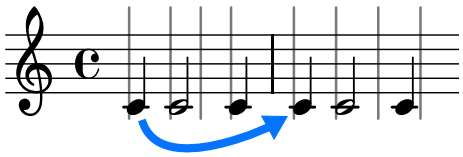
- eine Note mit gleicher Länge. Die Note kann eingefügt werden.
- eine kürzere oder längere Note. Die Note kann nicht eingefügt werden.
- eine Pause, die gleichlang oder länger ist. Die Note kann eingefügt werden.
- eine kürzere Pause. Die Note kann nicht eingefügt werden.
- keine Note, die vorige Note überlappt die Stelle. Die Note kann nicht eingefügt werden.

Pausen können explizit als Objekte in der Stimme vorhanden sein, werden aber auch automatisch für das Notenbild generiert, falls eine Lücke zwischen zwei Noten in der Stimme auftritt. Die explizit gesetzten Pausen müssen gegebenenfalls gekürzt werden, um Platz für die einzufügende Note zu schaffen.

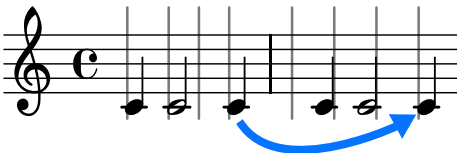
Wenn eine Note nicht in eine Stimme eingefügt werden kann, gibt es die Möglichkeit, eine neue Stimme zu erstellen und die Note in diese Stimme einzufügen. Auf dieses Vorgehen soll hier verzichtet werden, weil dadurch die musikalische Struktur zerstört werden kann.

5.8.2 Verschiebung von mehreren Noten in Richtung der x-Achse

Beim Verschieben von mehreren Noten tritt das Problem auf, dass eine feste Strecke auf dem Bildschirm je nach Lage eine andere metrische Distanz haben kann. Dies liegt daran, dass der Platzbedarf der Noten nicht proportional zur Länge der Noten ist. Werden jetzt also mehrere Noten um die gleiche Strecke auf dem Bildschirm bewegt, kann es passieren, dass eine Note zum Beispiel um 4 Achtel, eine andere aber um 5 Achtel verschoben wurde. Deshalb muss eine Note als Referenznote gewählt werden. Diese bestimmt den metrischen Abstand. Am Intuitivsten scheint es, wenn hierfür immer die Note unter dem Maus-Zeiger gewählt wird. Mit der in 5.5.3 beschriebenen Methode `getTime()` kann der zum Endpunkt des Verschiebevorgangs gehörige Zeitpunkt bestimmt werden. Alle mitverschobenen Noten müssen jetzt um die Differenz zwischen Startzeit der Note unter dem Cursor und diesem Zeitpunkt versetzt werden. Vorher muss jedoch für jede Note überprüft werden, ob sie an dem berechneten Zeitpunkt eingefügt werden kann. Dazu müssen die Fälle aus 5.8.1 untersucht werden. Es erscheint sinnvoll, den Vorgang nur abzuschließen, wenn alle Noten eingefügt werden können, ansonsten werden alle Noten wieder zurückgesetzt. Eine Alternative wäre, nur die einfügbaren Noten zu verschieben. Dieses Vorgehen wurde nicht implementiert, da es wahrscheinlich nicht den Erwartungen der meisten Benutzer entspricht.



(a) Verschieben mit erstem Ton als Referenz)



(b) Verschieben mit dem letzen Ton als Referenz)

Abbildung 5.4: Verschieben auf der x-Achse mit unterschiedlichen Referenztönen

Die Verwaltung der `MouseListener` und der grafischen Operationen übernimmt die Klasse `ScoreEditorMouseAdapter`. Die Veränderungen am Notensystem können über die Klasse `ScoreManipulator` gemacht werden.

5.9 Midi-Wiedergabe

Das Abspielen der Noten geschieht über den MUSITECH-`Object-Player`. Der `Object-Player` ist als Singleton realisiert. Zugriff bekommt man über die statische Methode `ObjectPlayer.getInstance()`. Mit der Methode `setContainer()` kann man dem Player ein Objekt vom Typ `de.uos.fmt.musitech.data.structure.container.Container` übergeben. Dies ist das `NotationSystem` des `ScoreEditor`s. Die Steuerung des `ObjectPlayer`s übernimmt der `PlayTimer`, der zentrale Timer für die Wiedergabe in MUSITECH. Hier können sich Player wie der `ObjectPlayer`, aber auch Displays (wie das `ScorePanel` von dem die Klasse `ScoreEditor` erbt) anmelden und Timing-Information empfangen bzw. abfragen. Das `ScorePanel` zum Beispiel kann sich immer über die aktuelle Position der Players informieren lassen und so einen Cursor beim Abspielen mitlaufen lassen.

5.10 Datenspeicherung

Für Serialisierung existiert im MUSITECH-Projekt die Klasse `MusiteXMLSerializer`, die es ermöglicht, MUSITECH-Objekte im Musite-XML-Format zu speichern. Für diese Arbeit wurde eine Datenbank eingerichtet, die die Notenbeispiele (in Form von Objekten vom Typ `Piece`) zentral auf dem MUSITECH-Server verwaltet. Als Datenbank wurde hierfür die XML-Datenbank Xindice[33] aus XML-Projekt der Apache Foundation gewählt. Sie speichert die serialisierten Stücke direkt und ohne Umwandlung zum Beispiel in ein relationales Modell.

Der Datenbankserver läuft ebenso wie die in Abschnitt 6 beschriebene Webanwendung als Servlet in dem verwendeten Webserver und Servlet-Container Jetty[34].

5.10.1 Zugriff auf die Datenbank

Für den Zugriff auf die Datenbank bieten sich mehrere Möglichkeiten an:

Core Server API Das Core Server API ist ein Low-Level-API, das direkten Zugriff auf die Daten bietet. Es kann jedoch nur in der gleichen Instanz der Java VM verwendet werden und ist Xindice-spezifisch. Die anderen APIs bauen auf diesem auf.

Xindice XML-RPC API XML-RPC[35] ist eine Spezifikation für Remote-Procedur-Calling über HTTP. Die Beschreibung der zu übertragenden Daten geschieht in XML. Der Vorteil dieses API ist seine Plattformunabhängigkeit.[36]

XML:DB XML Database API Das komfortabelste API für den Zugriff ist das XML:DB XML Database API. Die Datenbank enthält eine Implementierung in Java, die für Anwendung genutzt werden soll. Der Zugriff auf die XML-Datenbank

[33] `http://xml.apache.org/xindice/`
[34] `http://www.mortbay.org/`
[35] `http://www.xmlrpc.com/`
[36] Eine Liste über die verschiedenen Implementierungen findet sich unter `http://de.wikipedia.org/wiki/XML-RPC#Weblinks`

verläuft ähnlich wie ein Zugriff auf eine relationale Datenbank über JDBC. Als Abfrage-Sprache wird XPath[37] verwendet, für die Datenmanipulation XUpdate.[38]

Datenbank-Organisation

Die Datenbank ist hierarchisch in sogenannten Collections organisiert. Collections können entweder XML-Dokumente oder wiederum Collections enthalten. Von einer Collection aus können dann mit Hilfe von XPath-Ausdrücken Abfragen abgesetzt werden.

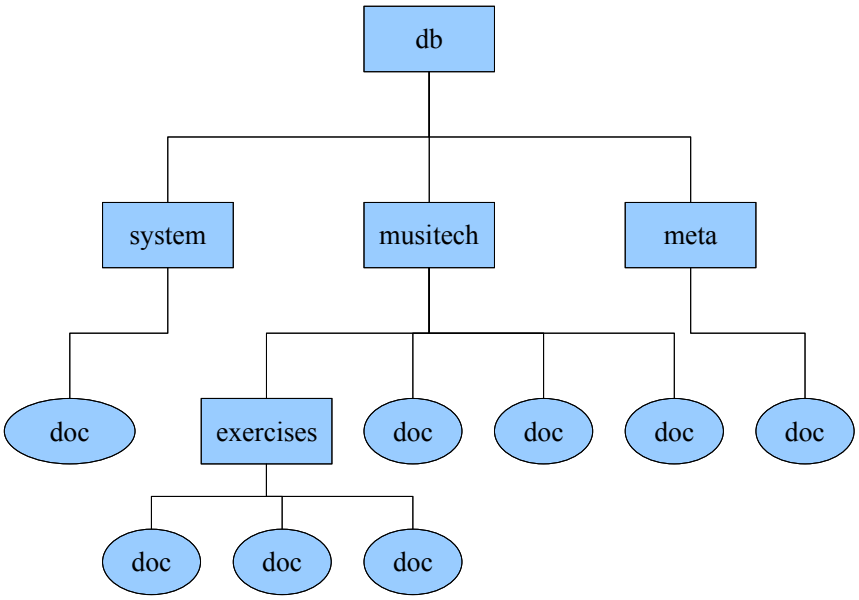


Abbildung 5.5: Datenbankorganisation in Collections

Herstellen einer Verbindung zur Datenbank

Das Herstellen einer Verbindung zur Datenbank geschieht ähnlich wie bei JDBC. Zunächst muss der Datenbanktreiber geladen werden. Mit Hilfe des Treibers kann ein Objekt vom Typ `Database` erstellt und beim `DatabaseManager` angemeldet werden. Über die statische Methode `DatabaseManager.getCollection()`, der als Argument die URI der Wurzel-Collection übergeben wird, bekommt man ein `Collection`-Objekt.

Zugriff auf die Dokumente

Für die beiden Anwendungen wird der Zugriff auf die Datenbank über die Klasse `DatabaseConnector` aus dem Paket `db` geregelt. Sie bietet Methoden zum Einfügen,

[37] `http://www.w3.org/TR/xpath`
[38] `http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html`

```
private static final String
    DB_URI = "xmldb:xindice://musitech.fmt.uos.de:8080/db/musitech";
private Collection getRootCollection() {
    if (rootCollection == null) {
        try {
            String driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
            Class c = Class.forName(driver);
            Database database = (Database) c.newInstance();
            DatabaseManager.registerDatabase(database);
            rootCollection = (Collection) DatabaseManager
                    .getCollection(DB_URI);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    return rootCollection;
}
```

Quellcode 5.8: Methode `getRootCollection()` aus der Klasse `DatabaseConnector`

Löschen und zum Auflisten der vorhandenen Dokumente. Die Dokumente werden über eine Id verwaltet, die eine String-Repräsentation der Systemzeit zum Zeitpunkt des Einfügens ist. In dem Dokument `index` können zu der Id (bzw. dem Dokument) noch Metainformationen, wie der Titel des Stückes und der Komponist, abgelegt werden. Abfragen (zum Beispiel nach den Stücken eines Komponisten) lassen sich dann mittels XPath formulieren und an die Methode `query()` übergeben.

```
<?xml version="1.0" encoding="UTF-8"?>
<piecelist>
  <piece>
   <id>1186948146931</id>
    <title>Konzert fuer Posaune und Orchester B-dur</title>
    <composer>Ernst Sachse</composer>
  </piece>
  <piece>
    <id>1186948147658</id>
    <title>Morceau Symphonique op. 88</title>
    <composer>Alexandre Guilmant</composer>
  </piece>
</piecelist>
```

Quellcode 5.9: Beispiel für ein Index-Dokument

Um ein Dokument (Stück) hinzuzufügen, wird die Id und das XML-Dokument als String der Methode `addDocument()` übergeben. Es bleibt noch die Index-Datei zu akualisieren. Dies geschieht mit Hilfe einer XUpdate-Anfrage (vgl. Quellcode 5.10).

```
<?xml version="1.0" encoding="UTF-8"?>
<xupdate:modifications version="1.0"
  xmlns:xupdate="http://www.xmldb.org/xupdate">
    <xupdate:append select="/piecelist" >
        <xupdate:element name="piece">
            <id>1186948148452</id>
            <title>Meditation</title>
            <composer>Frigyes Hidas</composer>
        </xupdate:element>
    </xupdate:append>
</xupdate:modifications>
```

Quellcode 5.10: Hinzufügen eines Stückes in die Index-Datei

Die (De-)Serialisierung der `Piece`-Objekte und das Laden und Speichern in der Datenbank über den `DatabaseConnector` ebenso wie die XPath- und XQuery-Abfragen geschehen in der Klasse `PieceAdmin` aus demselben Paket. Zusätzlich kann die Klasse auch neue, leere Stücke erzeugen.

6 Umsetzung mit SVG und AJAX

Die Umsetzung lehnt sich strukturell stark an die Java-Applet-Implementation an. Um den Wartungsaufwand zu minimieren, werden auf Server-Seite unter anderem die gleichen Klassen wie im Applet benutzt. Zum Benutzen der Programme wird ein SVG-fähiger Browser benötigt. Die Anwendungen wurden mit Mozilla Firefox und Opera entwickelt und getestet.

6.1 SVG-Erweiterung des GWT

Da das Google Web Toolkit von Hause aus keine Unterstützung von SVG bietet, musste es um diese Funktionalität erweitert werden. Eine Bibliothek von Robert Hanson, die unter `http://gwt-widget.sourceforge.net` verfügbar ist und elementare SVG-Funktionalität bietet, stellte sich unter anderem im Bezug auf das Event-Handling als zu eingeschränkt heraus.

Das Google Web Toolkit erzeugt seinen Inhalt per DOM-Manipulationen. Möchte man HTML- und SVG-Elemente gemeinsam benutzen, müssen die Elemente in einem DOM vorhanden sein. Dies kann man durch die Verwendung von XHTML und verschiedenen Namespaces erreichen.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:svg="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <head>
        <meta name='gwt:module' content='Musitech' />
        <link rel="stylesheet" href="styles.css" type="text/css"/>
        <title>Musitech</title>
    </head>
    <body>
        <script language="javascript" src="gwt.js"></script>
        <div id="score"></div>
        <div id="player"></div>
    </body>
</html>
```

Quellcode 6.1: HTML-Rumpf-Datei

6.1.1 Namespace-Erweiterung

Um mit unterschiedlichen Namensräumen zu arbeiten, muss das Toolkit um DOM-Methoden mit Namespace-Unterstützung erweitert werden. Dies geschieht in der Klasse `DOM2Impl` im Paket `client` mit Hilfe des JSNI. Im wesentlichen sind drei Methoden notwendig. Mit der Methode `createElementNS()` kann ein Element mit vorgegebenem Namensraum eingefügt werden. Diese Methode ist für das Einfügen von SVG-Elementen wichtig. Die Methoden `setAttributeNS()` und `getAttributeNS()` dienen zum Setzen und Auslesen von Attributen mit Namensräumen. Dies wird für XLINK-Verweise benötigt.

```
package client.svg;
import com.google.gwt.user.client.Element;
public class DOM2Impl {
  public native Element createElementNS(String ns, String tag)/*-{
     return $doc.createElementNS(ns, tag);
  }-*/;
  public native void setAttributeNS(String ns, Element elem,
        String attr, String value) /*-{
     elem.setAttributeNS(ns, attr, value);
  }-*/;
  public native String getAttribute(String ns, Element elem,
        String attr) /*-{
     var ret = elem.getAttributeNS(ns,attr);
     return (ret == null) ? null : String(ret);
  }-*/;
}
```

Quellcode 6.2: DOM2Impl.java

6.1.2 SVG Klassenhierarchie

SVGWidget

Die Basisklasse für alle SVG-Elemente ist die Klasse `SVGWidget`. Mit ihr können beliebige SVG-Elemente erzeugt werden. Dem Konstruktor muss lediglich der Tag-Name als String übergeben werden. Alle Subklassen erhalten automatisch den SVG-Namespace. Die Klasse unterhält Listen und Methoden zur Verwaltung von Click- und MouseListenern und leitet ankommende Events an die angemeldeten Listener weiter. Sie bietet ebenso Methoden zum vereinfachten Auslesen und Setzen von Attributen.

SVGPanel

Die Klasse `SVGPanel` repräsentiert das SVG-Wurzelelement `svg`. Sie bietet die Möglichkeit, andere SVGWidgets aufzunehmen und zu verwalten.

Defs

Die Klasse `Defs` dient der Einbindung der Musikschriftart "Gin Music Font". Diese wurde mit Hilfe des Apache Batik SVG Toolkit[39] in eine SVG-Schrift umgewandelt. Da die Unterstützung für SVG-Fonts in den Browsern im Moment noch sehr unvollständig ist, wird diese Klasse derzeit nicht eingesetzt. Stattdessen muss die Schriftart als TrueType-Schrift auf dem System installiert werden.

SVGContainer

`SVGContainer` ist eine Container-Klasse, die in der Lage ist, andere `SVGWidgets` aufzunehmen. Über eine Liste kann auf die einzelnen Widgets zugegriffen werden. Im DOM wird diese Klasse auf das SVG-Gruppierungs-Element `<g>` abgebildet.

Shape

Die Klasse `Shape` im Paket `client.svg.shapes` ist die Basisklasse für alle darstellbaren Elemente. Sie bietet allgemeine Methoden zum Setzen der Stiftfarbe, der Füllfarbe, der Transparenz.

Grundelemente

Für jedes der Grundelemente Line, Circle, Ellipse, Rectangle, Polygon, Polyline, Path und Text gibt es im Paket `client.svg.shapes` eine entsprechende Klasse. Die Attribute können über Getter- und Setter-Methoden gesetzt werden.

```
package client;
import client.svg.SVGPanel;
import client.svg.shapes.Circle;
import client.svg.shapes.Rectangle;
import com.google.gwt.user.client.ui.SimplePanel;
public class TestPanel extends SimplePanel {
    private SVGPanel panel;
    public TestPanel() {
        panel = new SVGPanel(400, 300);
        Rectangle r = new Rectangle(50, 50, 100, 100);
        r.setFill("yellow");
        r.setStroke("black");
        panel.add(r);
        Circle i = new Circle(100,100,30);
        r.setFill("none");
        r.setStroke("#80FF80");
        panel.add(r);
        setWidget(panel);
    }
}
```

Quellcode 6.3: Beispiel für ein SVGPanel mit einigen Elementen

[39] `http://xmlgraphics.apache.org/batik/`

6.2 Aufbau des Clients

Der Aufbau des SVG-Client-Moduls ähnelt dem Aufbau des Applets. Die zentrale Klasse ist `ScorePanel`. Sie ist für die Anzeige der Noten zuständig. Die Listener werden über die Klasse `ScoreMouseAdapter` verwaltet. Änderungen am Notensystem können über die Klasse `ScoreManipulator` vorgenommen werden.

6.2.1 ScorePanel

Die Klasse `ScorePanel` ist eine Unterklasse von `SVGPanel` und damit das `<svg>`-Element im DOM. Als grafische Elemente enthält sie mit dem `SVGScore` die Notendarstellung und in einem `SVGContainer` die Linien des Rasters.

Modi

Genau wie die Klasse `ScoreEditor` kann `ScorePanel` in einem von drei Modi (`VIEW`, `SELECT` und `INSERT`) sein, der bestimmte Aktionen zulässt oder untersagt.

Raster

Das Raster kann mit der Methode `setGridResolution(int gridResolution)` gesetzt werden. Der übergebene Integer-Wert gibt den Nenner der Auflösung an (also $1 \Rightarrow \frac{1}{1}$, $2 \Rightarrow \frac{1}{2}$, $4 \Rightarrow \frac{1}{4}$, ...). Um die Kommunikation mit dem Server minimal zu halten und eventuelle Latenzprobleme zu vermeiden, wird bei Aktualisierung der Notengrafik ein Sechzehntel-Raster in Form eines `Line`-Arrays mit übertragen. Soll nun zum Beispiel ein Achtel-Raster angezeigt werden, wird nur jede zweite Linie angezeigt, beim Viertel-Raster entsprechend jede vierte Linie. Die anzuzeigenden Linien werden dann dem `SVGContainer grid` hinzugefügt.

6.2.2 ScoreMouseAdapter

Diese Klasse dient dem Überwachen der Mausaktionen im `ScorePanel`. Je nach Modus muss anders auf Mausaktionen reagiert werden. Befindet sich das `ScorePanel` im `SELECT`-Modus und es wird mit der Maus in das Panel geklickt, wird die Methode `onClick(Widget w)` aufgerufen. Hier wird entschieden, ob sich eine Note unter dem Cursor befindet. Falls sich keine Note unter dem Cursor befindet, wird eine gegebenenfalls vorhandene Selektion gelöscht und an der Stelle ein Rechteck als Auswahlrechteck eingesetzt. Das `dragging`-Flag wird auf `true` gesetzt und damit der Auswahlmodus aktiviert. Befindet sich eine Note unter dem Cursor und sind Noten ausgewählt, wird das `moving`-Flag auf `true` gesetzt und so der Verschiebemodus aktiviert. In einem zweidimensionalen `int`-Array `oldpositions` werden die Bildschirmpositionen der ausgewählten Noten gespeichert. Für beide Fälle wird der Bildschirmpunkt, an dem geklickt wurde, in den Variablen `x` und `y` gespeichert.

Die Methode `onMouseMove(Widget w, int x, int y)` wird bei der Bewegung der Maus über das `ScorePanel` ausgelöst. Ist das `dragging`-Flag, gesetzt wird die

Größe des Auswahl-Rechteckes aktualisiert. Ist dagegen das `moving`-Flag gesetzt, werden die Bildschirmkoordinaten der ausgewählten Noten angepasst. Mit Hilfe der Variablen `x` und `y` können die Entfernungen zum Startpunkt der Aktion berechnet werden und Auswahlrechteck oder Notenpositionen um diesen Wert verändert werden.

Beim erneuten Klicken mit der Maus wird wieder die Methode `onClick(Widget w)` aufgerufen. Ist das `dragging`-Flag gesetzt, wird überprüft, welche Noten sich im Auswahlrechteck befinden und diese dann zur Selektion hinzufügt. Danach wird das Auswahlrechteck aus dem `ScorePanel` entfernt. Falls das `moving`-Flag gesetzt ist, wird die Methode `moveSelection` der Klasse `ScoreManipulator` aufgerufen. Falls sich das Panel im `INSERT`-Modus befindet, wird die Methode `insertNote(int x, int y, int d)` von `ScoreManipulator` aufgerufen, die versucht an der Stelle (x, y) eine Note mit der Länge $\frac{1}{d}$ einzufügen.

6.2.3 ScoreManipulator

Die Klasse `client.score.ScoreManipulator` enthält die Methoden zum Verändern des Notenmaterials. Es sind im Wesentlichen die gleichen Methoden wie in der Klasse `client.ScoreManipulator` des Applets. Aufrufe dieser Methoden werden per RPC an die entsprechenden Methoden des `ScoreManipulator` im Server weitergegeben.

6.2.4 Realisation der MIDI-Wiedergabe

Die Wiedergabe der Noten geschieht per MIDI. Hierfür muss im Browser ein Plugin installiert sein (z.B. Quicktime). Soll das Stück abgespielt werden, wird je nach Browser entweder ein `<embed>`- oder `<object>`-Element in das DOM eingefügt. Als `src`- bzw. `data`-Attribut wird ein Servlet angegeben, dass automatisch aus dem im Server gespeicherten `NotationSystem` ein Midi-File generiert und an den Client sendet. Der Browser beginnt dann mit der Wiedergabe. Soll diese beendet werden, wird das `<embed>`- oder `<object>`-Element wieder aus dem DOM entfernt.

Um diese Vorgänge zu vereinheitlichen gibt es das Interface `NotePlayer`. Es definiert die Methoden `start()` und `stop()`. Die statische Methode `getPlayer()` der Klasse `NotePlayerFactory` gibt je nach Browser entweder einen `EmbedNotePlayer` oder einen `ObjectNotePlayer` zurück. Beim Aufruf der `start()`-Methode hängen sich die Player in das DOM ein, beim Aufruf von `stop()` enfernen sie sich wieder.

6.3 Aufbau des Servers

Für jeden Benutzer wird eine Instanz von `ScoreEditor` erzeugt. Hierüber wird mit den gleichen Methoden wie beim Applet auf die Benutzeraktionen reagiert und das Notenbild erzeugt. Über ein Servlet werden diese Methoden für den Client zur Verfügung gestellt. Da bei einem Servlet immer nur eine Instanz für alle Benutzer erzeugt wird, kann der `ScoreEditor` nicht einfach in einem Feld in der Servlet-Klasse gespeichert werden. Eine Speicherung als Attribute im Session-Objekt ist

nicht möglich, da die Klasse `ScoreEditor` nicht serialisierbar ist. Stattdessen kann der `ScoreEditor` über statische Methoden der Klasse `ScoreEditorPool` in einer HashMap unter der Session-ID im Speicher abgelegt werden. Nach einer festgelegten Leerlaufzeit werden alte Instanzen aus dem Speicher entfernt.

Für das `NotationSystem` exitstieren in MUSITECH Serialisierungsmechanismen, so kann das Stück selbst zum Beispiel aus einer Datenbank gelesen werden oder in einer Datenbank abgespeichert werden (siehe Abschnitt 5.10).

6.4 Kommunikation zwischen Client und Server

Für die Übertragung der Daten zwischen Client und Server werden die RPC-Mechanismen des Google Web Toolkits benutzt. Um die zu übertragende Datenmenge zu minimieren, wurde ein Transport-Format für die Notendarstellung entwickelt, bei dem nur die nötigen Daten übertragen werden.

Alle Score-Objekte besitzen eine `paint()`-Methode, mit der sie sich auf einen übergebenen `java.awt.Graphics`-Kontext zeichnen können. Mit Hilfe der Klasse `TGraphics` aus dem Paket `score`, die von `java.awt.Graphics` erbt, kann ein Grafik-Kontext erzeugt werden, der bei jedem Methodenaufruf ein Objekt erzeugt, dass die relevanten Daten, wie zum Beispiel bei einer Linie Anfangs- und Endpunkt, festhält. Diese Objekte werden in einer Liste gespeichert.

Für jede grafische Grundform existiert eine entsprechende Klasse im Paket `client.transport`. Diese Klassen haben alle den Prefix "T" und erben von der Klasse `TShape`, die Eigenschaften, die alle Elemente haben sollen, implementiert. Die Klassen sind Java-Beans[40] und lassen sich deshalb leicht serialisieren. Für die RPC-Übertragung im Google Web Toolkit ist noch die Implementierung des Markerinterfaces `com.google.gwt.user.client.rpc.IsSerializable` nötig (seit Version 1.4 des GWT ist auch die Implementiertung von `java.io.Serializable` möglich).

Die in Abschnitt 5.2 beschriebene Objekthierachie, die eine Partitur darstellt wird in eine schlankere, übertragbare Form gebracht. Dafür existieren im Paket `client.transport` entsprechende Klassen. Basisklasse für diese Objekte ist die Klasse `TScoreObject`, die ein `TShape`-Array besitzt, in dem die zeichenbaren Elemente gespeichert werden können. Zum Beispiel legt die Klasse `TStaff` die Notenlinien als `TLine`-Objekte in diesem Array ab. Zusätzlich besitzt die Klasse ein Array, in dem die zugehörigen Takte in Form von `TMeasure`-Objekte gespeichert werden.

Die Klassen `TScore` und `TPitch` sind die einzigen Klassen, die über Zeichenobjekte hinaus noch zusätzliche Informationen beinhalten. Ein `TScore`-Objekt speichert die Größe der Partitur und den Abstand der Notenlinien eines Systems. Außerdem hat es in einem zusätzlichen Feld `TLine`-Objekte für das Raster. Objekte vom Typ `TPitch` entsprechen den Noten und enthalten als zusätzliche Information eine ID, die sie eindeutig identifiziert und mit deren Hilfe auf dem Server die Verknüpfung zum Note-Objekt hergestellt werden kann.

Im Client wird mit Hilfe eines `client.score.Transformer`-Objektes von einem

[40] vgl. [Ullenboom, 2007], S. 1347ff.

`TScore` in ein SVGScore tranformiert. Dazu gibt es im Paket `client.score.elements` entsprechende Klassen. Alle diese Klassen haben als Basisklasse `SVGScoreObject`, die andere `SVGScoreObject`s aufnehmen und verwalten kann. `SVGScoreObject` erbt dabei wiederum von `SVGContainer`, was bedeutet, dass die Elemente im DOM als `<g>`-Elemente repräsentiert werden.

Mit Hilfe des GWT-RPC können nun, wie in Abschnitt 4.6.3 beschrieben, entfernte Methoden aufgerufen werden. Ein Servlet stellt diese Methoden bereit. Der Rückgabewert ist in diesem Fall immer ein `client.rpc.ResultObject`. Dieses Objekt enthält ein `TScore`-Feld `tScore` und einen `boolean`-Feld `successfull`. `successfull` gibt an, ob die Operation erfolgreich war, während `tscore` bei Veränderungen des Notenbildes die neue Darstellung enthält oder sonst `null` ist.

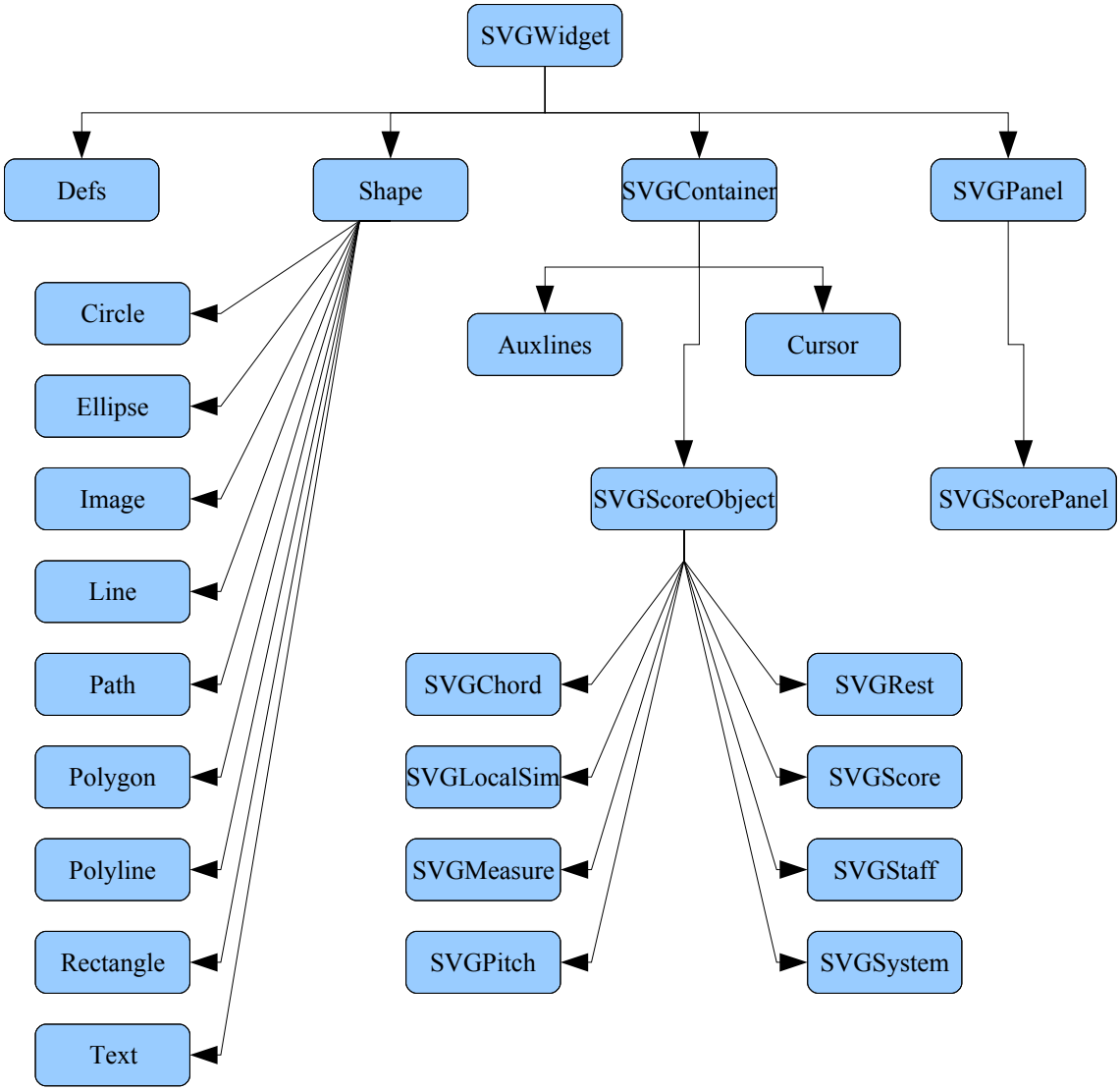


Abbildung 6.1: Vollständige SVG-Klassenhierarchie

7 Vergleich der beiden Umsetzungen anhand von Beispielapplikationen

Der Vergleich der beiden Lösungen wird anhand von zwei Beispielanwendungen, die jeweils für beide Lösungen umgesetzt worden sind, vorgenommen. Die Beispielanwendungen sollen nur Anregungen für mögliche Einsatzzwecke geben und die Funktionen demonstrieren.

7.1 Noteneditor

Der Noteneditor ist von den Funktionen her nicht mit Notensatzprogrammen wie Finale, Sibelius oder anderen vergleichbar, da in MUSITECH der Schwerpunkt auf der automatischen Notenbildgenerierung für Lehr- und Lernkontexte liegt.

7.1.1 Die Symbolleisten

Der Noteneditor besteht aus zwei Teilen. Im oberen Teil befinden sich drei Symbolleisten, die die Funktionen für den Benutzer bereitstellen. Da es sich bei den Applikationen um Webanwendungen handelt, wurde auf Menüleisten verzichtet.

Auf der ersten Symbolleiste gibt es zunächst Buttons für die Partitur-Verwaltung. Es können neue Notenblätter erstellt, Notenblätter geöffnet oder gespeichert werden. Die Partituren werden in beiden Anwendungen auf dem Server gespeichert, damit immer das gleiche Material zur Verfügung steht. Danach folgen drei Buttons zum Einstellen des Editier-Modus des Notenpanels. Der Ansichtsmodus erlaubt keine Veränderung. Im Auswahlmodus können Noten selektiert und bearbeitet werden. Der Einfügemodus ermöglicht es, Noten in die Partitur einzugefügen. Zum Verschieben und Einfügen ist es nötig, ein Raster über die Partitur zu legen. Die Rastergröße kann mit Hilfe der folgenden fünf Buttons auf der Symbolleiste festgelegt werden. Beim Einfügen von neuen Noten muss zusätzlich die Notenlänge der einzufügenden Note angegeben werden. Dies geschieht mit den nächsten fünf Buttons. Die letzten vier Buttons sind für die Steuerung der Abspielfunktion zuständig. In dem Textfeld kann das Tempo festgelegt werden.

Abbildung 7.1: Noteneditor: 1. Symbolleiste

Die zweite Symbolleiste stellt Buttons zum Verändern markierter Noten zur

Verfügung. Das Radiergummi symbolisiert die Löschfunktion. Mit der nächsten Buttongruppe ist es möglich, markierte Noten mit einem Vorzeichen zu versehen oder bestehende Versetzungszeichen aufzulösen. Die vorletzte Gruppe ermöglicht es, verschiedene Arten von Akzenten auf die markierten Noten anzuwenden. Mit der letzten Gruppe können Überbindungen erstellt werden oder übergebundene Noten zusammengefasst werden.

Abbildung 7.2: Noteneditor: 2. Symbolleiste

Die dritte Symbolleiste bietet Möglichkeiten zur Systemveränderung. Das aktive System wird mit der Combobox ausgewählt. Mit Hilfe der Buttons in der ersten Gruppe kann ein System hinzugefügt, das aktive System gelöscht, ein Takt an die Systeme angefügt oder es können überfüssige Takte gelöscht werden. Der Notenschlüssel wird mit den Buttons der zweiten Gruppe festgelegt. Aus Gründen der Übersichtlichkeit stehen hier nur die drei gebräuchlichsten Schlüssel, der Violin-, Alt- und Bassschlüssel zur Verfügung. Andere Schlüssel, wie zum Beispiel der Tenorschlüssel, könnten leicht hinzugefügt werden. Beim Klick auf den vorletzten bzw. letzten Button öffnet sich ein Pop-Up-Menü, über das die Tonart bzw. die Taktart des Stückes festgelegt werden kann.

Abbildung 7.3: Noteneditor: 3. Symbolleiste

7.1.2 Anzeigebereich

Im Anzeigebereich wird die Notengrafik angezeigt. Befindet sich der Bereich im VIEW-Modus, reagiert er nicht auf Mausereignisse. Im SELECT-Modus können Noten ausgewählt und verschoben werden. Hier gibt es einen Unterschied in der Bedienung der beiden Anwendungen. Einzelne Noten werden in beiden Fällen durch einen einfachen Klick ausgewählt. Mehrere Noten werden mit einem Auswahlrechteck ausgewählt. Beim Java-Applet geschieht dies durch Drücken der linken Maustaste und durch anschließendes Aufziehen eines Auswahlrechteckes. Beim Loslassen der Maustaste werden alle Noten im Auswahlrechteck selektiert. Dieses Verhalten ist man z.B. aus den gängigen DTP-Programmen gewohnt.

Bei der SVG-Version wurde ein anderer Weg gewählt, da bei einigen Browsern auf manchen Plattformen das Aufziehen des Auswahlrechtecks als Drag-And-Drop für die gesamte SVG-Grafik oder den in der Grafik enthaltenen Text interpretiert wird. Aus diesem Grund startet man die Auswahl mit einem Klick, zieht das Auswahlrechteck auf und klickt dann noch einmal, um den Auswahlvorgang zu beenden. Nachdem Noten ausgewählt wurden, können verschiedene Aktionen durchgeführt werden.

Das Verschieben von Noten geschieht auf die gleiche Weise wie das Auswählen: Bei der Applet-Version wird der Mauszeiger auf eine markierte Note gerichtet und die Noten dann per normalem Drag-And-Drop verschoben. Bei der SVG-Version tritt das gleiche Problem wie beim Auswählen der Noten auf. Deshalb wird auch hier der Verschiebe-Vorgang mit einem Klick auf eine markierte Note gestartet, die Noten dann durch Bewegen der Maus verschoben. Nach einem weiteren Klick wird der Verschiebe-Vorgang beendet. Sind Noten ausgewählt, können alternativ zum Verschieben auch die Aktionen der mittleren Symbolleiste (Abb. 7.2) ausgeführt werden.

7.2 Übungsprogramm

Während bei den beiden Noteneditoren ein Überblick über die meisten Funktionen gegeben wird, sollen die Übungsprogramme denkbare Einsatzmöglichkeiten zeigen. Der Test, ob eine Aufgabe richtig gelöst wurde, basiert auf einem Vergleich mit einer Lösung. Die Überprüfung läuft Note für Note ab, und es werden keine Hilfestellungen oder Korrekturvorschläge zur Lösung gegeben. Dies wäre bei einem richtigen Musiklehre-/Gehörbildungsprogramm natürlich nötig.

7.2.1 Datenverwaltung

Die Aufgaben werden in der XML-Datenbank in einer Collection `exercises` gespeichert. Die Klasse `ExerciseController` stellt die Verbindung zur Datenbank her und liefert die Übungen in Form von `Exercise`-Objekten. Eine Aufgabe besteht aus einer Beschreibung, einer Vorlage, mit der gestartet wird, und einer Lösung, mit der die Benutzereingaben verglichen werden. Zusätzlich wird noch die Anzahl der Versuche mitgezählt. Die Methode `check()` wertet die Benutzereingaben aus und liefert `true` zurück, wenn die Aufgabe richtig gelöst wurde. Ansonsten wird `false` zurückgeliefert und die Anzahl der Versuche um eins erhöht. Die verschiedenen Übungen werden in einem `Course`-Objekt organisiert, welches die Übungen der Reihe nach vom Server lädt, in Exercise-Objekte umwandelt und dann dem Client zur Verfügung stellt. Beim Java-Applet geschieht dies direkt beim Clienten, bei der SVG-Variante werden die `Course`-Objekte zentral auf dem Server verwaltet (analog zum `ScoreEditor`, Abschnitt 6.3).

7.2.2 Bedienelemente

Die Symbolleisten wurden auf die benötigten Elemente reduziert. Da keine Veränderungen an den Systemen vorgenommen werden müssen, wurden zum Beispiel die Combobox zur Auswahl und die Buttons zum Erstellen und Löschen von Systemen entfernt. Neu hinzugekommen ist ein Panel, das Informationen über die aktuelle Aufgabe anzeigt und auch einen Button zum Überprüfen und zum Abspielen der Lösung enthält.

Es stehen derzeit fünf Aufgaben zur Verfügung, die die verschiedenen Möglichkeiten zur Interaktion zeigen.

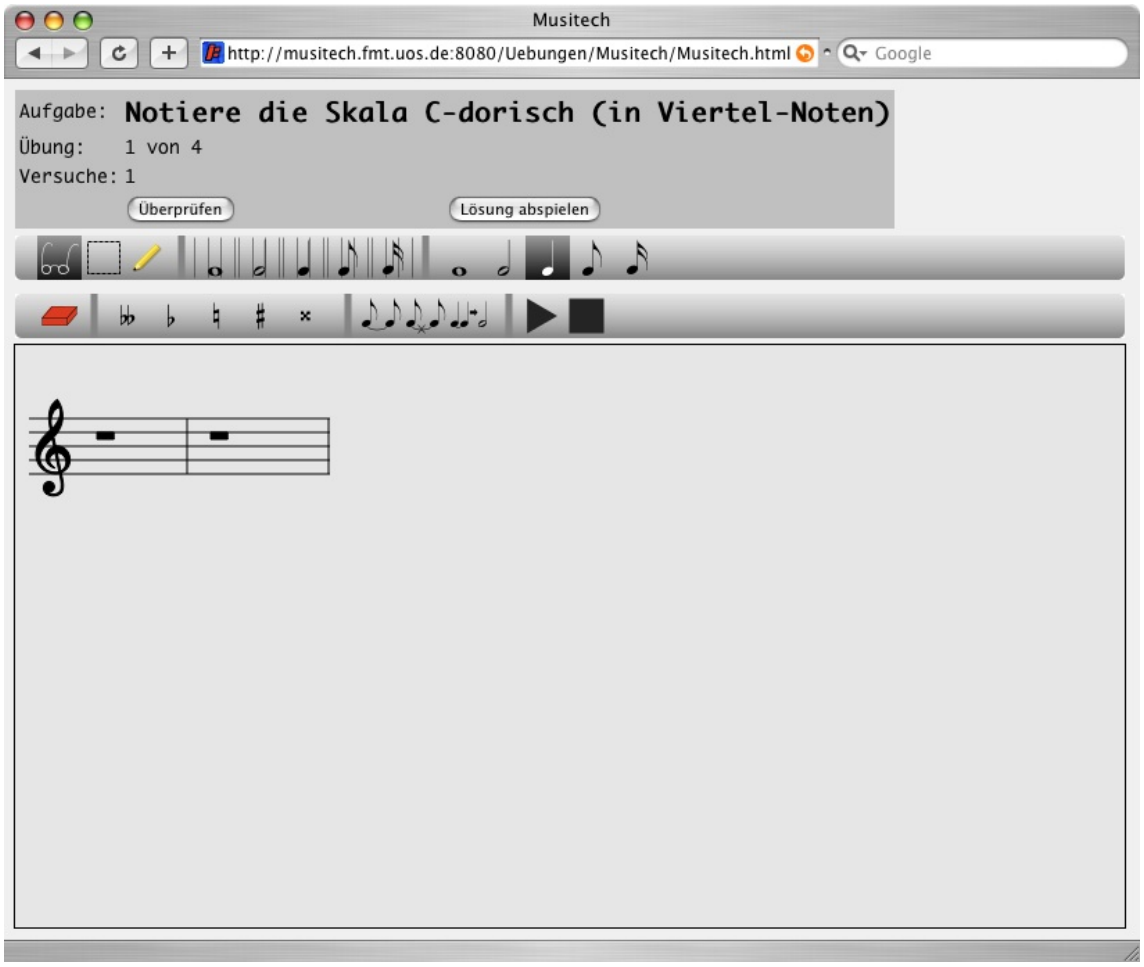


Abbildung 7.4: Übungsprogramm: 1. Übung

7.3 Vergleich der beiden Techniken

7.3.1 Java-Applet

Das Java-Applet kann auf die schon in MUSITECH enthaltene Möglichkeit der Notendarstellung zurückgreifen, die eine hochwertige Anzeige von Notengrafik ermöglicht. Durch die Verwendung der SWING-Bibliothek, die für alle Plattformen verfügbar ist, für die es auch die Java-VM gibt, ist eine einheitliche Event-Verarbeitung (im Wesentlichen Maus-Events) gewährleistet. Auch die Notengrafik, die mit Hilfe des Java 2D-API erzeugt wird, sieht auf allen Plattformen, unabhängig vom Browser, identisch aus.

Probleme ergeben sich bei der Integration in eine bestehende Web-Seite. Das

Java-Applet bietet die Möglichkeit, zwischen zwei Look-And-Feels zu wählen[41]: Es gibt das Cross-Platform-Look-And-Feel, das auf jeder Plattform identisch aussieht, und das System-Look-And-Feel, das sich am allgemeinen Look-And-Feel des Betriebsystems orientiert, teilweise sogar native Bestandteile nutzt. Bei beiden Alternativen wirkt das Applet aber wie ein fremdes Objekt in der Seite (das es ja auch ist) und integriert sich nicht in das Seiten-Design.

Damit die Notenbeispiele abgespielt werden können, benötigt das Applet vollen Zugriff auf die Soundkarte. Aus Sicherheitsgründen werden Applets aber in einer *Sandbox* ausgeführt und somit der Zugriff auf das System untersagt[42]. Der Autor des Applets kann jedoch das Applet mit dem Programm *jarsigner* signieren. Mit Hilfe eines Zertifikates kann der Benutzer dann nachher die Echtheit des Applets überprüfen und ihm mehr Rechte einräumen. Ein Restrisiko bleibt jedoch bestehen.

Ein anderer Nachteil ist die Wartezeit bis zum Start des Programms. Wenn die Seite aufgerufen wird, muss zunächst die Java-VM gestartet werden. Da das Applet wie ein normales Programm auf dem Client ausgeführt wird, muss dann noch das Programm mit den zugehörigen Bibliotheken heruntergeladen werden. Die MUSITECH-Bibliothek umfasst schon ca. 2,5 MB, kommen noch zusätzliche Funktionen hinzu, kann dies leicht noch mehr[43] werden.

7.3.2 SVG

Der Nachteil der unter Umständen langen Wartezeit tritt bei der SVG(-HTML)-Version nicht auf. Da die Berechnung der Notendarstellung und der Zugriff auf die Datenbank auf dem Server geschieht, muss dieser Code nicht zum Browser übertragen werden. Es muss allein der Java-Script-Code für die Aufbereitung der Darstellung und die Verarbeitung der Benutzereingaben übertragen werden. Da in den neueren Browsern die SVG-Unterstützung direkt integriert ist, muss kein Plugin mehr gestartet werden.

Leider ist die Anwendung sehr von der SVG-Implementation des Browsers abhängig. Die Implementationen sind sehr unterschiedlich; und um eine möglichst große Unabhängigkeit zu erreichen, können nur Funktionen benutzt werden, die von allen Browsern unterstützt werden. Ein Beispiel hierfür ist die Benutzung von SVG-Fonts. Die Symbole in der Notengrafik werden mit Hilfe einer True-Type-Schrift erzeugt. Java bietet die Möglichkeit, die Schriftdatei direkt zu laden und zu verwenden, damit die Schrift nicht zuvor auf dem System installiert werden muss. SVG bietet die Möglichkeit, im `def`-Element SVG-Fonts einzubetten oder aus einer externen Datei zu laden. Die einzelnen Buchstaben der Schrift werden dann durch Pfade beschrieben. Mit Hilfe des Apache Batik SVG Toolkits können - wie beschrieben - True-Type-Schriften in SVG-Fonts umgewandelt werden. Unter Mac OS X funktionieren SVG-Fonts mit jedem Browser, unter Windows jedoch nur mit Opera. Deshalb wurde diese Möglichkeit deaktiviert, und die Schriftdatei *gin.ttf* muss zuvor

[41] Von der Möglichkeit, ein eigenes Look-And-Feel zu erstellen, soll aufgrund der Komplexität abgesehen werden.

[42] vgl. [Ullenboom, 2007], S. 1372ff.

[43] In diesem Fall ist es die Xindice-Bibliothek für den Zugriff auf die XML-Datenbank.

auf dem System installiert werden.

Auch bei der Darstellung können Probleme auftauchen. Beim Java-Applet werden Veränderungen im Notenbild durch Neuberechnung der Notengrafik und durch anschließendes Neuzeichnen (`repaint()`) bewirkt. Bei SVG hat man keinen direkten Einfluss auf die Zeichenfunktionen. Änderungen werden im DOM vorgenommen (Änderungen von Attributen, Hinzufügen oder Löschen von Elementen). Der Browser ist dann für das Aktualisieren der Darstellung verantwortlich. Dies funktioniert aber nicht immer ganz zuverlässig. So kann es durchaus passieren, dass beim Ändern von einem feinen zu einem groben Raster zunächst noch einige Linien des feineren Rasters auf dem Bildschirm bleiben.

Fehler können auch bei der Eventverarbeitung auftreten: Fährt man im Selektionsmodus mit der Maus über eine Note, wird diese grau eingefärbt. Dafür wird den Notenköpfen ein Maus-Listener hinzugefügt, der den Kopf bei einem eintreffenden Mouse-Enter-Event einfärbt und bei einem Mouse-Leave-Event wieder zurücksetzt. Bei Opera unter Windows kann es nun bei einem Akkord mit mehreren Tönen übereinander vorkommen, dass beim obersten Ton die Maus-Events noch zuverlässig beim Überfahren ausgelöst werden, bei den darunterliegenden Tönen aber ein Versatz entsteht (vgl. Abb 7.5)

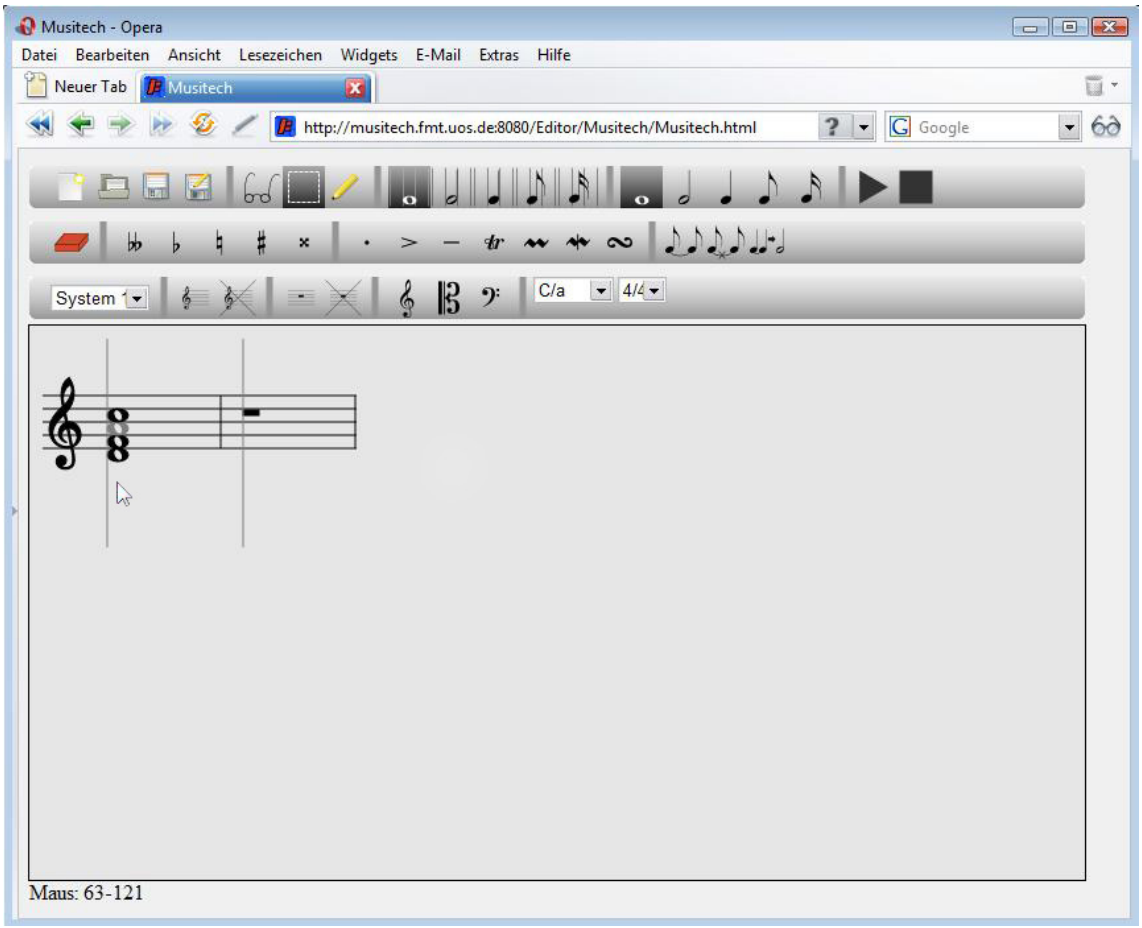


Abbildung 7.5: Versatz bei Opera unter Windows Vista

Generell kann man sagen, dass die SVG-Notengrafik nicht so performant ist wie die Java-Notengrafik. Bei größeren Stücken mit mehreren Systemen kann man das besonders beim Aufziehen des Auswahl-Rechecks und beim Einfügen am Noten-Cursor Performance-Probleme beobachten. Aber auch in diesem Bereich gibt es Unterschiede zwischen den Browsern.

Vom Sicherheitsaspekt her hat die SVG-Webanwendung Vorteile gegenüber dem Applet. Dadurch, dass die Anwendung im Browser ohne spezielle Plugins abläuft und nur die Techniken HTML, SVG und JavaScript nutzt, kann der Benutzer sicher sein, dass sie über den Browser hinaus keinen Zugriff auf das System bekommt.

8 Schlusswort

Die Untersuchung der Techniken zeigt, dass es auf beide Weisen möglich ist, Webinterfaces für interaktive Notengrafik zu entwickeln. Im direkten Vergleich weist jede Technik besondere Stärken, aber auch Schwächen auf.

Für die Java-Applets sprechen sicherlich die Qualität und Performance der Notengrafik und die Zuverlässigkeit. Um die Applets aber überhaupt ausführen zu können, muss die Java Virtual Machine installiert sein. Wenn die Seite mit dem ersten Applet geladen wird muss zunächst die Virtual Machine gestartet werden, danach muss das Applet heruntergeladen und gestartet werden. Auf diese Weise können bei normaler DSL-Internetverbindung beim ersten Start Wartezeiten von über 40 Sekunden auftreten. Wurde das Applet jedoch erst einmal in den Cache geladen, muss bei jedem weiteren Aufruf nur auf eine neue Version geprüft werden, und es startet innerhalb von wenigen Sekunden.

Für vollen Zugriff auf die Soundkarte müssen die Rechte des Applets ausgeweitet werden. Einem Applet Zugriff auf das System zu geben birgt natürlich Sicherheitsrisiken.

Schwierigkeiten gibt auch es bei der Integration in vorhandene Webseiten. Das Design der Anwendung lässt sich nur schwer an das Design der Webseite anpassen. Java verwendet standardmäßig das System-Look-And-Feel, das auch auf jeder Plattform anders aussieht. Zum Beispiel können die Buttons auf den Symbolleisten je nach System eine unterschiedliche Größe aufweisen. Dies ist bei Desktopanwendungen weniger ein Problem, da die Fenstergröße meist in gewissen Grenzen flexibel ist. Ein Applet jedoch bekommt von der umgebenen HTML-Seite genaue Größenangaben vorgegeben.

Genau in diesen Aspekten kann die SVG-Anwendung ihre Flexibilität zeigen. Das Notations-Panel kann in jedes beliebige Container-Element eingebettet werden. Auch die Buttons können in HTML designed werden und beliebig auf der Seite angeordnet werden. Leider ist die erzeugte Grafik nicht so hochwertig und auch die Eventbehandlung klappt beim Notation-Panel nicht so zuverlässig. Bei größeren Beispielen zeigen sich zudem auch die schon zuvor beschriebenen Performance-Probleme. Diese treten jedoch besonders bei Mozilla Firefox auf und fallen bei der Benutzung von Opera nicht so sehr ins Gewicht. Das zeigt, dass hier noch Optimierungsmöglichkeiten in der Rendering-Engine bestehen. Da die SVG-Unterstützung in den Browsern allgemein aber noch in den Kinderschuhen steckt und die Implementierungen noch weit von der vollständigen Unterstützung des Standards entfernt sind, kann man in Zukunft auf jeden Fall noch Verbesserungen erwarten.

Für jeden Anwendungszweck muss nun abgewogen werden, welche Lösung die meisten Vorteile bringt. Sollen nur kurze Beispiele mit wenigen Takten an verschie-

den Stellen der Webseite gezeigt werden oder sollen Noten in größeren Kontexten bearbeitet werden. Soll bei der Wiedergabe der Noten ein Cursor die Position im Notenbild anzeigen und die Möglichkeit bestehen die Wiedergabe zwischendurch zu unterbrechen, oder reicht es, wenn das ganze Stück im ganzen abgespielt wird? Je nachdem wie man die Prioritäten setzt, kommt entweder die eine oder die andere Lösung in Frage.

Die nächste wichtigste Aufgabe wird jetzt sein, die entwickelten Klassen in das MUSITECH-Framework zu integrieren, damit sich sich die Funktionen auch in normalen Java-Applikationen, die auf MUSITECH aufbauen, nutzen lassen. Darüber hinaus können je nach Bedarf auch Erweiterungen in der Funktionalität vorgenommen werden.

Erweiterungmöglichkeiten

Die für diese Arbeit entwickelten Klassen stellen die grundlegenden Möglichkeiten zur Interaktion mit dem Noten zur Verfügung. Darauf aufbauend lassen weitere Funktionen entwickeln. Einige naheliegende Funktionen, die besonders für größere Notenbeispiele von Bedeutung sind, sollen kurz erläutert werden.

Intelligentere Verarbeitung der Benutzereingaben Bisher werden Noten weitgehend unabhängig von ihrem unmittelbaren Kontext eingefügt. Man könnte zum Beispiel beim Einfügen von Achtelnoten die Nachbarnoten betrachten und gegebenenfalls Überbalkungen einfügen.

Mehrere Stimmen pro System Um auch polyphone Stücke zu bearbeiten ist notwendig mehrere Stimmen pro Notensystem zuzulassen.

Takt- und Schlüsselwechsel Für die hier beschriebenen kurzen Beispiele sind Takt- oder Schlüsselwechsel nicht notwendig. Falls in Zukunft aber größere Stücke möglich sein sollten, wäre dies ein nützliches Feature.

Zeilenumbrüche im Notentext Bisher werden die Noten ohne Zeilenumbrüche dargestellt. Größere Stücke lassen sich übersichtlicher in mehren Zeilen darstellen.

9 Literaturverzeichnis

[Eisenberg, 2002] Eisenberg, J. David:
SVG Essentials
Sebastopol: O'Reilly Media, 2002

[Enders (e.a.), 2004] Enders, Bernd (e.a.):
Abschlussbericht zum Projekt MUSITECH
Osnabrück: 2004

[Carl, 2006] Carl, Denny:
Praxiswissen AJAX
Köln: O'Reilly Media, 2006

[Garrett, 2005] Garrett, Jesse James:
Ajax: A New Approach to Web Applications.
`http://www.adaptivepath.com/publications/essays/archives/000385.php`

[Gieseking/Weyde, 2002] Gieseking, Martin; Weyde, Tillman:
Concepts of the MUSITECH Infrastructure for Internet-Based Interactive Musical Applications
Osnabrück: 2002

[Gieseking, 2001] Gieseking, Martin:
Code-basierte Generierung interaktiver Notengraphik
Osnabrück: Electronic Publishing Osnabrück, 2001

[Harrold/Means, 2002] Harold, Elliotte Rusty; Means, W. Scott:
XML in a Nutshell
Sebastopol: O'Reilly Media, 2002[2]

[Hempel, 1997] Hempel, Christoph:
Neue Allgemeine Musiklehre
Mainz: Schott, 1997

[Knuth, 1987] Knuth, Donald Ervin:
The TeXbook
Reading: Addison Wesley, 1987[12]

[Kühn, 2001] Kühn, Clemens:
Formenlehre der Musik
Kassel: Bäreneiter, 2001[6]

[McLaughlin, 2002] McLaughlin, Brett:
Java & XML
Köln: O'Reilly Media, 2002[2]

[O'Reilly, 2005] O'Reilly, Tim:
What Is Web 2.0
http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[Ullenboom, 2007] Ullenboom, Christian:
Java ist auch eine Insel
Bonn: Galileo Press, 2007[6]

[Vornberger, 2005] Vornberger, Oliver:
Datenbanksysteme (Vorlesungsskript SS 2005)
Osnabrück, 2005[9]

[Watt (e.a.), 2003] Watt, Andrew (e.a.):
SVG Unleashed
Indianapolis: Sams Publishing, 2003

[Weyde, 2002] Weyde, Tillman:
Auszug aus dem Projektbericht vom August 2002
Osnabrück: 2002

[Weyde, o.J.] Weyde, Tillman:
Modelling Cognitive and Analytic Musical Structures in the MUSITECH Framework
London: o.J.

[W3C, 2000] **Document Object Model (DOM) Level 2 Core Specification Version 1.0, W3C Recommendation 13 November, 2000**
http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

[W3C, 2003] **Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation 14 January 2003**
http://www.w3.org/TR/2003/REC-SVG11-20030114/

A Screenshots

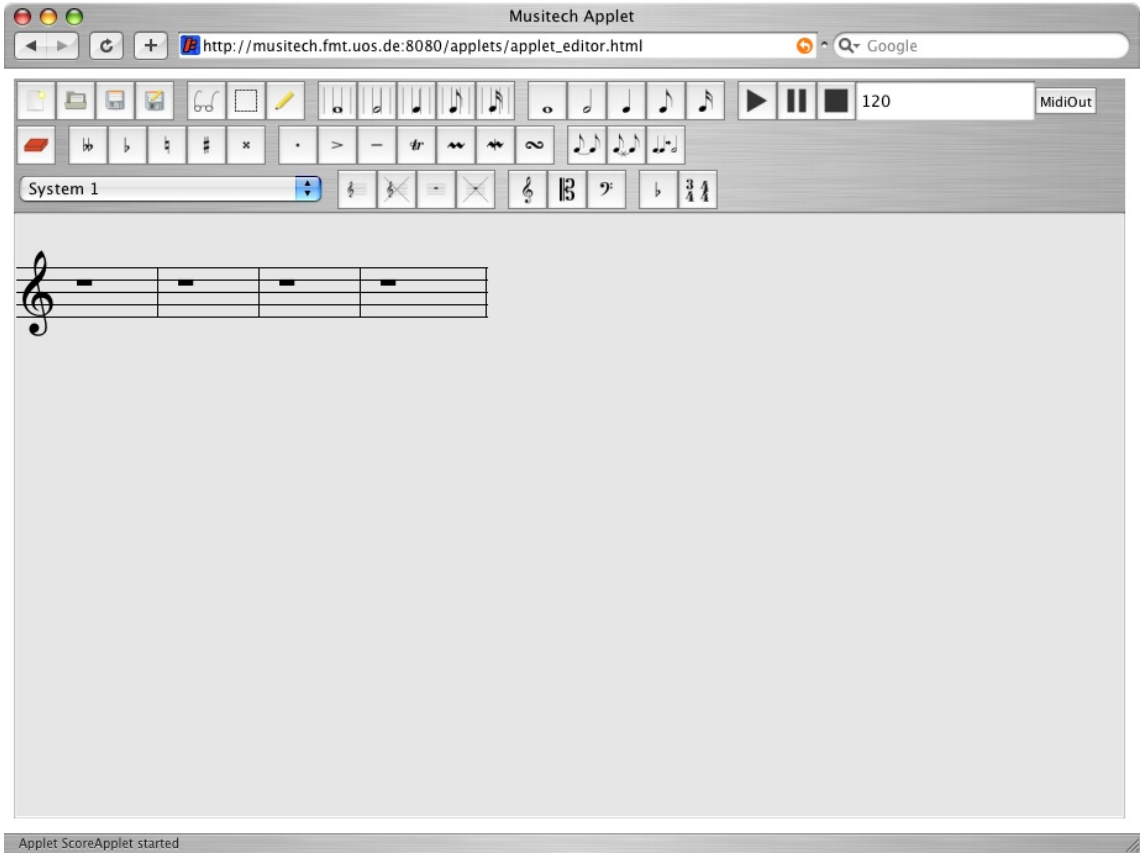


Abbildung A.1: Applet unter Safari 3.0.3 (Mac OS X 10.4)

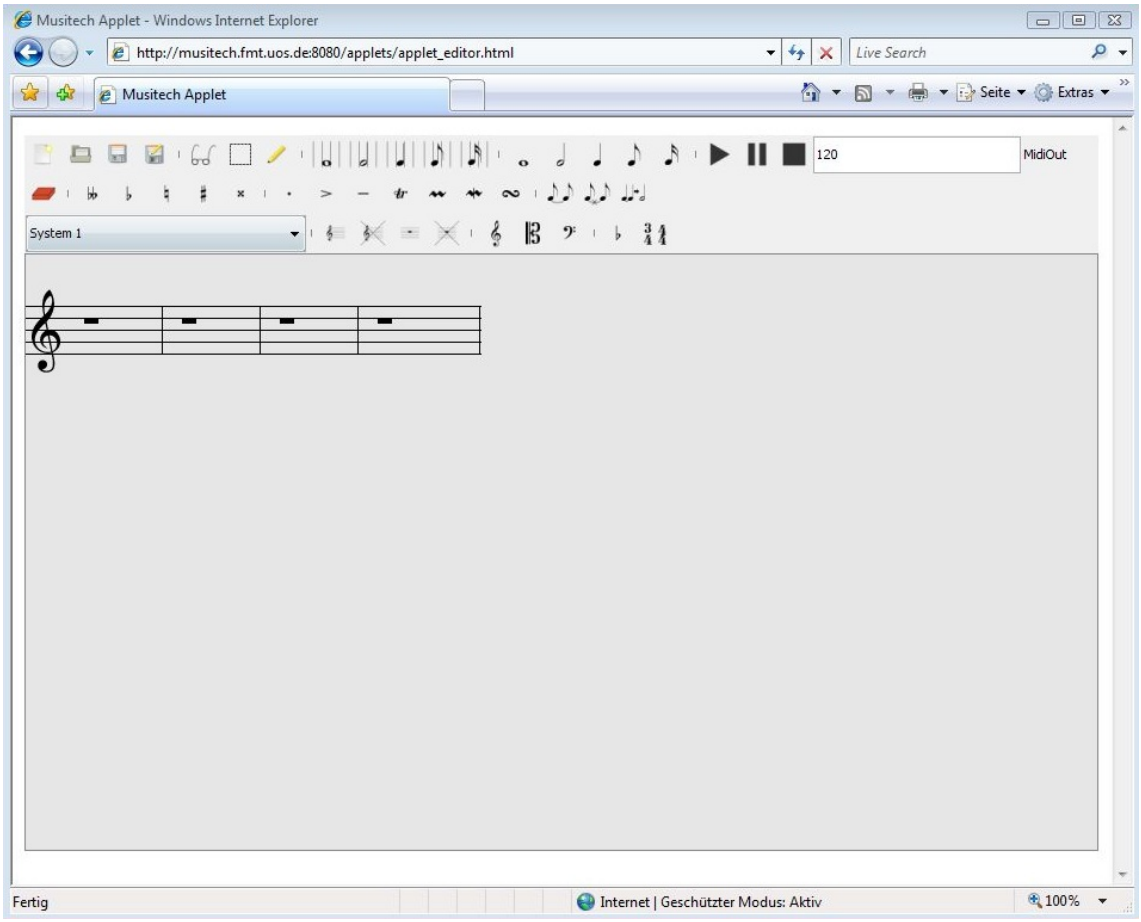


Abbildung A.2: Applet unter Internet Explorer 7.0.2 (Windows Vista x64)

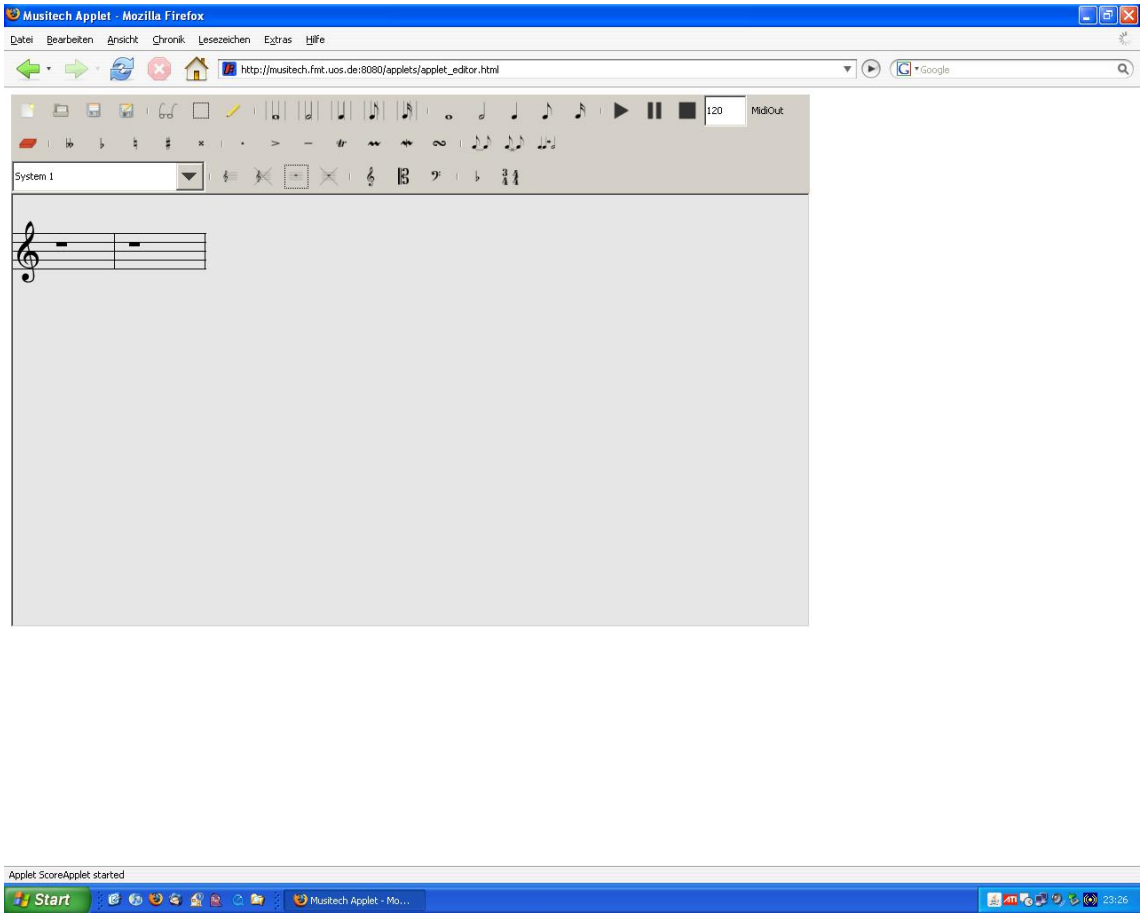


Abbildung A.3: Applet unter Mozilla 2.0.0.6 (Windows XP)

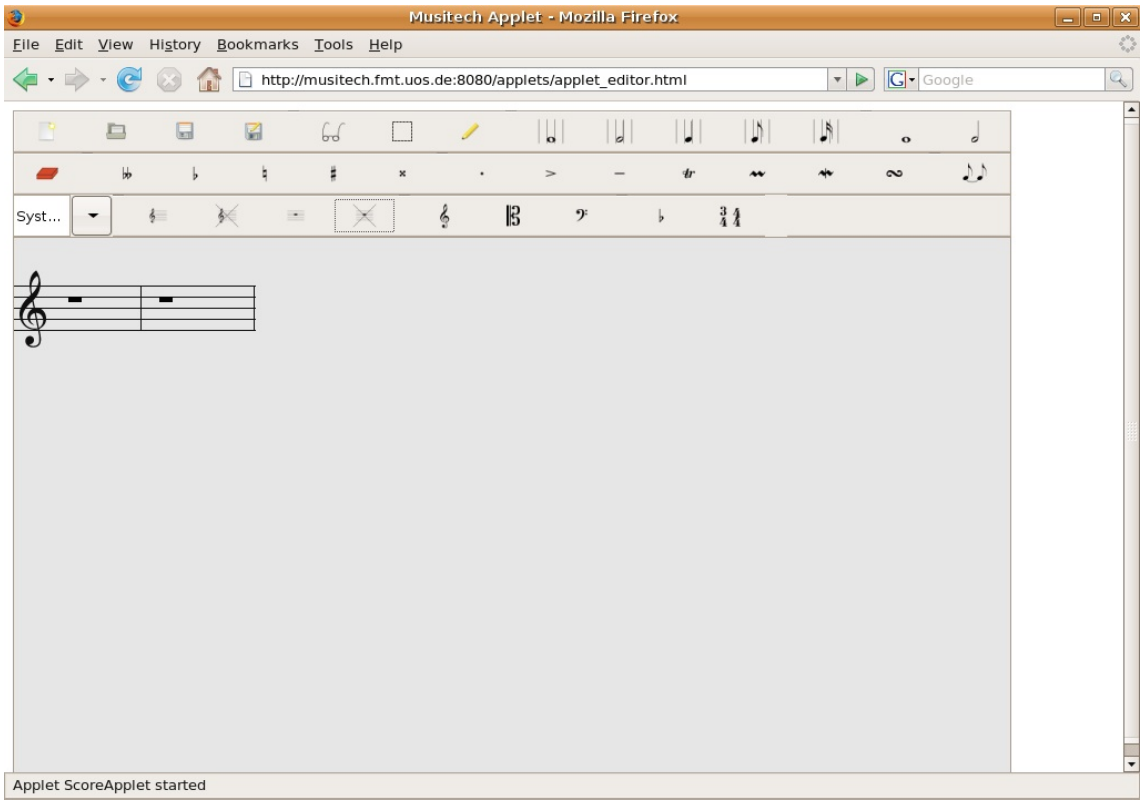


Abbildung A.4: Applet unter Mozilla 2.0.0.6 (Ubuntu 7.04)

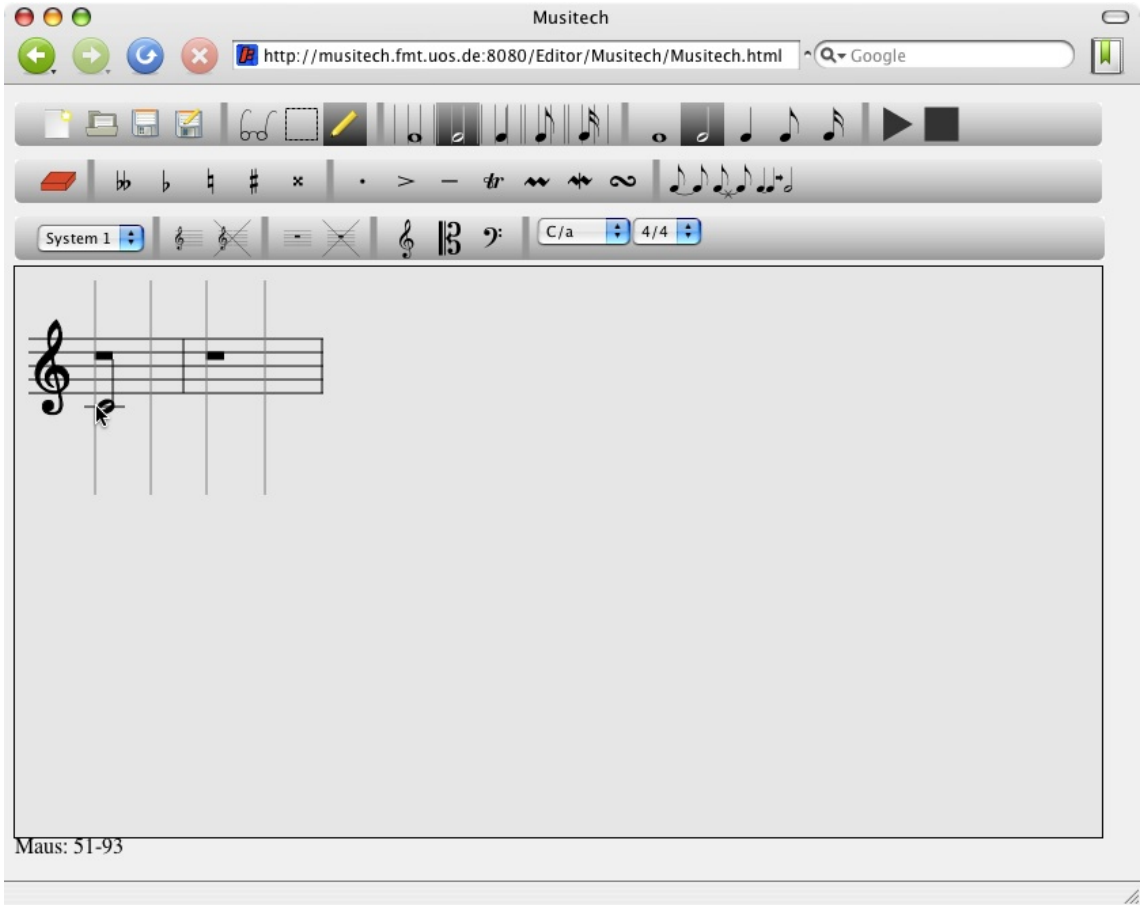


Abbildung A.5: SVG-Editor unter Camino 1.5.1 (Mac OS X 10.4)

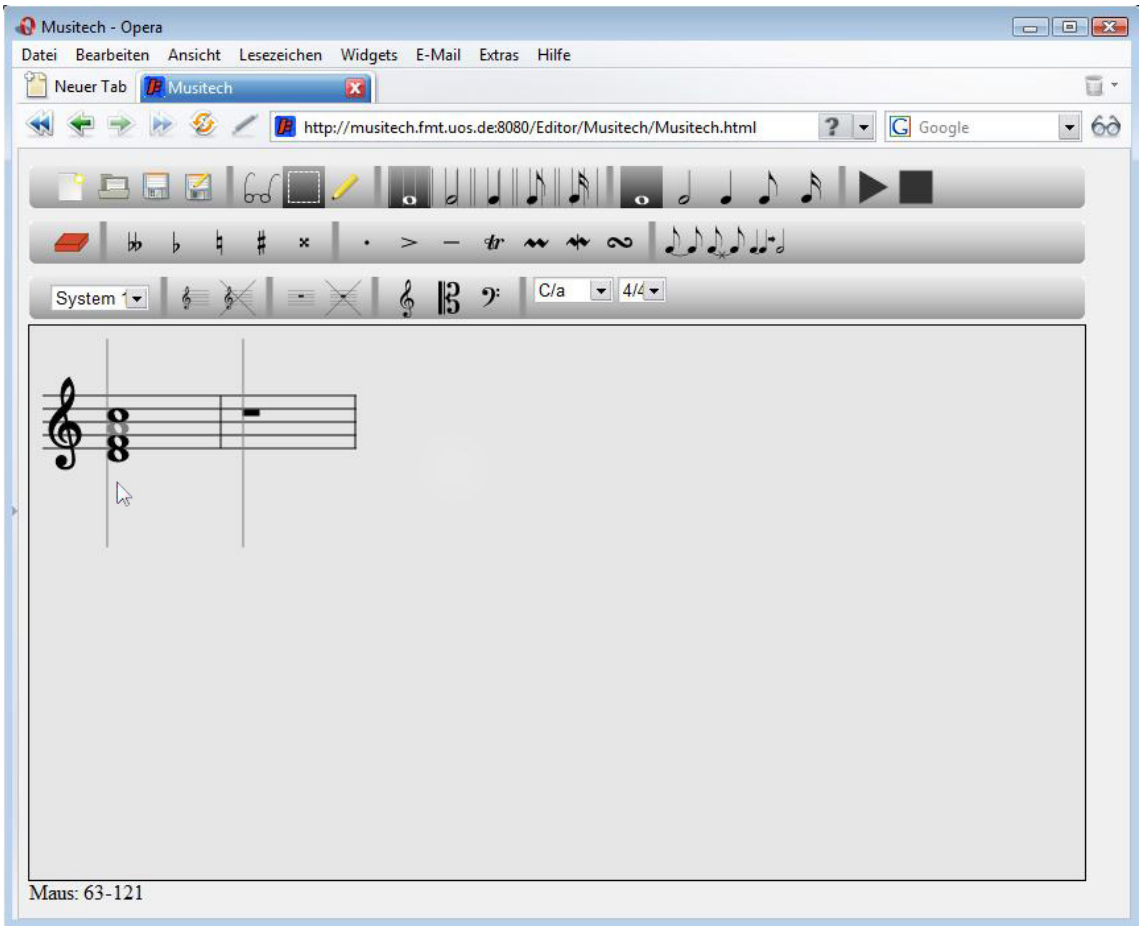


Abbildung A.6: SVG-Editor unter Opera 9.23 (Windows Vista x64)

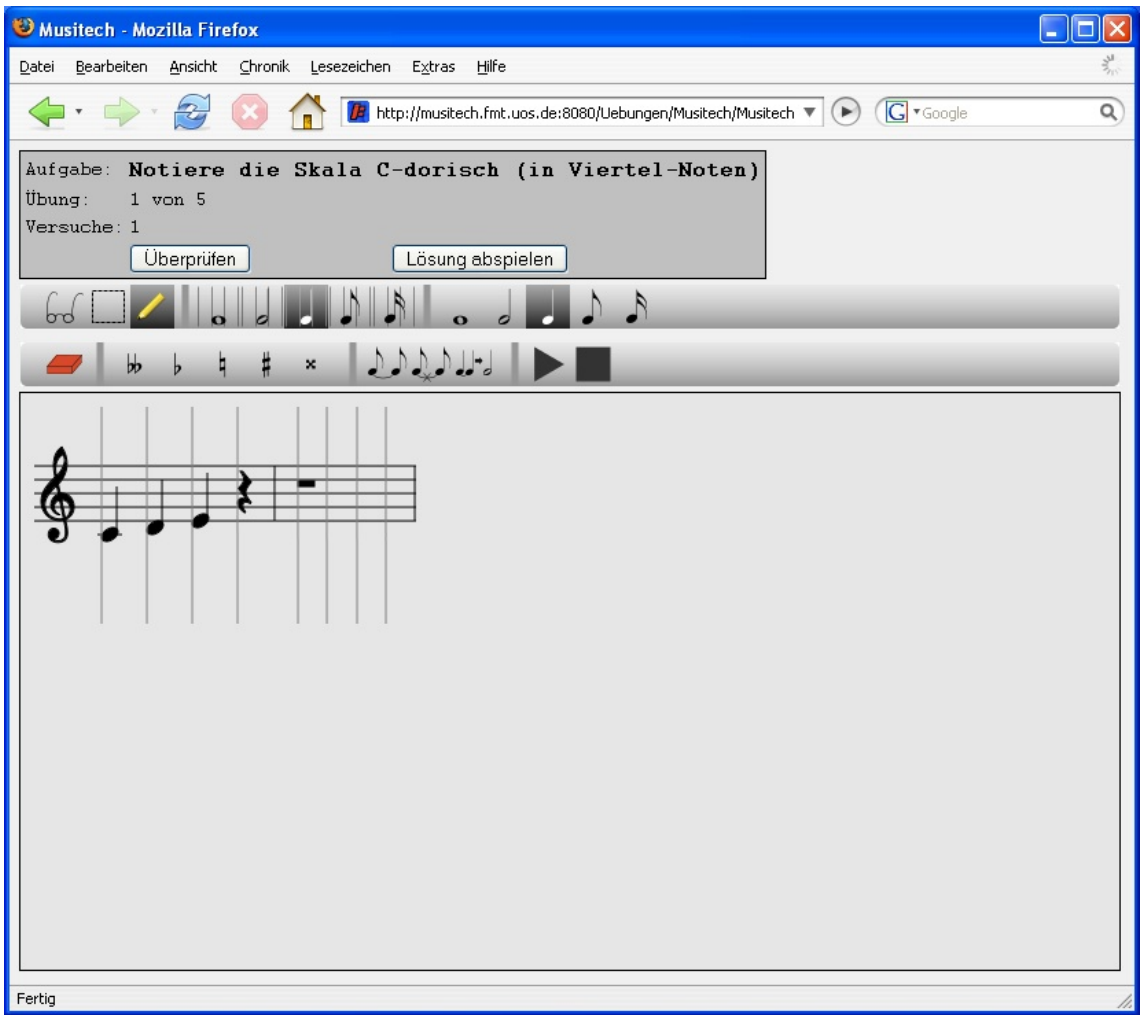


Abbildung A.7: SVG-Übungen unter Mozilla 2.0.0.6 (Windows XP)

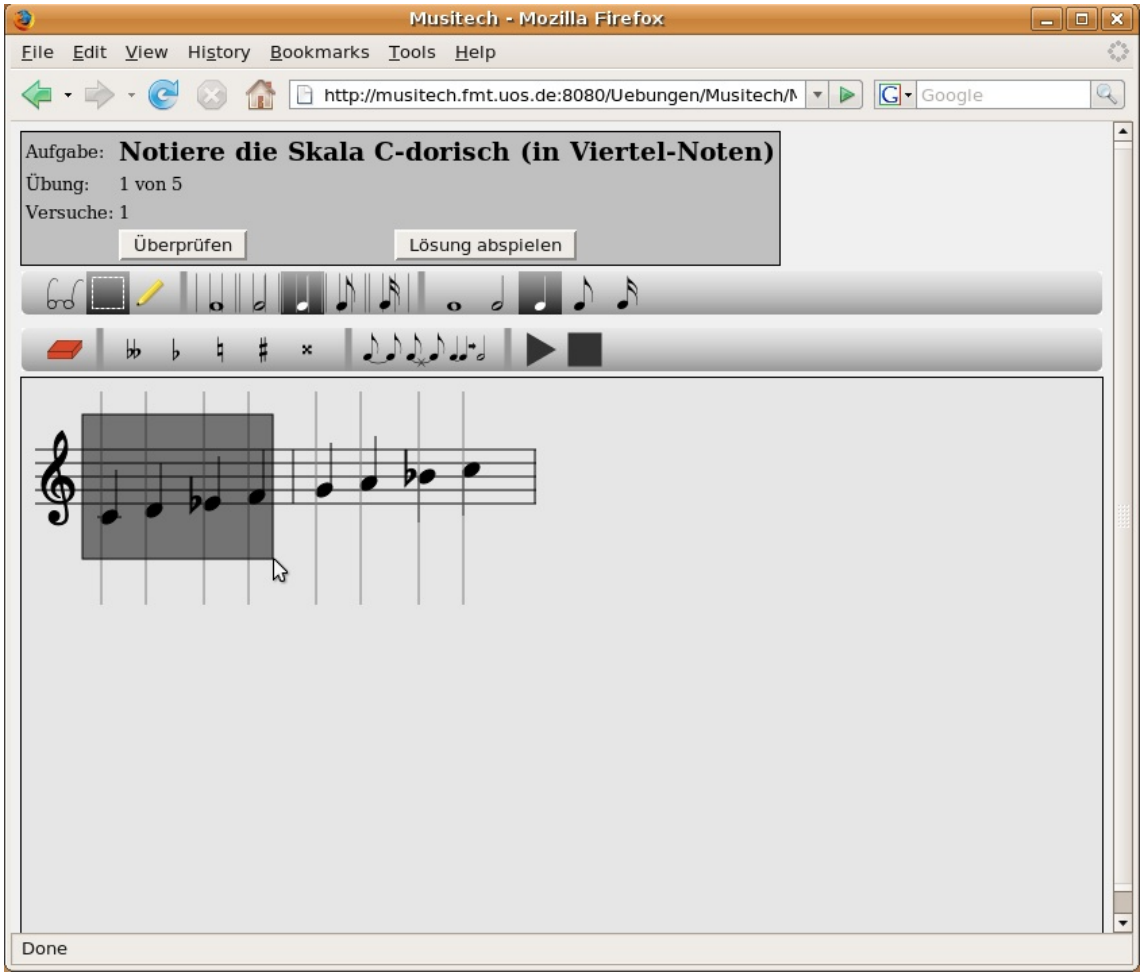


Abbildung A.8: SVG-Übungen unter Mozilla 2.0.0.6 (Ubuntu 7.04)

B Übersicht über die im Rahmen der Arbeit erstellten Klassen

B.1 `default`-Paket

ExerciseApplet

Einstiegsklasse für die Übungsanwendung (Applet-Version)

ExerciseApplication

Einstiegsklasse für die Übungsanwendung (Standalone-Version)

ScoreApplet

Einstiegsklasse für die Editor-Anwendung (Applet-Version)

ScoreApplication

Einstiegsklasse für die Editor-Anwendung (Standalone-Version)

B.2 Paket `client`

Logger

Die Klasse ermöglicht es dem Client, auf die Standardausgabe des Servers zu schreiben. Die definiert hierfür verschiedene statische Methoden.

Musitech

Einstiegsklasse für die Web-Anwendung. Es wird hier entweder die Editor- oder Übungs-Anwendung geladen.

B.3 Paket `client.exercises`

CourseInfo

Die Klasse `CourseInfo` ist eine Transport-Klasse und dient der Übertragung von Informationen über den Kurs und die aktuelle Aufgabe in der Übungsanwendung.

ExerciseInfoPanel

Das `ExerciseInfoPanel` ist ein Panel zu Anzeige von Informationen über die aktuelle Aufgabe. Es enthält auch die Buttons zum Überprüfen des Lösungsvorschlages und zum Abspielen der Lösung.

ExerciseToolBar1

Die `ExerciseToolbar1` ist die obere Symbolleiste in der Übungsanwendung. Sie enthält Buttons zum Einstellen des Rasters und der Notenlänge der einzufügenden Noten.

ExerciseToolBar2

Die `ExeciseToolbar2` ist die untere Symbolleiste in der Übungsanwendung. Sie enthält Buttons zum Löschen von Noten, zum Hinzufügen von Versetzungszeichen, zum Verbinden und Trennen von Noten und zum Abspielen der eingegebenen Noten.

SummaryDialog

Der `SummaryDialog` wird nach erfolgreichem Absolvieren des Kurses angezeigt. Er enthält die Anzahl der Versuche, die für jede einzelne Übung benötigt wurde.

B.4 Paket `client.player`

EmbedElement

Dieses Widget wird als `<embed>`-Element in das DOM eingefügt und dient als Grundlage für den `EmbedNotePlayer`. Es besitzt Methoden zum Setzen und Auslesen von Attributen.

EmbedNotePlayer

Diese Klasse ist eine Implementation des Interfaces `NotePlayer` auf Basis eines `EmbedElement`es.

MidiPlayer

Die Klasse `MidiPlayer` ist ein Widget, das es ermöglicht, ein Midi-File abzuspielen. Hierzu werden ein Start- und ein Stop-Button zur Verfügung gestellt. Über die Methode `setMidiFile(String)` kann die URL zu einem Midi-File angegeben werden.

NotePlayer

`NotePlayer` ist ein Interface, das alle Midi-Player implementieren müssen. Es legt folgende Methoden fest:

- `play()` zum Abspielen des vom Benutzer geladenen Stückes
- `play(int midiPitch)` zum Abspielen einer Note mit dem übergebenen Midi-Pitch
- `playSolution()` zum Abspielen der Lösung in der Übungsanwendung

Die Player sind in der Regel keine visuellen Komponenten.

NotePlayerFactory

Die Klasse `NotePlayerFactory` gibt über die statische Methode `getPlayer()` ein Player-Objekt zurück. Dies ist je nach Browser entweder ein `EmbedNotePlayer` oder ein `ObjectNotePlayer`. Die Klasse muss vor Verwendung über die Methode `init(Panel parentPanel)` initialisiert werden (`parentPanel` ist das Panel, in dem der Player eingefügt wird).

ObjectElement

Dieses Widget wird als `<object>`-Element in das DOM eingefügt und dient als Grundlage für den `ObjectNotePlayer`. Es besitzt Methoden zum Setzen und Auslesen von Attributen.

ObjectNotePlayer

Diese Klasse ist eine Implementation des Interfaces `NotePlayer` auf Basis eines `ObjectElement`es.

B.5 Paket `client.rpc`

AsyncCallbackAdapter

Die Klasse `AsyncCallbackAdapter` ist eine Adapter-Klasse für das Interface `AsyncCallback`

CourseRPC

`CourseRPC` ist ein Remote-Interface zur Verwaltung von Übungen. Hierüber kann in der Webanwendung Verbindung zum `Course`-Objekt des Benutzers auf dem Server hergestellt werden.

CourseRPCAsync

`CourseRPCAsync` ist die asynchrone Version des `CourseRPC`-Interfaces.

DbRPC

Über das Remote-Interface `DbRPC` kann vom Client aus die Verbindung zur Datenbank hergestellt werden. Es können z.B. Stücke geladen, gespeichert und gelöscht werden.

DbRPCAsync

Asynchrone Version des `DbRPC`-Interfaces.

ResultObject

Das `ResultObject` wird bei den meisten Anfragen an den Server zurückgeliefert. In einer `boolean`-Variable `successful` wird vermerkt, ob die Anfrage erfolgreich war, in einer Variable `tScore` wird ein `TScore`-Objekt gespeichert, falls eine Neuberechnung der Darstellung nötig war.

RPCConnection

Die Klasse `RPCConnection` ist die zentrale Klasse, die die Verbindungen zum Server herstellt und verwaltet. Zugriff auf die Verbindungs-Objekte geschieht über Getter-Methoden. Die Klasse selbst ist als Singleton realisiert.

ScoreRPC

`ScoreRPC` ist das Remote-Interface für alle entfernten Methoden, die mit der Veränderung des angezeigten Notenblattes zusammenhängen. Im Wesentlichen werden hier alle Methoden der Klasse `score.ScoreManipulator` verfügbar gemacht.

ScoreRPCAsync

Asynchrone Version des `ScoreRPC`-Interfaces.

B.6 Paket `client.score`

AuxLines

Die Klasse `AuxLines` ist ein `SVGContainer`, der sich um das Zeichnen von Hilfslinien über oder unter dem System im Einfüge-Modus kümmert. Über die Methode `setPosition(int x, int y)` wird die aktuelle Maus-Position übergeben. Die Lage zum ausgwählten Notensystem wird untersucht, und gegebenenfalls werden Hilfslinien hinzugefügt oder gelöscht.

Cursor

Die Klasse `Cursor` kümmert sich um das Zeichnen des Noten-Cursor im Einfüge-Modus. Dem Konstruktor wird die aktuell eingestellte Notenlänge übergeben, und der Cursor passt daraufhin sein Aussehen an. Mit den Methoden `setX(int x)` und `setY(int y)` wird die Maus-Position übergeben.

ScoreEditor

Das Panel `ScoreEditor` erzeugt einen Editor mit Symbolleisten und Anzeigefläche. Um das eigenliche `SVGPanel` wird noch ein Wrapper-Panel mit festgelegter Größe gelegt, das automatisch Scroll-Leisten anzeigt, wenn das `SVGPanel` zu groß wird.

ScoreListener

Klassen, die das Interface `ScoreListener` implementieren können sich beim `ScorePanel` anmelden und sich informieren lassen, wenn die Notendarstellung sich verändert hat.

ScoreManipulator

Die Klasse `ScoreManipulator` stellt analog zum `score.ScoreManipulator` die Methoden zum Verändern des Notenmaterials zur Verfügung. Die Aufrufe werden über ein `ScoreRPCAsync`-Objekt an den Server und den dort gespeicherten `score.ScoreEditor` (bzw. das damit verbundene `score.ScoreManipulator`-Objekt) weitergegeben. Vom Server wird eine neue Darstellung berechnet und zurückgegeben.

ScoreMouseAdapter

Die Klasse `ScoreMouseAdapter` kümmert sich um Maus-Event-Behandlung. Dazu implementiert sie die Interfaces `com.google.gwt.user.client.ui.ClickListener` und `com.google.gwt.user.client.ui.MouseListener` und agiert als Click- bzw. MouseListener für das `ScorePanel`.

ScorePanel

Das `ScorePanel` erbt von `SVGPanel` und kann `SVGScore`-Objekte anzeigen. Neue Notengrafiken können über die Methoden `setScore(TScore tScore)` und `setScore(SVGScore score)` übergeben werden. Ein mit dem Panel verbundener `ScoreMouseAdapter` ermöglicht das Verändern der Noten. Über die Methode `setGrid(int gridSize)` lässt sich die Rastergröße festlegen und über `setLength(int length)` die Notenlänge.

SelectionListener

Objekte, die das Interface `SelectionListener` implementieren, können sich beim `SelectionManager` anmelden und sich bei Veränderungen der Notenselektion benachrichtigen lassen.

SelectionManager

Die Klasse `SelectionManager` speichert die aktuelle Notenauswahl und benachrichtigt angemeldete Listener bei Veränderungen.

Transformer

Mit Hilfe der Klasse `Transformer` lassen sich `TScore`-Objekte in `SVGScore`-Objekte umwandeln. Diese können dann einem `ScorePanel` hinzugefügt werden.

B.7 Paket `client.score.dialogs`

NewDialog

Der `NewDialog` fragt den Benutzer beim Erstellen eines neuen Stückes nach Titel und Namen des Komponisten. Auch die Anzahl der vorgegebenen Takte kann festgelegt werden.

OpenDialog

Der `OpenDialog` ermöglicht es dem Benutzer, ein Stück aus der Datenbank zu laden. Dazu wird eine Liste der verfügbaren Stücke vom Server geladen und dem Anwender zur Auswahl angezeigt.

SaveAsDialog

Der `SaveAsDialog` kann ein Stück unter einem anderen Namen und/oder Komponisten in der Datenbank speichern.

B.8 Paket `client.score.elements`

SVGChord

Ein `SVGChord` stellt einen Akkord da. Er kann mehrere Noten in Form von `SVGPitches` enthalten. Der `SVGChord` kümmert sich auch um das Zeichnen des Notenhalses.

SVGLocalSim

Im `SVGLocalSim` sind alle Akkorde einer Zählzeit enthalten.

SVGMeasure

Ein `SVGMeasure` repräsentiert einen Takt. Er kann mehrere `SVGLocalSim`s enthalten.

SVGPitch

Ein `SVGPitch` repräsentiert einen Notenkopf. Er enthält eine Id, der auf dem Server eindeutig das zugehörige `de.uos.fmt.musitech.data.structure.Note`-Objekt zugewiesen werden kann.

SVGRest

Ein `SVGRest`-Objekt stellt eine Pause dar.

SVGScore

Die Klasse `SVGScore` steht an der Spitze der `ScoreObject`-Hierarchie und enthält zum Beispiel die Rasterlinien zur Darstellung des Rasters.

SVGScoreObject

Die `SVGScoreObject`-Klasse ist die Basisklasse für alle SVG-Objekte eines Notenbildes. Sie erbt von `SVGContainer` und kann daher andere SVG-Objekte aufnehmen.

SVGStaff

Ein `SVGStaff` repräsentiert ein Notensystem und enthält `SVGMeasure`-Objekte. Weiterhin zeichnet ein `SVGStaff` auch die Notenlinien des Systems.

SVGSystem

Ein `SVGSystem` repräsentiert eine Gruppe von zusammengefassten `SVGStaff`s (zum Beispiel durch eine Akkolade).

B.9 Paket `client.score.toolbar`

AbstractToolbar

Die Klasse `AbstractToolbar` ist die abstrakte Basisklasse für alle Symbolleisten der Webanwendung. Klassen, die von dieser Klasse erben, müssen die Methode `createWidgets()` implementieren.

AccentToolbar

Die Symbolleiste stellt Buttons zum Setzen von Akzenten zur Verfügung.

AlterationToolbar

Die Symbolleiste stellt Buttons zum Setzen von Versetzungszeichen zur Verfügung.

ClefToolbar

Die Symbolleiste stellt Buttons zum Verändern von Notenschlüsseln zur Verfügung.

FileToolbar

Die Symbolleiste stellt Buttons zum Erstellen, Öffnen und Speichern von Notenblättern zur Verfügung.

GridToolbar

Die Symbolleiste stellt Buttons zum Einstellen der Rastergröße zur Verfügung.

ImageButton

Mit Hilfe der Klasse ImageButton können aus Graphik-Dateien Buttons erstellt werden. Die Grafik muss dafür in drei Versionen für die drei Zustände des Buttons (normal, mouseOver, selected) vorliegen.

KeyToolbar

Die Symbolleiste stellt Elemente zum Einstellen der Tonart zur Verfügung.

MeasureToolbar

Die Buttons der `MeasureToolbar` ermöglichen es, neue Takte anzufügen oder überflüssige Takte zu löschen.

ModusToolbar

Die Symbolleiste stellt Buttons zur Auswahl des Modus zur Verfügung.

NoteLengthToolbar

Mit Hilfe der `NoteLengthToolbar` lässt sich die voreingestellte Notenlänge festlegen.

PlayerToolbar

Die `PlayerToolbar` bietet Elemente zu Steuerung der Abspielfunktionen.

RemoveNotesToolbar

Die `RemoveNotesToolbar` enthält einen Button zum Löschen der markierten Noten.

SystemToolbar

Die `SystemToolbar` bietet Elemente zur Auswahl, zum Löschen oder zum Hinzufügen von Systemen.

TieToolbar

Die `TieToolbar` enthält Buttons zum Überbinden, Trennen oder Verschmelzen von ausgewählten Noten.

ToolBar

Die Klasse `Toolbar` ist ein Panel, das die drei Symbolleisten für den Editor erstellt und anordnet.

B.10 Paket `client.svg`

Color

Die Klasse `Color` repräsentiert Farben. Die Color-Objekte können `Shape`s als Attributewerte - zum Beispiel für die Attribute `stroke` oder `fill` - zugewiesen werden. Dem Konstruktur der Klasse wird ein RGB-Farb-Tripel übergeben. Für die Grundfarben stehen vordefinierte Konstanten zur Verfügung.

Defs

Die Klasse `Defs` repräsentiert das SVG-Element `<defs>`. Sie kann Prototypen aufnehmen, die dann im `SVGPanel` verwendet werden können. Es können auch SVG-

Fonts eingebettet werden.

DOM2

Die Klasse `DOM2` erweitert die `DOM`-Klasse des GWT um Funktionen mit Namespace-Unterstützung.

DOM2Impl

In dieser Klasse befinden sich die Implementationen der in der Klasse `DOM2` bereitgestellten Funktionen.

SVGContainer

Ein `SVGContainer`-Objekt kann andere `SVGWidget`s aufnehmen und gruppiert diese im DOM mit einem `<g>`-Element.

SVGPanel

Das `SVGPanel` stellt im DOM das `<svg>`-Element dar. Es kann andere `SVGWidgets` enthalten. Als `Widget` kann es jedem beliebigen Panel hinzugefügt werden.

SVGWidget

Das `SVGWidget` ist die Basisklasse für alle SVG-Elemente. Es bietet Methoden zum Setzen und Auslesen von Attributen und übernimmt das Event-Handling.

B.11 Paket `client.svg.shapes`

Circle

Die Klasse `Circle` stellt einen Kreis dar. Dem Kontruktor werden Mittelpunkt und Radius übergeben.

Ellipse

Die Klasse `Ellipse` stellt eine Ellipse dar. Dem Kontruktor werden Mittelpunkt und Radius in x- und y-Richtung übergeben.

Image

Mit Hilfe der Image-Klasse ist es möglich eine Pixelgrafik in ein SVGPanel einzufügen.

Line

Mit Hilfe der `Line`-Klasse können Linien dargestellt werden.

Path

Mit Hilfe der `Path`-Klasse können Pfade dargestellt werden. Der Pfad kann dem Kontruktor als String übergeben werden.

Polygon

Mit Hilfe der `Polygon`-Klasse können Polygone dargestellt werden.

Polyline

Mit Hilfe der `Polyline`-Klasse können Polylinien dargestellt werden. Sie werden im Gegensatz zu Polygon nicht automatisch geschlossen.

Rectangle

Mit Hilfe der `Rectangle`-Klasse können Rechtecke dargestellt werden.

Shape

`Shape` ist die abstrakte Basisklasse für alle SVG-Grundformen. Sie bietet allgemeine Methoden wie zum Beispiel zum Setzen der Strichfarbe oder der Füllfarbe.

Text

Mit Hilfe der `Text`-Klasse kann einem `SVGPanel` Text hinzugefügt werden.

B.12 Paket `client.transport`

TChord

Ein `TChord` ist die Transportklasse für einen Akkord da. Er kann mehrere Noten in Form von `TPitches` enthalten. Die zusätzlich gespeicherten `TShape`-Objekte stellen zum Beispiel den Notenhals und die Fähnchen dar.

TLocalSim

Ein `TLocalSim` gruppiert alle Akkorde einer Zählzeit.

TMeasure

Ein `TMeasure` repräsentiert einen Takt. Er kann mehrere `TLocalSim`s enthalten.

TPitch

Ein `TPitch` repräsentiert einen Notenkopf. Er enthält eine Id, der auf dem Server eindeutig das zugehörige MUSITECH-`Note`-Objekt zugewiesen werden kann.

TRest

Ein `TRest`-Objekt stellt eine Pause dar.

TScore

Die Klasse `TScore` steht an der Spitze der `TScoreObject`-Hierarchie. Sie kann mehrere `TSystem`-Objekte enthalten. Zusätzlich speichert sie zum Beispiel noch die Linien zur Darstellung des Rasters.

TScoreObject

Die `TScoreObject`-Klasse ist die Basisklasse für alle Transportobjekte. Sie implementiert das Interface `com.google.gwt.user.client.rpc.IsSerializable` und stellt damit die Serialisierbarkeit sicher. In einem `TShape`-Array werden zu zeichnende Objekte gespeichert.

TStaff

Ein `TStaff` repräsentiert ein Notensystem und enthält `TMeasure`-Objekte. Weiterhin speichert ein `TStaff` auch die Notenlinien des Systems.

TSystem

Ein `TSystem` repräsentiert eine Gruppe von zusammengefassten `TStaff`s (zum Beispiel eine Akkolade).

B.13 Paket `client.transport.shapes`

TArc

Die Klasse `TArc` repräsentiert einen elliptischen Bogenabschnitt.

TLine

Die Klasse `TLine` repräsentiert eine Line.

TOval

Die Klasse `TOval` repräsentiert ein Oval.

TPolygon

Die Klasse `TPolygon` repräsentiert ein Polgyon.

TPolyline

Die Klasse `TPolyline` repräsentiert eine Polyline.

TRectangle

Die Klasse `TRectangle` repräsentiert ein Rechteck.

TShape

Die Klasse `TShape` ist Oberklasse für alle Klassen dieses Pakets

TString

Die Klasse `TString` repräsentiert einen String.

B.14 Paket `db`

DatabaseConnector

Die Klasse `DatabaseConnector` ist als Singleton realisiert und übernimmt die Kommunikation mit der XML-Datenbank. Sie bietet Methoden zum Navigieren in der Datenbank und zum Laden, Löschen und Hinzufügen von Dokumenten. XPath-Abfragen oder Dokumentänderungen per XUpdate sind auch möglich.

PieceAdmin

Die Klasse `PieceAdmin` übernimmt das Serialisieren und Deserialisieren von Stücken (`Piece`-Objekte) und die Verwaltung der Metadaten.

B.15 Paket `exercises`

Course

Ein `Course`-Objekt gruppiert verschiedene Aufgaben (`Exercise`s) zu einem Kurs.

Exercise

Ein `Exercise`-Objekt repräsentiert eine Übungsaufgabe. Es enthält das Lösung-Notensystem und das Notensystem des Benutzers. Beim Aufruf der `isCorrect()`-Methode werden die Systeme miteinander verglichen und bei Übereinstimmung `true`, sonst `false`, zurückgegeben.

ExerciseController

Die Klasse `ExerciseController` übernimmt das Laden der verschiedenen Aufgaben aus der Datenbank.

ExerciseInfoPanel

Dies ist die SWING-Version des `ExerciseInfoPanel`s aus dem Paket `client.exercises`.

ExercisePanel

Die Klasse `ExercisePanel` sammelt alle Bedienelemente und den `ScoreEditor` und richtet sie auf einem Panel aus.

ExerciseToolbar

Auf der `ExerciseToolbar` werden alle für das Übungapplet benötigten Bedienelemente mit Hilfe von zwei Symbolleisten ausgerichtet.

B.16 Paket `score`

NoteCursor

Mit Hilfe der Klasse `NoteCursor` kann im Einfüge-Modus ein Noten-Cursor erzeugt werden. Die Methode `setPosition(int x, int y)` legt die Position des Cursors fest. Die anzuzeigende Note kann mit der Methode `setLength(int i)` angegeben werden. Über die Methode `paint(Graphics g)` kann der Cursor dann auf ein Panel gezeichnet werden.

ScoreEditor

Die Klasse `ScoreEditor` ist eine Kernklasse der Arbeit. Sie erbt von der Klasse `ScorePanel` die Fähigkeiten zur Anzeige von Notensystemen. Über die Methode `setModus(Modus m)` kann sie in einen von drei möglichen Modi versetzt werden

ScoreEditorMouseAdapter

Der `ScoreEditorMouseAdapter` wird als Maus-Listener beim `ScoreEditor` angemeldet. Je nach Modus des `ScoreEditor`s reagiert er unterschiedlich auf Maus-Events. Im Auswahl-Modus wird zum Beispiel bei einem `mousePressed`-Event ein Auswahlrechteck erzeugt. So lange die Maustaste gedrücktgehalten wird, wird die Größe dieses Rechtecks bei einem `mouseDragged`-Event verändert. Wird die Maustaste dann losgelassen, werden alle Noten, die sich innerhalb des Rechteckes befinden ermittelt und über den MUSITECH-`SelectionManager` markiert.

ScoreEditorPanel

Das `ScoreEditorPanel` ordnet den `ScoreEditor` zusammen mit einem `ToolBarPanel` auf einem Panel an.

ScoreManipulator

Die Klasse `ScoreManipulator` ist eng verknüpft mit dem `ScoreEditor`. Sie bietet Methoden zum Verändern des zum `ScoreEditor` gehörenden `NotationSystem`. Dazu gehört zum Beispiel das Löschen und Einfügen von Noten, aber auch das Ändern von Takt- und Tonart.

TGraphics

Diese Klasse erbt von `java.awt.Graphics`. Bei jedem Aufruf einer Zeichenmethode wird ein zugehöriges `TShape`-Objekt erzeugt und in einer Liste gespeichert. Man kann einen `ScoreEditor` auf diesen Grafikkontext zeichnen lassen und enthält so eine Darstellung des Notenbildes mit `TShape`-Objekten.

Transformer

Lässt man den `ScoreEditor` direkt auf den `TGraphics`-Kontext zeichen, kann man die einzelnen Notationsobjekte in der Liste nicht identifizieren. Die Klasse `Transformer` baut daher einen Objektbaum mit Objekten aus dem Paket `client.transport` auf. Diese enthalten dann die `TShape`-Objekte

B.17 Paket `score.dialogs`

NewDialog

Der `NewDialog` fragt den Benutzer nach Titel, Komponist und Länge des Stückes und erstellt ein neues Stück nach den Vorgaben.

OpenDialog

Der `OpenDialog` zeigt dem Benutzer eine Liste der verfügbaren Stücke. Das ausgewählte Stück wird dann geladen.

SaveAsDialog

Der `SaveAsDialog` fragt den Benutzer nach Titel und Komponist und speichert das geöffnete Stück dann mit diesen Informationen in der Datenbank.

B.18 Paket `score.icons`

Icons

Die Klasse `Icons` enthält eine Methode `getIcon(int size)`, mit der die Grafiken in diesem Paket als `ImageIcon`s mit der vorgegebenen Größe geladen werden.

B.19 Paket `score.toolbar`

AbstractToolbarSection

Die `AbstractToolbarSection` ist die abstrakte Basisklasse für alle Symbolleistenabschnitte. Sie enthält zum Beispiel eine `update()`-Methode, die die Unterklassen implementieren müssen, wenn sie bei Änderungen des `ScoreEditor`s ihre Darstellung aktualisieren müssen. In der `createItems()`-Methode sollen die Elemente des Abschnittes erstellt werden. Eine `addItems(JToolbar tb)`-Methode fügt sie die dann der übergebenden `JToolbar` hinzu.

AccentSection

In der `AccentSection`-Klasse werden `Action`[44]-Objekte zum Hinzufügen von Akzenten erstellt.

[44] javax.swing.Action

AlterationSection

Mit den `Action`-Objekten der `AlterationSection` können Versetzungszeichen erstellt werden.

ClefSection

Die `ClefSection` stellt `Action`-Objekte zum Verändern der Notenschlüssel bereit.

FileSection

Mit den `Action`-Objekten der `FileSection` werden die Dialoge zum Öffnen, Speichern usw. angezeigt.

GridSection

Die `GridSection` stellt fünf `Action`-Objekte zum Einstellen der Rastergröße zur Verfügung.

KeySignatureSection

Mit der `KeySignatureSection` wird der übergebenen `JToolbar` ein Popup-Menü hinzugefügt, das es ermöglicht, die Tonart des Stückes zu verändern.

ModusSection

Mit Hilfe der `Action`-Objekte der `ModusSection` kann der Modus des `ScoreEditors` festgelegt werden.

NoteLengthSection

Die `NoteLengthSection` stellt fünf `Action`-Objekte zum Einstellen der Notenlänge zur Verfügung.

PlayerSection

Mit den `Action`-Objekten der `PlayerSection` kann das `NotationSystem` des `ScoreEditor` abgespielt werden. In einem Textfeld wird das Tempo festgelegt.

SectionController

Das Interface `SectionController` schreibt zwei Methoden vor: Die Methode `getScoreEditor()` soll den `ScoreEditor` zurückgeben. Die Methode `update()` wird aufgerufen, wenn die Darstellung aktualisiert werden soll.

SystemSection

Die `SystemSection` fügt der Toolbar eine Combobox zum Festlegen des aktuellen Systems hinzu. Weiterhin gibt es noch `Action`-Objekte zum Erstellen und Löschen von Systemen,

TieSection

Die `TieSection` fügt der übergebenen ToolBar drei `Action`-Objekte hinzu: Eine `Action` zum Erstellen von Überbindungen, eine zum Löschen von Überbindungen und eine zum Verschmelzen von übergebundenen Noten.

ToolBarPanel

Das `ToolBarPanel` implementiert das Interface `SectionController`. Es kann also dazu benutzt werden, die Sections zu erstellen. Es besitzt drei `JToolbars`, die den verschieden Sections zum Füllen übergeben werden.

B.20 Paket `server`

CoursePool

Der `CoursePool` ist eine Klasse, die die verschiedenen `Course`-Objekte unter der Session-Id der Clients der Webanwendung im Speicher hält.

CourseRPCImpl

Das Servlet `CourseRPCImpl` implementiert das Interface `client.rpc.CourseRPC` und speichert die einzelnen `Course`-Objekte im `CoursePool`.

DbRPCImpl

Das Servlet `DbRPCImpl` implementiert das Interface `client.rpc.DbRPC` und stellt für die Clients die Schnittstelle zu Datenbank dar. Geladene Stücke werden im `ScoreEditorPool` gespeichert.

PlaySolutionServlet

Das Servlet wandelt die Lösung einer Aufgabe in ein Midi-File um und sendet dies.

PlayUserInputServlet

Das Servlet wandelt das Stück des `ScoreEditors` des Clients in ein Midi-File um und sendet dies.

ScoreEditorPool

Im `ScoreEditorPool` werden die `ScoreEditor`-Instanzen der einzelnen Clients unter der Session-Id abgelegt.

ScoreRPCImpl

Das Servlet `ScoreRPCImpl` implementiert das Interface `client.rpc.ScoreRPC` und stellt dem Client die Funktionen zum Verändern des Stückes zur Verfügung. Dafür wird die zum Client gehörige Instanz des `ScoreEditor`s aus dem `ScoreEditorPool` geladen und der Funktionsaufruf an den `ScoreEditorManipulator` weitergegeben.

ScoreServlet

Das `ScoreServlet` ist Oberklasse der anderen Servlets und implementiert Funktionen für den direkten Zugriff auf den zum Client gehörenden `ScoreEditor`.

SingleNoteServlet

Dem SingleNoteServlet wird per GET-Parameter ein Midi-Pitch übergeben. Es generiert dann ein Midi-File mit einem Ton dieser Tonhöhe.

C Begleit-CD

Inhalt der Begleit-CD

- Arbeit als PDF-Datei
- Programmcode mit benötigten Bibliotheken
- MUSITECH-Projekt
- Google Web Toolkit, Version 1.4
- Jetty Webserver und Servlet-Container, Version 6.1.5

Erklärung

Hiermit erkläre ich, dass ich die Diplomarbeit selbstständig angefertigt und keine Quellen und Hilfsmittel außer den in der Arbeit angegebenen benutzt habe.

Lübbecke, den 05.09.2007 .....