Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Felix Jensen

Entwicklung einer DSL zur Komposition von Jazz-Musik

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Felix Jensen

Entwicklung einer DSL zur Komposition von Jazz-Musik

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Sarstedt
Zweitgutachter: Prof. Dr. Fohl

Eingereicht am: 15. März 2016

Felix Jensen

Thema der Arbeit
Entwicklung einer DSL zur Komposition von Jazz-Musik

Stichworte
Domänenspezifische Sprache, DSL, Jazz, Notation, Musik

Kurzzusammenfassung
Während für klassische Musik zumeist eine sehr präzise Notationsform gewählt wird, werden im Jazz häufig nur wenige Aspekte eines Stücks notiert. Die in diesem Genre üblichen Notationsformen sind für das Medium Papier ausgelegt und sind für Rechner schwer einzulesen, obwohl diese bei der Komposition sehr hilfreiche Werkzeuge sein können. Daher soll in dieser Arbeit ein System zur Notation von Musikstücken und der Interaktionen mit ihnen vorgestellt werden, dessen Fokus auf Jazz-Musik liegt.
Felix Jensen

Title of the paper
Development of a DSL for Jazz-Music Composition

Keywords
domain-specific language, DSL, Jazz, notation, music

Abstract
While the usual approach to music notation in the classical context is very detailed, in jazz music the notation is often much less precise. The common ways of notation in this genre are designed to be written on paper and are hard to read for computers, even though they could be a very helpful tool for composition. That is why this paper presents a system for notation of musical pieces in the genre of jazz and for interaction with them.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation und Zielsetzung

Ein großer Bestandteil des Alltags vieler Musiker ist das Lesen und Notieren von Musik. In der klassischen Musik sind Stücke oft akribisch bis ins letzte Detail ausnotiert, was es den Interpreten ermöglicht, sie mit allen Nuancen nachzuspielen. In anderen Musikrichtungen wird es mit der Notation nicht so genau genommen. Ein besonderes Beispiel ist der Jazz.

Die Jazz-Musik lebt in ihrem Wesen von Veränderung und Individualität. Die Improvisation ist ein wesentlicher Bestandteil, der einen großen Teil des Jazz ausmacht. Gleichzeitig hat er ein breites Fundament an theoretischen Grundlagen, die dem Musiker beim Musizieren helfen können. Aber bei aller Freiheit kommt der Jazz dennoch nicht ohne die Musiknotation aus. Dabei die gleiche Notationsform wie die klassischen Musik zu nutzen würde ihm aber nicht gerecht werden. Daher wird hier oft eine andere Form gewählt, beispielsweise die des Leadsheets.

In der heutigen Zeit sind Computer in jedem Haushalt zu finden und Kommunikation findet häufig darüber statt. Dagegen wurden die meisten bekannten Notationsformen noch vor den ersten Rechnern erfunden. Sie sind darauf ausgelegt, gut auf Papier notiert und gelesen zu werden. So können sie von einem Rechner nur schwierig eingelesen und interpretiert werden. Die Verwendung eines Computers könnte bei der Notation und Arbeit mit einem Musikstück aber auf verschiedene Art hilfreich sein.

Das Ziel dieser Arbeit ist daher die Entwicklung eines Systems zur Notation von Jazz-Musik, in dessen Zentrum eine Domain-Specific Language (engl. Domänenspezifische Sprache, DSL) steht. Beim Entwurf der Sprache sollen die wesentlichen Eigenschaften des Jazz beachtet werden. Darüberhinaus soll die interaktive Arbeit mit einem notierten Musikstück möglich sein und die enthaltenen Informationen des Stücks zum Abspielen genutzt werden können. Damit soll erreicht werden, dass die notierte Musik als Basis zur Improvisation dienen kann.

Das Ziel der Arbeit ist nicht, eine vollständige Abbildung aller möglichen Objekte der musikalischen Domäne in eine Sprache zu bringen, sondern eine elegante Lösung für die in

der Jazz-Musik wesentlichen Elemente zu finden. Auch eine harmonische Analyse soll hier nicht durchgeführt werden.

1.2. Gliederung der Arbeit

In Kapitel 2 werden zunächst einige musikalischen Grundbegriffe erläutert. Anschließend werden verschiedene Formen der Musiknotation besprochen. Außerdem erfolgt eine Einführung in Domain-Specific Languages.

Kapitel 3 beginnt mit einer Beschreibung der Anforderungen an das System. Anschließend erfolgt eine Überprüfung vorhandener Lösungen. In Kapitel 4 wird ein Konzept für eine Lösung vorgestellt und anschließend eine Architektur dieser Lösung erstellt.

Als nächstes folgt das Kapitel 5, in dem auf einige Implementierungsdetails eingegangen wird. Anschließend wird in einer Auswertung ein Vergleich zwischen den Anforderungen aus der Analyse und dem hier vorgestellten System durchgeführt. Dann werden noch einige Verbesserungsmöglichkeit vorgestellt. Das Kapitel 6 schließt die Arbeit mit Fazit ab.

2. Grundlagen

2.1. Musikalische Grundbegriffe

Im folgenden werden einige Grundbegriffe der Musik erläutert, die zum Verständnis dieser Arbeit benötigt werden. Die hier beschriebenen Informationen basieren auf den Werken *Neue Jazz-Harmonielehre* von Frank Sikora (Sikora, 2012), sowie *Die neue Harmonielehre* von Frank Haunschild (Haunschild, 1998).

2.1.1. Stammtöne

In der Musiktheorie werden Töne in sieben Stammtöne (A, B, C, D, E, F, G) gegliedert. Diesen Stammtönen sind Frequenzen zugeordnet. Bei der Verdoppelung der Frequenz wird von einer Oktave gesprochen. Die Oktave eines Tons hat denselben Namen, wobei die einzelnen Oktavräume zur Differenzierung vorangestellte Beiwörter besitzen. Die Oktavräume gehen vom Ton C aus und haben von tiefer Tonhöhe ausgehend die Namen Subkontra-Oktave, Kontra-Oktave, große Oktave, kleine Oktave, und ein- bis fünfgestrichene Oktave. (vgl. Grabner, 2011, S. 7)

Im deutschen wird das B mit H bezeichnet. Da sich die Arbeit aber auf Jazz bezieht und es dort üblich ist, die internationale Schreibweise zu verwenden, wird hier ebenfalls die internationale Schreibweise verwendet.

Der Tonabstand zwischen den einzelnen Stammtönen ist unterschiedlich. Die kleinstmögliche Tonabstand wird Halbtonschritt genannt. Zwischen den Tonschritten A-B, C-D, D-E, F-G und G-A beträgt der Tonabstand jeweils einen Ganzton, zwischen B und C, sowie E und F einen Halbton. Mit Hilfe von Versetzungszeichen können Stammtöne um einen Halbtonschritt erhöht (durch ein „♯“), bzw. erniedrigt (durch ein „♭“) werden. Dadurch ergibt sich ein gesamtes Tonspektrum von zwölf Tönen (A, A♯, B, C, C♯, D, D♯, E, F, F♯, G, G♯).

Der Abstand zwischen zwei Tönen wird als Intervall bezeichnet. Dieser Abstand wird in Halbtönen gemessen, wobei jedes dieser Intervalle einen festgelegten Namen hat. Eine Auflistung der Intervalle mit ihren Namen ist im Buch *Die neue Harmonielehre* von Frank Haunschild zu finden. (Haunschild, 1998, S.34)

2.1.2. Tonleitern

Eine Menge von Tönen innerhalb einer Oktave kann als Tonleiter bezeichnet werden. Die Folge jedes einzelnen der zwölf in einer Oktave enthaltenen Töne wird als chromatische Tonleiter bezeichnet. Reiht man die Stammtöne zu einer Folge auf, so erhält man von C ausgehend die siebentönige C-Dur-Tonleiter, von A ausgehend die A-Moll-Tonleiter. Um die Dur- und Moll-Tonleiter auf andere Grundtöne zu übertragen, kann man die Skalen als Intervallreihen darstellen:

dur	1	2	3	4	5	6	7	8
moll	1	2	♭3	4	5	♭6	♭7	8

Tabelle 2.1.: Struktur von Dur- und Moll-Tonleitern als Intervallreihen

In Tabelle 2.1 erkennt man, dass die Intervalle zwischen den einzelnen Stufen in der Dur-Tonleiter unterschiedlich sind als in der Moll-Tonleiter. Nimmt man die hier aufgelisteten Intervalle und legt diese auf einen Grundton, so erhält man eine Dur- oder Moll-Tonleiter in der entsprechenden Tonart. (vgl. Sikora, 2012, S. 41ff)

2.1.3. Das Ionische System

In der Jazz-Harmonielehre werden neben den Dur- und Moll-Tonleitern noch weitere Tonleitern verwendet. Diese wurden von dem mittelalterlichen System der Kirchentonarten abgeleitet und bilden das Ionische System.

Im Ionischen System werden sieben Modi von einer Dur-Tonleiter abgeleitet. Ausgehend von der ersten Stufe der C-Dur-Tonleiter erhält man die Ionische Skala C-Ionisch, die der Dur-Tonleiter entspricht. Ausgehend von der sechsten Stufe erhält man A-Äolisch, was den gleichen Tönen wie A-Moll entspricht. Außerdem gibt es noch dorisch, phrygisch, lydisch, mixolydisch und lokrisch (vgl. Haunschild, 1998, S. 69).

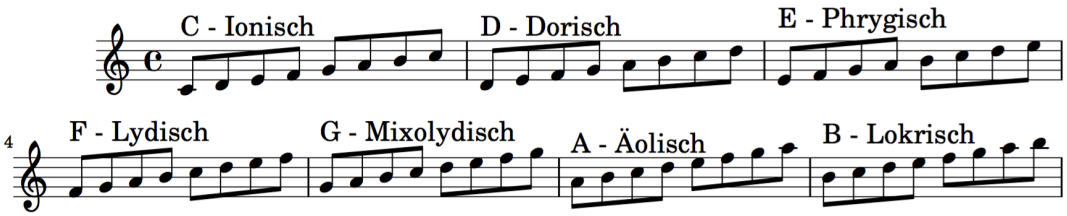

Abbildung 2.1.: Modi im Ionischen System

Abbildung 2.1 zeigt die unterschiedlichen Modi, die mit dem Tonmaterial von C-Ionisch von unterschiedlichen Stufen ausgehend möglich sind.

2.1.4. Akkorde

Als Akkord bezeichnet man mehrere gemeinsam klingende Töne. Die einfachste Art eines Akkords ist ein Dreiklang. Dabei wird von einem Grundton ausgehend eine Note im Terzabstand hinzugefügt. Fügt man von dieser Note ausgehend eine weitere Terz hinzu, ergibt es einen Dreiklang. Dabei ist in den Intervallen das Muster Prim - Terz - Quinte erkennbar. Dadurch, dass es zwei unterschiedliche Arten von Terzen gibt, die große und die kleine Terz, gibt es somit vier verschiedene Arten von Dreiklängen: Dur, Moll, Übermäßig und Vermindert.

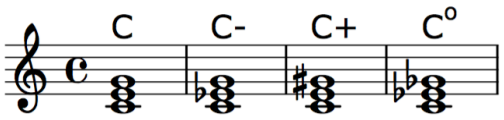

Abbildung 2.2.: Arten von Dreiklängen

Den Akkorden werden je nach Literatur und Musikrichtung unterschiedliche Symbole zugeordnet. Im Jazz wird vorwiegend folgende Notation verwendet:

Akkordtyp	Symbol	Intervallstruktur
Dur (major)	C	1 – 3 – 5
Moll (minor)	C-	1 – ♭3 – 5
Vermindert (diminished)	C°	1 - ♭3 – ♭5
Übermäßig (augmented)	C+	1 – 3 – ♯5

Tabelle 2.2.: Akkordgrundtypen

Fügt man diesen Dreiklängen jeweils eine weitere Terz hinzu, so erhält man eine Reihe von Vierklängen. Eine Ausnahme ist der Quartvorhalt (engl. suspended fourth), bei dem die große Terz durch die reine Quarte ersetzt wird.

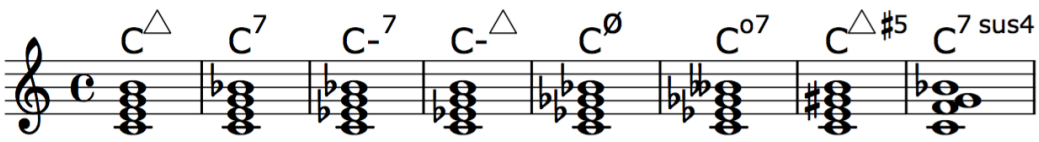

Abbildung 2.3.: Vierklänge

Akkordtyp	Symbol	Intervallstruktur
großer Septakkord (major 7)	C△7	1 – 3 – 5 – 7
Dominantseptakkord (dominant)	C7	1 – 3 – 5 – ♭7
Mollseptakkord (minor 7)	C-7	1 – ♭3 – 5 – ♭7
Moll-Major-7 (minor major 7)	C-△7	1 – ♭3 – 5 – 7
Halbverminderter Septakkord (halfdiminished)	Cø7 / C-7(♭5)	1 – ♭3 – ♭5 – ♭7
Verminderter Septakkord (diminished 7)	C°7	1 – ♭3 – ♭5 – ♭♭7
Quartvorhalt (suspended 4)	C7(sus4)	1 – 4 – 5 – ♭7

Tabelle 2.3.: Akkordgrundtypen

Darüber hinaus können mithilfe weiterer Töne viele weitere Akkorde erzeugt werden. Dabei hat jedes Intervall zum Grundtons des Akkords eine entsprechende Bezeichnung bzw. ein Symbol.

Intervallname	Symbol
kleine Sekunde	♭9
große Sekunde	9, add9
übermäßige Sekunde	♯9
kleine Terz	- oder °
große Terz	Dur oder +
reine Quarte	sus4, 11
übermäßige Quarte	♯11
verminderte Quinte	♭5, bei ° enthalten
reine Quinte	bei Dur und Moll enthalten
übermäßige Quinte	♯5, bei + enthalten
kleine Sexte	♭6, ♭13
große Sexte	6, 13
verminderte Septime	°7
kleine Septime	7
große Septime	maj7, △7

Tabelle 2.4.: Intervalle und deren Bezeichnung im Akkordsymbol

2.1.5. Skalentheorie und Stufendreiklänge

Bildet man auf den Stufen einer Dur-Tonleiter Dreiklänge, die nur aus den Tönen der jeweiligen Tonleiter bestehen, so erhält man eine Reihe von leitereigenen Akkorden. Ihnen werden vom Grundton ausgehend die römischen Zahlen I bis VII zugeordnet. Sie werden auch als Stufendreiklänge oder allgemeiner als Stufenakkorde bezeichnet.

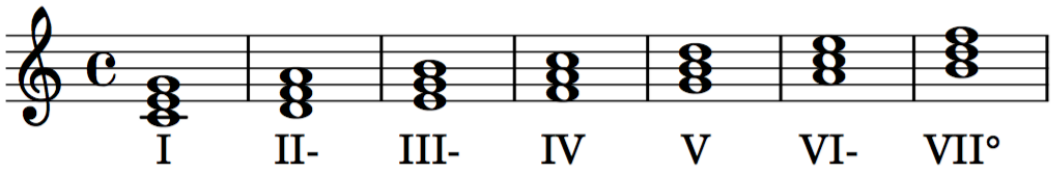

Abbildung 2.4.: Stufendreiklänge der C-Dur-Tonleiter

Die Tabellen 2.2 und 2.3 basieren auf (Sikora, 2012, S.30). Die Tabelle 2.4 basiert auf (Sikora, 2012, S. 32) und (Haunschild, 1998, S. 56).

2.2. Musiknotation

In der Musik gibt es verschiedene Systeme, um Stücke zu notieren. Bei der Notation sind je nach Musikrichtung unterschiedliche Informationen wichtig. Im folgenden werden einige übliche Notationssysteme vorgestellt.

2.2.1. Moderne westliche Notenschrift

Die bekannteste Notationsform ist die moderne Notenschrift. In ihr können Musikstücke sehr präzise aufgezeichnet werden. Ihr Kernelement ist das Notensystem, bestehend aus fünf Linien, einem Notenschlüssel, sowie weiteren Informationen wie Tempo, Tonart, Taktart und Dynamik.

Die einzelnen Stammnoten werden abgebildet auf die Linien und dazwischenliegenden Zwischenräume. Je höher eine Note in diesem System liegt, desto höher ist ihr Ton. Die konkrete Tonhöhe wird durch den Notenschlüssel festgelegt. Die Tonlänge wird durch verschiedene Notensymbole bestimmt. Eine ganze Note wird mittels eines leeren Kreises dargestellt und sie hat den Notenwert von vier Viertelnoten.

Eine halbe Note erhält zusätzlich einen Strich. Ihr Notenwert entspricht zwei Viertelnoten. Bei einer Viertelnote wird der Kreis zusätzlich gefüllt.

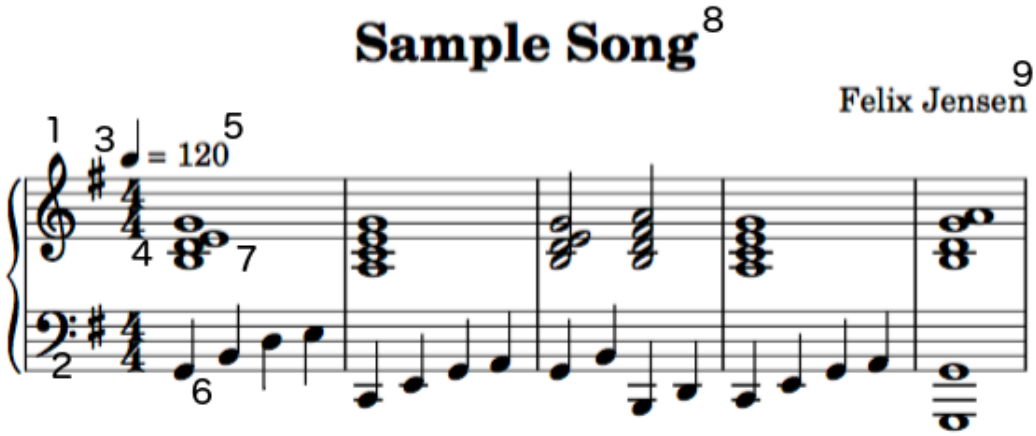

Abbildung 2.5.: Moderne westliche Notation

In Abbildung 2.5 sieht man zwei Notensysteme, die mit einer Klammer verbunden sind. So werden üblicherweise Klaviernoten notiert. Das obere der Systeme enthält die Noten für die rechte Hand, das untere die für die linke. Die Tonhöhe des Notensystems wird durch einen Notenschlüssel festgelegt. Im oberen System wird der Violinschlüssel (1) verwendet, der besagt, dass die zweite Linie von unten auf den Ton g' festgelegt wird. Im unteren System wird der Bassschlüssel (2) genutzt. Hier ist die vierte Linie von unten mit dem Ton f definiert.

Hinter dem Notenschlüssel folgt die Angabe der Vorzeichen. Auf der F-Linie befindet sich ein ♯ (3). Es sorgt dafür, dass alle f-Noten automatisch zu f♯ werden. Diese Angabe deutet zudem auf daraufhin, dass das Stück in der Tonart G-Dur ist.

Nach der Angabe der Vorzeichen folgt die Taktart, in diesem Fall ein Viervierteltakt (4). Jeder Takt enthält vier Viertelnoten, wobei der Akzent auf dem ersten liegt, und ein weiterer etwas schwächerer auf der dritten.

Das Tempo wird oberhalb des Notensystems angegeben (5). Neben einer meist italienischen Tempobezeichnung gibt es in modernen Stücken oft eine genauere Angabe in Beats per Minute (BPM). In diesem Fall hat das Lied 120 Viertelnoten pro Minute.

Im unteren Notensystem ist eine Folge von Viertelnoten notiert (6). Es handelt sich um einzelne Töne, die nacheinander gespielt werden. im oberen Notensystem sind ausschließlich Akkorde notiert. Der erste Akkord (7) ist ein G6 in der 1. Umkehrung, bestehend aus den Noten b, d, e und g.

Über dem Notensystem ist außerdem noch der Titel (8) und der Komponist (9) des Stücks angegeben.

Neben den oben angegebenen Informationen können noch viele weitere Parameter angegeben werden, wie beispielsweise Dynamik, Taktart- und Tempoänderungen, Haltebögen und andere Hinweise, die der Komponist für wichtig hält. So ist es möglich, ein Lied genauso zu notieren, wie es gespielt werden soll.

2.2.2. Akkordsymbolschrift

Während die Notenschrift für die Notation klassischer Musik sehr gut geeignet ist, werden in vielen moderneren Musikrichtungen, wie dem Blues oder Jazz, Improvisationen wichtiger. Des weiteren gibt es hier häufig begleitende Spuren, die primär aus dem Notenmaterial eines Akkords zur Zeit bestehen. Hier wäre die konventionelle Notenschrift zu einschränkend und zu aufwändig zu lesen und schreiben. Stattdessen verwendet man eine Akkordsymbolschrift. Hierin werden den einzelnen Akkordtypen Symbole zugeordnet: Ein Akkordsymbol besteht aus dem Grundton des Akkords (groß geschrieben), gefolgt von einem oder mehreren Symbolen.

Akkordsymbole werden über das Notensystem an der Position geschrieben, an der der Akkord gespielt werden soll. Dabei reicht auch bei einer Folge von mehreren Takten desselben Akkords das einmalige Notieren des Akkordsymbols aus. (vgl. Sikora, 2012, S. 29ff)

Je nach Literatur werden für Akkordtypen unterschiedliche Symbolalternativen verwendet.

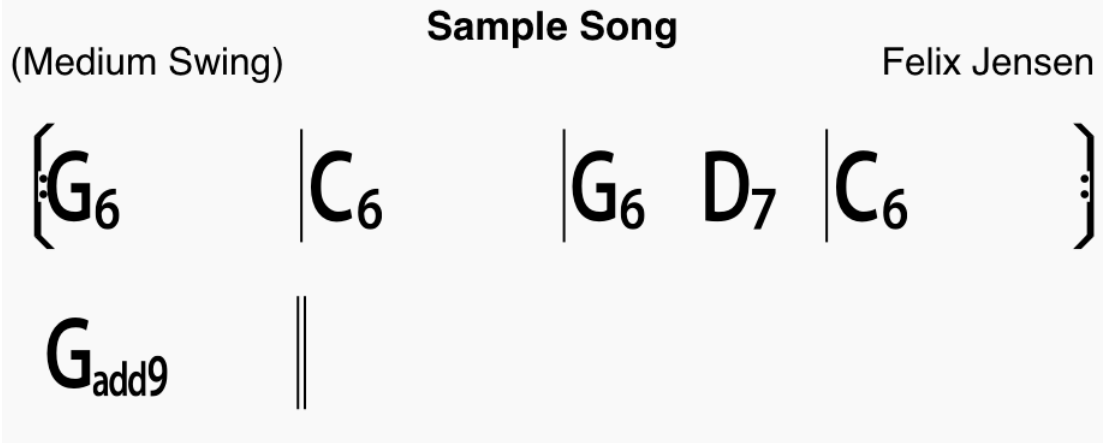

Abbildung 2.6.: Beispiel eines Notation in Akkordsymbolschrift

In Abbildung 2.6 ist beispielhaft die Notation eines Stücks nur mit Akkordsymbolen gezeigt.

2.2.3. Leadsheet

Das Leadsheet verbindet die Akkordsymbolschrift mit der konventionellen Notation. Dabei wird nur die Melodie eines Stückes in Notenform notiert. Die verwendete Akkordfolge wird

zusätzlich über die Takte geschrieben. Da Leadsheets auch als Improvisationsgrundlage dienen, sind die Informationen bewusst vage gehalten. Dies bedeutet, dass sowohl Melodie als auch Akkordfolge vereinfacht dargestellt sein kann (vgl. Sikora, 2012, S. 64).

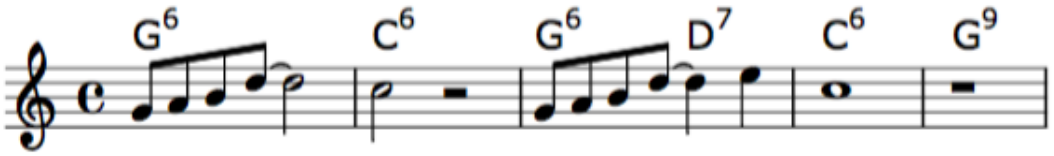

Abbildung 2.7.: Leadsheet-Notation

Im Abbildung 2.7 ist das Beispielstück in Leadsheet-Notation angegeben. Die Begleitung wird mithilfe der Akkordsymbolnotation über den Takten angegeben. Im Notensystem ist eine Melodie in einer simplen Form angegeben. Die konkrete Spielart der Begleitung ist frei wählbar, solange die angegebenen Akkorde gespielt werden.

2.2.4. Notation mit Stufenakkorden

Nutzt man nur leitereigene Akkorde, ist es möglich, diese mittels der jeweiligen Stufe zu notieren. In dieser Notationsart werden die Stufenakkorde der Stufentheorie mit der Akkordsymbolschrift verbunden. Beispielsweise würde in der Tonart C-Dur der Akkord C△ als I△ oder 1△ notiert werden.

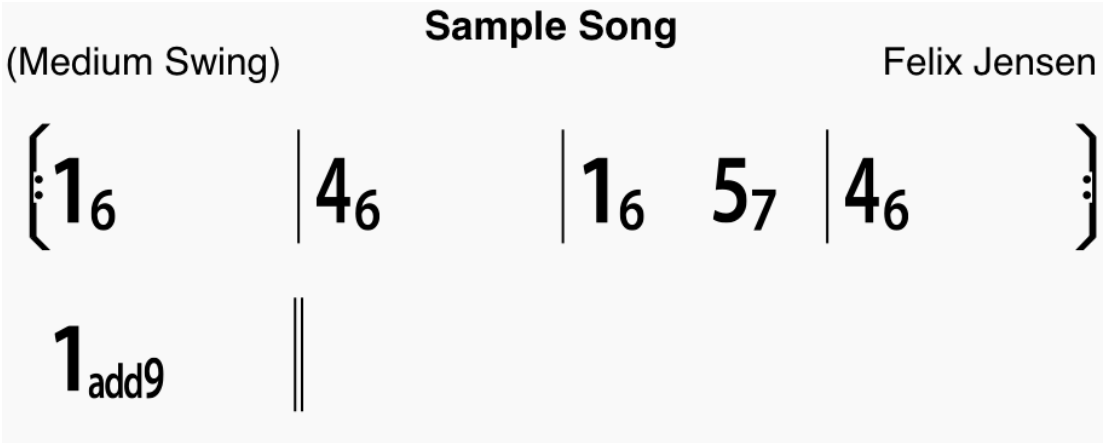

Abbildung 2.8.: Notation mit Stufenzahl der Akkorde

Ein Vorteil dieser Notation ist, dass man Akkordfolgen unabhängig von einer konkreten Tonart notieren kann. Dies ermöglicht die Analyse auf einer abstrakteren Ebene.

2.3. Domain-Specific Languages

In diesem Abschnitt sollen die wichtigsten für diese Arbeit relevanten Grundlagen zum Thema Domain-Specific Languages beschrieben werden. Der Abschnitt basiert auf dem Werk *Domain-Specific Languages* von Martin Fowler Fowler (2010).

Eine domain-specific language (DSL) oder domänenspezifische Sprache ist eine Programmiersprache, die auf ein bestimmtes Anwendungsfeld ausgerichtet ist. Martin Fowler beschreibt in dem Buch *Domain-Specific Languages* vier Eigenschaften von DSLs:

- Computer programming language: Eine DSL ist eine für Menschen verständliche Sprache, die dazu genutzt wird, einen Computer dazu anzuleiten, etwas zu tun.
- Language nature: Die Ausdruckskraft einer Sprache kommt nicht nur von einzelnen Ausdrücken, sondern auch aus dem Zusammenhang mehrerer Ausdrücke.
- Limited expressiveness: Im Gegensatz zu einer general-purpose programming language (GPL, engl. universell einsetzbaren Programmiersprache) unterstützt eine DSL nur diejenigen Funktionalitäten, die in dem jeweiligen Einsatzfeld benötigt werden.
- Domain focus: Der Fokus einer DSL liegt auf einer kleinen Domäne. Sie sollte nicht mehr als diese Domäne abbilden.

DSLs werden als Werkzeug eingesetzt, um eine Problemdomäne verständlich beschreiben zu können. Da sie nur die Probleme einer Domäne enthalten, können Experten dieser Domäne sie ohne besonderes Zusatzwissen bedienen. Es wird zwischen internen und externen DSL unterschieden.

2.3.1. Interne DSL

Eine interne DSL ist eine Sprache, die innerhalb eine vorhandene universell einsetzbare Programmiersprache eingebettet ist. Sie nutzen eine Untermenge der Funktionalität der Wirtssprache. Somit ist ein gültiges Programm einer internen DSL auch ein gültiges Programm ihrer Wirtssprache. Die interne DSL sieht ihrer Wirtssprache oft nur noch bedingt ähnlich, da die für die Wirtssprache charakteristischen Kontrollstrukturen häufig nicht mehr sichtbar sind.

Ein Vorteil in der Nutzung einer internen DSL ist, dass der Implementationsaufwand im Vergleich zu einer externen DSL geringer ist, da kein Parser entwickelt werden muss. Innerhalb eines Softwaresystems kann die Verwendung einer DSL in bestimmten Einsatzgebieten den

Code besser lesbar machen. Ein Beispiel für eine interne DSL ist Rake[1], ein Build-Management-Werkzeug, dass als Wirtssprache Ruby verwendet.

```
1  (song
2    (title "Sample Song")
3    (composer "Felix Jensen")
4    (tempo 120)
5    (progression
6      (bar (chord :I :6))
7      (bar (chord :IV :6))
8      (bar
9        (chord :I :6 1/2)
10       (chord :V :7 1/2))
11     (bar (chord :IV :6))
12     (bar (chord :I :add9))))
```

Abbildung 2.9.: Ein Beispiel einer DSL zur Musiknotation in einem Lisp-Dialekt

Abbildung 2.9 stellt eine mögliche interne DSL in einem Lisp-Dialekt dar. Die für Lisp charakteristische Klammersetzung muss weiterhin durchgeführt werden, aber die einzelnen Elemente der Sprache sind auch für Benutzer lesbar, die keine Erfahrung mit Lisp haben.

2.3.2. Externe DSL

Eine externe DSL ist eine DSL, die von anderen Sprachen unabhängig ist. Die Syntax kann somit von den Entwicklern frei gewählt werden. Ein in einer DSL entwickeltes Programm wird üblicherweise von der Parserkomponente eines Softwaresystems eingelesen und dort dann weiterverarbeitet. Da im Vergleich zur internen DSL die Wirtssprache fehlt, müssen alle für die Sprache gewünschten Funktionalitäten selbst implementiert werden. Dafür ist die Syntax dann befreit von den Einschränkungen der Wirtssprache. Im Beispiel der Lisp-DSL in Abbildung 2.9

[1] https://github.com/ruby/rake

könnten so zum Beispiel die Klammern wegfallen. Auch für die Akkordsymbole könnte eine geeignetere Notation gewählt werden.

2.4. Grammatik

Eine formale Grammatik ist eine Form zur Beschreibung der Struktur einer Programmiersprache. Sie beschreibt lediglich die syntaktischen Regeln der Sprache. Eine kontextfreie Grammatik besteht aus einer Menge von Ersetzungsregeln, und einem Startsymbol und jeweils einer Menge von Terminalsymbolen und Nichtterminalsymbolen. Eine Ersetzungsregel besteht aus einer linken Seite und einer rechten Seite. Die linke Seite enthält genau ein Nichtterminalsymbol, die rechte Seite beschreibt mögliche Ersetzungen. Dies können weitere Nichtterminale oder Terminale sein.

Zur Darstellung einer kontextfreien Grammatik wird zumeist die erweiterte Backus-Naur-Form (EBNF) verwendet. Sie erlaubt die Notation der Ersetzungsregeln. (vgl. Grune u. a., 2000, S. 34-40)

```
1  Akkordfolge ::= Akkord Akkord*
2  Akkord ::= Grundton Dreiklang?
3  Grundton ::= Ton Versetzungszeichen?
4
5  Ton ::= "C" | "D" | "E" | "F" | "G" | "A" | "B"
6  Versetzungszeichen ::= "#" | "b"
7  Dreiklang ::= "–" | "+" | "o"
```

Abbildung 2.10.: Darstellung einer Grammatik zur Notation von Akkordfolgen in der EBNF

Abbildung 2.10 zeigt eine simple Grammatik zur Notation von Akkordfolgen, notiert in der EBNF. In Zeile 2 wird eine Ersetzungsregel beschrieben, die den Aufbau eines Akkords definiert. ein Akkord besteht aus einem Grundton, gefolgt von einem optionalen Dreiklangsymbol. Der Grundton wiederum besteht aus einem Ton und einem Versetzungszeichen. Ein Fragezeichen bedeutet, dass das vorstehende Symbol optional ist. Zeile 6 erläutert, dass ein Versetzungszeichen entweder eine Raute oder ein b sein kann. Diese Alternative wird durch den senkrechten Strich markiert.

Zeile 1 sagt aus, dass eine Akkordfolge aus einem Akkord gefolgt von beliebig vielen weiteren Akkorden bestehen kann. Wenn ein oder beliebig viel mehr von einem Symbol kommen soll,

kann auch das Plus verwendet werden. Symbole in Anführungszeichen sind Terminalsymbole, die anderen sind Nichtterminalsymbole.

Eine für die dargestellte Grammatik gültige Eingabe könnte folgendermaßen aussehen: *C+ D#-*.

2.5. Von der Eingabe zur Datenstruktur

Im folgenden wird der Ablauf von der Eingabe eines Programms bis zur fertig eingelesenen Datenstruktur beschrieben. Zur Veranschaulichung werden die einzelnen Schritte anhand des Beispiels *C+ D#-* mit Grafiken versehen.

2.5.1. Syntax-Directed Translation

Eine EBNF kann dazu benutzt werden, mithilfe eines Parsergenerators einen Parser zu erzeugen. Dies wird syntax-directed translation (engl., Syntaxgesteuerte Übersetzung) genannt.

Vor dem Parse-Vorgang steht die Lexikalische Analyse, die von einem Lexer durchgeführt wird. Er liest einen eingehenden Text ein und erkennt Grundelemente der Sprache, wie Operatoren, Schlüsselwörter und Bezeichner. Anschließend reicht er sie weiter an den Parser. Dieser wandelt anhand seiner Grammatik die Folge von Schlüsselwörtern in eine neue Struktur um, sofern die Syntax korrekt ist. Diese Struktur wird dann verwendet, um eine zur Sprache gehörende semantische Analyse durchzuführen und entsprechende Aufgaben durchzuführen.

2.5.2. Parsebaum

Das Ergebnis eines Parse-Vorgangs ist ein Parsebaum (engl. Parsetree). Er stellt die Eingabe als Baum dar, wobei jedes Element dem entsprechenden Grammatiksymbolen zugeordnet ist. Ein Parsebaum ist zur weiteren Verarbeitung der Eingabe nützlich, ist in der Regel aber für die semantische Analyse noch nicht optimal geeignet.

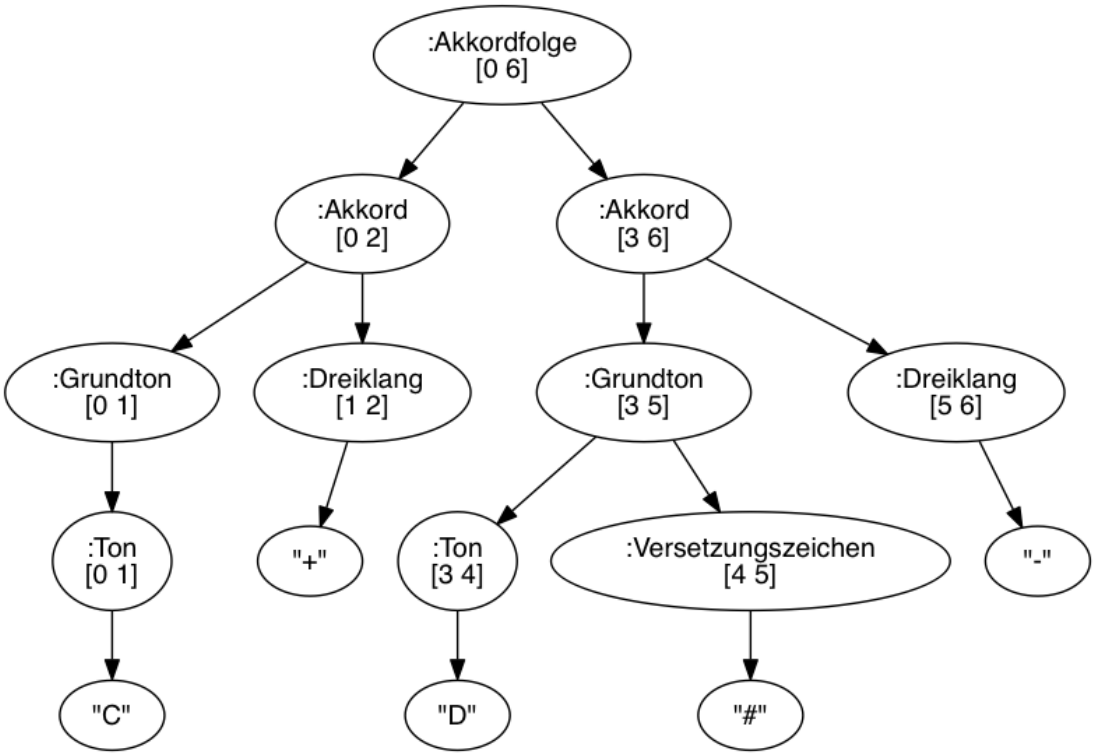

Abbildung 2.11.: Parsebaum für die Grammatik von Akkordfolgen

Ein Beispiel für einen Parsebaum ist in Abbildung 2.11 zu sehen. Als Blätter sind einzelne Teile der Eingabe zu erkennen. Ihre Elternknöten entsprechen den in der Grammatik vorgegebenen Übergangsregeln.

2.5.3. Abstract Syntax Tree

Ein Abstrakter Syntaxbaum (AST, engl. Abstract Syntax Tree) ist eine weitere Darstellung der Eingabe als Baum. Er wird aus einem Parsebaum erzeugt, in dieser umgeformt wird. Dabei werden für die weitere Verarbeitung unnötige Knoten entfernt. Auch können Strukturen verändert und so vereinheitlicht werden. Es ist nicht unüblich, dass vom Weg vom Parsebaum zur entgültigen Repräsentation mehrere Zwischenformate verwendet werden.

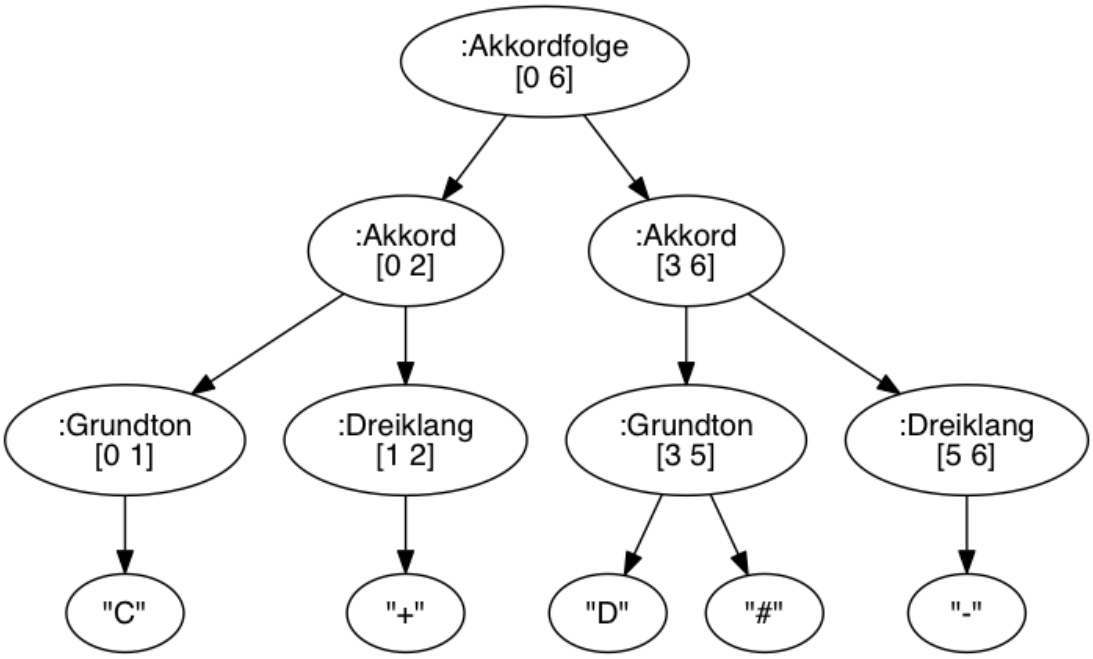

Abbildung 2.12.: Ein möglicher AST für die Grammatik von Akkordfolgen

In der Abbildung 2.12 sieht man, wie im Vergleich zum Parsebaum einige Knoten entfernt wurden. Die Entscheidung, welche Knoten behalten, welche verworfen und welche verändert werden sollen, ist je nach Sprache und Nutzen unterschiedlich und muss individuell getroffen werden.

2.5.4. Semantic Model

Ein Semantic Model (engl., semantisches Modell) ist eine Darstellung der Domäne als Datenstruktur. Es wird dazu verwendet, den Sachverhalt in einer GPL darzustellen. Es wird nach dem Parsen aus dem AST befüllt. Anschließend kann eine semantische Prüfung durchgeführt werden, um sicherzustellen, dass die Eingabe auch semantisch korrekt ist.

Der Aufbau des Semantic Models ist abhängig von der zukünftigen Verwendung. Es kann schon vor der Grammatik erstellt werden, sollte spätestens aber parallel zur Grammatik entwickelt werden.

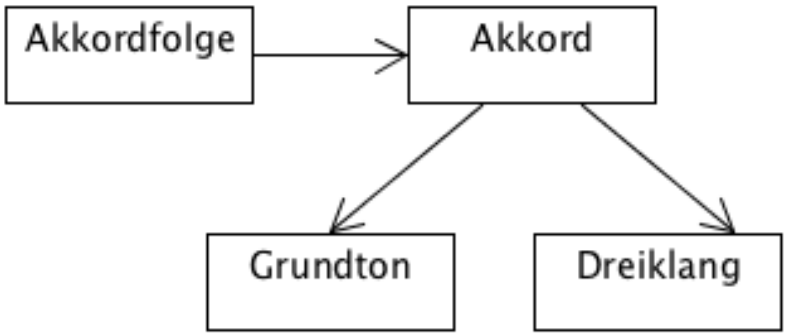

Abbildung 2.13.: Semantic Model zur Darstellung von Akkordfolgen

Das Semantic Model für die Beispielgrammatik ist in Abbildung 2.13 zu sehen. Es zeigt den Aufbau der Thematik, wie es in objektorientierter Programmierung implementiert werden könnte.

2.5.5. Compiler und Interpreter

Ein Compiler ist ein Programm zur Übersetzung der Eingabe in einer Sprache in eine ausführbare Form. Das Resultat ist also ein ausführbares Programm in einer anderen Sprache. Ein Interpreter liest ebenfalls die Eingabe ein, führt sie aber sofort aus. Bei beiden muss neben der syntaktischen Analyse auch eine semantische Analyse erfolgen, um die Gültigkeit der Eingabe sicherzustellen.

Auch wenn die Benutzung der beiden Programme unterschiedlich ist, so kann häufig trotzdem dieselbe Komponente zum Einlesen genutzt werden. Ein Vorteil eines Compilers ist, dass durch die Umwandlung in eine andere Sprache eine große Anzahl von Optimierungen möglich ist. Der Interpreter dagegen läuft innerhalb einer Laufzeitumgebung und ist daher langsamer. Dafür kann er sich die von der Laufzeitumgebung angebotenen Funktionalitäten zunutze machen.

3. Analyse

Dieses Kapitel befasst sich mit der Analyse der eingangs genannten Problemstellung. Dabei sollen zunächst einige Anforderungen vorgestellt werden, die im Vordergrund zur Bewertung und Entwicklung einer Lösung stehen. Anschließend werden vorhandene Lösungen auf Eignung für die Problemstellung untersucht.

3.0.1. Anforderungen

Um den Rahmen des zu entwickelnden Systems festzulegen, wurden einige Anforderungen erarbeitet, die hier vorgestellt werden sollen. Sie dienen als Grundlage zur Bewertung von vorhandenen Lösungen und zur Erstellung eines Entwurfs.

Sikora schreibt in seinem Buch *Neue Jazz-Harmonielehre*, dass eine Notenschrift die Musik reflektieren müsse, die sie darzustellen versucht (Sikora, 2012, S. 29). Im Jazz sind die Anforderungen an eine Notationssprache anders als in der klassischen Musik, da hier Improvisation und Komposition in einem besonderen Verhältnis zueinander stehen. Die klassische Notation ist sehr präzise, während es im Jazz improvisatorischen Freiraum geben muss.

Bei der Entwicklung einer DSL soll einen Sachverhalt in eine für den Computer verständliche Form zu bringen. Auch hier ist eine hohe Präzision notwendig, damit die Eingabe vom Computer so interpretiert wird, wie sie gemeint ist. Die in dieser Arbeit zu entwickelnde Lösung soll die für den Jazz so wichtige Kombination von Improvisation und Komposition ermöglichen, ohne dabei in eine sehr förmliche und techniknahe Syntax zu rutschen. Stattdessen soll die Notation auf den musiktheoretischen Grundlagen des Jazz aufbauen.

Die Notationsform eines Musikstücks muss eine Reihe verschiedener Informationen beinhalten, damit es vom Leser verstanden und interpretiert werden kann. Dazu gehören zum Beispiel Tempo, Tonart, Titel und natürlich die Akkordfolge. Je nach Umfang des Stücks können aber auch Strukturinformationen von Bedeutung sein, aber auch Dynamik- oder Stilangaben. Im Gegensatz zu der Fülle an möglichen und benötigten Informationen steht das Bedürfnis, ein Stück möglichst schnell und knapp notieren zu können. Dies ist schon deshalb von Interesse, da der kreative Vorgang eines Musikers nicht planbar ist, und spontane Ideen schnell notiert

werden müssen, damit sie nicht vergessen werden. Dabei soll die Syntax trotzdem lesbar und verständlich sein.

Während die Notation eines Musikstücks auf dem Papier nur die Interpretation des geschriebenen durch einen Musiker zulässt, bietet die Eingabe in einen Computer mehr Möglichkeiten. Die Notation eines Musikstücks ist schließlich auch in der Regel nicht Notation der Notation wegen, sondern hat den Zweck der Speicherung oder Weitergabe des Liedes, oder auch als Grundlage zur Improvisation und Weiterentwicklung. Die Fähigkeiten eines Computers können hier auf verschiedene Arten zur Unterstützung bei der Erarbeitung und Interpretation eines Stückes angewendet werden. Eine besonders interessante Art der Unterstützung sieht der Autor dabei im Abspielen des Stückes. Sie hilft bei der Notation, indem durch Anhören der geschriebenen Musik geprüft werden kann, ob das Ergebnis erreicht wurde, was dem Musiker vorschwebt. Darüber hinaus kann dann die notierte Musik zur Begleitung für die Improvisation dienen.

Daraus lassen sich einige Kernanforderungen ableiten, die im folgenden zur Besprechung vorhandener Lösung und zum Entwurf einer eigenen Lösung genutzt werden:

- Improvisation und Komposition im Einklang
- eine Notenschrift mit musiktheoretischer Basis
- ein ausgewogenes Verhältnis von Informationsdichte und Lesbarkeit
- Unterstützung bei Notation und Improvisation durch den Computer

3.1. Vorhandene Lösungen

Im Folgenden sollen einige vorhandenen Lösungen für die oben genannte Problemstellung auf die in Abschnitt 3.0.1 genannten Anforderungen hin untersucht werden. Dabei wird die Lösung jeweils kurz beschrieben. Anschließend wird ein Beispiel der Notation gegeben und schließlich wird überprüft, ob die Lösung den Anforderungen entspricht.

3.1.1. Notationsbeispiel

Zur Besprechung der Notation wird in jeder der in diesem Kapitel vorgestellten Lösungen ein Notationsbeispiel gegeben. Um eine gewisse Vergleichbarkeit zu gewährleisten, wird in jedem Beispiel der in Abbildung 3.1 gezeigte Notenausschnitt nachgestellt. Das Beispiel ist nicht unbedingt repräsentativ für die Notation ganzer Stücke, die in der Notenschrift über mehrere

Seiten gehen können. Die Angabe einer so umfangreichen Notation würde den Rahmen dieser Arbeit sprengen.

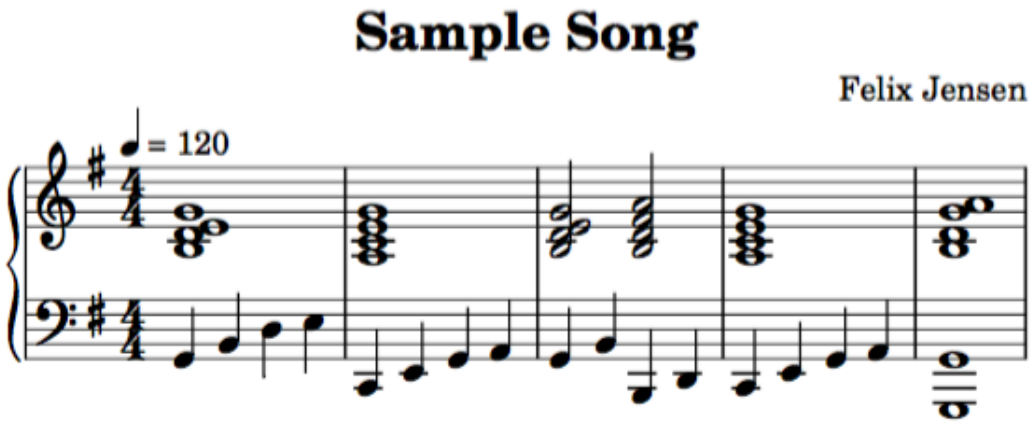

Abbildung 3.1.: Notationsbeispiel in moderner westlicher Notation

3.1.2. Lilypond

LilyPond[1] ist eine Notensatz-Software, deren Ziel es ist, eine möglichst hohe Qualität im Bereich des Notendrucks zu erreichen. Die Software nutzt dabei zur Eingabe von Musik eine textbasierte Sprache.

[1] http://lilypond.org

```
1  rhMusic = \relative c' {
2    <b d e g>1
3    <a c e g>
4    <b d e g>2 <b d fis a> }
5  lhMusic = \relative c {
6    g4 b d e
7    c, e g a
8    g b b, d }
9  \header {
10   title = "Sample Song"
11   composer = "Felix Jensen" }
12 \score {
13 \new PianoStaff <<
14   \new Staff = "RightHand" <<
15     \tempo 4 = 120
16     \key g \major
17     \rhMusic
18   >>
19   \new Staff = "LeftHand" <<
20     \key g \major
21     \clef bass
22     \lhMusic
23   >>
24 >> }
```

Abbildung 3.2.: Beispiel einer modernen Notation in LilyPond mit zwei Notensystemen

Abbildung 3.2 zeigt die Notation der ersten drei Takte des Musikstücks. Die Ausgabe dieser Notationsart ist eine moderne westliche Notation mit je einer Spur für die linke und die rechte Hand zum Spiel auf einem Klavier. Zunächst wird ein Part für die rechte Hand definiert. Mithilfe des Befehls
relative c’ wird festgelegt, dass der Inhalt des folgenden Blockes in geschweiften Klammern in der Tonhöhe bei c’ liegt. Innerhalb des Blockes werden Noten mit deren Namen notiert. Hinter dem Notennamen wird mit einer Zahl die Notenlänge bestimmt. Eine *1* steht dabei für

eine ganze Note, eine *2* für eine halbe, und so weiter. Wenn an einer Note keine Notenlänge angegeben ist, so übernimmt sie den Wert der vorherigen Note.

Im Fall der rechten Hand (Zeilen 2-4) sind ausschließlich Akkorde notiert. Sie werden Diese werden mithilfe der Zeichen < und > gruppiert. Die Zeichenfolge *<b d e g>1* bedeutet somit, dass hier ein G6-Akkord in der dritten Ableitung notiert ist, der die Dauer einer ganzen Note hat.

In der linken Hand (Zeilen 6-8) sind ausschließlich einzelne Viertelnoten zu sehen. Während die Noten zu Beginn im Bereich der Note c liegen, wird in Zeile 7 mit der Eingabe *c,* die Tonhöhe um eine Oktave abgesenkt. Das g zu Beginn der Zeile 8 ist trotzdem in derselben Oktave wie das zu Beginn von Zeile 6. Das liegt daran, dass bei der Wahl der Tonhöhe von der vorherigen Note ausgegangen und die nähere Oktave verwendet wird.

Im Beispiel sieht man auch den Einsatz von Variablen. Die Spuren der linken und rechten Hand werden jeweils der Variablen lhMusic bzw. rhMusic zugeordnet. Diese Variablen werden später innerhalb des score-Blocks mittels *\lhMusic* und *\rhMusic* verwendet.

Innerhalb des score-Blocks wird ein PianoStaff mit zwei Staffs definiert. Innerhalb dieser Staffs werden Tempo und Tonart festgelegt und anschließend mithilfe der Variablen die jeweiligen Noten eingefügt.

```
1  <<
2  \chords {
3    g1:6
4    c:6
5    g2:6 d2:7
6  }
7  \relative c'' {
8    g8 a b d~ d2
9    c2 r
10   g8 a b d~ d4 e
11 }
12 >>
```

Abbildung 3.3.: Beispilnotation eines Leadsheets in Lilypond

LilyPond unterstützt neben der Erzeugung klassischer Notation auch die Ausgabe als Leadsheets. In Abbildung 3.3 ist die Notation eines Leadsheets zu sehen. Im chords-Block wird die Akkordnotation festgelegt. Hierfür gibt es neben der oben bereits genannten Form für Akkorde noch den sogenannten *chord mode*. In diesem ist es möglich, eine besondere Syntax für Akkorde zu verwenden. Wie im Beispiel zu sehen, wird hinter dem Grundton des Akkords mithilfe eines Doppelpunktes ein Akkordtyp angehängt, in diesem Fall die *6* für einen Sextakkord. Zusätzlich zu dem ersten Akkordtyp können mit einem Punkt separiert weitere Töne hinzugefügt und mit einem ^ entfernt werden.

Die beiden Notationsformen sind strukturell unterschiedlich. Dies wird dadurch ermöglicht, dass LilyPond fehlende Befehle zur Strukturierung automatisch hinzufügt.

Die Noteneingabe in LilyPond ist mit etwas Übung und mithilfe der umfangreichen Dokumentation[2] in angemessener Zeit zu erledigen. Neben den hier genannten Notationsformen, bietet die Sprache weit mehr, vor allem in Bezug auf Notensatz. Der kreative Prozess zur Schaffung der Musik ist bei LilyPond zeitlich vor der Notation einzuordnen. Die Notation wird mittels Symbolen für Akkorde und Töne durchgeführt, womit die Nähe zur Theorie gegeben ist. Es gibt keine Möglichkeit zur Improvisation oder zum Abspielen der Noten, aber sie können in andere Formate exportiert werden und so in anderen Programmen geöffnet werden. Eine direkte Unterstützung bei der Notation oder Komposition ist dabei aber nicht gegeben. Die Kernkompetenz von LilyPond liegt im Notensatz.

3.1.3. Sonic Pi

Sonic Pi[3] ist ein Live Coding Synthesizer, der als Eingabesprache eine interne DSL in der Programmiersprache Ruby verwendet. Das Ziel dieser Software ist, eine Plattform zum Lernen von Programmierung und Musik zu bieten und gleichzeitig mächtig genug zu sein, um als Instrument für professionelle Musiker zu dienen.

Bei Sonic Pi liegt die Erzeugung von elektronischer Musik im Vordergrund, die bevorzugt als improvisierter Auftritt erfolgt. Die Struktur eines Stücks wird in der Regel nicht explizit notiert, sondern ergibt sich während des Auftritts durch Veränderungen im Quellcode. Dabei könnte ein Stück damit beginnen, dass eine Bassdrum einen Takt definiert. Anschließend könnte eine harmonische Untermalung auf Basis einer E-Moll-Tonleiter erfolgen, deren Abfolge mithilfe eines Zufallsgenerators gesteuert wird.

[2] http://lilypond.org/doc/v2.18/Documentation/web/index.html
[3] http://sonic-pi.net

Ein weiteres Element könnte eine ebenfalls zufallsgenerierte Melodie sein. Abschließend werden nacheinander die Melodiespur, die Harmoniespur und die Bassdrum ausgeschaltet und das Lied beendet.

Um eine solche Folge zu erstellen, wird für jede der Spuren eine Endlosschleife in einem eigenen Thread erzeugt, in der Töne abgespielt und anschließend eine bestimmte Zeit geschlafen wird. Wenn eine Spur ausgeschaltet werden soll, muss die Endlosschleife im Code terminiert werden. Anschließend wird dieser Part des Quellcodes neu evaluiert und nach dem nächsten ausführen der Schleife wird sie verlassen.

```
1  use_bpm 120
2
3  q = 1.0
4  h = q*2
5  f = q*4
6
7  in_thread do # right hand
8    play_chord( chord( :G3, '6', invert: 1))
9    sleep f
10   play_chord( chord( :C3, '6', invert: 3))
11   sleep f
12   play_chord( chord( :G3, '6', invert: 1))
13   sleep h
14   play_chord( chord( :D3, '7', invert: 1))
15 end
16
17 in_thread do # left hand
18   play_pattern_timed [:G2, :B2, :D3, :E3], [q, q, q, q] # Bar 2
19   play_pattern_timed [:C2, :E2, :G2, :A2], [q, q, q, q] # Bar 3
20   play_pattern_timed [:G2, :B2, :B1, :D1], [q, q, q, q] # Bar 4
21 end
```

Abbildung 3.4.: Sequenzielle Notation in Sonic Pi

Eine Notation, die der westlichen Notation mehr ähnelt, sieht man in Abbildung 3.4. In den ersten Zeilen wird das Tempo auf 120 BPM gesetzt, anschließend werden einige Variablen für die einzelnen Tonlängen definiert. In Zeile 7 wird ein neuer Thread erzeugt, der den folgenden Block ausführt. Mithilfe der *sleep* und der *play_chord*-Funktionen ist es möglich, das Abspielen der Töne zeitlich zu platzieren. Zeilen 9 bis 11 beschreiben den ersten Takt.

Die *sleep*-Aufrufe entsprechen jeweils einer Pause. *play_pattern_timed* erwartet zwei Listen, wobei die erste die zu spielenden Töne enthält und die zweite Pausenwerte enthält, die jeweils zwischen den Tönen eingehalten werden. Sonic Pi hat für die gängigsten Töne Symbole definiert

(z. B. *:C4*), welche auch von den Funktionen verstanden werden. Diese Symbole stehen für Ganzzahlen, die den Midi[4]-Noten des Tons entsprechen. Da es keine vordefinierten Symbole für erhöhte oder erniedrigte Töne gibt, wird hier für den Ton D♯ der Notenwert von *:D4* um eins erhöht.

Zur Darstellung von Akkorden nutzt Sonic Pi Funktionen. Die Funktion *chord(tonic, name)* erwartet einen Grundton und Akkordtypen. Für die üblichsten Akkordtypen gibt es Symbole (z. b. *:major*), komplexere können mittels eines Strings notiert werden. Anders als in LilyPond sind diese Akkordtypen jedoch fest vorgegeben; einzelne Töne hinzufügen oder entfernen geht über diese Syntax nicht. Da Akkorde allerdings als Listen von Tönen zu verstehen sind, kann man diese nachträglich manuell bearbeiten oder sich eigene Akkorde definieren.

Ähnlich wie die *chord*-Funktion gibt es auch eine Funktion *scale(tonic, name)*, die anhand eines Grundtons und eines Skalennamens eine Tonleiter als Liste von Tönen zurückgibt. So ist es beispielsweise möglich, eine zufallsgenerierte Tonfolge innerhalb einer Tonleiter abzuspielen.

Da Sonic Pi eine interne DSL in Ruby verwendet, ist es möglich, alle üblichen Features der Programmiersprache zu verwenden. Somit ist es auch möglich, eigene Funktionen zu schreiben, und so die Funktionalität und Lesbarkeit nach eigenen Wünschen zu erweitern.

Sonic Pi ist für Musiker ohne Programmiererfahrung nur mit sorgfältiger Einarbeitung nutzbar. Die Software enthält ein eingebautes Hilfesystem, das ein umfangreiches Tutorial, eine Reihe von Beispielen und eine Sprachreferenz enthält. Neben einigen Funktionen für Akkorde und Skalen bietet Sonic Pi nur wenige Schnittpunkte zur klassischen Notation. Dafür ist es möglich, die Mächtigkeit einer Programmiersprache zu nutzen, womit eine sehr dynamische und kreative Komponier- und Notationsform möglich ist.

Eine schnelle Notation ist für den geübten Nutzer in einer individualisierten Umgebung möglich, erfordert allerdings Erfahrung in der Programmierung. Das technische Umfeld der Sprache birgt die Gefahr, dass sich der Nutzer in der komplexen Syntax verliert, statt sich mit den musikalischen Aspekten zu befassen. Durch die Möglichkeit, Tonleitern, Akkorde und Tonfolgen abspielen zu können, kann man diese als Grundlage für Improvisation, wie auch als Unterstützung bei der Notation nutzen.

3.1.4. iReal Pro

iReal Pro[5] ist ein Werkzeug zur Begleitung eines Musikers beim Üben und zum Improvisieren. Die Software besteht im wesentlichen aus einer Liste von Liedern und einer Abspielkomponente,

[4] Musical Instrument Digital Interface. Ein Protokoll zum Austausch von musikalischen Steuerinformationen.
[5] http://irealpro.com

mit der man das Lied in verschiedenen Stilen, Tempi und Tonarten spielen kann. Es ist auch möglich, die Instrumentierung zu beeinflussen.

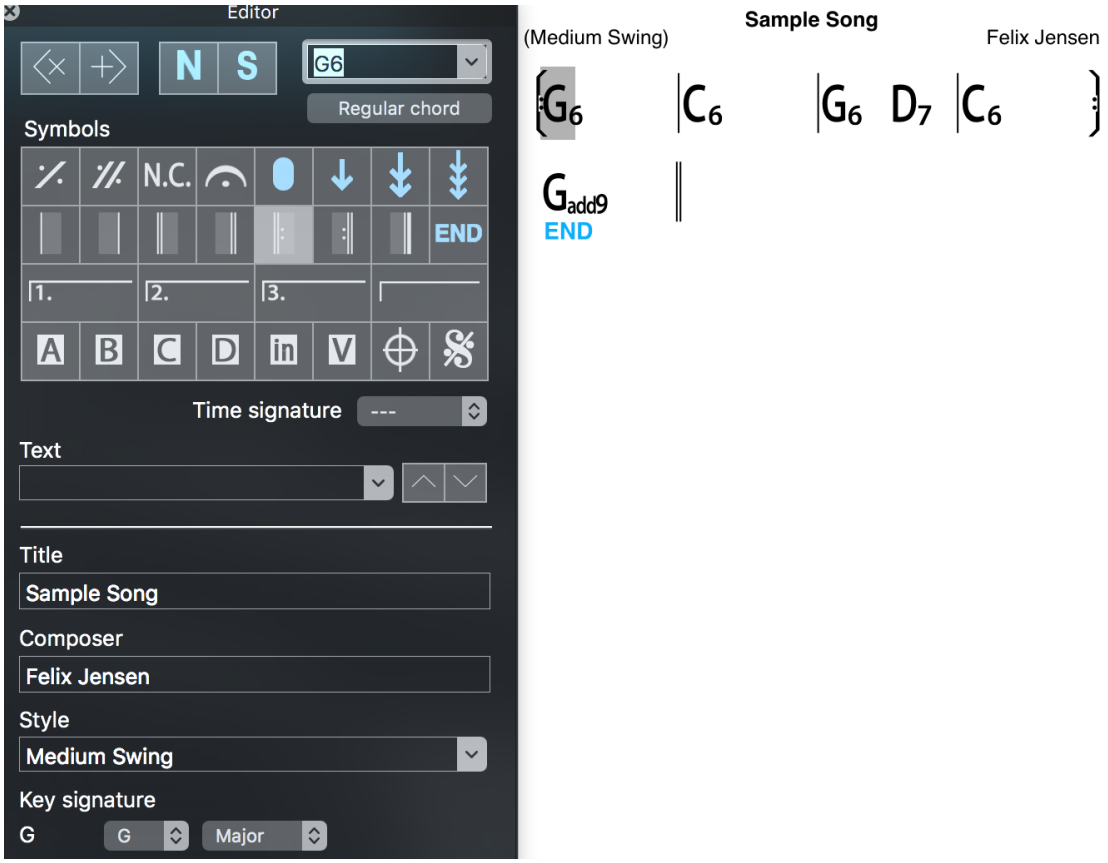

Abbildung 3.5.: Die Notationsoberfläche von iReal Pro

Neben den voreingestellten Liedern gibt es auch die Möglichkeit, eigene Lieder zu notieren. Die Notation erfolgt dabei in einer graphischen Notationssprache. Abbildung 3.5 zeigt die Benutzeroberfläche, die zur Notation verwendet wird. Auf der rechten Seite ist das zu bearbeitende Dokument. Hier können in einer Art Raster Symbole platziert werden. Die Symbole werden mittels Klicken auf das jeweilige Icon links an der aktuellen Stelle eingefügt. Neben Symbolen, die aus der westlichen Notation bekannt sind, gibt es noch einige, die für Anweisungen an die Abspielkomponente benötigt werden. Für Akkorde gibt es ein Textfeld, in das man die Akkorde notieren kann. Außerdem ist hier ein Auswahlmenü, dass nach Eingabe eines Grundtons alle möglichen Akkorde anzeigt.

Weitere Informationen, die zur Erstellung eines Liedes nötig sind, sind Titel, Komponist und Tonart, die in die entsprechenden Felder auf der linken Seite des Editors eingetragen werden. Mit dem Style-Feld kann ein Stil des Players ausgewählt werden.

Erstellte Lieder werden von der Community über ein Forum geteilt. Neben einem eigenen Dateiformat können sie auch als PDF, Audiodatei oder MusicXML exportiert werden. Außer der zur Notation verwendeten Ansicht gibt es noch die Möglichkeit, die Noten in anderer Form darzustellen. Für Nutzer von Klavier, Gitarre und Ukulele gibt können die Akkorde mit Griffbrett- bzw. Tastaturdiagrammen angezeigt werden. Außerdem gibt es die *Number Notation*, in der die Akkorde in der Stufe der Tonleiter notiert werden.

Die Notation eines Liedes in iReal Pro ist relativ schnell erledigt, die Sprache ist größtenteils selbsterklärend. Eine Möglichkeit zur Notation von Melodien gibt es nicht. Die Abspielkomponente ermöglicht ein sofortiges Anhören der notierten Akkordfolge und lädt zur Improvisation ein. Die Anzahl der möglichen Abspielstile ist jedoch relativ klein, eine eigene Veränderung dieser Stile ist nicht möglich. Die Notation als Text und außerhalb der Software ist nicht möglich.

3.2. Fazit

In diesem Kapitel wurden zunächst einige Anforderungen an mögliche Lösungen vorgestellt. Diese Anforderungen lassen sich in den folgenden Stichpunkten zusammenfassen.

- Improvisation und Komposition im Einklang
- eine Notenschrift mit musiktheoretischer Basis
- ein ausgewogenes Verhältnis von Informationsdichte und Lesbarkeit
- Unterstützung bei Notation und Improvisation durch den Computer

Anschließend wurden drei verschiedene vorhandene Lösungen besprochen. In Abschnitt 3.1.2 wurde LilyPond vorgestellt, eine Notensatzsoftware mit mächtiger Texteingabesprache zur Notation. Ihre Stärken liegen im Notensatz, Unterstützung bei Komposition oder Improvisation gibt es keine. Dafür ist die Syntax der eigentlichen Musiknotation gut lesbar, wenn auch die Strukturierung zu Verwirrungen führen kann.

Abschnitt 3.1.3 zeigte mit Sonic Pi eine interne DSL in Ruby, die als Kernkompetenz Livecoding hat. Außerdem hat sie als Ziel, Grundlagen in Musikerzeugung und Programmierung zu vermitteln. Sie macht es relativ leicht, algorithmische Musik zu machen. Wenn man Programmierung beherrscht, hat man ein mächtiges Werkzeug. Das Abspielen der Noten ist eine

zentrale Funktionalität der Software. Die Nähe der Syntax an allgemeiner Programmierung kann allerdings abschreckend wirken.

Zuletzt wurde in Abschnitt 3.1.4 die Software iReal Pro vorgestellt. Sie bietet eine Abspielkomponente mit verschiedenen Stilrichtungen und der Möglichkeit, das Tempo, die Instrumentierung und die Tonart zu ändern. Die Lieder werden in einer graphischen Akkordnotation niedergeschrieben. Dadurch, dass es sich um eine graphische Sprache handelt, ist der Benutzer auf die Anwendung einer Maus gebunden, dafür muss er sich aber keine Syntax merken. Die Flexibilität der unterschiedlichen Abspielstile ist eingeschränkt, da es nicht sehr viele gibt und sie nicht angepasst werden können.

4. Entwurf

Aus den in Kapitel 3 beschriebenen Anforderungen und den dort vorgestellten vorhandenen Lösungen kann ein Konzept für ein Notationssystem erstellt werden, das im folgenden vorgestellt werden soll. Anschließend werden die wichtigsten Anwendungsfälle beschrieben. Zum Verständnis der Strukturen wird dann das Datenmodell des Systems beschrieben. Daraufhin werden die Designentscheidungen in Bezug auf die Grammatik erklärt. Dann folgt ein Blick auf den Systemkontext und die Systemstruktur, gefolgt von einer genaueren Untersuchung der zu entwickelnden Komponenten.

4.1. Konzept

Das folgende Konzept hat zum Ziel, die bereits vorgestellten Anforderungen zu erfüllen. Der Name des Systems lautet *Jazzler*.

In der Jazz-Musik spielen Improvisation und Komposition eine ebenbürtige Rolle. Dies soll von dem hier vorgestellten System berücksichtigt werden, indem die Notation auf Akkordfolgen beschränkt ist. Melodien sind zwar ein Teil der Jazzmusik, können aber zugunsten der Improvisation in den Hintergrund treten.

Es soll sich bei der DSL um eine textliche Form handeln. Dies liegt darin begründet, dass eine textuelle Sprache schnell zu nutzen ist, wenn die Syntax gut verständlich ist. Außerdem kann die Notation so auch ohne die notwendige Software erfolgen. Die Wahl fällt zudem auf die Form einer externen DSL. Dies hat den Grund, dass die Syntaxelemente einer Wirtssprache ablenkend wären und die Ästhetik beeinflussen würden.

Die Notation der Musik soll auf Basis musiktheoretischer Grundlagen erfolgen. Dies soll durch die Verwendung einer Akkordsymbolnotation erfolgen, die den existierenden Notationsformen ähnelt. Dadurch kann das vorhandene Wissen genutzt werden, um in geeigneter Form Akkordfolgen zu notieren. Die Akkorde sollen zudem in römischen Zahlen notiert werden, die ihre Stufe in der Tonleiter darstellen. Dies ermöglicht eine Sicht auf die musikalischen Zusammenhänge und ermöglicht die schnelle Änderung der Tonart ohne du Nutzung eines Transponierungswerkzeugs.

Die Strukturierung eines Liedes kann je nach Art recht kompliziert werden und Wiederholungen beinhalten. Um die Syntax einfach zu halten und die Struktur leicht sichtbar zu machen, soll in der Notation ein Element zur Strukturierung vorhanden sein. Akkordfolgen sollen in verschiedene Figuren aufgeteilt werden, die anschließend in einer Liedstruktur in eine Abfolge gebracht werden sollen. Dies ermöglicht auch, den Aufbau eines Stücks schnell zu ändern, ohne die konkreten Akkordfolgen bearbeiten zu müssen.

Die Notation der Musik soll durch eine interaktive Umgebung unterstützt werden. Dazu wird als Benutzerschnittstelle eine REPL (Read-eval-print Loop) verwendet. Sie liest Befehle ein, wertet sie aus und gibt ein Ergebnis zurück. Sie wird dazu genutzt, nach und nach ein Musikstück zu erstellen und mit diesem zu interagieren. Neben der Funktionalität zur Notation des Stückes soll es eine Möglichkeit zum Abspielen des notierten Liedes geben. Das ermöglicht die Überprüfung der Stückes, wie auch die Improvisation auf dessen Grundlage. Eine Skizze der Oberfläche ist in Abbildung 4.1 zu sehen.

```
Song> Title: Sample Song
Sample Song> Tempo: 120
Sample Song> Key: G Major
Sample Song> song
{ :bpm 120
  :key {:root :G :mode :major}
  :title "Sample Song"}
```

Abbildung 4.1.: Ansicht der REPL-Oberfläche mit einigen Beispieleingaben

Um das Stück und dessen Interpretation weiter zu präzisieren, soll es möglich sein, den Rhythmus festzulegen. Lieder haben oft eine weitgehend einheitliche Form des Rhythmus über viele Takte hinaus, während einige wenige Takte rhythmische Veränderungen enthalten. Die rhythmische Notation soll daher an verschiedenen Stellen möglich sein: Auf Ebene des gesamten Liedes, im Rahmen einer Akkordfolge und für einzelne Takte. Um das Abspielen von Anfang an zu ermöglichen, sollen im Fall fehlender Elemente im Lied auf Standardwerte zurückgegriffen werden.

Ein Zweck der Notation ist die Speicherung einer Idee oder eines Liedes. Dafür soll eine Speicherfunktion eingebaut werden, die die Musik als Klartext in eine Datei ablegt und bei

Bedarf wieder lädt. Der Text soll dabei der Syntax der Grammatik entsprechen, was die Notation eines Liedes außerhalb der REPL ermöglicht.

4.2. Anwendungsfälle

Im folgenden Abschnitt werden die wesentlichen Anwendungsfälle des zu beschreibenden Systems beschrieben.

4.2.1. Notation eines Musikstückes

Die Notation eines Musikstückes besteht aus mehreren Schritten. Die wesentlichen Informationen über ein Stück umfassen Titel, Komponist, Tonart, Figuren und Struktur des Stückes, sowie Rhythmusinformationen auf verschiedenen Ebenen. Da die Abläufe in diesen verschiedenen Formen der Eingabe sehr ähnlich sind, wird hier nur ein Anwendungsfall beschrieben, der exemplarisch für die anderen steht.

Titel:	Notation einer Akkordfolge als Figur
Akteur:	Musiker
Ziel:	Notation einer Akkordfolge als Figur
Auslöser:	Der Musiker hat eine Akkordfolge, die er notieren und als Figur speichern möchte.
Vorbedingungen:	Das System ist gestartet.
Nachbedingungen:	Die Akkordfolge ist als Figur in einem Lied im System geladen.

Erfolgsszenario:

1. Der Musiker gibt in der Syntax der Sprache eine Akkordfolge als Figur ein.
2. Das System prüft die Befehlsstruktur und prüft, ob es sich um einen Befehl der REPL oder der Sprache handelt.
3. Das System liest die Eingabe ein und prüft die Syntax anhand der Sprache.
4. Das System speichert die Figur und die Akkordfolge in einer Liedstruktur.
5. Das System überprüft die Liedstruktur auf semantische Korrektheit.
6. Das System übernimmt die Liedstruktur als Basis für weitere Eingaben.

Erweiterungen:

2a. Falls ein Fehler in der Befehlsstruktur auftritt, wird ein entsprechender Fehler angezeigt und der Befehl verworfen.

3a. Falls ein Fehler in der Syntax auftritt, wird dieser dem Musiker angezeigt und die Befehlsausführung abgebrochen.

5a. Falls die Liedstruktur ungültig ist, wird der Fehler dem Benutzer angezeigt und die Liedstruktur verworfen.

Tabelle 4.1.: Anwendungsfall: Notation einer Akkordfolge als Figur

4.2.2. Laden eines Musikstückes

Der folgende Anwendungsfall beschreibt das Öffnen eines Musikstückes aus einer Datei.

Titel:	Laden eines Musikstückes
Akteur:	Musiker
Ziel:	Laden eines Musikstückes aus einer Datei in das System
Auslöser:	Musiker hat eine Datei, die ein Lied enthält und möchte dies ins System laden.
Vorbedingungen:	System ist gestartet und hat kein Lied geladen. Die Datei enthält Lieddaten.
Nachbedingungen:	Das Lied ist im System geladen.

Erfolgsszenario:

1. Der Musiker führt unter Angabe eines Dateipfades und eines Dateinamens den Ladebefehl aus.
2. Das System prüft die Befehlsstruktur.
3. Das System öffnet die Datei und liest sie ein.
4. Das System erzeugt aus den gelesenen Daten eine Liedstruktur.
5. Das System überprüft die semantische Korrektheit der Liedstruktur.
6. Das System zeigt dem Musiker die Liedstruktur an.

Erweiterungen:

2a. Falls ein Fehler in der Befehlsstruktur auftritt, wird ein entsprechender Fehler angezeigt und der Befehl verworfen.

3a. Falls ein Fehler beim Lesen der Datei auftritt, wird er dem Benutzer angezeigt und der Ladevorgang abgebrochen.

5a. Falls die Liedstruktur ungültig ist, wird der Fehler dem Benutzer angezeigt und die Liedstruktur verworfen.

Tabelle 4.2.: Anwendungsfall: Laden eines Musikstückes

4.2.3. Speichern eines Musikstückes

Die Speicherung eines Liedes in einer Datei wird im folgenden beschrieben.

Titel:	Speichern eines Musikstückes
Akteur:	Musiker
Ziel:	Festschreiben eines Musikstückes auf ein Speichermedium
Auslöser:	Musiker hat eine Musikstück im System notiert und möchte dies speichern.
Vorbedingungen:	Das System ist gestartet und hat eine Liedstruktur geladen.
Nachbedingungen:	Das Musikstückes ist auf einem Speichermedium gespeichert.
Erfolgsszenario:	

1. Der Musiker führt unter Angabe eines Dateipfades und eines Dateinamens den Speicherbefehl aus.
2. Das System prüft die Befehlsstruktur.
3. Das System erstellt eine Datei mit dem Dateinamen unter dem gegebenen Dateipfad und schreibt die Informationen des Musikstückes in diese Datei.
4. Das System informiert den Benutzer über den Erfolg des Speicherns.

Erweiterungen:

2a. Falls ein Fehler in der Befehlsstruktur auftritt, wird ein entsprechender Fehler angezeigt und der Befehl verworfen.

3a. Falls ein Fehler beim Speichern der Datei auftritt, wird er dem Benutzer angezeigt und der Speichervorgang abgebrochen.

Tabelle 4.3.: Anwendungsfall: Speichern eines Musikstückes

4.2.4. Abspielen eines Musikstückes

Zum Abspielen eines Musikstückes mithilfe des Systems sind die im folgenden Anwendungsfall aufgelisteten Schritte notwendig.

Titel:	Abspielen eines Musikstückes
Akteur:	Musiker
Ziel:	Abspielen eines im System geladenen Musikstückes
Auslöser:	Der Musiker hat ein Musikstück im System notiert und möchte es abspielen.
Vorbedingungen:	Das System ist gestartet und hat eine Liedstruktur geladen.
Nachbedingungen:	Das System hat das Musikstück abgespielt.

Erfolgsszenario:

1. Der Musiker führt den Abspielbefehl aus.
2. Das System prüft die Befehlsstruktur.
3. Das System erzeugt anhand der geladenen Liedstruktur eine Abfolge von Abspielinformationen.
4. Das System sendet die Abspielinformationen an die Abspielkomponente.
5. Die Abspielkomponente spielt das Lied ab.

Erweiterungen:

2a. Falls ein Fehler in der Befehlsstruktur auftritt, wird ein entsprechender Fehler angezeigt und der Befehl verworfen.

3a. Falls nicht alle für das Abspielen notwendigen Informationen in der Liedstruktur enthalten sind, verwendet das System dafür Standardwerte.

Tabelle 4.4.: Anwendungsfall: Laden eines Liedes

4.3. Datenmodell

Zur Entwicklung eines Softwaresystems ist die Modellierung eines Datenmodells sinnvoll. Im Fall von Jazzler setzt sich diese aus zwei wesentlichen Teilen zusammen: Das für die Speicherung eines Musikstückes notwendige Semantic Model, und der Aufbau generellen Software.

4.3.1. Semantic Model

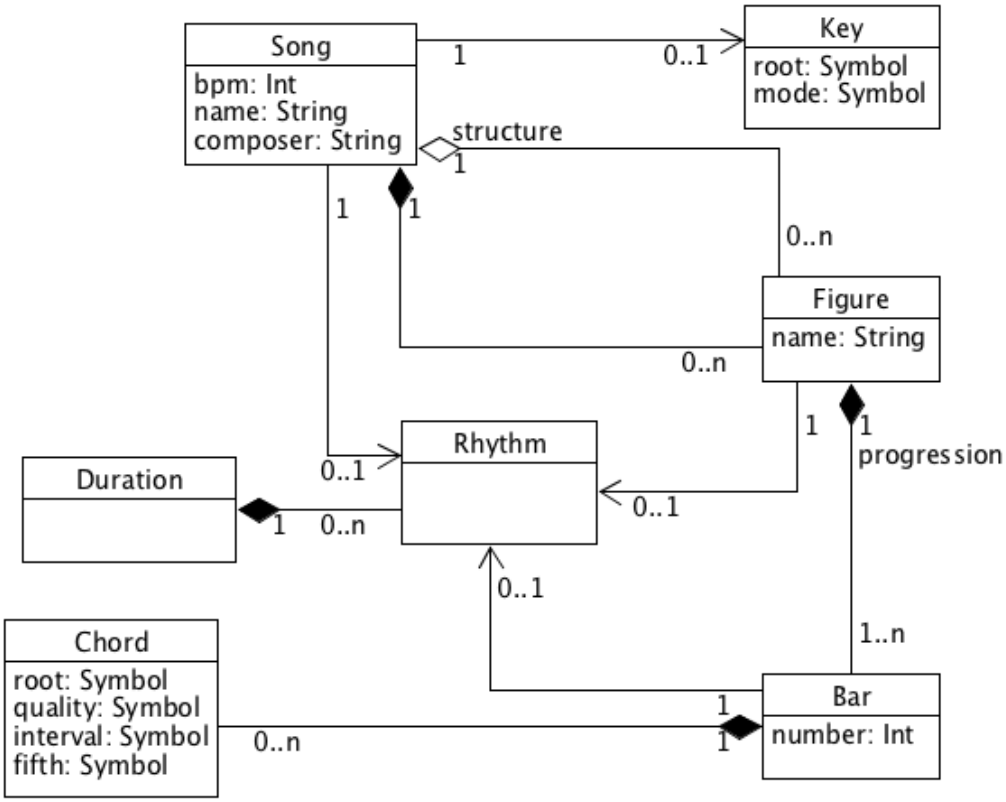

Abbildung 4.2.: Semantisches Datenmodell für ein Musikstück

Für die Darstellung des Semantic Model wurde ein UML-Klassendiagramm gewählt. Das Kernstück ist der Song. Dieser enthält neben Informationen wie Titel, Komponist und Abspieltempo auch die Tonart (Key), welcher aus einem Grundton und einer Modus besteht. Alle möglichen Figures werden in einer Liste gehalten. Mit dem Feld Structure wird der Ablauf des Liedes beschrieben, der durch eine Reihe von Figures realisiert wird. Die Figures müssen im Song erstellt worden sein, bevor sie in der Structure benutzt werden können.

Eine Figure stellt ein Liedsegment dar. Sie besteht aus einem Namen und einer Progression. Ein Beispiel für eine Figure könnte ein Refrain sein. Die Progression stellt eine Reihe von aufeinanderfolgenden Takten dar, die hier als Bar gespeichert werden. Ein Bar wiederum hat eine Taktnummer und eine Anzahl von Akkorden.

Ein Akkord besteht aus root, quality, interval und fifth. Root ist dabei der Grundton, quality entspricht der Bezeichnung eines Dreiklangs. Das interval-Feld wird für die Darstellung eines

Vierklangs verwendet. Das Feld fifth enthält Informationen, ob die Quinte erhöht oder vermindert ist, oder durch eine sus4 ersetzt werden soll. Ein *C-7♭5*-Akkord wird folgendermaßen aufgeteilt: Root ist das C; quality entspricht dem -, also minor; interval enthält ein Symbol für ein dom7 und fifth enthält die Information, dass die Quinte vermindert ist.

Ein Rhythmus-Objekt kann sowohl zu einer Figure, als auch zu einer Bar, als auch zum gesamten Lied gehören. Beim Abspielen kann auf jeder Ebene geprüft werden, ob ein Rhythmus vorhanden ist. Ansonsten wird auf der nächst höheren Ebene nachgeschaut.

4.3.2. Datenmodell des Softwaresystems

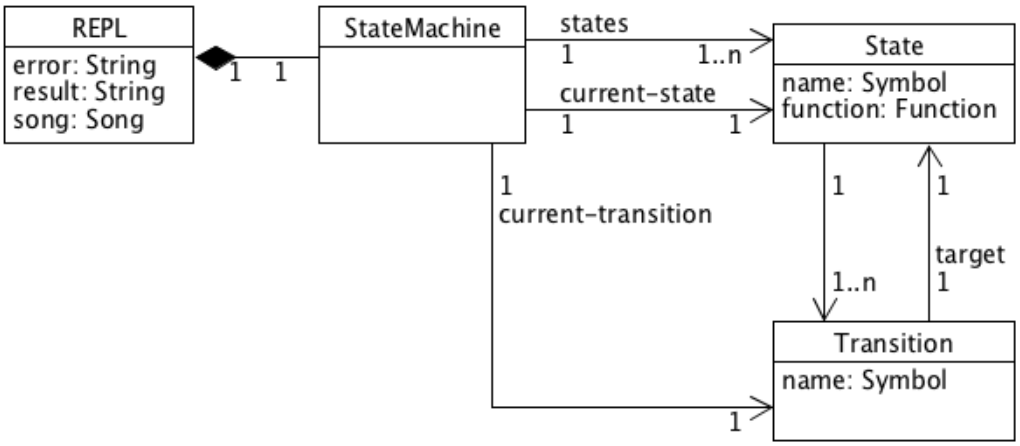

Abbildung 4.3.: Datenmodell der REPL

Abbildung 4.3 zeigt das Datenmodell des Systems. Die REPL enthält neben einigen Feldern für Rückgaben eine StateMachine. Sie enthält einen aktuellen State und eine aktuelle Transition, sowie eine Liste von allen möglichen States. Ein State ist ein Zustand der StateMachine, hat einen Namen und wird durch eine Funktion realisiert. Außerdem hat sie eine Anzahl von möglichen Transitions. Eine Transition ist ein Zustandsübergang. Sie hat einen Namen und einen Zielzustand. Außerdem enthält die REPL einen Song. Dieser entspricht dem in Abschnitt 4.3.1 vorgestellten Semantic Model.

4.4. Grammatikentwurf

Zur Beschreibung der für die Notation eines Stückes verwendete Sprache, soll im folgenden Abschnitt ein Überblick über die Grammatik erfolgen. Nach einer kurzen Übersicht der Struktur folgen genauere Erklärungen zu einzelnen Bestandteilen der Sprache.

Bei dem Entwurf der Grammatik wurde wert darauf gelegt, dass eine gewisse Abstraktion angewendet wird, die dazu führt, dass eine Wiederholung derselben Elemente selten auftritt, wodurch Änderungen leichter durchzuführen sind und die Liedstruktur von einer gehobenen Position dargestellt wird.

4.4.1. Struktur der Sprache

Bevor die einzelnen Elemente der Sprache im Detail besprochen werden, folgt hier eine kurze Übersicht der Sprachstruktur.

Zunächst gibt es Befehle für das Notieren von Autor und Titel, sowie Geschwindigkeit und Tonart des Stücks. Um Akkordfolgen zu notieren, gibt es Progressions. Sie werden mit einer Klammernotation angegeben und stellen eine Abfolge von Akkorden auf verschiedenen Stufen dar. Die Akkorde innerhalb der Progressions werden mittels einer Akkordsymbolnotation notiert.

Progressions werden bei der Definition von Figures verwendet. Diese geben einer Progression einen Namen und sind verwendbar in der Festlegung der Struktur eines Musikstückes. Damit aus einer Sammlung von Figures ein Stück wird, kann eine Structure definiert werden. Sie legt den Ablauf des Musikstückes und damit eine Reihenfolge der Figures fest.

Mithilfe eines Rhythmus-Blockes kann eine Progression, ein Takt oder auch das gesamte Stück mit einer rhythmischen Präferenz versehen werden. Rhythmusblöcke werden mit geschweiften Klammern notiert. Die Trennung von rhythmischen und harmonischen Werten hat die Absicht, dass die jeweiligen Werte so in andere Zusammenhänge gebracht werden können.

```
1  Title: Sample Song
2  Composer: Felix Jensen
3  Tempo: 120
4  Key: G Major
5  Rhythm: {1/4 1/4 1/4 1/4}
6
7  Part1 = [I6 IV6 [I6 V7] IV6]
8  End = [Iadd9]
9
10 Structure: Part1 End
```

Abbildung 4.4.: Sprachbeispiel

Ein Beispiel für ein in dieser Sprache notiertes Musikstück ist in Abbildung 4.4 zu sehen. Es beginnt mit der Notation von Titel, Komponist und Geschwindigkeit des Stückes. Anschließend wird die Tonart festgelegt und der allgemeine Rhythmus des Stückes auf vier Anschläge der Länge einer Viertelnote definiert. In den Zeilen 7 und 8 werden jeweils eine Figure erzeugt. In der letzten Zeile wird die Struktur des Liedes festgelegt.

4.4.2. Titel, Autor und Geschwindigkeit

Um den Titel des Stücks festzulegen, gibt es das Keyword *Title:*, auf das ein String folgen muss. Ähnlich steht es um den Komponisten, der mit dem Keyword *Composer:* und einem String definiert wird. Die Geschwindigkeit des Stücks kann mit dem Schlüsselwort *Tempo:* und einem darauffolgenden Integer-Wert gesetzt werden.

4.4.3. Tonart

Die Tonart besteht aus einem Grundton und einem Modus. Zum Festlegen der Tonart gibt es das Keyword *Key:*. In Abbildung 4.4 wird gezeigt, wie die Tonart auf G-Dur festgelegt wird. Da die Anzahl der möglichen Modi sehr groß ist, wird sich in dieser Arbeit auf die Tonleitern des Ionischen Systems, sowie Dur und Moll beschränkt.

4.4.4. Akkorde

Die Modellierung der Akkorde ist aufgrund der großen Flexibilität der üblichen Akkordsymbole etwas aufwändiger. Ein Akkordsymbol besteht üblicherweise aus einem Grundton, gefolgt von

einem Teil, der im englischen *chord quality* genannt wird. Darauf folgt eine Intervallnummer, zuletzt eine Veränderung der Quinte oder eine *sus4*.

Um eine leichte Änderung der Tonart zu ermöglichen, wird anstelle des Grundtons die römische Zahl der Stufe des Akkords in der Tonleiter verwendet.

Die hier verwendete Akkordschrift bildet nicht alle möglichen Akkorde ab, da nur eine Intervallnummer notiert werden kann. In Jazz kommen jedoch auch komplexere Akkorde vor. Um den Umfang der Arbeit nicht zu sprengen, wird hier auf eine genauere Darstellung der Akkorde verzichtet.

4.4.5. Progressions

Akkordfolgen werden hier Progressions genannt. Um Progressions zu notieren, soll es möglich sein, mehrere Takte zu notieren und dabei kein starres Format zu fordern. Außerdem müssen sowohl Takte mit nur einem Akkord notiert werden können, als auch Takte, die einen oder mehrere Akkordwechsel enthalten. Um dies zu ermöglichen, wird die Akkordfolge von eckigen Klammern umschlossen. Innerhalb dieser Klammern können Takte notiert werden. Wenn ein Takt mit nur einem Akkord gefüllt wird, so kann er mit der Notation des Akkords notiert werden. Wenn mehrere Akkorde im Takt enthalten sind, so wird der Takt mittels eines weiteren eckigen Klammernpaares dargestellt, in das beliebig viele Akkorde eingefügt werden können.

4.4.6. Figures

Die Notation eines Stücks erfolgt in Figures. Diese Figures ähneln Variablen einer Programmiersprache. Die Syntax der Figures lautet *Figurename = Progression*. Der Figurename besteht dabei aus einem Großbuchstaben, gefolgt von beliebig vielen Kleinbuchstaben. Leer- oder Sonderzeichen sind nicht erlaubt. Durch den Einsatz von Figures ist es möglich, wiederholte Liedsegmente nur einmal notieren zu müssen. Außerdem bietet es eine Möglichkeit der Strukturierung des Stücks.

4.4.7. Structure

Die Struktur eines Liedes besteht aus einer Folge von Figures. Sie wird als Structure bezeichnet. Zur Notation wird das *Structure*-Keyword angegeben. Danach können Figurenames angegeben werden, wobei die Reihenfolge die Struktur des Liedes festlegt. Jede Figure kann auch mehrmals vorkommen.

4.4.8. Rhythm

Zum Abspielen des Liedes ist es notwendig, eine Form des Rhythmus zu definieren. Ansonsten müsste eine von dem System definierte Form gewählt werden. Um eine zufriedenstellende Lösung zu erreichen, ist die Platzierung des Rhythmus-Elements wichtig.

Für die grundsätzliche Syntax der Rhythmuselemente ist die Wahl auf die geschweiften Klammern gefallen. Sie umschließen eine Folge von Brüchen, die der Länge eines Anschlags entsprechen.

Diese Form eines Rhythm-Blocks bietet die Möglichkeit, den Rhythmus an verschiedenen Stellen zu definieren. Zum einen kann als Fallback-Lösung für den gesamten Song der Block mit dem Keyword *Rhythm:* angegeben werden. Am Ende einer Figure kann ebenfalls ein solcher Rhythm-Block angehängt werden, was den Rhythmus auf die Progression dieser Figure anwendet, nicht aber auf andere Progressions des Liedes. Außerdem kann ein solcher Block innerhalb einer Progression nach einem Takt verwendet werden, um den Rhythmus nur dieses Taktes zu beeinflussen. Die verschiedenen Anwendsungsformen des Rhythm-Blocks sind in Abbildung 4.5 zu sehen.

```
1  Rhythm: {1/2 1/8 1/8 1/4}
2
3  Figure = [I II] {1/4 1/2 1/4}
4  Figure2 = [I{1/1} [I II]{1/2 1/2}]
```

Abbildung 4.5.: Verschiedene Verwendungsformen des Rhythm-Blocks

4.5. Systemkontext

Abbildung 4.6 zeigt das Systemumfeld. Der Benutzer gibt Musikdaten in das System ein. Zum Abspielen sendet das System Abspielinformation an SuperCollider[1]. Bei SuperCollider handelt es sich um ein System zur Audiosynthese und algorithmischen Musikerzeugung. Dies spielt dann die Musik ab.

[1] https://supercollider.github.io

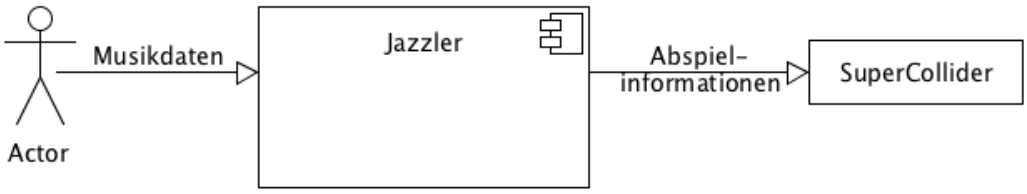

Abbildung 4.6.: Kontextsicht des Systems

4.6. Systemstruktur

Die Architektur der Software setzt sich aus einigen Komponenten zusammen. Sie ist in Abbildung 4.7 zu sehen. Die zentrale Komponente ist die REPL-Komponente. Sie stellt die Schnittstelle zum Benutzer bereit und beinhaltet die weitere Ablaufsteuerung. Sie benötigt die Parserkomponente, um eingehende Befehle zu übersetzen und die Player-Komponente, um die Möglichkeit zum Abspielen bereitzustellen. Die Song-Komponente wird von allen Komponenten verwendet. Sie enthält das Semantic Model und somit die Datenstruktur, die im Zentrum der Anwendung steckt.

In der Parser-Komponente ist die Grammatik definiert. Hier werden Programme der DSL eingelesen und über verschiedene Transformationen ins Format des Semantic Models gebracht.

Die Player-Komponente übersetzt die interne Datenstruktur des Musikstücks in eine sequenzielle Form und sendet sie über eine Schnittstelle an SuperCollider.

Die Song-Komponente enthält ein Datenmodell zur Verarbeitung von Musikstücken. Es bietet die Möglichkeit, Attribute eines Musikstückes zu setzen und auszulesen. Darüberhinaus kann hier die semantische Korrektheit des Stückes überprüft werden.

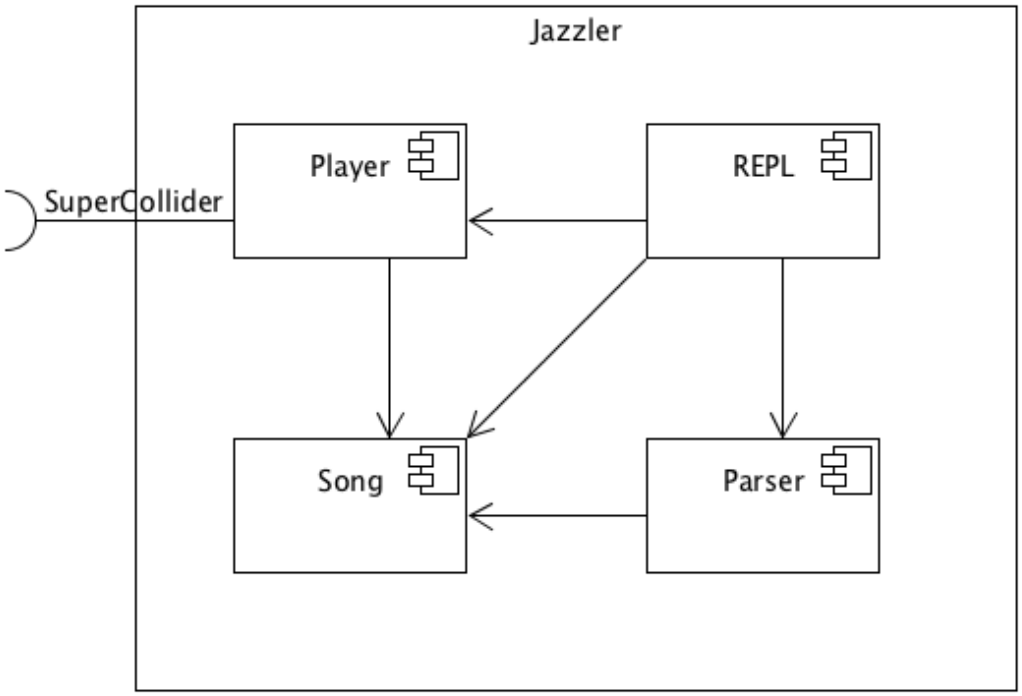

Abbildung 4.7.: Bausteinsicht des Systems

4.7. Komponenten

Die im vorherigen Abschnitt vorgestellten Komponenten des Systems sollen im folgenden detailliert besprochen werden. Zunächst wird die Song-Komponente beschrieben, anschließend die REPL, dann der Parser und zuletzt die Abspielkomponente.

4.7.1. Song

Die Song-Komponente besteht aus einem Song-Objekt, dass Möglichkeiten zur Speicherung von Song-Informationen bietet. Außerdem wird hier die semantische Korrektheit geprüft. Abbildung 4.8 zeigt diese Komponente.

Diese Komponente ist größtenteils eine passive Datenstruktur. Da sie jedoch eine zentrale Rolle in allen Bereichen der Anwendung spielt, wurde sie als eigene Komponente modelliert. Bei einer eventuellen Weiterentwicklung des Systems können hier weitere Funktionalitäten

eingebaut werden, die Auswertungen und weitere musikalische Unterstützung ermöglichen könnten.

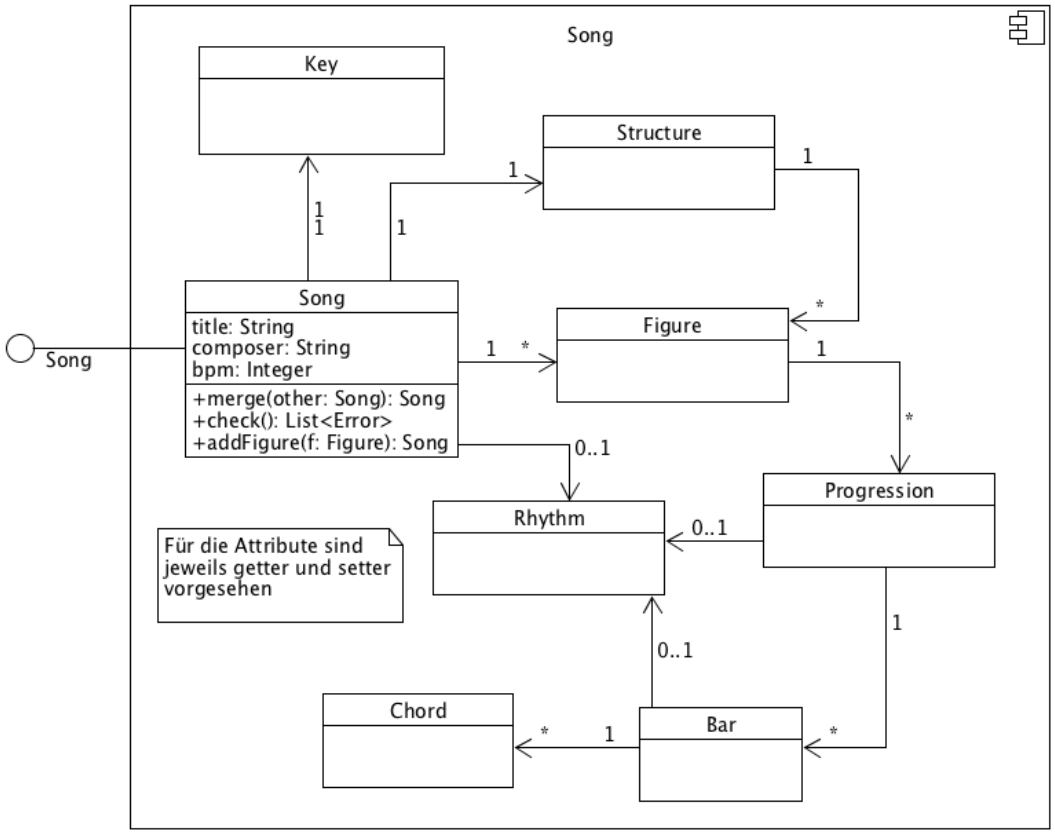

Abbildung 4.8.: Bausteinsicht der Song-Komponente

4.7.2. REPL

Die REPL-Komponente ist das zentrale Stück der Ablaufsteuerung des Systems. Sie empfängt Eingaben vom Benutzer, wertet die Befehle aus und zeigt dem Benutzer Ausgaben an. Die Komponente besteht aus mehreren Elementen. Bevor aber der Aufbau der Komponente vorgestellt wird, soll der strukturelle Ablauf mithilfe eines Zustandsautomaten beschrieben werden.

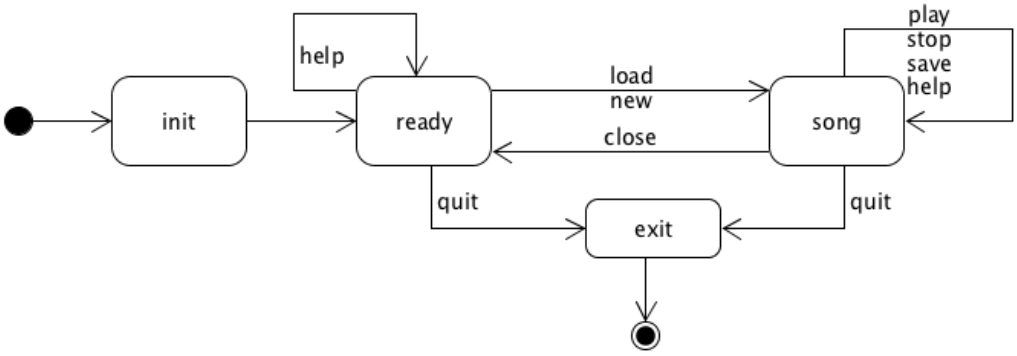

Abbildung 4.9.: Zustandsautomat der REPL

Abbildung 4.9 zeigt das Zustandsdiagramm der REPL. Beim Start wird zunächst der Zustand *init* durchlaufen. In diesem werden zum Start notwendige Aktionen ausgeführt. Eine Willkommensmeldung wird ausgegeben. Anschließend wechselt der Automat in den *ready*-Zustand. Bei erfolgreichem Ausführen der Befehle *load* und *new* wird der Zustand verlassen und zum *song*-Zustand gewechselt. Dieser bietet Funktionen zum Abspielen, Speichern und Verändern eines Liedes. Beim Schließen des Liedes wird zurück in den *ready*-Zustand gewechselt. In beiden Zuständen gültig sind die *help*-Befehle, die einen Hilfetext anzeigen, sowie die *quit*-Befehle, die zum Zustand *exit* führen. Hier wird dafür gesorgt, dass die REPL sauber herunterfährt. Anschließend beendet der Automat.

Jeder Zustand des Systems erwartet gewisse Eingaben durch den Benutzer. Dieselbe Eingabe kann in unterschiedlichen Zuständen verschiedene Auswirkungen haben, da die Abwicklung der Eingabe durch den jeweiligen Zustand erfolgt.

Die Wahl der Architektur für einen Zustandsautomaten kommt daher, dass so die Flexibilität der Anwendung erhöht wird. Unterschiedliche Zustände können unterschiedliche Arbeitsschritte abbilden. So könnte das Abspielen beispielsweise auch in einem anderen Zustand erfolgen, wenn die Komplexität der Abspielkomponente größer wird. In diesem Stadium der Anwendung wurde sich dagegen entschieden, da der Aufwand des Zustandswechsels für den Benutzer im Verhältnis zum Mehrwert eines eigenen Zustandes zu groß ist.

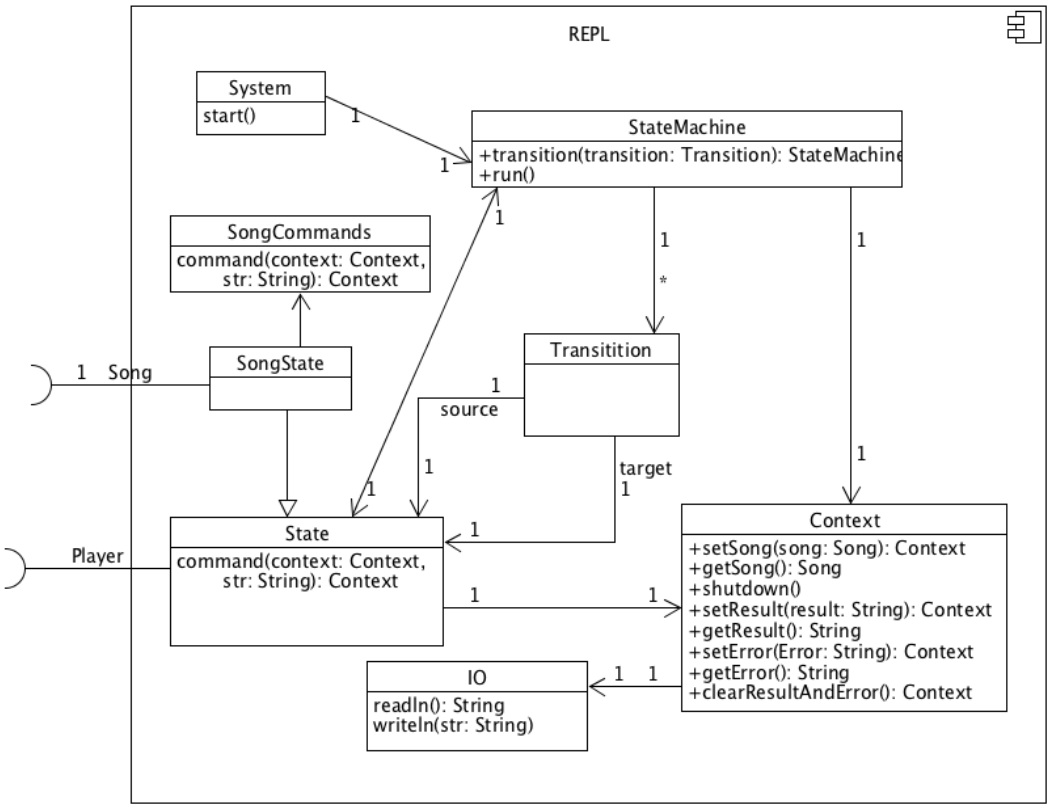

Abbildung 4.10.: Bausteinsicht der REPL-Komponente

Als Basis der REPL wird ein Zustandsautomat verwendet. Die *StateMachine* bietet einen abstrakten Zustandsautomaten. Er enthält einen Systemkontext, in dem der Zustand des Systems gespeichert ist. Dazu gehört die Song-Struktur, sowie Informationen, die zur Ausgabe bestimmt sind, Result und Error. Ein Result ist eine Ausgabe des Systems an den Benutzer. Beim Befehl *help* beispielsweise wäre der Hilfetext das Result. Wenn bei der Bearbeitung eines Befehls ein Fehler auftritt, so wird er in Error gespeichert. Die Trennung von Result und Error ermöglicht eine unterschiedliche Verarbeitung und Ausgabe.

Neben dem Kontext hat die StateMachine einen aktuellen State, der für die Abwicklung der eingehenden Befehle zuständig ist. Jeder Zustand kann in der StateMachine einen Zustandswechsel bewirken. Dazu hat sie eine Liste von Transitions, die von einem Zustand zu einem anderen Zustand führen. Die Übergänge sind dabei genau definiert.

Das Systemmodul bietet eine Konfigurationsstelle und dient zum Starten der REPL. In ihr werden die für den Zustandsautomaten benötigten Maps definiert. Die Funktionen, die für

die Zustände verwendet werden, sind im Modul *States* definiert. Jeder der Zustände erwartet gewisse Befehle. Die Befehle für den Song-Zustand werden im *SongState* bearbeitet. Dieses Modul hat eine Funktion *command*, die einen String erwartet. Je nach Inhalt des Strings wird eine Funktion aufgerufen, die den Befehl weiterverarbeitet. Handelt es sich um einen gültigen Befehl, so gibt es eine spezielle Funktion dafür, ansonsten wird die Funktion *unknown* aufgerufen, die für die Ausgabe eines Fehlers sorgt. Die oben genannten Befehle können keine direkten Ausgaben erzeugen. Stattdessen werden die Ausgaben in der Datenstruktur gehalten und in der Zustandsfunktion ausgegeben. Funktionen für diese Art von Ausgaben und zur Durchführung von Zustandsübergängen werden im Context definiert. Zum erleichterten Wechsel der konkreten Ein- und Ausgabe-Implementation wurde ein *IO*-Modul konzipiert, das eine Schnittstelle für die wichtigsten Les- und Schreibfunktionen anbietet.

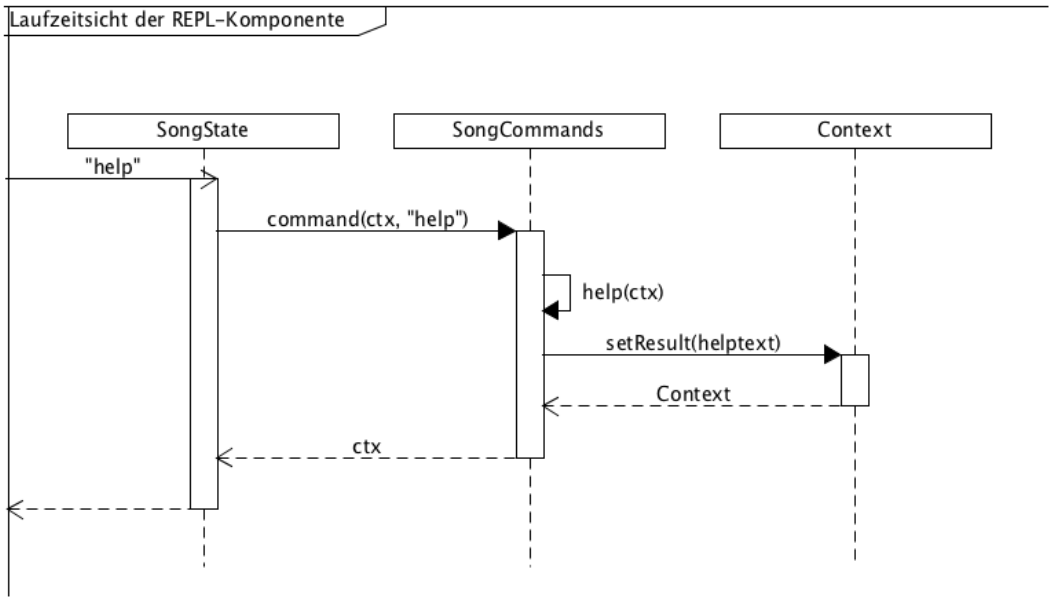

Abbildung 4.11.: Laufzeitsicht der REPL-Komponente

Abbildung 4.11 zeigt den groben Ablauf der Verarbeitung eines Befehls in der REPL-Komponente. Die Zustandssteuerung ist dabei außen vor gelassen. Der SongState erhält eine neue Nachricht, diesem Fall den String *help*. Er gibt den Aufruf an SongCommands weiter. Hier wird er analysiert und die passende Funktion *help* aufgerufen. Diese analysiert den Aufruf auf Parameter und belegt im Context-Objekt das Result-Feld mit einem Hilfetext. Dieses Context-Objekt wird über SongCommands und SongState zurück an die StateMachine gegeben.

4.7.3. Parser

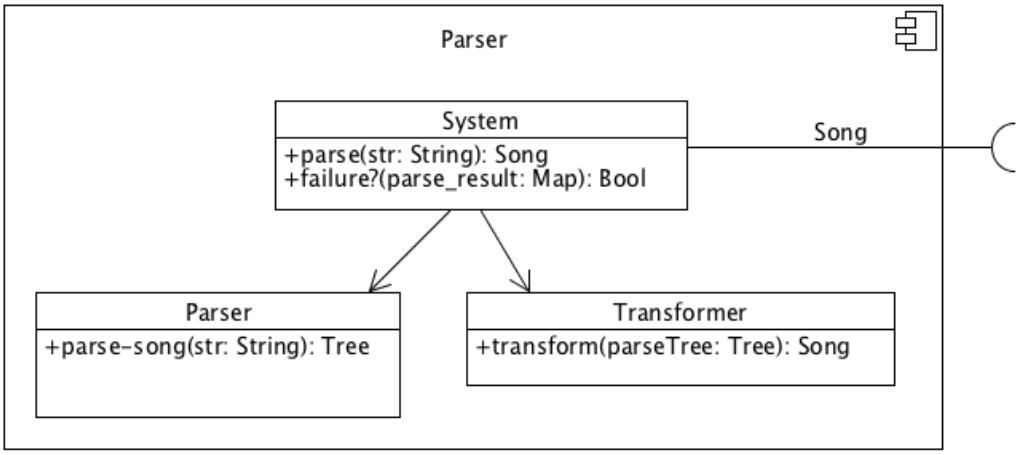

Abbildung 4.12.: Bausteinsicht der Parser-Komponente

Die Parser-Komponente ist zuständig für das Parsen der Grammatik der DSL und der Transformation der daraus resultierenden Baumstrukturen in die Song-Struktur. Zur Generierung der notwendigen Parse-Funktionen wird die Instaparse-Bibliothek verwendet. Sie benötigt dafür eine Grammatik in der Form einer EBNF, die im *Parser* definiert wird.

Zur Transformation der von den Parse-Funktionen ausgegeben Datenstrukturen in die eigene, bietet der *Transformer* eine Funktion an. *System* verbindet den Ablauf vom Parsen und Transformieren und bietet eine Schnittstelle nach außen an.

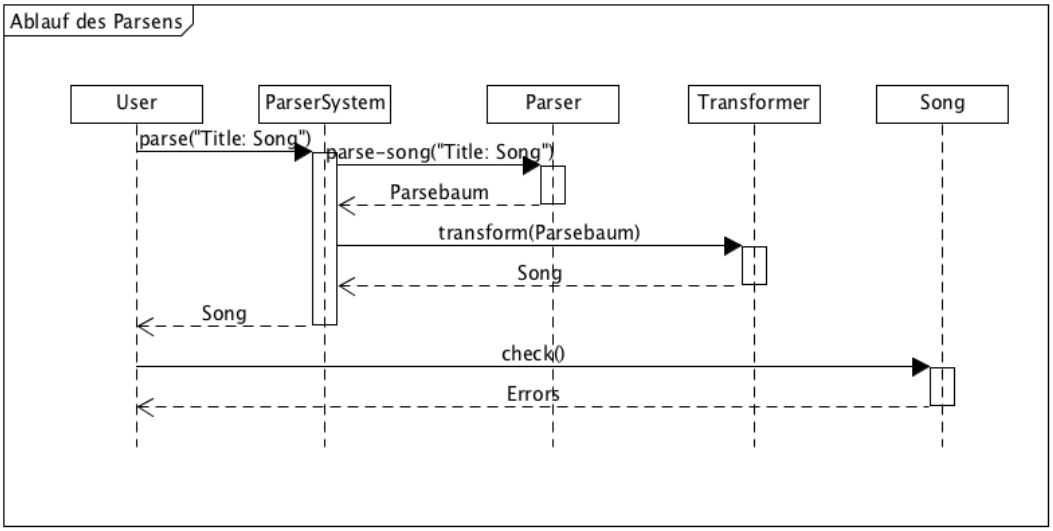

Abbildung 4.13.: Laufzeitsicht eines Parse-Vorgangs

In der Abbildung 4.13 ist eine Laufzeitsicht des Parsevorgangs zu sehen. Der Benutzer der Komponente ruft auf dem ParserSystem die Funktion *parse* mit einem String als Parameter an. Das ParseSystem delegiert den Aufruf an den Parse, der Anhand des Strings und seiner Grammatikregeln einen Parsebaum generiert. Er gibt diesen Parsebaum zurück ans ParserSystem, das mit der Funktion *transform* den Parsebaum an den Transformer weiterreicht. Dieser erstellt anhand einer Reihe von Transformationsregeln aus dem Parsebaum ein Song-Objekt. Dieses wird dem Benutzer zurückgegeben.

Die Arbeit der Parser-Komponente ist hier vorbei. Der Benutzer hat anschließend die Möglichkeit, das Song-Objekt mithilfe der *check*-Funktion auf semantische Korrektheit zu überprüfen.

4.7.4. Player

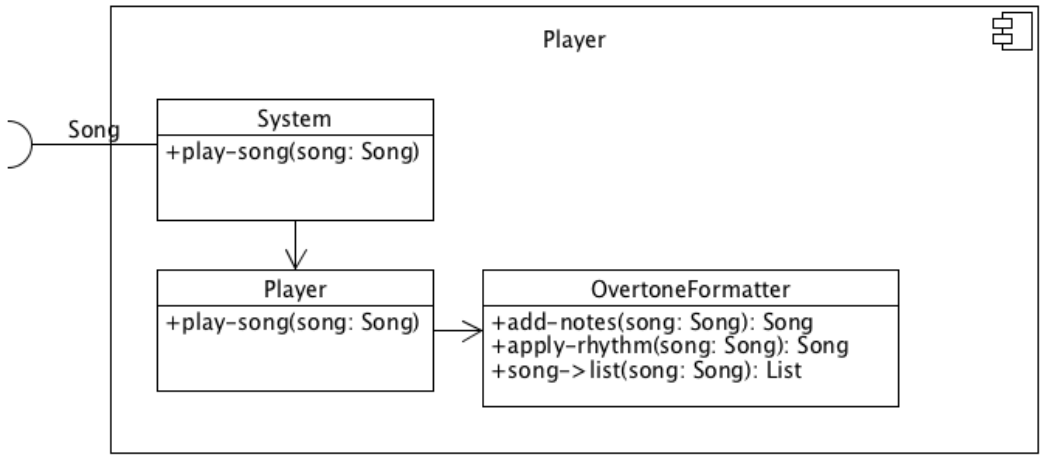

Abbildung 4.14.: Bausteinsicht der Player-Komponente

Die Schnittstelle der Player-Komponente bietet lediglich eine Funktion an: *play-song*. Diese nimmt eine Song-Struktur an. Anschließend fügt sie mithilfe der OvertoneFormatter-Funktion *add-notes* zu jedem Akkord im Lied Notenwerte in Midi-Form hinzu. Außerdem wird der im Lied festgelegte Rhythmus auf die einzelnen Takte angewendet. Damit werden die Takte vervollständigt. Mithilfe der Funktion *song->seq* wird eine Liste von Takten erstellt. Diese wird dann durchlaufen. Die einzelnen Akkorde werden dann zum Abspielen angemeldet. Das Abspielen geschieht asynchron.

4.8. Fazit

In diesem Kapitel wurde gezeigt, wie aus den in der Analyse erarbeiteten Anforderungen ein Entwurf für das Notationssystem Jazzler erstellt wurde. Im Zentrum lag dabei die Erstellung einer Sprache zur Notation von Musikstücken und die Architektur eines interaktiven Systems zur Eingabe und Nutzung der durch die Sprache erzeugten Musikstücke.

5. Implementierung und Auswertung

In diesem Kapitel werden einige Entscheidungen für die Implementierung besprochen. Anschließend wird eine Bewertung des Systems getroffen.

5.1. Programmiersprache und Bibliotheken

Für die Implementierung wurde die Sprache Clojure[1] gewählt. Es handelt sich dabei über einen Lisp-Dialekt, der auf der Java Virtual Machine läuft. Durch besondere Syntax von Clojure, in der Programmiercode als Listen dargestellt werden, ist die Veränderung von Datenstrukturen sehr einfach zu erledigen, was die Baumtransformationen erleichtert. Clojure ist eine dynamisch typisierte funktionale Programmiersprache.

Zur Erzeugung eines Parsers fiel die Wahl auf die Bibliothek Instaparse[2]. Zur Definition von Grammatiken nutzt sie eine EBNF-Syntax. Als Ausgangsformat bietet Instaparse zwei verschiedene Formate, für die es umfangreiche Bibliotheken zur Transformation. Außerdem bietet es eine eigene Funktionalität, mit der simple Transformationen einfach möglich sind. In dieser Arbeit wurde für die Transformation Instaparse verwendet.

5.2. Modellierung in funktionaler Programmierung

In Abschnitt 4.7.2 wurde zur Modellierung der REPL-Komponente ein UML-Klassendiagramm benutzt. Diese Art von Diagramm wird zur Darstellung von Strukturen in der objektorientierten Programmierung verwendet. Bei der objektorientierten Programmierung ist ein wichtiger Aspekt die Verbindung von Zustand und Verhalten in Klassen und Objekten. Dagegen wird bei der funktionalen Programmierung Verhalten und Daten getrennt.

Zur Implementierung der Architektur wurden daher keine Klassen verwendet. Die meisten der in den Diagrammen als Klassen dargestellten Elemente wurden stattdessen mithilfe von Modulen realisiert. So gibt es beispielsweise für die Implementierung des Datentyps *Song* ein Modul namens Song. Innerhalb dieses Moduls gibt es eine Funktion zur Erstellung einer neuen

[1] http://clojure.org
[2] https://github.com/Engelberg/instaparse

Songstruktur, sowie Funktionen zur Manipulation einer solchen Struktur. Die Datenstruktur des Songs ist als HashMap realisiert.

Neben den Funktionen, die direkt den Song betreffen, enthält das Modul aber auch Funktionen, die Attribute des Songs betreffen. So gibt es hier auch Funktionen zur Erzeugung von Akkorden und Akkordfolgen. Die im Diagramm angegebenen Relationen gelten weiterhin.

Eine Besonderheit ist die Implementierung der Zustände und Commands der REPL. Ein Zustand wird durch eine Funktion repräsentiert, in der der allgemeine Ablauf beschrieben wird. Zusätzlich hat er eine Reihe von Funktionen, die jeweils einen Command abbilden. Während die Commands im Klassendiagramm als Klassen dargestellt werden, sind es hier Funktionen, die für den jeweiligen Befehl verantwortlich sind. Es gibt keine Vererbung, sondern nur eine strukturelle Ähnlichkeit.

5.3. Audioausgabe

Um das Abspielen der Musik zu ermöglichen, wurde Overtone[3] verwendet. Dabei handelt es sich um eine Clojure-Library, die eine Schnittstelle zu SuperCollider bietet.

SuperCollider bietet eine Sprache zur Definition von Synthesizern und zur programmatischen Erzeugung von Kompositionen, die eine syntaktische Ähnlichkeit zu der Programmiersprache C hat. Zum Ausführen der Töne und Kompositionen bietet es darüber hinaus einen Server an, der mittels einer offenen Schnittstelle Steuerungsbefehle entgegennimmt. Die Library Overtone ermöglicht es, die SuperCollider-Umgebung unter Nutzung von Clojure-Code anzusprechen.

Zu Overtone gehört auch die Library at-at[4]. Mit ihr ist es möglich, die Ausführung von Funktionen zu einem Zeitpunkt in der Zukunft festzulegen. Dies wird in der Player-Komponente genutzt, um das Abspielen der einzelnen Töne zu realisieren.

Der allgemeine Ablauf des Players ist so aufgebaut, dass beim Ausführen der Abspielfunktion die relevanten Takte durchlaufen werden. Für jeden Takt wird ein Soundereignis geplant, das einen Abspielzeitpunkt und eine Dauer erhält. Aufgrund der funktionalen Natur von Clojure ist es sehr einfach, die konkrete Funktion, die das Soundereignis bei SuperCollider auslöst, auszutauschen. So ist es möglich, die Abspielkomponente zu ersetzen oder zu erweitern. Es wäre beispielsweise möglich, nicht nur ein Soundereignis bei SuperCollider auszulösen, sondern zusätzlich eine Nachricht an eine grafische Oberfläche zu senden, die den aktuell abgespielten Akkord anzeigt.

[3] https://github.com/overtone/overtone
[4] https://github.com/overtone/at-at

5.4. DSL

Für die Realisierung der DSL wurde die Instaparse-Library verwendet. Sie erzeugt anhand einer in einem EBNF-Format notierten Grammatik eine Parser-Funktion, die dann zum Parsen eines Programms dieser Sprache verwendet werden kann. Das Ausgabeformat dieses Parsevorgangs ist eine Baumstruktur, die als Liste von Listen realisiert ist. Darüberhinaus bietet Instaparse eine Funktion zur Transformation dieser Baumstruktur. Sie nutzt dazu eine Map, in der für jeden Knoten, der verändert werden soll, eine Funktion eingetragen werden kann. Diese Funktion ist dann für die Umstrukturierung verantwortlich. Dadurch, dass Clojure sehr gute Unterstützung zur Umformung von Listen bietet, sind die Transformationsfunktionen sehr kurz und leicht lesbar.

5.5. Stand der Entwicklung

Die im Rahmen des Entwurfs geplante Anwendung Jazzler wurde zu einem großen Teil entsprechend der Anforderungen implementiert. Der komplette Featureumfang des Entwurfs wurde dabei aber nicht erreicht. So sind die Funktionalitäten zum Laden und Speichern eines Liedes als Datei nicht implementiert worden.

Auch die Verwendung von Rhythmusblöcken ist noch nicht eingebaut, allerdings sind diese als Prototyp realisiert und im Source Code enthalten.

5.6. Auswertung

Die hier vorgestellte Softwarelösung Jazzler hatte das Ziel, einige Anforderungen zu erfüllen. Im folgenden soll kurz untersucht werden, wie diese Anforderungen erfüllt wurden.

5.6.1. Improvisation und Komposition im Einklang

Eine Anforderung war, dass der für Jazzmusik typische Zusammenhang von Improvisation und Komposition beachtet werden soll. Im vorgestellten System sollte diese Anforderung dadurch gelöst werden, dass eine Melodienotation in der Sprache nicht unterstützt wird. Stattdessen sollte eine Abspielkomponente die Improvisation ermöglichen. Außerdem sollten Lieder im Fall fehlender Werte Standardwerte erhalten, die das Abspielen ermöglichen.

5.6.2. Eine Notenschrift mit musiktheoretischer Basis

Die Notationssprache des Systems sollte nicht die Verbindung zu den musiktheoretischen Prinzipien und Grundlagen verlieren. Um dieser Anforderung gerecht zu werden, wurde die Notation der Akkorde durch Akkordsymbole realisiert. Zusätzlich dazu wurden die Akkord-Grundtöne durch römische Zahlen dargestellt, was der Stufe des Akkords in der Tonleiter entspricht. Die Notation der Rhythmuswerte erfolgt durch Bruchzahlen, die entsprechend der üblichen Notenlängen genutzt werden können.

5.6.3. Ein ausgewogenes Verhältnis von Informationsdichte und Lesbarkeit

Den Informationsgehalt der Sprache hoch zu halten, während die Lesbarkeit erhalten bleibt, war eine weitere Anforderung. Auf diesen Aspekt sprechen mehrere Elemente der Sprache an. Zunächst wurden Akkordsymbole ähnlich der Akkordsymbolnotation verwendet. Sie sind kürzer zu notieren, als unter Angabe der einzelnen Noten, aus denen ein Akkord besteht.

Bei der Notation von Akkordfolgen ist es möglich, Takte mit einem einzelnen Akkord anzugeben, wenn der Takt nur einen Akkord enthält. Enthält er mehrere Akkorde, so muss dieser Takt mittels einem Klammerpaar markiert werden. Dabei steht die Notation eines Akkords nicht für einen Anschlag, sondern notiert nur, dass an dieser Stelle dieser Akkord gespielt werden soll. Das konkrete Abspielen des Akkords wird erst durch den Rhythmus festgelegt.

Des weiteren wurde die Gliederung des Liedes in Figures eingeführt. Dies ermöglicht eine Verringerung der Anzahl zu schreibenden Takte, da mehrfach vorkommende Teile in der Struktur mehrmals mit dem Figurenamen aufzählen kann.

Bei der Nutzung des Systems ist es nicht nötig, jede einzelne Information über das zu notierende Stück anzugeben. Fehlende Informationen werden bei der Verwendung durch Standardwerte ersetzt. Dadurch kann die Notation verkürzt werden.

5.6.4. Unterstützung bei Notation und Improvisation durch den Computer

Die letzte Anforderung sagte aus, dass der Benutzer bei der Notation und der Improvisation unterstützt werden soll. Dazu fiel die Wahl auf ein interaktives REPL-System, dass die Verwendung von Funktionen ermöglicht. Dies umfasst auch die Abspielfunktion, die aus diesem REPL-System heraus ausgeführt wird. Weitere unterstützende Funktionen sind durch den Aufbau des Systems möglich, waren aber nicht Teil der Arbeit.

5.7. Verbesserungsmöglichkeiten

Neben der bereits genannten nicht fertig implementierten Funktionalitäten bietet das System noch einige Verbesserungsmöglichkeiten.

5.7.1. DSL

Die DSL ermöglicht primär die Notation von Akkordfolgen. Dabei sind die möglichen Töne an die im System definierten Tonleitern gebunden. Hier könnte eine Notation der Halbtonschritte der Tonleiter einen flexibleren Ansatz ermöglichen. Außerdem ist es in Musikstücken nicht unüblich, die Tonart zu wechseln. Die Möglichkeit des Tonartwechsels würde die Sprache mächtiger machen.

Als Interessant könnte sich auch die Komposition von Figures aus anderen Figures erweisen. Auch kann eine Art Variablenkonstrukt ähnlich den Figures für Rhythmusblöcke weitere Duplikation verringern.

Eine Notationsform für Melodien entspricht zwar nicht den ursprünglichen Anforderungen, könnte aber bei einer guten Syntax die Sprache sinnvoll erweitern.

Weitere Gedanken sind bei Takten mit mehreren Akkorden nötig. Dabei ist es momentan schwierig zu erkennen, wie ein solcher Takt abgespielt wird, wenn kein passender Rhythmusblock vorhanden ist. Wie dieses Problem zu lösen ist, lässt sich hier nicht beantworten.

5.7.2. System

Die REPL kann durch weitere Funktionalitäten, wie dem Speichern von Musikstücken und Hilfsmitteln für das Improvisieren ausgestattet werden. So könnte es beispielsweise eine Auflistung der geeigneten Töne für den aktuellen Akkord geben.

Eine größere Weiterentwicklung wäre die Umstrukturierung des Systems in eine Client-Server-Architektur. Der Server würde eine zentrale Anmeldestelle für beliebige Clients bieten. Die Clients könnten verschiedene Formen haben. Zur Nutzung bei der Improvisation könnte ein Client geeignete Töne Anzeigen, während ein anderer anhand der gegebenen Musikinformationen eine begleitende Basslinie erzeugt und an SuperCollider sendet. Auch könnte ein Client Signale an digitale Instrumente senden.

6. Fazit

Im Rahmen dieser Arbeit wurde Softwaresystem zur Notation von Jazz-Musik konzipiert und umgesetzt. Es besteht aus einer REPL, einer Komponente zum Abspielen, sowie einer DSL, die die zentrale Rolle der Anwendung spielt. Das System ist so entworfen, dass Erweiterungen in der Funktionalität ohne großen Aufwand möglich sind.

Die domänenspezifische Sprache ist darauf ausgelegt, in einer für Musiker lesbaren Form Musik zu notieren, die als Untermalung für Improvisationen dienen kann. Im Fokus liegt die Definition von Akkordfolgen im Kontext einer festgelegten Tonleiter. Die Akkordfolgen werden benannt und können in eine beliebige Reihenfolge gebracht die Liedstruktur definieren. Dazu ist es möglich, den einzelnen Takten auf verschiedenen Ebenen Rhythmusmuster zuzuweisen, die beim Abspielen beachtet werden.

Die REPL bietet eine Schnittstelle für den Benutzer, Lieder in der DSL zu notieren und sie abzuspielen. Dadurch bietet sie die Möglichkeit der Interaktion mit dem zu notierenden Stück.

Die Notation eines Musikstücks ist aufgrund der vielen verschiedenen Notationsarten und Werkzeuge eine indiviuelle Sache. Die Komposition von Musik ist ein kreativer Prozess und die Möglichkeit, unterschiedliche Notationswerkzeuge zur Auswahl zu haben, kann eine positive Auswirkung auf diesen Prozess haben. Und auch die Art der Interaktion mit dem Notationsmedium kann eine immense Hilfe bei der Komposition und Improvisation sein.

Das hier entwickelte System kann sicherlich keine etablierte Notationsform ersetzen, aber ein andere Blickwinkel auf dieselbe kann dabei helfen, einen anderen Aspekt dieser Thematik zu erkennen.

A. Anhang

Inhalt der CD:

1. Bachelorarbeit als PDF: *Bachelorarbeit Felix Jensen.pdf*
2. Quellcode im Ordner *jazzler*
3. Infotext zum Quellcode: *infotext.txt*

Literaturverzeichnis

[Fowler 2010] Fowler, Martin: *Domain-Specific Languages*. First printing. Pearson Education, Inc, 2010. – ISBN 978-0-321-71294-3

[Grabner 2011] Grabner, Hermann: *Allgemeine Musiklehre*. 25. Auflage. Bärenreiter-Verlag Karl Vötterle GmbH & Co. KG, 2011. – ISBN 978-3-7618-0061-4

[Grune u. a. 2000] Grune, Dick ; Bal, Henri E. ; Jacobs, Ceriel J. ; Langendoen, Koen G.: *Modern Compiler Design*. John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, 2000. – ISBN 978-0-471-97697-4

[Haunschild 1998] Haunschild, Frank: *Die neue Harmonielehre*. Erw. und überarb. Neuaufl. 1998. AMA Verlag GmbH, 1998. – ISBN 978-3-7618-0061-4

[Sikora 2012] Sikora, Frank: *Neue Jazz-Harmonielehre*. 5. Auflage. Schott Music GmbH & Co. KG, 2012. – ISBN 978-3-7957-5124-1

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 15. März 2016 Felix Jensen