



Pós-Graduação em Ciência da Computação

**“BLEEM BLOOM: Um ambiente para musicologia
assistida por computador”**

Por

DÍDIMO VIEIRA DE ARAÚJO JUNIOR

Dissertação de Mestrado



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, AGOSTO/2011



**UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

DÍDIMO VIEIRA DE ARAÚJO JUNIOR

“BLEEM BLOOM: Um ambiente para musicologia assistida por computador”

*ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO
EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE
INFORMÁTICA DA UNIVERSIDADE FEDERAL DE
PERNAMBUCO COMO REQUISITO PARCIAL PARA
OBTEÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA
COMPUTAÇÃO.*

ORIENTADOR: GEBER LISBOA RAMALHO

CÓ-ORIENTADOR: GIORDANO CABRAL

RECIFE, AGOSTO/2011

**Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571**

**Araújo Junior, Dídimº Vieira de
BLEEM BLOOM: um ambiente para musicologia
assistida por computador / Dídimº Vieira de Araújo Junior
- Recife: O Autor, 2011.
115 folhas: il., fig., tab., quadro**

**Orientador: Geber Lisboa Ramalho.
Dissertação (mestrado) - Universidade Federal de
Pernambuco. CIn, Ciência da Computação, 2011.**

Inclui bibliografia e apêndice.

**1. Interação Homem - computador. 2. Engenharia de
software. I. Ramalho, Geber Lisboa (orientador). II. Título.**

004.019

CDD (22. ed.)

MEI2011 – 159

Dissertação de Mestrado apresentada por **Dídimo Vieira de Araújo Junior** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título “**Bleem Bloom: Um Ambiente Para Musicologia Assistida Por Computador**” orientada pelo **Prof. Geber Lisboa Ramalho** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

Prof. Márcio Leal de Melo Dahia
D'accord Music Software

Prof. Carlos Sandroni
Departamento de Música / UFPE

Visto e permitida a impressão.
Recife, 18 de agosto de 2011.

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

A meus pais.

Agradecimentos

Aos colegas e professores do Centro de Informática, em especial Scholz, Raphael, Fúlvio, Ernesto, Valmir, Rodrigo, Giordano e Márcio. A Geber, pela oportunidade, orientação e conhecimento.

Aos professores e alunos do Departamento de Música, em especial Sandroni, Nelson, Lara, Érico, Manassés e Jaziel.

Aos amigos que me ajudaram indiretamente, com compreensão pela minha ausência quando precisaram de mim, ou diretamente, em especial Mitsuo, Cecília, Henrique, Walter e Marcílio.

A minha família e a Danielle em especial pela paciência.

Resumo

Pesquisas na área de análise musical são beneficiadas pela possibilidade de resolver questões que envolvem análises complexas de grandes quantidades de dados musicais com a utilização do computador. Diferentemente da maioria das pesquisas sobre expressividade musical - que se concentram em música clássica para piano - o projeto "Um País, Um Violão", realizado no Centro de Informática da Universidade Federal de Pernambuco, estuda a dimensão rítmica da bossa nova ao violão. Vários temas sobre aquisição de conhecimento a respeito da expressividade musical já foram trabalhados dentro do projeto como o reconhecimento de padrões rítmicos, análise harmônica, microrrítmo e microdinâmica.

Em se tratando de pesquisa preliminar e pioneira, um dos problemas em torno do projeto é que seus trabalhos foram realizados sem foco no público-alvo, os pesquisadores em musicologia, o que implica deficiências na usabilidade, fazendo com que, em alguns casos, seja praticamente impossível sua utilização por um usuário sem conhecimentos avançados em computação. Um outro problema do projeto é que seus trabalhos foram implementados em momentos distintos, sem qualquer preocupação em como eles poderiam ser integrados, mantidos e expandidos.

Este trabalho de dissertação objetiva resolver os problemas estruturais e de usabilidade relacionados. Para tanto, a abordagem utilizada envolve a aplicação de métodos de interação humano-computador - para conceber uma solução adequada ao público-alvo - e a análise dos trabalhos atuais, para o projeto e construção de uma nova arquitetura baseada em *plugins*. O objetivo é oferecer novas possibilidades de análise através da integração também dos resultados de cada trabalho, estimulando o uso prático dos trabalhos científicos realizados no projeto "Um País, Um Violão".

As avaliações dos protótipos e experimentos, realizadas por colaboradores representantes do público-alvo, mostraram considerações positivas a respeito do software proposto.

Palavras-chaves: musicologia, usabilidade, interação humano-computador, interface gráfica, engenharia de software, arquitetura de software, plugin, computação musical, "Um país, um violão", Bleem Bloom.

Abstract

Research in music analysis are benefited by the opportunity to resolve issues that involve complex analysis of large amounts of music data using the computer. Unlike most research on musical expression - that focus on classical music for piano - the project "One Country, One Guitar" held at the Computer Center of the Federal University of Pernambuco, is studying the rhythmic dimension of the bossa nova guitar. Several issues regarding the acquisition of knowledge about musical expression have been worked into the project as the recognition of rhythmic patterns, harmonic analysis, microtiming and microdynamic.

In the case of pioneer and preliminary research, one of the problems surrounding the project is that its works were performed without a focus on target audience, researchers in musicology, which implies deficiencies in usability, so that in some cases, be practically impossible to use by a user without advanced knowledge in computing. Another problem is that your project works were implemented at different times, without any concern as they could be integrated, maintained and expanded.

This dissertation aims to solve structural problems and usability related. To this end, the approach involves the application of methods of human-computer interaction - to design an appropriate solution to the target audience - and the analysis of the current work, for the design and construction of a new architecture based on plugins. The goal is to offer new possibilities for analysis by integrating also the results of each work by stimulating the practical use of scientific work on the project "One Country, One Guitar".

The evaluations of the prototypes and experiments, made by employees representatives of the target audience, showed positive considerations about the proposed software.

Keywords: musicology, usability, human-computer interaction, graphic interface, software engineering, software architecture, plugin, computer music, "One country, one guitar," Bleem Bloom.

SUMÁRIO

1 INTRODUÇÃO.....	13
2 O PROBLEMA	17
2.1 Problemas Gerais.....	17
2.1.1 Manutenção de Software.....	17
2.1.2 Usabilidade	19
2.2 Problemas Específicos.....	21
3 ESTADO DA ARTE	28
3.1 Engenharia de Software para Integração e Evolução de Software	28
3.1.1 Integração com Troca de Dados.....	28
3.1.2 Evolução de Legados e Mudanças em Software	31
3.1.3 Modelos de Arquiteturas	34
3.2 IHC para Concepção Focada no Usuário e Usabilidade	36
4 CONCEPÇÃO E ANÁLISE.....	42
4.1 Concepção do Produto.....	42
4.1.1 Plano de Pesquisa.....	43
4.1.2 Público-Alvo.....	44
4.1.3 Coleta de Dados	45
4.1.4 Análise dos Dados	47
4.1.5 Análise de Similares.....	50
4.1.6 Prototipação e Validação	61
4.1.7 O Produto	65
4.2 Projeto da Arquitetura do Software	66
5 IMPLEMENTAÇÃO E EXPERIMENTOS.....	71
5.1 Detalhes Sobre a Implementação	71
5.1.1 Escolhas Tecnológicas (linguagem, bibliotecas e toolkits)	71
5.1.2 Processo	75
5.1.3 Detalhamento	76
5.2 Experimentos.....	82
5.2.1 Características e Funcionalidades Avaliadas	83
5.2.2 Método de Avaliação	84
5.2.3 Resultados	85

6 CONCLUSÃO.....	88
6.1 Objetivos e Contribuições	88
6.2 Trabalhos Futuros	89
REFERENCIAS BIBLIOGRÁFICAS	94
APÊNDICE A - Dados Coletados	100
Dados coletados durante entrevistas:.....	100
Dados coletados nas validações dos protótipos:	102
Análise das anotações de avaliação com protótipos	104
APÊNDICE B - Redução em Categorias.....	106
APÊNDICE C – Requisitos Funcionais	109
APÊNDICE D - Casos de Uso.....	111

ÍNDICE DE ILUSTRAÇÕES

Figura 1 Articulações realizadas pelo intérprete na interface do trabalho de Lima. As seis primeiras linhas representam as cordas do violão, enquanto as duas últimas linhas indicam respectivamente as marcações de tempo e a junção de todas as cordas em uma representação. Retirado de (LIMA, 2007).....	22
Figura 2 Grafo de equipolência como resultado da análise de similaridade entre os padrões rítmicos. Retirado de (LIMA, 2007).....	22
Figura 3 Interface do trabalho estudo em microdinâmica retirado de (HOLANDA, 2010).....	23
Figura 4 Gráfico gerado por um software de planilha eletrônica utilizando os dados fornecidos pelo processamento dos trabalhos de microdinâmica e microrrítmo, retirado de (HOLANDA, 2010).....	24
Figura 5 Interface do trabalho de integração do projeto 1P1V. Retirado de (SENA, 2008).....	25
Figura 6 Comparação entre formatos de dados. (a) Arquivo com campos mapeados por índices (b) Formato CSV com campos separados por vírgulas (c) Formato XML.....	30
Figura 7 Comparação dos níveis de mudanças em software. Na situação retangular podem ser feitas pequenas correções ou alguma melhoria estrutural mais relevante, enquanto a situação circular indica uma implementação completamente nova. A estrela nas duas situações indica que a essência do software em termos de objetivo foi mantida.	32
Figura 8 Técnica de wrapping para adaptar uma porção de software em uma nova situação utilizando sua estrutura original quando a implementação desta pode ser abstraída pelo restante dos elementos envolvidos.....	33
Figura 9 Representação das relações entre as categorias definidas. Focos de estudo, que surgem de uma necessidade, são manipulados por meio de ações funcionais que possuem um objetivo. Esse objetivo é o que leva à análise de quadros de situações, que podem ser a relação entre intérpretes e suas interpretações.....	50
Figura 10 Interface do trabalho de integração de Sena (2008). Retirado de (SENA, 2008).....	51
Figura 11 Interface gráfica do JRing. Retirado de (KORNSTADT, 2001).....	54

Figura 12 Interface gráfico do OpenMusic 5 retirado de (BRESSON, 2011).....	55
Figura 13 Interface gráfica do DAW Cakewalk SONAR X1 retirado de (CAKEWAL, 2011).....	59
Figura 14 Interface do coach mode do TD-4 em seu display. Retirado de (ROLAND, 2009).....	60
Figura 15 Interface do <i>scope function</i> do TD-9 em seu <i>display</i> . Retirado de (ROLAND, 2008)	60
Figura 16 Processo de prototipação iterativa. Com base nos dados previamente coletados e interpretados, o processo consiste em reformular o protótipo confeccionado até que este seja aceito pelos usuários, quando então é implementado. Os dados coletados durante uma validação são utilizados na reformulação do protótipo.....	62
Figura 17 Protótipo baseado em cenários em formato de apresentação digital.....	64
Figura 18 Protótipo funcional desenvolvido para testar a arquitetura proposta.....	64
Figura 19 Protótipo final em papel. A imagem mostra uma cena (<i>timeline</i>) obtida a partir do menu "Arquivo, Nova cena...". Também estão presentes a janela do gerenciador de materiais (no canto inferior esquerdo) e o menu de contexto de um bloco ativando a caixa de diálogo para entrada de parâmetros para a pesquisa de blocos similares.	65
Figura 20 Diagrama de casos de uso das principais funcionalidades	66
Figura 21 Comportamento padrão dos trabalhos. Os programas recebem materiais como entrada, realizam seus processamentos e devolvem resultados.	68
Figura 22 As camadas de uma aplicação baseada em <i>plugins</i> . Retirado de (MAYER, MELZER e SCHWEIGERT, 2003).....	69
Figura 23 O <i>plugin</i> como um <i>wrapper</i>	69
Figura 24 Arquitetura simplificada	70
Figura 25 As atividades de um ciclo	75
Figura 26 Ciclo composto por ciclos paralelos	76
Figura 27 Processo de mapeamento das classes dos <i>plugins</i> e <i>Descriptor</i> realizado pelos <i>class loader</i> do <i>host</i>	79
Figura 28 Atividades realizadas durante uma solicitação de transformação.....	80
Figura 29 Representação dos objetos que compõem uma cena	81
Figura 30 Atualização da interface disparada pela atualização de objetos de dados	82

ÍNDICE DE TABELAS

Tabela 1 Tabela de agrupamento e classificação dos dados coletados. Cada linha da tabela representa a etapa do roteiro de entrevista (indicada na primeira coluna) onde os dados foram coletados. A segunda coluna contém os dados que possuem relação direta com o projeto, e a terceira coluna contém os dados que não possuem relação direta com o projeto.	48
Tabela 2 Resultado dos experimentos.....	86

ÍNDICE DE CÓDIGOS

Código 1 Representação do Humdrum para Nun danket alle Gott, arr. J.S. Bach. Retirado de (HURON, 1999).....	53
Código 2 Exemplo de linha de comando para utilização do Humdrum que localiza quintas paralelas entre as vozes baixo e alto. Retirado de (HURON, 1999).....	53
Código 3 Exemplo da sintaxe de JMusic retirado de (SORENSEN e BROWN, 2011)	57
Código 4 Exemplo da sintaxe que exibe a partitura do conteúdo do arquivo. Retirado de (CUTHBERT e ARIZA, 2011b)	57

1 INTRODUÇÃO

Um dos motivos do crescente interesse dos pesquisadores em estudar música através de máquinas é que ela oferece um conjunto de problemas cujas formalização e investigação via computador são bastante complexas (BALABAN et al., 1992). Considerando que o computador é capaz de detectar eventos tão mínimos que os sentidos humanos não são capazes de identificar e que alguns tipos de análises levariam muito tempo para serem concluídas manualmente, a máquina se torna uma ferramenta indispensável em pesquisas onde está envolvida uma grande quantidade de dados, inclusive dados musicais. Neste contexto, um dos problemas que tem despertado a atenção de pesquisadores é a compreensão da interpretação musical (LIMA, 2007). No momento em que um *software* utiliza uma partitura digital para executar uma música, ali estão instruções matematicamente precisas que o computador é capaz de reproduzir. Entretanto, qualquer músico com um mínimo de experiência, sabe que tocar uma música da forma exata como ela está grafada resulta em algo mecânico e artificial (LIMA, 2007), e, por isso, existem situações em que é desejável e intencional (SLOBODA, 2000) que uma obra musical seja executada de forma um pouco diferente da partitura, considerando elementos interpretativos de forma subjetiva, intuitiva e pessoal (PALMER, 1997). A esse fenômeno é dado o nome de interpretação.

Palmer (1997), citada em Lima (2007), percebeu que a maioria dos estudos sobre interpretação são focadas em música clássica para piano. Ainda que atualmente existam pesquisas em torno de ritmos latinos (MCGUINNESS, 2005) e até mesmo do samba (GOUYON, 2007; NAVEDA et al., 2009), sua constatação continua válida. Pensando nisso, o projeto "Um país, um violão" (1P1V), realizado no Centro de Informática (CIn) da Universidade Federal de Pernambuco (UFPE), se concentra em dar maior foco a elementos rítmicos dentro da bossa nova ao violão, com pesquisas em torno de expressividade e descoberta de conhecimentos musicais. Um dos motivos dessas escolhas é que, no âmbito da Computação Musical (ROADS et al., 1996), área da Ciência da Computação que tem como domínio de aplicação a música, as pesquisas e estudos da interpretação musical concentram-se, via de regra, na análise das dimensões melódica e harmônica (CAMBOUROPOULOS e WIDMER, 2000; PARDO e BIRMINGHAM, 1999). Apenas

recentemente, alguns aspectos da dimensão rítmica, a despeito de sua evidente importância estrutural, têm recebido alguma atenção (DIXON et al., 2004; DESAIN e HONING, 2003). Além disso, na bossa nova, o violão - instrumento que melhor representa o Brasil - exerce papel fundamental no ritmo (LIMA, 2007).

Com o objetivo de compor um conjunto de dados para análise, necessário nos estudos propostos pelo projeto 1P1V, foram gravadas execuções de dois intérpretes, utilizando um violão MIDI. Essas interpretações foram registradas em dois formatos simultaneamente: Pulse-Code Modulation (PCM), que é o sinal de áudio produzido pelo instrumento em formato digital, capturado com um microfone, e Standard Midi File (SMF), que é um arquivo cuja estrutura são dados simbólicos, que representam eventos musicais realizados durante a interpretação, como tocar uma nota com determinada intensidade em determinado momento. Esse conjunto de dados foi utilizado para estudar vários assuntos dentro do projeto 1P1V. Lima (2007) realizou um trabalho sobre padrões rítmicos, que diz respeito às batidas, no jargão dos músicos, realizadas com a mão direita no acompanhamento ao violão, onde são identificadas batidas semelhantes dentro de um dicionário de padrões previamente identificados. Scholz (2008) desenvolveu um processo para identificação de acordes utilizando regras de harmonia jazzística, utilizadas na bossa nova, e técnicas de inteligência artificial. Silvestre (2009) realizou um trabalho sobre Microrritmo, que estuda padrões dos pequenos desvios de tempo, em termos de antecipações e atrasos, aplicados pelo intérprete nos momentos de ataque do acompanhamento durante sua interpretação. Holanda (2010), por sua vez, realizou um estudo sobre padrões de Microdinâmica, analisando as pequenas variações na intensidade dos ataques aplicadas pelos intérpretes ao executar os padrões rítmicos.

Contudo, existem dois problemas que retardam o avanço do projeto 1P1V. Um deles é que os trabalhos desenvolvidos foram construídos sem a preocupação em como o público-alvo - musicólogos e pesquisadores entusiastas - interagiriam com seus programas, focando apenas na contribuição científica. Alguns desses trabalhos possuem uma interface gráfica, mas que não foram construídas consultando tal público-alvo, tornando suas usabilidades inadequadas ao uso prático. Outros trabalhos, em suas situações atuais, exigem que o usuário tenha conhecimentos avançados em computação, pois não possuem interface de interação ou apenas parte dela. O outro problema é que não existe qualquer tipo de integração estrutural entre esses trabalhos. Cada programa foi construído de forma

independente e, por isso, funcionam como programas isolados. Essa característica impede também que os resultados de processamento de cada programa sejam analisados conjuntamente.

Considerando a importância das contribuições, benefícios e resultados que o projeto 1P1V pode oferecer, assim como os problemas existentes em torno do mesmo, a proposta deste trabalho é o desenvolvimento de um ambiente de software para auxílio à musicologia. Tal proposta envolve dois desafios: um de engenharia de software, que visa resolver os problemas estruturais de integração; e outro de usabilidade, que visa oferecer ganhos como novas funcionalidades e possibilidades de análise de maneira amigável, incentivando o uso prático dos trabalhos acadêmicos realizados no CIn da UFPE.

A abordagem utilizada nesse trabalho é composta de duas partes principais. A primeira, relacionada a concepção do produto quanto às suas funcionalidades e usabilidade, busca entender os principais problemas e necessidades do público-alvo e do projeto 1P1V, através de técnicas de pesquisa da área de Interação Humano Computador (IHC). A segunda parte, dedicada ao software e sua estrutura interna, se concentra em projetar um software mais robusto, que ofereça suporte a integração e melhores condições de manutenção a partir da análise dos trabalhos já realizados.

Este documento está organizado em seis capítulos, dos quais os próximos cinco se dedicam a descrever os métodos utilizados e as atividades realizadas durante o desenvolvimento deste trabalho. O capítulo 2 fala sobre os problemas gerais de usabilidade, integração de sistemas legados e arquiteturas de software, além tratar detalhadamente dos problemas específicos do projeto 1P1V, mostrando as condições em que seus trabalhos foram desenvolvidos e o estado atual de cada um deles. O capítulo 3 se dedica a apresentar o estado da arte nos tipos de pesquisas relacionadas a este trabalho, que serviram como base teórica de orientação para o desenvolvimento da solução proposta. No capítulo 4 é onde estão descritos os métodos de concepção de produto utilizados, assim como as atividades realizadas em todas as suas etapas, e as análises realizadas para a definição de uma nova arquitetura para o produto concebido. A implementação, por sua vez, está no capítulo 5, onde são descritas características técnicas, o processo utilizado, os principais elementos da estrutura do ambiente e como estes interagem entre si, além de, ao final do capítulo, os experimentos realizados. Por último, o capítulo 6, que fica

reservado à um resumo do que foi realizado, conclusões, contribuições esperadas e sugestões de trabalhos futuros.

2 O PROBLEMA

Neste capítulo serão apresentados os problemas gerais em temas relacionados a este trabalho, como sistemas legados, integração, usabilidade, interfaces gráficas, além de problemas específicos identificados no contexto do projeto em questão, pois contribuem na motivação deste trabalho.

2.1 Problemas Gerais

Esta seção apresenta problemas gerais a respeito de sistemas legados e suas condições de manutenção, pois causam complicações estruturais que impactam em questões de integração, e de interface gráfica, pois influenciam na qualidade de uso para os usuários.

2.1.1 Manutenção de Software

Segundo Weiderman e colegas, é muito comum que os desenvolvedores não se deparem com um quadro em branco, mas com sistemas legados desenvolvidos ao longo de muitos anos a um custo significativo, que se encontram numa situação de uma verdadeira colcha de retalhos de diversas plataformas e tecnologias (WEIDERMAN, et al. 1997). Por volta de 70% do esforço investido no desenvolvimento de um *software* se concentra em manutenção (REZENDE, 2005). Esses sistemas manipulam dados e tratam de assuntos de vital importância para o funcionamento de uma organização e, por isso, falhas costumam impactar seriamente nos negócios de uma empresa (BISBAL, et al. 1999). Existe, portanto, a necessidade de se manter esses sistemas ativos.

Um sistema legado pode ser definido como "qualquer sistema de informação que resiste significativamente a modificação e evolução" (BRODIE e STONEBRAKER, 1995), resistência esta devida a problemas que são formados ao longo do tempo por diversos motivos. É muito difícil que uma pessoa conheça um sistema por completo, que permaneça no mesmo projeto ou mesmo na empresa por muito tempo. Essa realidade pode gerar problemas de falta ou desatualização da

documentação, assim como mudanças de estilo de gestão que resultam em vários tipos de soluções a medida que novas necessidades são identificadas.

Segundo Sommerville (2004), as empresas que possuem uma grande quantidade de sistemas legados correm o risco de cair no dilema de decidir entre a manutenção ou substituição desses sistemas, pois ambas as opções oferecem custos e esforços. Ele afirma ainda que os diversos programas dos sistemas legados podem ser escritos por pessoas diferentes em linguagens diferentes, e hoje em dia pode ser bastante difícil de encontrar no mercado profissionais com conhecimentos em tecnologias obsoletas. Um grande problema é que softwares que se tornam obsoletos costumam gerar cada vez mais custos associados à sua manutenção e continuidade de uso (WEIDERMAN, et al. 1997), tornando-se uma responsabilidade cara, pouco produtiva e de valor decrescente. Os problemas mais sérios costumam ficar em torno de custos de manutenção de *software* e *hardware* obsoletos nos quais costumam executar, dos grandes esforços de integração devido à falta de interfaces definidas, e do tempo demandado na detecção de avarias devido à falta de documentação e, consequentemente, da dificuldade de compreensão de seu funcionamento interno (BISBAL, et al. 1999). Um outro grande problema é a constante inserção de novas regras de negócio e a falta de atualização da documentação (ARAÚJO, 2009). Isso, mais na frente, torna bem mais difícil qualquer tentativa de solução em cima de um sistema problemático. Quando os sistemas são pequenos e cobrem somente um pequeno número de atividades de uma organização, é possível considerar a reformulação e substituição de um sistema ou subsistema que não mais satisfaz as necessidades da empresa. Mas os sistemas que crescem e cobrem diversas atividades tornam-se investimentos substanciais cuja substituição é mais difícil. Portanto, um grande problema é sobre como criar softwares que irão alavancar a construção de outros no futuro (WEIDERMAN, et al. 1997). Em outras palavras, como criar softwares flexíveis com a capacidade de integração e de fácil manutenção?

Em geral softwares desenvolvidos sem um método, planejamento, documentação e em prazos curtos costumam gerar problemas de manutenção e expansão. Ao se depararem com a necessidade de modificação em softwares em situações críticas, é comum que gestores tomem decisões para que sejam adotadas soluções rápidas, mas estas soluções costumam alimentar uma bola de neve de problemas que se torna cada vez maior, principalmente quando se fala em

integração com novas funcionalidades ou outros produtos. A medida que novas necessidades são identificadas, percebe-se que será difícil atende-las com qualidade devido à realidade caótica na qual o *software* se encontra.

Simplesmente criar interfaces para sistemas legados não costuma resolver problemas, uma vez que tal atividade em geral não considera problemas de sobrecarga, incapacidade de evoluir para fornecer novas funcionalidades e os excessivamente altos custo de manutenção (BISBAL, et al. 1999). Alguns dos problemas citados costumam influenciar negativamente também na usabilidade do produto. Um exemplo disso é a falta de lógica ou uniformidade entre os roteiros de uso das funcionalidades, devido à partes do programa que foram desenvolvidas por pessoas ou em momentos diferentes. Além disso existem outros problemas relacionados à usabilidade e interface.

2.1.2 Usabilidade

Do ponto de vista de Engenharia de Software, sistemas devem ser construídos para que sejam internamente de fácil manutenção, livres de erros, robustos, eficientes e seguros, além de considerar a minimização de custos. Já do ponto de vista da área de IHC (Interação Humano Computador), sistemas devem ser construídos para que ofereçam qualidade de uso (BARBOSA e SILVA, 2010). Essa divergência de objetivos desmotiva a união dessas duas importantes perspectivas. Segundo Barbosa e Silva (2010), pouca ou nenhuma atenção é dada ao que fica fora do sistema e a como ele será usado durante seu desenvolvimento, o que resulta em baixa qualidade de usabilidade. Talvez essa seja a principal fonte de problemas.

O mal uso de analogias em definições de interfaces gráficas costuma ser problemático para os usuários. Um sistema deve ser capaz de comunicar ao usuário o que ele é capaz de fazer e como se faz, pois a falta de clareza sobre o efeito de suas ações pode reduzir a confiança do usuário (BARBOSA e SILVA, 2010). O conceito de comunicabilidade está relacionado com a capacidade da interface transmitir ao usuário as intenções do *designer*, assim como a lógica de uso da aplicação (DE SOUZA e LEITÃO, 2009). As técnicas de IHC visam atingir esta e outras características pois, interfaces com baixa qualidade de uso aumentam a necessidade de treinamento e causam confusão aos usuários, induzindo-os a erros

e gerando insatisfação, o que baixa sua produtividade (PRATES e BARBOSA, 2003).

Partindo de observações de casos reais, problemas de interface gráfica e usabilidade podem ser facilmente identificadas. A utilização de linguagem inadequada nas interfaces dos sistemas é percebida quando se encontra, por exemplo, a expressão "Descrição Processo" em vez de "Descrição do Processo". Tal característica costuma ser obtida a partir da tradução de funcionalidades semelhantes em programas estrangeiros, quando o designer tenta, sem sucesso, abreviar um texto para que este se encaixe em uma tela já sobrecarregada de informações, ou mesmo por vícios dos desenvolvedores em utilizar expressões usadas em código fonte, estruturas de dados e nomes de colunas de uma tabela no bancos de dados.

Práticas comuns também podem trazer outros tipos de problemas. Modificações contínuas no *software* costumam resultar na adição de novos elementos onde existe espaço disponível na tela, sem considerar o fluxo lógico de manipulação para realização de uma tarefa específica. Também é muito comum não existir documentação do usuário e nem sua atualização após modificações nos casos de uso. Além disso, protótipos são mais usados como base para os programadores do que como parte de um método focado em atender às necessidades dos usuários durante validações.

Resolver um problema de design está relacionado com a questão "Como melhorar a situação?", situações essas que podem ser desde uma condição de trabalho sem uma intervenção em *software* ou uma intervenção existente, que em seu estado original atenda ou não as necessidades dos usuários (BARBOSA e SILVA, 2010). Talvez a maioria dos problemas de design sejam originados pela não utilização de métodos de IHC na definição de interfaces e pela ausência de um profissional especializado no assunto como integrante do time de desenvolvimento, pois é muito comum que os próprios desenvolvedores realizem as atividades de design. Isso pode resultar, na melhor das hipóteses, em produtos sem características estéticas adequadas.

2.2 Problemas Específicos

Obviamente o projeto 1P1V não compõe um sistema de grande porte e nem é tão antigo quando comparado a sistemas corporativos de 10 ou até 20 anos, mas também está sujeito a vários dos mesmos problemas de usabilidade e integração, pois seu histórico a respeito de tempo e evolução é bastante semelhante ao de sistemas legados, porém em menor escala. Para mostrar os problemas específicos relacionados ao projeto e seus trabalhos realizados, cada um deles será apresentado a seguir.

Lima (2007) em sua tese, defendeu um estudo sobre descoberta automática de conhecimento em interpretações musicais, no caso do acompanhamento rítmico ao violão. O objetivo era obter novos conhecimentos a respeito de características das batidas típicas da bossa nova, chamadas aqui de padrões rítmicos, relacionadas aos intérpretes. Em seu trabalho, houve a preocupação em construir uma interface gráfica, pois seus usuários seriam mais musicólogos que cientistas da computação. Desenvolvido em Java, para que o programa final da pesquisa seja executado, é necessária uma configuração de expansão da memória utilizada pela Java Virtual Machine (JVM), pois é comum que esta se esgote ao carregar a interface gráfica, que é composta por duas telas principais. A primeira, como mostra a **Figura 1**, é uma janela onde é exibido o resultado da transformação do arquivo MIDI de origem em padrões rítmicos, representados por símbolos textuais, que representam as articulações realizadas pelo intérprete em cada corda do violão. A segunda é um grafo de equipolência, como mostra a **Figura 2**, onde os nós representam os padrões rítmicos encontrados e as arestas indicam que existe similaridade entre os nós. O peso da aresta indica o grau de similaridade entre os padrões conectados. O conteúdo em texto, rotulado nos nós do grafo de equipolência, contém os índices iniciais e finais do trecho considerado padrão dentro do conteúdo total da transformação e sua quantidade de ocorrências na obra. Essa visualização, apesar de eficiente, pode se tornar muito confusa dependendo da quantidade e complexidade das relações de similaridade entre os padrões. Além disso, o nó não mostra o conteúdo do padrão em si, apenas o local de ocorrência.

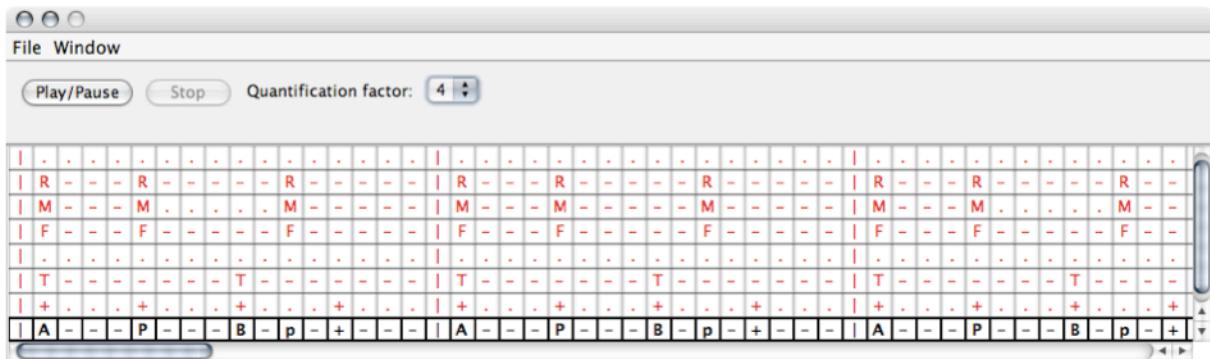


Figura 1 Articulações realizadas pelo intérprete na interface do trabalho de Lima. As seis primeiras linhas representam as cordas do violão, enquanto as duas últimas linhas indicam respectivamente as marcações de tempo e a junção de todas as cordas em uma representação. Retirado de (LIMA, 2007).

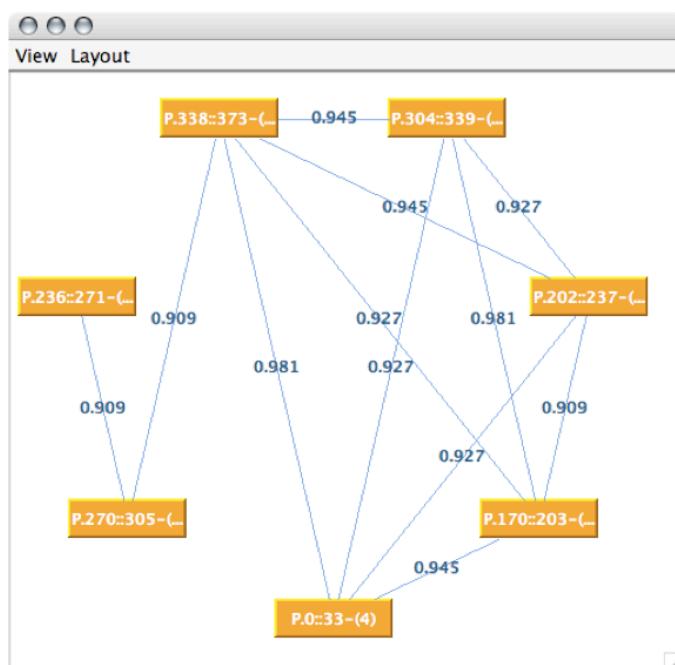


Figura 2 Grafo de equipolência como resultado da análise de similaridade entre os padrões rítmicos. Retirado de (LIMA, 2007).

Scholz (2008) desenvolveu um processo, denominado COCHONUT (Complex Chords Nutting), que é capaz de detectar acordes em sequências midi capturadas a partir de um violão MIDI e, em situações ruidosas onde a detecção não foi possível, fazer uma análise harmônica e deduzir acordes, baseado em regras que consideram o contexto de tal evento. Neste trabalho não houve a oportunidade de oferecer aos usuários uma interface gráfica que os possibilitassem sua utilização, sendo necessária a escrita e execução de um programa roteiro para a obtenção de seus resultados.

Silvestre (2009) realizou um trabalho sobre descoberta automática de conhecimento em interpretações musicais, abordando microrrítmo e microdinâmica. Holanda (2010) complementou esse estudo, que responde questões a respeito de padrões em pequenos desvios de tempo (antecipações ou atrasos) e dinâmica (intensidade) nos ataques realizados por um intérprete ao executar padrões rítmicos. A forma de utilização desses trabalhos, assim como o formato dos dados e expressão de seus resultados, são bastante semelhantes mas, apesar de possuir interface gráfica, suas funcionalidades estão ainda parcialmente implementadas, pois ainda é necessária alguma manipulação dos resultados exibidos (**Figura 3**) para que sejam obtidas suas representações gráficas (**Figura 4**) e conclusões a respeito do material processado. Os resultados das análises, como apresentados no documento de Dissertação, só podem ser obtidos após o processamento do material com a utilização de uma outra ferramenta, como as planilhas eletrônicas, capaz de gerar gráficos com os dados fornecidos.

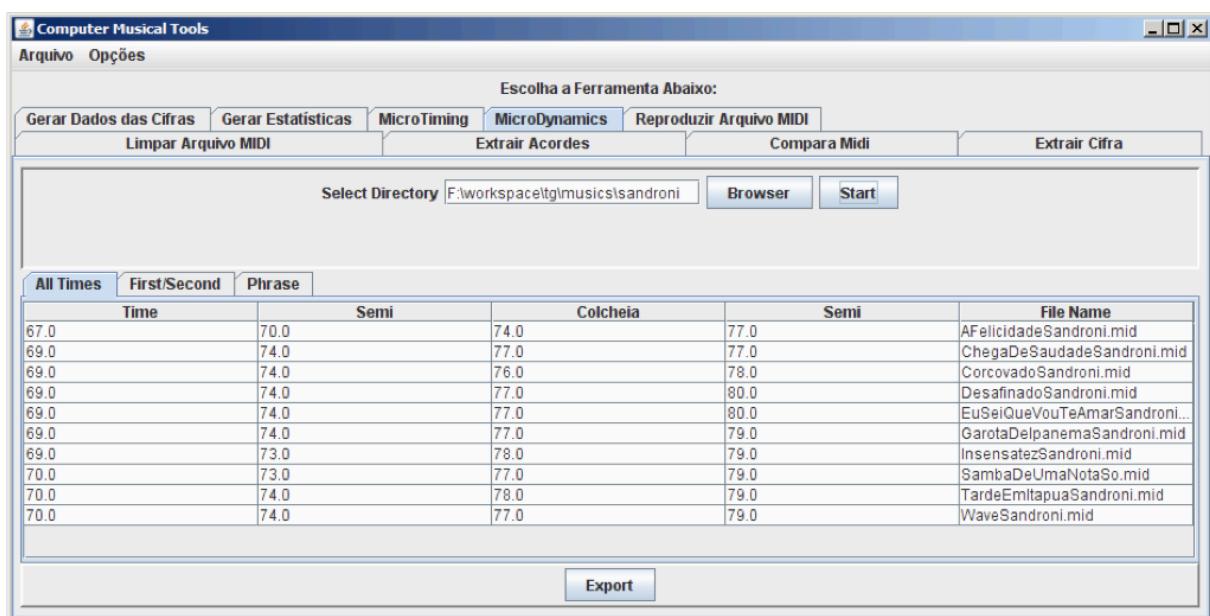


Figura 3 Interface do trabalho estudo em microdinâmica retirado de (HOLANDA, 2010)

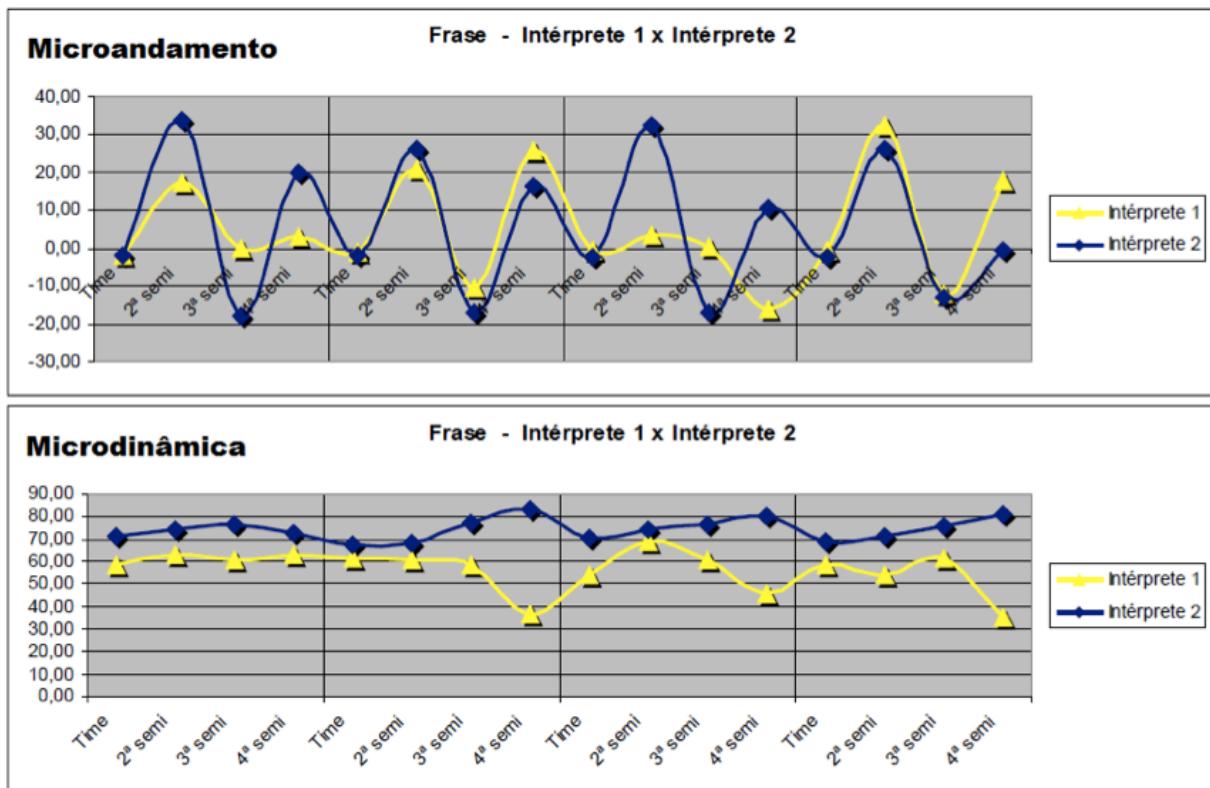


Figura 4 Gráfico gerado por um software de planilha eletrônica utilizando os dados fornecidos pelo processamento dos trabalhos de microdinâmica e microrritmo, retirado de (HOLANDA, 2010).

O trabalho de graduação de Sena, em 2008, criou formas de utilização com interface gráfica para funcionalidades sem tal recurso - como o caso da limpeza de arquivos necessária antes do processamento de trabalhos como os de Lima e Scholz, que remove ruídos capturados durante a gravação do material - e centralizou a utilização de todos os trabalhos citados em um ambiente que torna possível o uso prático por meio de uma interface gráfica. Também foram realizadas correções quanto a conflitos em nomes de classes e duplicação de código, e foi utilizado o padrão de projeto MVC (Model View Controller) para melhorar a qualidade do código fonte, separando código de interface gráfica do código funcional.

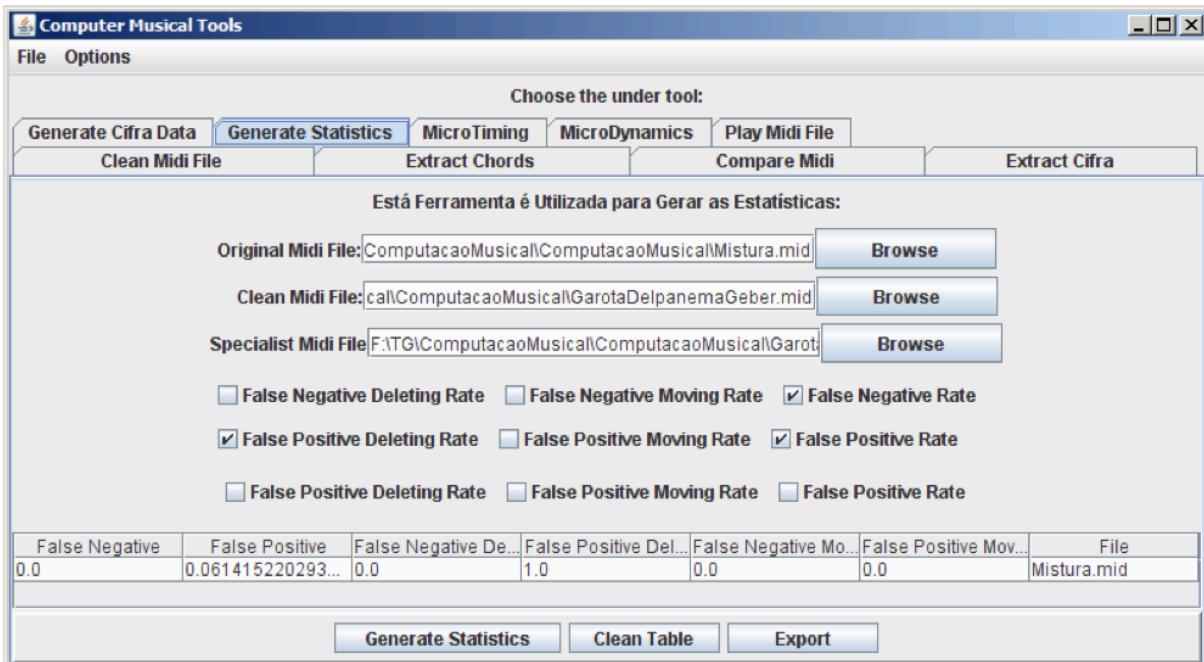


Figura 5 Interface do trabalho de integração do projeto 1P1V. Retirado de (SENA, 2008)

Um problema é que o resultado do trabalho de integração é, de forma simplificada, a soma dos pacotes de todos os trabalhos produzidos individualmente. A tendência é que este se torne um grande projeto a medida que novas necessidades venham a surgir, mas grandes projetos costumam estar mais sujeitos a problemas de manutenção. Essas novas necessidades podem ser desenvolvidas por novos colaboradores com estilos de programação diferentes e em outras localizações geográficas. O avanço das comunicações através da internet e o sucesso de grandes projetos *open source* tem motivado o desenvolvimento distribuído de software (SOARES, et al. 2007) e, apesar da existência de ferramentas de controle de versão que auxiliam o trabalho simultâneo e distribuído de desenvolvedores, ainda é comum que ocorram conflitos em arquivos modificados que precisam ser resolvidos. Dessa forma, a manutenção de um grande projeto com possíveis colaboradores geograficamente distribuídos pode se tornar caótica.

Uma realidade em torno dos trabalhos do projeto 1P1V é que eles foram realizados em momentos distintos e concluídos um a cada ano entre 2007 e 2010. Dessa forma não foram consideradas nem a integração desses trabalhos e nem as relações entre cada um dos focos de estudo¹.

¹ Para fins deste trabalho, **focos de estudo** são os temas estudados pelos trabalhos dentro do projeto. Microdinâmica é um exemplo de foco de estudo.

² **Marcadores**, também conhecidos como **tags**, são estruturas que definem e delimitam os dados

Quanto a integração, pode-se dizer que alguns padrões de projeto foram utilizados, mas isso não significa que foram seguidos os mesmo padrões. Essa diversidade de identidades estruturais, assim como os vários estilos de programação, dificultou a integração desses trabalhos em um padrão global predominante. Quando vários subsistemas seguem padrões semelhantes ou possuem características similares é mais fácil prever o comportamento de um subsistema desconhecido a partir do conhecimento prévio de um outro, tornando o entendimento do código fonte mais fácil e, com isso, acelerando sua manutenção. No modelo atual, a compreensão do código fonte por qualquer desenvolvedor que realizasse algum tipo de manutenção no ambiente seria bastante custosa.

Quanto a cooperação entre os trabalhos, existe uma situação que pode ser melhorada. Apesar deles estarem centralizados em um único ambiente de onde os usuários podem acessar todas as suas funcionalidade, esses trabalhos permanecem isolados uns dos outros, como se não fizessem parte do mesmo projeto, que poderia oferecer análises mais complexas caso os resultados de cada trabalho fossem analisados conjuntamente. Cada foco de estudo se concentra apenas em suas análises, mas não consideram encontrar relações entre suas informações e informações dos outros trabalhos para produzirem novos resultados. Dessa forma, as possibilidades de resultados e conclusões que o projeto pode oferecer dependem unicamente do que cada trabalho é capaz de produzir isoladamente, o que não maximiza o potencial do projeto.

Em termos de usabilidade, pode-se dizer que as formas de utilização de suas funcionalidades não foram projetadas com foco no público-alvo por meio de pesquisa, tornando seus roteiros inadequados. A medida que o projeto cresce, sua interface se torna cada vez mais carregada pois, para cada foco de estudo é criada pelo menos uma aba dedicada na janela principal do ambiente. Sendo assim, localizar uma funcionalidade nessa forma de visualização pode ser confuso, pois existem muitas palavras aglomeradas em sua interface. Como cada cenário de utilização de suas ferramentas se concentra basicamente em uma única aba, esta deve conter todos os componentes de entrada de dados necessários para sua completa realização, o que torna o conteúdo de algumas dessas abas demasiadamente carregado de componentes. Isso porque os roteiros de utilização não se fragmentam em etapas. Quando o roteiro de utilização de uma funcionalidade é dividido em etapas, estas podem ser individualmente mais simples

do que uma única grande etapa. Um exemplo disso seria a execução de um roteiro onde é necessário informar arquivos de entrada, que poderiam ser selecionados em uma caixa de diálogo, e diversos outros parâmetros definidos em uma segunda caixa de diálogo. Por fim, os resultados obtidos em cada foco de estudo são apresentados apenas em sua forma mais elementar e com poucos recursos gráficos, o que torna as atividades de leitura e interpretação mais cansativas.

Dessa forma ficam identificados os seguintes problemas:

- Não maximização do potencial do projeto musicológico quanto às possibilidades de visualizações, pois os resultados de análises de diferentes focos de estudo ficam isolados uns dos outros, quando poderiam estar juntos para permitir a descoberta de situações comuns;
- Roteiros de execução das funcionalidades não direcionados à um público-alvo específico e não discutida com algum representante grupo, o que resulta em um ambiente de difícil aprendizagem, telas demasiadamente carregadas quanto a quantidade de componentes apresentados e representação das informações ainda não amadurecida;
- Integração estrutural dos programas dos focos de estudo e expansão do projeto baseada em uma estratégia aditiva, que dificulta o desenvolvimento descentralizado de novos trabalhos por tornar a manutenção complexa, sujeita a conflitos e com forte necessidade de compreensão total dos códigos fonte escritos.

3 ESTADO DA ARTE

Este capítulo apresenta as técnicas, estratégias, abordagens e soluções para os problemas em questão, dentro dos temas envolvidos, que serviram de base referencial para o desenvolvimento deste trabalho. Devido a vasta diversidade de conceitos, estes serão apenas brevemente introduzidos, ficando qualquer detalhamento necessário para ser apresentado quando apropriado ao longo deste documento.

3.1 Engenharia de Software para Integração e Evolução de Software

Existem várias formas de integração de software, que normalmente envolvem o compartilhamento de dados, modificação do produto ou algum tipo de adaptação. Esta seção se dedica a apresentar uma análise crítica a respeito das possíveis formas de integração com o objetivo de identificar soluções adequadas para os problemas estruturais citados no capítulo 2.

Os critérios adotados para a realização da análise aqui apresentada refletem as necessidades do projeto quanto a: possíveis formas de troca de dados; o nível de adaptação necessário, sejam pequenas ou grandes mudanças; e o tipo de estruturação adequada para a integração.

3.1.1 Integração com Troca de Dados

Segundo Reynolds (2002), aplicações precisam trocar dados para se integrarem. Essa troca pode ser feita através do compartilhamento de arquivos em algum formato padrão ou mesmo através de bancos de dados compartilhados (VIGDER et al., 1996).

No contexto do projeto, os trabalhos dos focos de estudo são independentes entre si e por isso não precisam trocar dados uns com os outros diretamente, apenas com a aplicação principal do ambiente, que exerce o papel de integrador desses conteúdos de origens diferentes. Nenhum desses trabalhos lida com uma quantidade relevante de informações que justifique a utilização de um Sistema Gerenciador de Bancos de Dados (SGBD), mas os dados resultantes de suas

análises podem ser úteis também quando analisados por ferramentas externas que utilizam aprendizagem de máquina para mineração de dados, como o WEKA (Hall, et al. 2009).

Uma forma antiga e muito comum de troca de dados, e até hoje em prática, é a utilização de arquivos de exportação. Quando um programa precisa exportar dados para um outro, ele cria uma arquivo que contém uma massa de dados com campos delimitados por índices de início e fim. Dessa forma o programa que fará a leitura do arquivo sabe como localizar cada dado. Uma vez que o arquivo não armazena informações sobre como localizar cada dado, os programas se tornam muito dependentes do tamanho e ordem dos dados no arquivo e, por isso, é comum que *scripts* ou pequenos programas isolados da aplicação principal realizem as tarefas de exportação e importação, pois dessa forma é mais fácil manter a aplicação principal livre de possíveis modificações. Esse método é muito utilizado como tarefas que precisam ser executadas automática e periodicamente sem intervenção humana para compartilhar grandes quantidades de dados cuja disponibilidade imediata não é essencial, pois os dados mais recentes só estariam disponíveis a partir da próxima execução da tarefa.

Arquivos Comma-Separated Values (CSV), como uma implementação particular de arquivos de texto, dão uma certa flexibilidade ao formato dos arquivos utilizados para importação e exportação de dados. CSV é um formato bastante simples e utilizado em diversas plataformas para exportação e importação de dados em planilhas eletrônicas (SHAFRANOVICH, 2005) e SGBDs. Sua estrutura é capaz de armazenar dados tabelados onde os campos são separados por vírgulas e linhas são separadas quebras de linhas, excluindo a necessidade de fixar os limites iniciais e finais dos dados.

Tanto para troca e armazenamento de dados como para manter configurações de aplicações, Extensible Markup Language (XML) tem sido cada vez mais reconhecida e utilizada. Em 1997 a W3 Consortium (W3C) publicou a versão 1.0 do XML, um padrão internacional, não proprietário e de código aberto, com as vantagens de que seus marcadores² são definidos pelo autor do documento, seu formato ser legível por pessoas (ALMEIDA, 2002) e também ser muito útil para

² **Marcadores**, também conhecidos como **tags**, são estruturas que definem e delimitam os dados contidos em arquivos HTML, XML e semelhantes.

armazenar dados (REYNOLDS et al., 2002; W3SCHOOLS, 2011). Exemplos de conteúdo de arquivos simples, CSV e XML são mostrados na **Figura 6**.

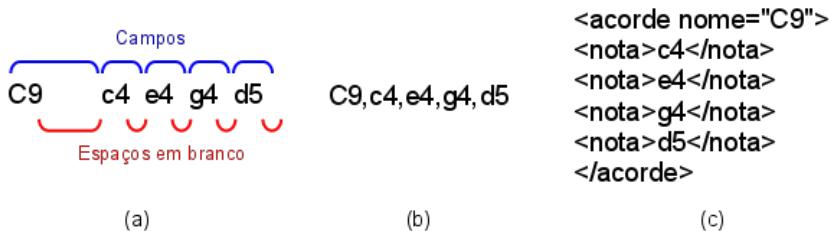


Figura 6 Comparação entre formatos de dados. (a) Arquivo com campos mapeados por índices (b) Formato CSV com campos separados por vírgulas (c) Formato XML.

No caso de uma aplicação precisar mudar a estrutura de seus dados exportados, é provável que a aplicação que importará esses dados precise ser modificada ou que os dados sejam convertidos em um formato reconhecido. Muitos arquivos podem ser convertido para outros formatos semelhantes, mas utilizar XML minimiza esse esforço, pois existe um mecanismo para essa tarefa. Um arquivo XML pode ser transformado em qualquer outra coisa, inclusive outros formatos de XML, através de uma transformação XSLT (eXtensible Stylesheet Language Transformations), que é um formato específico de XML (Holman 2000) que contém regras para modificação estrutural do arquivo de entrada original XML. Dessa forma uma aplicação não precisa ser modificada quando os dados fornecidos por outros serviços são modificados, bastando apenas modificar o arquivo XSLT.

Apesar das vantagens de padronização e flexibilidade, a manipulação de dados em XML pode ser complexa para situações simples, e sua representação pode fazer com que um arquivo seja grande mesmo para carregar pouco conteúdo. CSV está na situação inversa, pois é um formato bem mais simples e leve, mas não tão flexível, além de estar sujeito a incompatibilidades devido às várias definições que podem ser encontradas.

JavaScript Object Notation (JSON) surge como uma alternativa leve para serialização e troca de dados estruturados (JSON, 2011) no lugar de XML. Sua notação é legível por humanos, de fácil análise por softwares e independente de linguagem (CROCKFORD, 2006). O problema é que JSON é utilizado em aplicações web e isso não faz sentido no contexto de um projeto que não se trata de uma aplicação web.

Uma combinação que pode ser considerada como razoável para a troca de dados seria a utilização de XML definido em um nível de detalhamento moderado com um mecanismo de conversão para outros formatos por meio de uma transformação XSLT. O detalhamento moderado pode ajudar a simplificar a manipulação dos dados, enquanto o formato para o qual seria transformado pode ser o CSV. Esse método pode ainda maximizar as possibilidades de troca de dados com ferramentas externas. Apesar do domínio mais especializado de formatos como MusicXML³, sua utilização pode ser arriscada, uma vez que a representação de outros tipos de informações pode ser necessária em futuros estudos ainda não previstos.

3.1.2 Evolução de Legados e Mudanças em Software

A Engenharia do Software - disciplina da engenharia que se ocupa de todos os aspectos da produção de software (SOMMERVILLE, 2004) - trata de assuntos como mudanças em sistemas que podem ser feitas em vários níveis dependendo de cada caso. Muitos estudos sobre evolução de *software* vêm de sistemas legados, pois lidar com esse tipo de sistema é uma situação comum para desenvolvedores. No caso do projeto em questão, existe a necessidade de integrar trabalhos já implementados em um ambiente de *software* completamente novo com a capacidade de suportar futuras integrações, o que implica adaptações desejavelmente simples e um desenvolvimento completo.

Para Comella-Dorda e colegas (2000), evolução de *software* pode ser feita de três formas: Manutenção, onde são feitas pequenas melhorias e correções de bugs, mas nunca grandes mudanças estruturais; Substituição, que é usada em sistemas legados que não conseguem acompanhar a evolução dos negócios e nem podem ser modernizados nem estendidos, ou não são documentados; Modernização, que envolve mudanças estruturais mais significativas ou novos elementos, mas preserva boa parte do sistema atual.

Segundo Sommerville (2004), mudanças em *softwares* podem ser por: Manutenção, onde apenas pequenas correções pontuais ou novos requisitos são considerados, mas sem modificações estruturais; Transformação de arquitetura,

³ Music XML é um formato específico de XML para representar informações musicais.

onde ocorrem mudanças mais radicais, normalmente por questões de custos de manutenção; Reengenharia, quando não são consideradas novas funcionalidades nem ocorrem grandes mudanças na arquitetura do *software*, mas podem ocorrer algumas alterações estruturais que reduzam sua complexidade. O termo evolução é associado à ganhos em questões de manutenção, novos requisitos e capacidades, além de melhorias quanto à possibilidades de integração. Para alguns autores a expressão reengenharia é muitas vezes sinônimo de migração, para outros, significa mudanças de ambientes (BISBAL, et al. 1999). Bisbal e colegas (1999) afirmam que essas divergências são causadas pela falta de padronização das terminologias. Para fins deste trabalho, a relação entre os níveis de mudanças e os conceitos foi interpretada conforme a **Figura 7**:

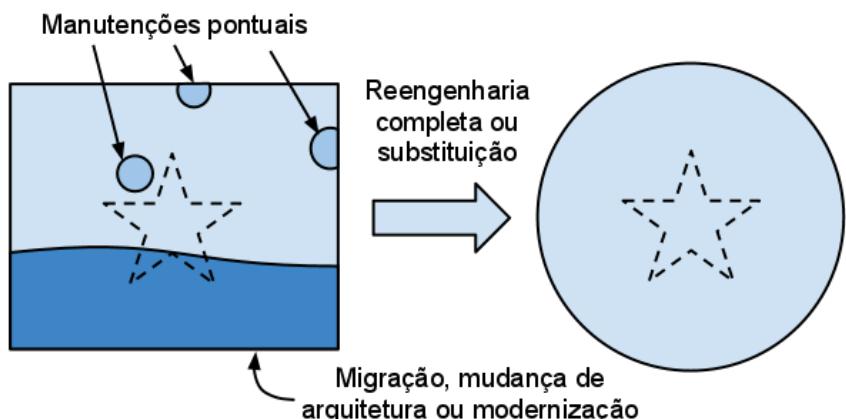


Figura 7 Comparação dos níveis de mudanças em software. Na situação retangular podem ser feitas pequenas correções ou alguma melhoria estrutural mais relevante, enquanto a situação circular indica uma implementação completamente nova. A estrela nas duas situações indica que a essência do software em termos de objetivo foi mantida.

Em situações em que não é possível uma reengenharia completa de um sistema, também são consideradas estratégias de evolução de arquitetura com *wrapping* (BISBAL, et al. 1999), que consiste em envolver uma porção de *software* e dados em novos componentes ou estruturas, para que outros elementos não precisem saber nada a respeito de sua implementação. Essa pode ser uma estratégia adequada para a adaptação de componentes em uma atividade de integração, como se vê na **Figura 8**.

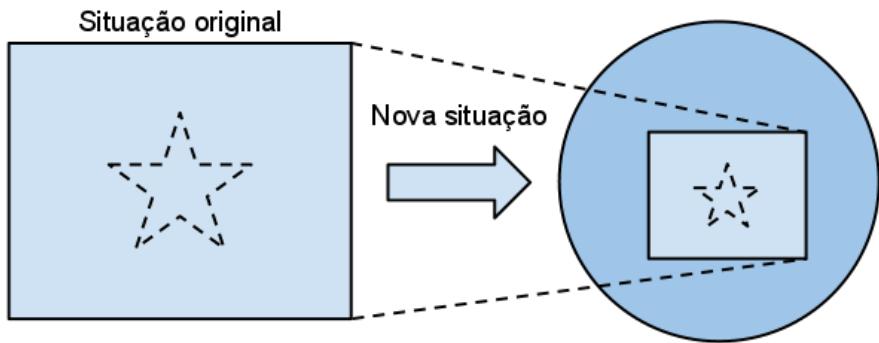


Figura 8 Técnica de wrapping para adaptar uma porção de software em uma nova situação utilizando sua estrutura original quando a implementação desta pode ser abstraída pelo restante dos elementos envolvidos.

É muito comum que os desenvolvedores não conheçam a fundo o produto a ser modificado. Bisbal e colegas (1999) apontam o Entendimento do Software como uma atividade necessária no processo de migração de sistemas legados. A compressão do software é fundamental para que as modificações sejam bem sucedidas, e são possíveis estratégias para esta tarefa (WEIDERMAN, et al. 1997): *Bottom-up*, que visa reconstruir o projeto de alto nível de um sistema começando pelo código fonte, através de uma série de vinculação de conceitos a fragmentos de programa. Técnicas de extração de arquitetura, por exemplo, devem começar pela análise do código-fonte (KAZMAN, 1997); *Top-down*, que busca, a partir de um prévia compreensão do sistema, marcar componentes individuais responsáveis por tarefas específicas; Refinamento Iterativo, que segue o roteiro de criar, verificar e modificar hipóteses até que o sistema inteiro seja explicado por um conjunto consistente de hipóteses; e Combinação, que explora pistas, oportunisticamente, de forma *top-down* e *bottom-up* a medida que se tornam disponíveis.

Uma combinação das estratégias *top-down* e *bottom-up* para o entendimento do software pode resultar numa atividade ágil e aprofundada tanto quanto necessário, numa tentativa de abstrair detalhes da implementação ao máximo com *top-down* e entender apenas o necessário para tornar um elemento compatível com uma nova situação com *bottom-up*. Essa estratégia combinada pode ser aplicada conjuntamente com a técnica de *wrapping*, que preserva ao máximo a estrutura do trabalho original.

3.1.3 Modelos de Arquiteturas

O Software Engineering Institutue (2011) define Arquitetura de Software como as definições dos elementos de *hardware* e *software* de um sistema e suas interações. É o elemento base para compreensão do sistema e por onde uma análise prévia acusa se a abordagem do projeto é aceitável, com objetivo de reduzir a probabilidade de insucesso (SEI, 2011). Vários modelos de arquitetura foram elaborados para priorizar determinadas características que variam de acordo com cada tipo de aplicação, mas é muito comum que sistemas utilizem algum tipo de combinação de diversos modelos de arquitetura (GARLAN e SHAWY, 1994). Para fins deste trabalho, como já citado, é importante considerar modelos que proporcionem suporte a integrações futuras e troca de dados.

O modelo de repositórios centraliza e compartilha uma coleção de dados entre os diversos componentes do sistema que atuam sobre esse conteúdo, e a interação entre esses componentes varia de acordo com cada projeto (SOMMERVILLE, 2004; GARLAN e SHAWY, 1994). Esse modelo pode utilizar, por exemplo, bancos de dados ou mesmo uma coleção de arquivos como sua massa de dados. Como dito anteriormente, o projeto não utiliza bancos de dados, mas um conjunto de arquivos para serem analisados. Mesmo utilizando uma adaptação desse modelo para centralizar esses arquivos, algo mais seria necessário para integração de novas necessidades, uma vez que, nesse modelo, a interação entre os elementos são específicas para cada projeto.

Sistemas distribuídos são compostos por mais de um centro ativo colaborando entre si como uma única aplicação de maior escala (SILBERSCHATZ, GALVIN e GAGNE, 2000), e são capazes de realizar muito bem a tarefa de compartilhar recursos e trocar dados. O problema das arquiteturas distribuídas, neste caso, é que o projeto em questão funciona melhor em um ambiente centralizado onde a disponibilidade das aplicações integradas é garantida, e por isso não são consideradas adequadas para este trabalho.

Uma arquitetura em camadas tem a qualidade de separar responsabilidades em níveis de abstração (CARVALHO, 2001), de forma que uma camada utiliza os serviço da camada abaixo e é utilizada pela camada acima. O modelo possui as vantagens de quebrar a complexidade de um problema grande, permite reuso e fácil substituição de componentes, além de que a manutenção interna em uma camada

não interfere nas outras. O desempenho pode ser prejudicado pela distância de chamadas de métodos entre as camadas de níveis mais extremos e o nível adequado de abstração também pode ser difícil de ser encontrado (GARLAN e SHAWY, 1994). Apesar das vantagens da divisão de responsabilidades e da substituição de componentes, camadas não possuem diretamente algum benefício em relação a integração de aplicações, embora possam ser base para organizar outros modelos de arquitetura que se dediquem mais a este mérito.

Uma arquitetura baseada em *plugins* é muito eficiente em adicionar novas funcionalidades a uma aplicação existente (VIGDER, GENTLEMAN e DEAN, 1996), pois permite que a aplicação principal (*host*) seja expandida por meio da adição de programas menores (*plugins*). *Plugins* não executam sozinhos, mas por requisição do *host* conforme necessário, e se comunicam com ele por meio de serviços disponibilizados e definidos por interfaces (CHATLEY, EISENBACH e MAGEE, 2003), pois nada se sabe sobre os *plugins* em tempo de desenvolvimento e compilação do *host*. O modelo permite que sejam desenvolvidas e adicionadas novas funcionalidades sem precisar modificar e reconstruir a aplicação inteira, além de modularizar e reduzir a complexidade da aplicação (MAYER, MELZER e SCHWEIGGERT, 2003). Um problema com esse modelo é que *plugins* precisam ser escritos, e provavelmente isso não seria realizado por usuários finais, pois precisariam de conhecimento aprofundado em tecnologias de desenvolvimento. Ainda assim, este é um modelo bastante apropriado para integração de aplicações como exposto pelos problemas apresentados no capítulo 2.

Quando aplicações disponibilizam suas bibliotecas de classes e métodos de forma bem estruturadas, é possível a integração através da escrita de *scripts*, que são fragmentos de códigos executáveis que podem ser escritos pelos próprios usuários finais com o auxílio de sua equipe técnica, utilizando essas bibliotecas de forma a combinar suas funcionalidade conforme necessário (VIGDER, GENTLEMAN e DEAN, 1996). Uma desvantagem desse tipo de integração é que cada funcionalidade precisa de um *script* de integração dedicado, sendo necessária a escrita de vários *scripts* para uma integração mais abrangente ou total. Além disso o usuário se torna dependente de alguém capaz de realizar a codificação necessária. Já padrões e conceitos de Programação Orientada a Objetos (POO), como a herança, permitem que partes específicas de uma aplicação sejam especializadas quando se tem acesso aos binários de uma aplicação mas não aos fontes (VIGDER,

GENTLEMAN e DEAN, 1996). Esse tipo de recurso pode ser útil quando se deseja que o componente especializado utilize outros componentes durante sua execução, mas pode exigir um conhecimento técnico ainda mais avançado que a situação com *scripts* citada.

É provável que a maioria das soluções que proporcione um nível de integração satisfatório exija algum tipo de conhecimento técnico em algum momento da integração. A construção de *plugins* utilizando uma biblioteca comum que contenha um conjunto reduzido de classes e definições para um domínio específico pode ser uma decisão equilibrada de complexidade e benefícios, pois a aplicação teria as vantagens do modelo e forte relação com seus *plugins*, o que traria possibilidades interessantes em termos de funcionalidades gerais. Um outro ponto importante é que se aplicações usam estruturas e objetos compatíveis, como classes especializadas por meio de herança, elas podem mais facilmente trocar dados entre si. Se existir um apoio técnico para a tarefa de implementação dos *plugins* a partir das necessidades dos usuários, e esta é uma realidade do projeto 1P1V, a solução teria a vantagem de ser um ambiente dedicado e oferecer fácil utilização e instalação dos *plugins* quando implementados.

3.2 IHC para Concepção Focada no Usuário e Usabilidade

A área de Interação Homem-Máquina (IHC) é dedicada a conceber produtos focados nos usuários, considerando uma abordagem de fora para dentro ao desenvolver sistemas, ao contrário da engenharia de *software*, que costuma se dedicar aos aspectos internos técnicos e estruturais do *software* (BARBOSA e SILVA, 2010). Isto significa sempre considerar a realidade e contexto dos usuários, assim como a forma como eles interagem com a tecnologia, buscando entender suas necessidades e limitações (FERRE, et al., 2001).

Soluções de *design* estão relacionadas com a melhoria de situações por meio de uma intervenção, que pode ser com a introdução de um artefato artificial produzido com um objetivo (BARBOSA e SILVA, 2010). A produção de uma solução está relacionada com as atividades de design, que podem ser resumidas como (LAWSON, 2006; LOWGREN, 2004):

- Análise de uma situação;

- Síntese de uma intervenção;
- Avaliação da nova situação.

Apesar de alguns processos de *design* definirem a ordem exata das atividades de seus ciclos de vida, Lawson (2006) defende que não importa a ordem nem a quantidade de vezes que as atividades são executadas, bastando começar por um problema, executar as atividades do processo iterativamente e chegar a uma solução. Também é possível distribuir o tempo dedicado a cada atividade, afim de definir possíveis estratégias (KRUGER e CROSS, 2006): design dirigido pelo problema, que dedica mais tempo à análise da situação atual; e design dirigido pela solução, que dedica mais tempo à explorar possíveis intervenções. Segundo o experimento de Kruger e Cross (2006), a qualidade da solução produzida não está diretamente relacionada com a estratégia adotada.

Na situação de desenvolvimento deste trabalho, isto é, com uma equipe extremamente pequena, é importante considerar processos que envolvam as atividades mais relevantes, diretas e objetivas. Adquirir conhecimento a respeito dos usuários e suas relações com a tecnologia também é importante, pois cada realidade é única e precisa ser compreendida, principalmente numa área tão específica quanto a musicologia. Com o objetivo de identificar e adotar um método adequado, alguns dos processos disponíveis na literatura foram analisados como descritos a seguir e posteriormente comentados.

O ciclo de vida em estrela, proposto por Hix e Hartson (1993), define as seguintes atividades de processo (HIX e HARTSON, 1993):

- Análise de tarefas, usuários e funções, com o objetivo de estudar a situação atual dos usuários;
- Especificação de requisitos, onde são definidos os problemas a serem resolvidos;
- Projeto conceitual e especificação de design, onde uma solução é concebida;
- Prototipação, onde são criadas réplicas interativas da solução proposta para validação;
- Implementação, onde o produto idealizado é de fato desenvolvido;
- Avaliação, realizada continuamente após a realização de cada uma das atividades.

O ciclo de vida da engenharia de usabilidade de Nielsen (1993) é composto pelas seguintes atividades:

- Conhecer o usuário, para estudar os usos pretendidos;
- Realizar uma análise competitiva, para conhecer os pontos positivos e negativos de produtos similares ou concorrentes ao que será desenvolvido;
- Definir as metas de usabilidade, para priorizar os fatores de qualidade do projeto;
- Fazer *designs* paralelos, para possibilitar a escolha entre várias alternativas de design;
- Adotar o *design* participativo, disponibilizando acesso a uma amostra representativa do população-alvo à equipe de *design*;
- Fazer o *design* coordenado, para evitar que existam inconsistências entre a interface projetada e o produto;
- Definir diretrizes e garantir que não sejam violadas;
- Fazer protótipos, utilizando estratégias para simplificar esses modelos;
- Realizar testes empíricos, para observar os usuários realizando tarefas;
- Praticar *design* iterativo.

Ao final, deve ser realizada uma coleta de dados para planejar a próxima versão e avaliar o retorno do investimento.

Mayhew (1999) também propôs um ciclo de vida para a engenharia de usabilidade, mas com uma visão mais integral dos fenômenos e atividades organizadas de forma a otimizar a solução. Nesse processo existem apenas três fases, mas que são detalhadas interiormente:

- Análise de requisitos, onde são aplicados os princípios de design de IHC para que sejam definidas as metas baseadas no perfil dos usuários, suas tarefas e as características da plataforma;
- Design, avaliação e desenvolvimento, que visa conceber a solução nos seguintes níveis de validação interativa:
 - Nível 1: aplicar reengenharia para alcançar os objetivos do usuário em um modelo conceitual, sempre verificando se as falhas foram resolvidas antes de passar para o próximo nível;

- Nível 2: aplicar padrões de design e construir protótipos de média fidelidade, verificando se as metas foram atingidas;
- Nível 3: projeto de alta fidelidade e implementação.
- Instalação, onde são coletadas opiniões dos usuários para verificar se as questões foram resolvidas ou para serem aplicadas em melhorias nas futuras versões.

O *design* contextual se concentra na imersão da equipe no contexto dos usuários, realizando um estudo profundo de suas necessidades e seus modos de trabalho. Este processo defende que os dados obtidos dos clientes é a base para decidir que necessidades serão abordadas, o que o sistema deve fazer e como deve ser estruturado (BEYER e HOLTZBLATT, 1999). Seu ciclo de vida é composto das seguintes atividades:

- Investigação contextual, que revela detalhes a respeito do trabalho e necessidades dos clientes e utiliza dados do cliente para tomada de decisões;
- Modelagem do trabalho, para descobrir e definir a estrutura do trabalho dos clientes e fornecer uma linguagem para que as equipes falem sobre o trabalho;
- Consolidação é quando se faz a síntese de dados qualitativos e dá significado ao trabalho realizado por cada perfil;
- Re-projeto de trabalho, está relacionado com a melhora das práticas de trabalho, com a integração das ideias de toda a equipe e a certificação de que os sistemas se encaixam no trabalho dos clientes;
- Projeto do ambiente do usuário, que foca a equipe no que o sistema faz e em sua estrutura, mantendo a coerência com o ponto de vista dos usuários;
- Prototipação e testes com clientes, focada em encontrar e corrigir erros antes de entrar em código.

O design baseado em cenários é um processo iterativo que utiliza estórias que simulam situações reais de pessoas realizando tarefas como base durante todas as atividades do processo de concepção de uma solução, onde são constantemente transformados (ROSSON e CARROLL, 2002). Esses cenários descrevem atores, metas e todas as suas interações, e podem ser elaborados como

protótipos (CARSOLL, 2000). Suas atividades são basicamente (BARBOSA e SILVA, 2010):

- Análise do problema, onde são criados os cenários dos problemas;
- Projeto da solução, onde são desenvolvidas ideias para a solução utilizando os cenários das atividades, do fluxo de informações e detalhes sobre a interação dos usuários com o sistema;
- Prototipação e avaliação.

O processo centrado na comunicação, que aparece complementando o *design* baseado em cenários, tem seu objetivo concentrado em produzir uma solução de alta comunicabilidade, ou seja, capaz de transmitir meta-informações do *designer* aos usuários por meio da interface, pois é baseado em engenharia semiótica que considera a interação como uma forma de comunicação (BARBOSA, et al., 2004). O processo considera as três atividades básicas do processo de *design* - análise, projeto e avaliação - e tem como diferenciais a utilização de dúvidas comuns dos usuários durante a interação ao longo de todo o processo e o fato de incentivar a construção dos projetos da interface e da interação separados (BARBOSA e SILVA, 2010).

O processo dirigido por objetivos tem o diferencial de explorar tecnologias já disponíveis para ajudar os usuários a atingirem seus objetivos por meio de ações e tarefas (COOPER, REIMANN e CRONIN, 2007), mas abre pouco espaço para soluções criativas, pois não foca em explorar novas possibilidades. A princípio esta abordagem não se enquadra na proposta deste trabalho pois a utilização de soluções disponíveis pode implicar uma maior necessidade de algum tipo de integração, o que já é um problema abordado neste trabalho. Além disso, novas ideias são fundamentais.

Os processos analisados, de um modo geral, apresentam uma lógica bastante semelhante entre si inclusive com uma série de atividades em comum. As diferenças ficam basicamente em torno de algum elemento ou atividade específica que recebe maior foco. Como praticamente todos os processos analisados envolvem algum tipo de análise do problema ou compreensão da situação, além de alguma atividade relacionada à prototipação, a decisão fica por conta de alguns pontos positivos ou negativos.

Os processos propostos pela engenharia de usabilidade são bastante completos e, por isso, algumas de suas atividades podem se tornar inviáveis para equipes pequenas, como a realização de design paralelo. Isso também é verdade para uma imersão completa da equipe de desenvolvimento no contexto do usuário como propõe o *design* contextual. Contudo, as técnicas de simplificação de protótipos e priorização dos fatores de qualidade propostos pela engenharia de usabilidade podem ser adaptadas a outros métodos. Já uma abordagem centrada na comunicação é bastante adequada para a construção de produtos autoexplicativos, mas este tipo de objetivo também pode ser atingido com um grau de aceitação razoável em outros processos.

Apesar de cenários serem muito focados em necessidades específicas e, com isso, proporcionar objetividade, um método menos específico pode abrir mais espaço para novas ideias. O processo em estrela definido por Hix e Hartson se mostrou como uma síntese mais equilibrada das principais atividades de processos de IHC, no sentido de ser algo entre completo e objetivo. Atividades de fácil adaptação, como a prototipação simplificada, e interfaces autoexplicativas do processo centrado na comunicação poderiam ainda ser consideradas conforme necessário.

4 CONCEPÇÃO E ANÁLISE

Este capítulo tem o objetivo de descrever o processo de concepção de uma solução em *software* para os problemas do projeto 1P1V mencionados no capítulo 2, assim como os métodos utilizados ao longo de cada etapa da metodologia adotada visando criar um produto adequado aos usuários e fins da pesquisa. Este capítulo está dividido em duas etapas: a concepção de um produto para o usuário, isto é, quanto a sua interface, usabilidade e funcionalidades; e a concepção de uma nova arquitetura de *software* e sua estrutura interna de funcionamento. Ao final do capítulo, estará definido o conjunto de requisitos para o desenvolvimento do *software* proposto.

4.1 Concepção do Produto

O processo de *design*, em geral, visa elaborar soluções para situações que podem melhorar, e é semelhante ao processo de pesquisa científica, onde a interpretação da situação atual é equivalente à construção de uma hipótese, as propostas de soluções representam a experimentação, e a avaliação é uma atividade fundamental (BARBOSA e SILVA, 2010). Essas situações podem ser desde uma realidade de trabalho para a qual será criado um *software* até um produto atualmente em uso mas que precisa ser melhorado. No caso deste trabalho, a solução se refere a um *software* para melhorar as condições de usabilidade e possibilidades do *software* atual do projeto 1P1V.

A necessidade de compreender melhor uma situação para a elaboração de novas ideias a partir de uma análise interpretativa de dados, além do fato do público-alvo ser tão restrito, levou à decisão de utilizar métodos qualitativos de pesquisa (BOOTH, et al., 2004), pois estes são adequados às características citadas que refletem a realidade deste trabalho. A pesquisa se mostra importante pois existem problemas de integração e usabilidade a serem resolvidos mas não existem as informações necessárias para a produção da solução. O objetivo é elaborar um produto de usabilidade adequada a seus usuários, que são pessoas da área musical. Como base para todo o processo, foram utilizadas as seguintes questões de pesquisa:

- **O que é importante e se deseja saber a respeito de um material musical?** O desenvolvimento desta pergunta visa capturar ideias a respeito do que poderia ser considerado novas funcionalidades do produto e informações que ele seria capaz de fornecer. Em outras palavras, está ligada a possibilidades de requisitos funcionais.
- **Como se deseja que seja a manipulação do software para que se chegue à resposta?** Esta questão está relacionada às futuras possibilidades da solução em termos de usabilidade, casos de uso e em como os usuários utilizariam o produto para que ele fornecesse o resultado desejado.
- **Como devem ser representadas e fornecidas essas respostas?** A questão diz respeito a como os usuários gostariam que fossem representadas as informações fornecidas pelo produto como resultados das análises realizadas após sua manipulação.

O produto concebido após a aplicação do método descrito neste capítulo foi batizado como *Bleem Bloom*, inspirado no título da canção “Bim Bom”, obra de João Gilberto, e na palavra *bloom*, que em inglês significa lupa, transmitindo a ideia de observar a música em detalhes.

4.1.1 Plano de Pesquisa

Com o objetivo de entender as necessidades, a forma de uso e objetivos de um determinado perfil de pessoas, no contexto da musicologia, foi elaborado um plano de pesquisa simplificado, e aqui apresentado. Cada etapa deste plano será discutida em detalhes em suas respectivas seções.

1. **Definição de Público Alvo:** Partindo do pré-suposto de que é melhor atender eficientemente à um determinado grupo de pessoas do que tentar atender à todos de forma superficial e, muitas vezes, não ser satisfatório, tal definição é fundamental para que o produto reflita diretamente as necessidades desse perfil de consumo. Nesse ponto será definido o tipo de usuário ao qual se destina o produto.
2. **Coleta de Dados:** A coleta de dados representa o ato de aquisição de conteúdo dos usuários que foram definidos como público alvo, considerando o projeto em questão, para que seja possível interpretar a situação. Nessa

etapa serão detalhados: O método de coleta, a amostra do público alvo da qual os dados serão coletados e os dados coletados propriamente ditos.

3. **Análise dos Dados:** Para que qualquer coisa possa ser feita a partir de dados coletados, estes precisam ser analisados. Organização, classificação e interpretação dos dados são essenciais para as etapas seguintes da pesquisa. Por fim, os dados serão trabalhados e associados à uma utilidade, tudo baseado nos objetivos e propósitos do projeto.
4. **Análise de Similares:** Nessa etapa serão observados produtos considerados similares ao software proposto, seja por objetivos, funcionalidades ou aparência, visando observar os pontos positivos e negativos de cada um deles. Essa visão permite conhecer um referencial de inspiração para elaboração de novas ideias.
5. **Prototipação e Validação:** Município de informações adquiridas após as etapas anteriores, constroem-se protótipos do produto para a realização de validações de usabilidade. O objetivo é realizar validações até que o modelo final seja definido.
6. **O Produto:** Nessa seção será apresentada a concepção do produto como um conjunto de requisitos e funcionalidades, que é a base para a construção do *software* proposto.

4.1.2 PÚBLICO-ALVO

Baseado no tema e motivação da pesquisa, nos objetivos do projeto e nos trabalhos já realizados, foi definido como público-alvo, primeiramente, musicólogos e pesquisadores interessados, podendo ser estendido à músicos e entusiastas curiosos. É importante ressaltar que esse conjunto de pessoas, mesmo munido do computador como ferramenta de auxílio à suas pesquisas, não necessariamente possui conhecimentos avançados em informática, devendo ser classificado apenas como um grupo de usuários. Essa informação é importante no momento da definição dos roteiros de utilização das funcionalidades do produto, pois não se deve contar com grande perícia por parte dos usuários durante uso e configuração de softwares.

4.1.3 Coleta de Dados

O desenvolvimento de um produto inovador depende também de criatividade, mas principalmente do conhecimento sobre o público-alvo, por isso deve-se coletar dados a respeito do mesmo, com o objetivo de descobrir suas necessidades e preferências, assim como tendências do grupo ao qual se destina o produto. Os dados devem ser coletados de fontes confiáveis, relevantes e representativas, pois do contrário poderia prejudicar o trabalho (BARBOSA e SILVA, 2010).

Um possível problema sobre as fontes de dados é a baixa representatividade da amostra utilizada. Isso aconteceu na pesquisa aqui relatada, pois seu público-alvo é muito restrito e o número de representantes da classe disponíveis era pequeno. Por esta razão também foram utilizados os trabalhos realizados e suas documentações como uma espécie de dados secundários (HAIR JR, et al., 2010) para complementar a pesquisa, considerando que eles também surgiram de necessidades referentes ao projeto.

Para posterior análise, normalmente são feitos registros fonográficos das reuniões e entrevistas de coleta de dados, o que permite maior concentração do entrevistador nos entrevistados. Também é comum a utilização de mais de uma técnica de coleta, pois dessa forma é possível ter um resultado mais confiável. Infelizmente, devido à limitação de tempo e ao tamanho da equipe de pesquisa, só foram aplicadas entrevistas em profundidade apenas com os membros mais representativos (apenas 4), para que mais detalhes em maior profundidade pudessem ser descobertos considerando que seu roteiro flexível permite que as perguntas e tópicos sejam aprofundados ou modificados caso necessário (BARBOSA e SILVA, 2010), e grupos focais para os demais, pois esses estimulam novas ideias e permitem a observação da interação entre os membros do grupo (HAIR JR, et al., 2010), também com o objetivo de tentar equilibrar a representatividade do público-alvo. Pelos mesmos motivos, optou-se por realizar anotações escritas durante as entrevistas, pois gravações levariam muito tempo para serem analisadas por uma única pessoa. A amostra do público-alvo selecionada foi composta por professores, músicos e alunos do departamento de música da UFPE. Foram ouvidos um total de 12 colaboradores.

Os dados foram coletados com o auxílio de um roteiro de entrevista baseado em questões, onde cada etapa recebeu uma codificação identificadora que possui o

objetivo de relacionar os dados às etapas onde foram coletados. O roteiro de entrevista e suas questões servem apenas como um guia para ajudar o entrevistador a manter o foco nos objetivos da pesquisa, pois as questões podem ser aprofundadas conforme as necessidades de cada entrevistado. Uma possível forma de aprofundamento da questão da etapa E3b (abaixo) sobre como manipular o *software*, por exemplo, é a aplicação de questões a respeito da experiência do usuário em manipular componentes de interface gráfica específicos, como caixas de texto, botões, menus e tabelas. O roteiro de entrevista semiestruturada foi definido da seguinte maneira:

- **E1: Apresentação do projeto musicológico.** Objetiva fazer com que os entrevistados conheçam o projeto e suas motivações, além dos trabalhos sobre padrões rítmicos, harmonia, microrítmo e microdinâmica já realizados, para que possam elaborar ideias e opinar a respeito dos mesmos.
- **E2: Aplicar questão: "Em sua atuação profissional, que funcionalidades ou recursos computacionais auxiliam ou auxiliariam em seu trabalho?".** Essa é uma questão que foca em ampliar a visão das necessidades e ideias de cada tipo de usuário dentro da amostra do público alvo. De maneira resumida, o objetivo é entender a situação atual e saber o que é útil até o momento.
- **Aplicação das questões de pesquisa (E3).** As questões de pesquisa (citadas na seção 4.1) serão aplicadas aos entrevistados com a intenção de registrar as ideias que serão, mais tarde, analisadas e transformadas em requisitos. É importante notar que tais questões são apenas a base da entrevista, podendo ser reformuladas para um melhor entendimento por parte dos entrevistados ou detalhadas para buscar respostas mais específicas. Tais questões, citadas abaixo, são aplicadas sempre no contexto da musicologia e do projeto, mas podem ser generalizadas ou simplificadas dependendo da segurança do entrevistado no assunto.
- **E3a: O que é importante e se deseja saber a respeito de um material musical?**
- **E3b: Como se deseja que deva ser a manipulação do software para que se chegue à resposta?**
- **E3c: Como devem ser representadas e fornecidas essas respostas?**

Definidas as questões de pesquisa na seção 4.1, a entrevista em profundidade e os grupos focais como métodos de coleta de dados, a amostra de público-alvo na seção 4.1.2 e o roteiro de entrevista acima, foi possível capturar o que os possíveis futuros usuários tem a dizer. Cada anotação foi numerada como um Dado Coletado, abreviado para DC<Número> e podem ser encontradas no APÊNDICE A - Dados Coletados. O motivo da utilização desta codificação é a simplificação da identificação e associação de cada elemento nas próximas seções.

4.1.4 Análise dos Dados

Na primeira etapa da entrevista, que é a etapa de apresentação E1, foi observado que alguns entrevistados tiveram dificuldade em compreender a motivação do desenvolvimento de um *software* com os propósitos apresentados e, na etapa E2 ficou claro o motivo: O perfil desse subgrupo não é relacionado à musicologia.

Foi constatado que parte dos entrevistados possuem o computador como ferramenta de sala de aula, para auxílio ao ensino e treinamento prático. Esse subgrupo gerou anotações, principalmente na etapa E3a, que nem sempre possuíam relação direta com a proposta do projeto e, por isso, foram classificadas como **Indireta** quanto à sua utilidade, sendo apreciadas como fontes adicionais de inspiração para definição de elementos de usabilidade e interface gráfica. A outra porção do grupo tem o computador como ferramenta de pesquisa. Nesse caso, as anotações possuem relações diretas com o projeto e, por isso, foram classificadas como **Direta** quanto à sua utilidade. A **Tabela 1** mostra essas duas realidades em cada etapa da entrevista na qual cada dado foi coletado:

Etapa/Utilidade	Utilidade Direta	Utilidade Indireta
E2: Requisitos	DC01, DC02, DC03, DC04, DC05, DC28, DC36	DC06, DC07, DC08, DC20, DC24, DC32
E3a: Requisitos	DC09, DC10, DC11, DC37	DC21, DC25, DC29, DC33
E3b: Casos de Uso	DC12, DC13, DC14, DC38	DC22, DC26, DC30, DC34
E3c: Interface	DC15, DC16, DC17, DC18, DC19, DC31, DC39	DC23, DC27, DC35

Tabela 1 Tabela de agrupamento e classificação dos dados coletados. Cada linha da tabela representa a etapa do roteiro de entrevista (indicada na primeira coluna) onde os dados foram coletados. A segunda coluna contém os dados que possuem relação direta com o projeto, e a terceira coluna contém os dados que não possuem relação direta com o projeto.

A organização dos dados também por etapa é importante pois estas refletem a aplicação dos dados da seguinte forma: Os dados das etapas E2 e E3a estão mais relacionadas aos requisitos do *software*, ou seja, aquilo que ele deve oferecer aos usuários como funcionalidades; a etapa E3b está relacionada à usabilidade e casos de uso, que se referem à como o *software* deve ser manipulado pelos usuários para realizar a tarefa desejada; enquanto a etapa E3c está relacionada a como as informações serão apresentadas aos usuários na interface gráfica do *software*.

A sequência da numeração dos dados e a **Tabela 1** mostraram que algumas ideias foram sugeridas praticamente completas, por possuírem dados coletados em várias etapas da entrevista, de E2 à E3c (requisitos, casos de uso e representação na interface gráfica), sobre um mesmo assunto. Essas situações foram mais encontradas no grupo de dados de utilidade indireta.

Além de considerar com mais atenção os dados que foram classificados como de utilidade direta, foi realizada uma redução em categorias, que consiste em classificar as transcrições dos dados em grupos de acordo com seu conteúdo, de forma a serem relevantes para tomadas de decisões (HAIR JR, et al., 2010). A interpretação das categorias e seus significados ajudaram a elaborar as ideias para o desenvolvimento dos protótipos e do produto. Tais categorias foram nomeadas e codificadas com uma numeração da palavra “categoria” abreviada, resultando no formato CT<Número>, para simplificar suas identificações em outras seções deste documento. O detalhamento das relações dos dados coletados, as expressões recorrentes e categorias geradas podem ser encontradas no APÊNDICE B - Redução em Categorias. As categorias são representadas graficamente na **Figura 9** e abaixo relacionadas:

- **CT01 Foco de Estudo:** Categoria que resume os trabalhos já realizados para o projeto 1P1V e a outros que venham a ser realizados futuramente que venham a surgir de uma necessidade. Cada trabalho possui seu tema básico e estuda características específicas da música.

- **CT02 Dimensão de Análise:** Palavras ou expressões que são relacionados aos resultados, elementos específicos de cada trabalho (foco de estudo), suas análises e conclusões.
- **CT03 Exibição e Representação:** O conteúdo deste agrupamento está imerso no contexto de interface gráfica ou ligadas a possibilidades de representações das informações ou conteúdo das obras que fazem parte do corpus.
- **CT04 Intérprete:** Se refere a uma pessoa que interpretou uma das obra que foi gravada e adicionada ao corpus do projeto. A identificação desses intérpretes é extremamente importante para o estudo de microrrítmo e microdinâmica. A categoria também pode ser dirigida a futuros colaboradores ou em situações hipotéticas, para descrever procedimentos de usabilidade de funcionalidade.
- **CT05 Material:** Categoria que representa referências aos arquivos contidos no *corpus* do projeto, que são interpretações de obras musicais e estão disponíveis para análise.
- **CT06 Ação Funcional:** Indica ações com uma utilidade esperada, podendo ser aplicadas aos materiais disponíveis no *corpus*. Pode ser relacionada à formas de usabilidade ou mesmo, em algumas situações, funcionalidades propriamente ditas.
- **CT07 Quadro:** Categoria que se refere ao conceito de refletir uma realidade ou situação, que pode ser uma combinação de variáveis.

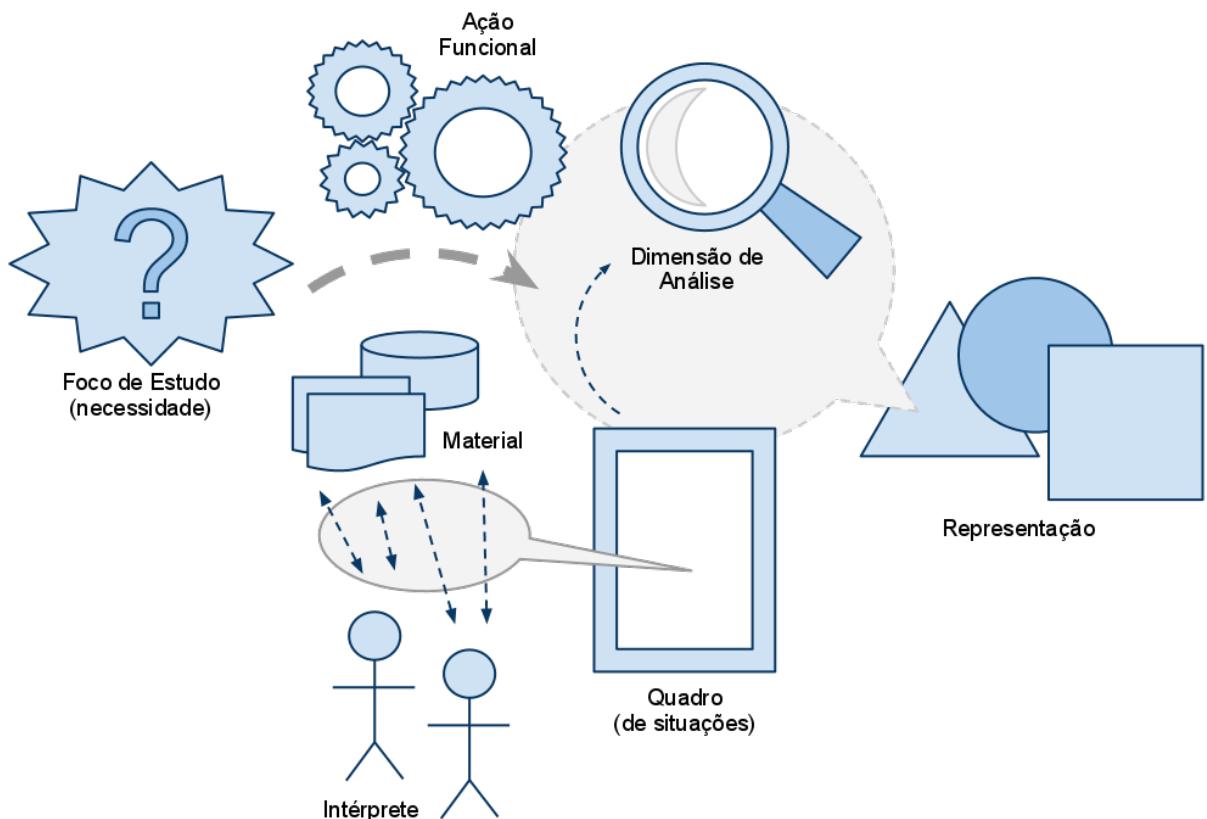


Figura 9 Representação das relações entre as categorias definidas. Focos de estudo, que surgem de uma necessidade, são manipulados por meio de ações funcionais que possuem um objetivo. Esse objetivo é o que leva à análise de quadros de situações, que podem ser a relação entre intérpretes e suas interpretações.

4.1.5 Análise de Similares

Nessa seção serão apresentados produtos que possuem objetivos ou funcionalidades similares ao do produto proposto. Como dito anteriormente, esses similares serão analisados para que sejam levantados seus pontos positivos, negativos e características relevantes para o desenvolvimento deste trabalho. Os critérios utilizados para classificar um produto como similar são baseados nos objetivos deste trabalho, nos dados coletados e suas análises. Cada similar também está codificado, para identificação em outras seções deste documento, como SM<Número>.

SM01: Ambiente Atual do Projeto Um País, Um Violão

Sena (2009), em seu trabalho de graduação, elaborou uma interface gráfica para melhorar a usabilidade e centralizar os trabalhos em computação musical

desenvolvidos para o projeto 1P1V. Cada funcionalidade agora possui uma forma de ser executada sem que seja necessária a utilização de linhas de comandos ou programação de roteiro da tarefa.

Este trabalho tornou possível a utilização dos programas por usuários que não necessariamente possuam conhecimentos avançados em informática, além de centralizar tudo em um único *software*, a partir do qual é possível usar todos os recursos.

Uma das principais características positivas desta proposta é que, para os projetos que originalmente possuíam uma interface gráfica, ficam preservadas suas formas de utilização originais. Os trabalhos ou funcionalidades que não possuíam uma interface gráfica ganharam suas primeiras formas de utilização. O produto dessa integração, internamente, agora é uma grande junção dos pacotes e classes de cada um dos trabalhos.

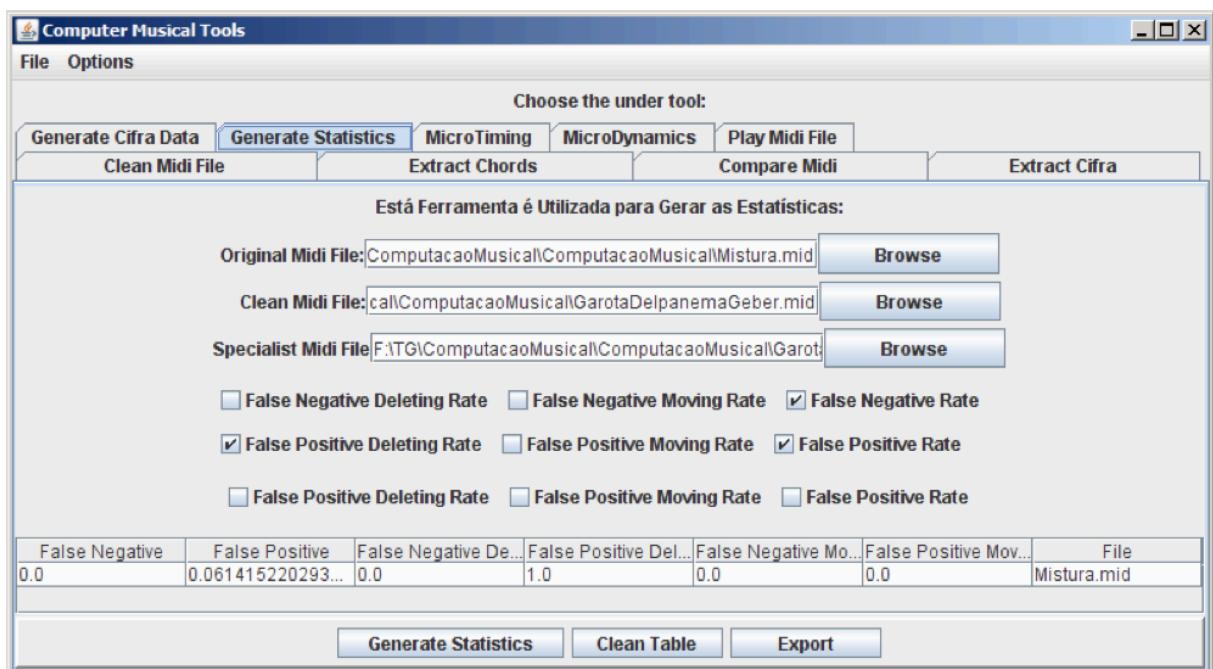


Figura 10 Interface do trabalho de integração de Sena (2008). Retirado de (SENA, 2008).

Apesar das vantagens, existe um problema com o modelo: Para cada nova necessidade que venha a surgir, será criado um novo trabalho e, para que este seja integrado aos demais, será necessária uma completa fusão ao ambiente, cada uma de forma única. Isso traz dificuldades para o crescimento do ambiente e sua manutenção.

Um outro problema é que cada trabalho possui sua maneira única e bastante especializada de ser utilizado. Quando existem muitas funcionalidades, cada uma com suas formas distintas de utilização, o aprendizado intuitivo fica difícil para os usuários. Além disso, mesmo estando em um mesmo ambiente, esses trabalhos continuam praticamente como se não compartilhassem relações uns com os outros.

SM02: Humdrum

Desenvolvido por David Huron, professor da Ohio State University, Humdrum é um conjunto de mais de sessenta ferramentas de *software* inter-relacionadas, criadas para auxílio à pesquisa em música. Seu funcionamento é baseado em manipulação de arquivos com conteúdo ASCII. A sintaxe dos arquivos utilizados pelo Humdrum é uma gramática que representa simbolicamente informações musicais (HURON, 2001; KNOPKE, 2008).

**kern	**kern	**kern	**kern
*staff2	*staff2	*staff1	*staff1
*cleffF4	*cleffF4	*clefG2	*clefG2
*k[f#c#g#]	*k[f#c#g#]	*k[f#c#g#]	*k[f#c#g#]
*M4/4	*M4/4	*M4/4	*M4/4
4AA	4c#	4a	4ee
=1	=1	=1	=1
8a	4c#	4a	4ee
8B	.	.	.
8c#	4c#	4a	4ee
8A	.	.	.
8D	4d	4a	4ff#
8E	.	.	.
8F#	4d	4a	4ff#
8D	.	.	.
=2	=2	=2	=2
2A;	2c#;	2a;	2ee;
4r	4r	4r	4r
4A	4e	4a	4cc#
=3	=3	=3	=3

4G#	4e	4b	4dd
4A	4e	4a	4cc#
8E	4e	4g#	4b
8D	.	.	.
8C#	4e	[4a	8 .cc#
8AA	.	.	.
.	.	.	16dd
=4	=4	=4	=4
2E	8e	8a]	2b
.	16d	8f#	.
.	16c#	.	.
.	4d	4g#	.
4AA;	4c#;	4e;	4a;
= : !	= : !	= : !	= : !
* -	* -	* -	* -

Código 1 Representação do Humdrum para Nun danket alle Gott, arr. J.S. Bach. Retirado de (HURON, 1999)

Quanto a sua capacidade e flexibilidade em termos de extração de informação, Humdrum é uma ferramenta bastante competente e talvez a mais reconhecida devido a sua vasta coleção de ferramentas, que também podem ser conectadas umas as outras por meio de *pipes*, para expressar resultados ainda mais específicos.

É de senso comum que pessoas são resistentes a atividades complexas. Em termos de usabilidade, Humdrum é uma ferramenta complicada. Além da vasta quantidade de extratores, sua execução é realizada a partir de linhas de comandos. Cada uma dessas ferramentas possui vários parâmetros de entrada, o que também dificulta o domínio por parte dos usuários. Um ponto positivo é que o usuário pode contar com uma documentação completa, mas essa documentação é composta de mais de 500 páginas em língua inglesa, o que pode fazer com que muitos desistam dela e, possivelmente, da ferramenta.

```
echo P5 > P5
echo '=' '*' >> P5; echo P5 >> P5
extract -i '*Ibass,*Ialto' file | hint -c | pattern -s = P5
```

Código 2 Exemplo de linha de comando para utilização do Humdrum que localiza quintas paralelas entre as vozes baixo e alto. Retirado de (HURON, 1999).

Por ter sido desenvolvido para sistemas UNIX, sua execução em outras plataformas requer o uso de ferramentas capazes de emular sua plataforma original, o que torna ainda mais complicada sua utilização por usuários sem conhecimentos avançados em computação, pois o processo de instalação e configuração de tantas ferramentas pode ser complicado e desmotivador para eles.

SM03: JRing

JRing surge, tecnicamente, como uma interface gráfica para um conjunto de ferramentas do Humdrum e funciona sobre suas operações de IR (Information Retrieval), oferecendo usabilidade intuitiva e operações de busca e comparação flexíveis (KORNSTADT, 2001). É uma ferramenta de utilidade bastante justificada, uma vez que Humdrum não é fácil para qualquer usuário.

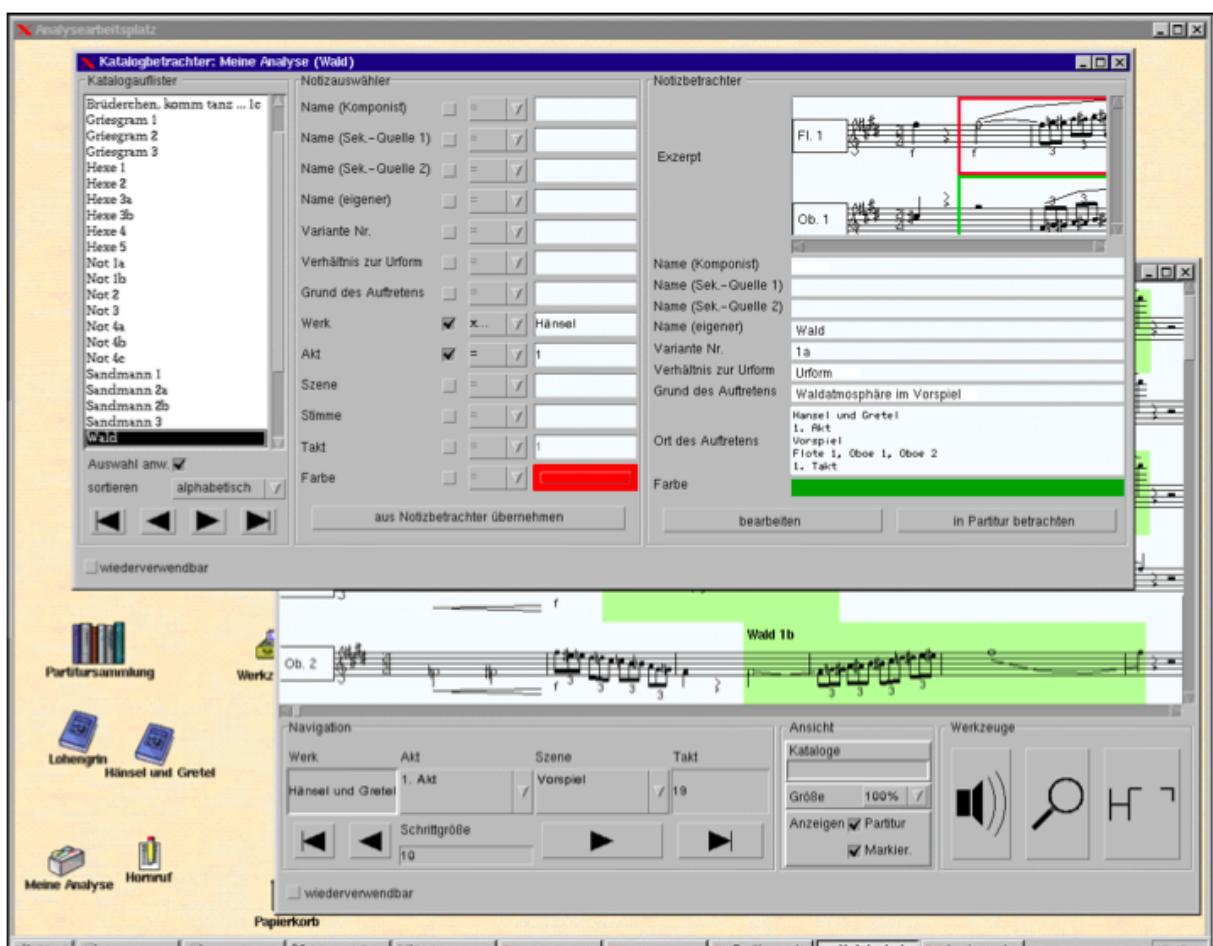


Figura 11 Interface gráfica do JRing. Retirado de (KORNSTADT, 2001)

JRing não é meramente uma interface gráfica, chegando a oferecer recursos não disponíveis originalmente no Humdrum. Conjuntamente com uma visualização em partitura, funciona um catálogo de anotações, que armazena, busca, seleciona e exibe informações adicionais sobre elementos relevantes extraídos e marcados em uma obra (KORNSTADT, 2001). Apesar de não abranger todas as funcionalidades e expressividade de Humdrum, JRing é uma ferramenta inovadora por possibilitar o uso de Humdrum por musicólogos.

SM04: OpenMusic 5

OpenMusic é um exemplo de uma linguagem de programação completa, em Lisp, dedicada a composição de música, que possui um ambiente de composição visual orientado a objetos. Esse ambiente dispõe de funções, representados como caixas, que podem ser conectadasumas as outras com estruturas de fluxo, criando um *patch* que, para o OpenMusic, é um algoritmo visual (BRESSON, AGON e ASSAYAG, 2005) (BRESSON, 2011).

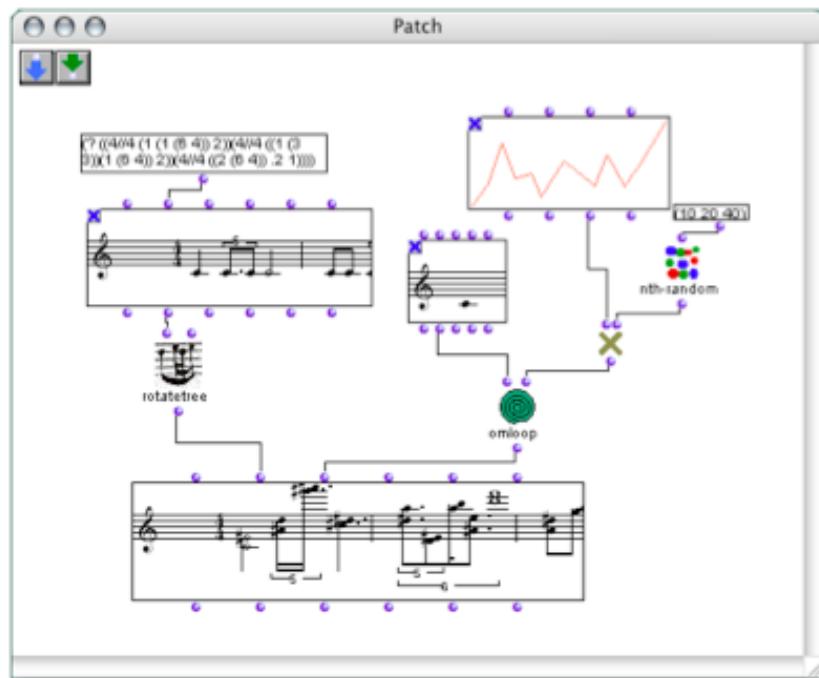


Figura 12 Interface gráfica do OpenMusic 5 retirado de (BRESSON, 2011)

OpenMusic também é um ambiente experimental para vários tipos de pesquisas sobre análise harmônica, reconhecimento de padrões, redes neurais e análise de estilo musical e simulações. OpenMusic é usado em musicologia como

suporte para experiências em modelos formais e matemáticos da música. A reconstituição de formalismos musicais e peças musicais neste ambiente proporciona uma abordagem intuitiva e interativa de análise musical (BRESSON, AGON e ASSAYAG, 2005; BRESSON, 2011).

Apesar de ser possível, talvez esta não seja a solução mais intuitiva para qualquer usuário, principalmente quando se trata de montar um *patch* para propósitos mais focados em pesquisa musicológica em MIR (Musical Information Retrieval), devido ao fato do foco principal ser a composição e que para praticamente tudo será necessário montar um algoritmo. Ainda assim OpenMusic é usado em várias universidades e conservatórios, pois é uma ferramenta prática e inovadora, permitindo que material musical e processos musicais coexistam em uma mesma representação (BRESSON, AGON and ASSAYAG 2005).

SM05: JMusic

JMusic, criado por Andrew Sorensen e Andrew Brown, é um projeto de pesquisa em música da Universidade de Tecnologia de Queensland (QUT), apoiado pelo departamento de música da própria universidade, em Birsbane, Austrália. O projeto é focado no auxílio à composição e processamento de áudio, mas pode ser usado para análise musical e ensino de computação musical (SORENSEN e BROWN, 2011; SORENSEN e BROWN, 2000).

Tecnicamente JMusic é uma biblioteca desenvolvida puramente em Java e, portanto, disponível para qualquer plataforma que possua uma máquina virtual. JMusic oferece implementações de classes que representam objetos de estrutura musical e métodos de manipulação como principal recurso. Utilizando sua API (Application Programing Interface) também é possível construir instrumentos (SORENSEN e BROWN, 2011; SORENSEN e BROWN, 2000).

```
Score s = new Score("JMDemo1 - Scale");
Part p = new Part("Flute", FLUTE, 0);
Phrase phr = new Phrase("Chromatic Scale", 0.0);
for(int i=0;i<12;i++) {
    Note n = new Note(C4+i, CROTCHET);
    phr.addNote(n);
```

```
}
```

Código 3 Exemplo da sintaxe de JMusic retirado de (SORENSEN e BROWN, 2011)

JMusic dá suporte a muitas possibilidades pois permite a utilização de todos os recursos de Java. Além disso, é um produto de fácil aprendizado e utilização por pessoas providas de conhecimentos de programação em Java, pois não possui novas sintaxes ou conceitos, e conta com documentação online. Deve ficar claro que qualquer possibilidade de uso da ferramenta é baseada em programação e, caso o usuário não possua tais conhecimentos, terá grandes dificuldades em utilizar o produto.

SM06: Music 21

Music21 é um conjunto poderoso e bastante expressivo de ferramentas focadas em musicologia assistida por computador. Com ele é possível que questões a respeito de elementos estruturais da música sejam respondidas ou, até mesmo, questões mais subjetivas, dependendo apenas da criatividade do usuário em montar scripts. Com Music21 é possível manipular música como um conjunto de dados simbólicos e exibir informações representando-as graficamente. As informações simbólicas de alto nível são representadas e definidas pelas classes contidas no pacote e, o funcionamento básico da ferramenta consiste em manipular essas estruturas (CUTHBERT e ARIZA, 2010; CUTHBERT e ARIZA, 2011a).

```
from music21 import *
sBach = corpus.parseWork('bach/bwv7.7')
sBach.show()
```

Código 4 Exemplo da sintaxe que exibe a partitura do conteúdo do arquivo. Retirado de (CUTHBERT e ARIZA, 2011b)

A ferramenta pode ser usada para escrever programas mais específicos ou executada por meio de linhas de comando em qualquer plataforma com suporte a Python mas, para utilizá-la, o usuário precisa ter conhecimento além do básico em computação, mais precisamente em programação, instalação e configuração de pacotes adicionais, uma vez que se trata de uma biblioteca em Python.

SM07: Outras Ferramentas e Recursos

Softwares do tipo DAW (Digital Audio Workstation) tem o propósito de produzir fonogramas, munido de funcionalidades para os processos desde captação de áudio, edição e mixagem até a masterização. Apesar de não serem exatamente produtos similares, possuem recursos relacionados e também servem como base de inspiração para elaboração de elementos de interface gráfica.

DAW's suportam extensão através de *plugins* VST (Virtual Studio Technology) DirectX e AU (Audio Unit), que permitem a adição de efeitos e instrumentos virtuais (VSTi - Virtual Studio Technology Instrument) para serem usadas nos projetos musicais. Esses *softwares* possuem normalmente, como sua interface básica, uma visualização em *timeline*, onde as gravações de áudio ou MIDI são exibidas ao longo de uma linha de tempo horizontal com medidas contadas da esquerda para a direita. A vantagem desta forma de visualização é a noção clara dos eventos ao longo do tempo e possibilitar o uso de mais de uma dimensão para expressar informações. Existem outras visualizações comuns em DAW's, como: *Piano Roll*, onde os eventos MIDI são exibidos como barras horizontais; e *Console*, onde são exibidos canais como em uma mesa de som.

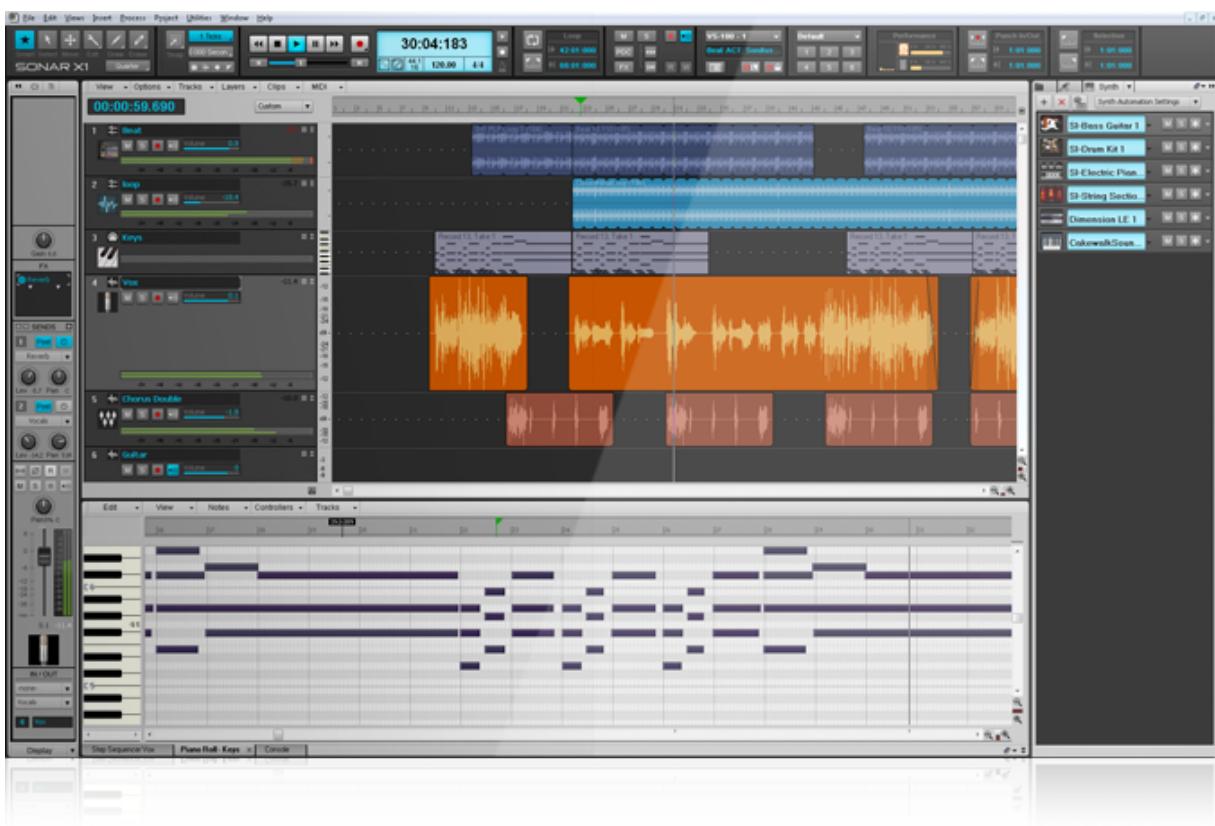


Figura 13 Interface gráfica do DAW Cakewalk SONAR X1 retirado de (CAKEWAL, 2011)

Assim como outras ferramentas, os DAW's - especialmente os que permitem edição de *tracks* MIDI (Musical Instrument Digital Interface) - possuem uma forma de selecionar eventos baseada em parâmetros. A esse tipo de funcionalidade é dado o nome de Filtro. Os filtros podem ser configurados e combinados de forma tão complexa que fornecem resultados bastante específicos, como um tipo simplificado de IR (Information Retrieval), que é um processo importante na musicologia.

TD-4 e TD-9 São módulos de som para *kits* de bateria eletrônica produzidos pela Roland e, apesar de também não serem produtos com objetivos similares, oferecem funcionalidades com interfaces gráficas interessantes. O modelo TD-4 possui uma funcionalidade chamada *Coach Mode*, que permite ao músico avaliar sua velocidade, precisão e estabilidade temporal durante a execução (ROLAND, 2009).

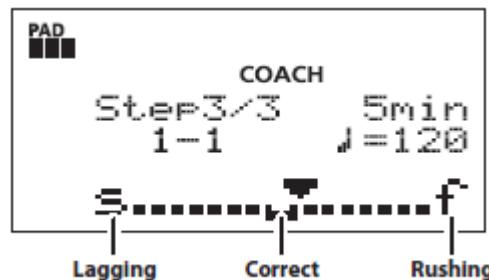


Figura 14 Interface do coach mode do TD-4 em seu display. Retirado de (ROLAND, 2009)

O modelo TD-9, por sua vez, possui a funcionalidade *Scope*, com objetivos bastante semelhantes ao *Coach Mode* do TD-4. A diferença entre as funcionalidades é que no TD-9 o usuário possui uma visualização simultânea da precisão temporal em cada *pad* (instrumento), onde é possível visualizar, graficamente, as batidas adiantadas e atrasadas de cada instrumento, como em uma visualização *piano roll* (ROLAND, 2008).

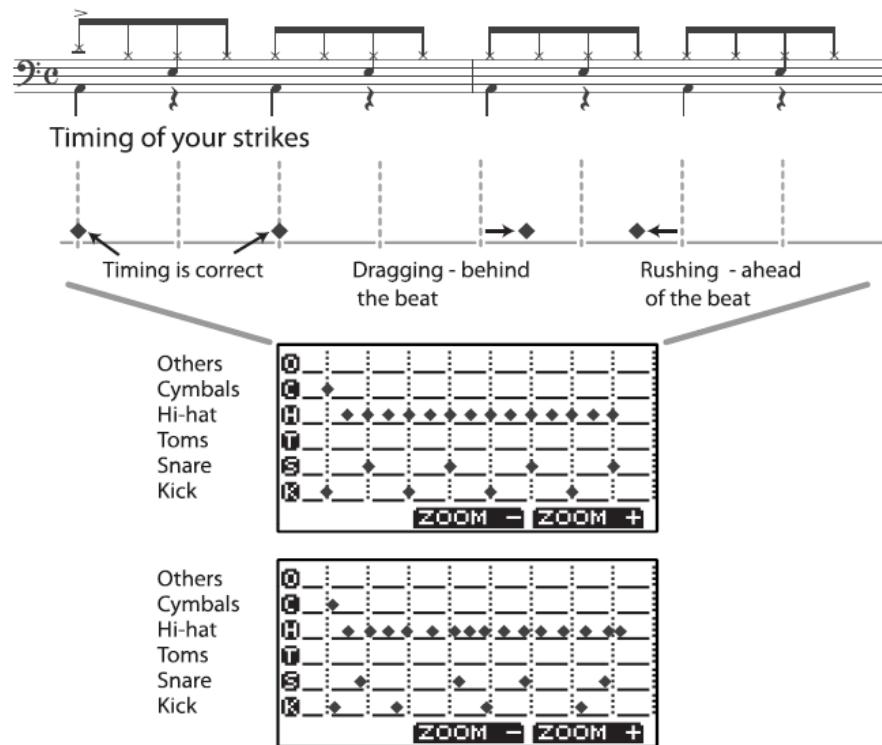


Figura 15 Interface do *scope function* do TD-9 em seu display. Retirado de (ROLAND, 2008)

4.1.6 Prototipação e Validação

Durante as entrevistas foi observado que grande parte dos entrevistados tinham maior facilidade em dizer se uma determinada funcionalidade apresentada era útil ou de boa utilização do que propor novas funcionalidades e formas de utilização. Baseado nisso e nas informações de cada uma das etapas anteriores, foram construídos protótipos baseados em cenários. As ideias iniciais a seguir foram criadas a partir da reflexão e interpretação dos dados obtidos anteriormente, e estão apontados como referências:

- A Interface principal foi definida como uma visualização em *timeline* (CT03 e SM07), onde cada *track* contém os objetos resultantes de uma transformação⁴ (CT02) realizada por um programa de foco de estudo em um material (CT01). Esses objetos são chamados de blocos. Futuros focos de estudo poderão ser adicionados à aplicação como *plugins* (DC01, DC03 e SM07), e suas transformações são realizadas a partir da escolha do material (CT06) e de suas configurações específicas (CT01 e CT02), daí o *software* processa a requisição e exibe os resultados (CT03). Assim como um DAW, o material transformado em *timeline* pode ser ouvido através de um *player* (CT06). Os blocos que compõem o *timeline* podem ser manipulados para correções de eventuais imprecisões, com operações de exclusão e edição de conteúdo (CT02 e CT06). O conteúdo de um *timeline* (*tracks* e blocos relacionados a um arquivo) é chamado de cena.
- A seleção de material para transformação e processamento é feita a partir da seleção de arquivos individuais ou classificadores em uma caixa de diálogo (CT06). Classificadores, ou *tags*, são palavras que caracterizam e agrupam arquivos em uma coleção (ou biblioteca) que pode ser organizada através de um gerenciador (CT07). Quando o usuário selecionar uma *tag* ele estará, na verdade, selecionando todo o material classificado por ela.
- Podem ser feitas análises de similaridade nos blocos contidos em um *timeline*, onde comparações entre blocos retornam um grau de similaridade (CT06) como o trabalho de Lima (2007). Sequências de blocos também

⁴ O material pode ser transformado em, por exemplo, sequências de acordes ou padrões rítmicos. O tipo de resultado depende de cada foco de estudo.

podem ser detectadas para revelação de padrões (CT06), assim como combinações comuns de blocos de *plugins* diferentes em um mesmo momento (CT06 e CT07), revelando padrões de situações recorrentes.

- Cada *plugin* é capaz de oferecer suas funcionalidades específicas (CT01 e CT02), que são recursos necessários para que cada trabalho expresse seus resultados da maneira adequada e sem maiores limitações para representações de informações ou procedimentos (CT03). Um exemplo disso seria o grafo de equipolência do trabalho de Lima (2007) ou os gráficos de Microdinâmica do trabalho de Holanda (2010). Essas funcionalidades são organizadas em um menu, e as funcionalidades específicas de cada *plugin* ficam agrupadas no sub-menu próprio do *plugin*.
- Sempre que possível é dado algum tipo de *feedback* ao usuário durante a utilização, como barras de progresso, ponteiros de *mouse* animados ou dicas sobre os componentes da interface gráfica (CT03). Uma janela de visualização de informações textuais fica visível no ambiente para que as funcionalidades expressem resultados de forma natural (CT03).

O processo de prototipação e validação utilizado foi realizado iterativamente como mostra a **Figura 16** e, sendo assim, os protótipos apresentados refletem as versões iniciais, enquanto as definições e os próximos capítulos refletem a situação final do produto. As anotações de sugestões de melhorias coletadas durante as validações podem ser encontradas no APÊNDICE A - Dados Coletados.

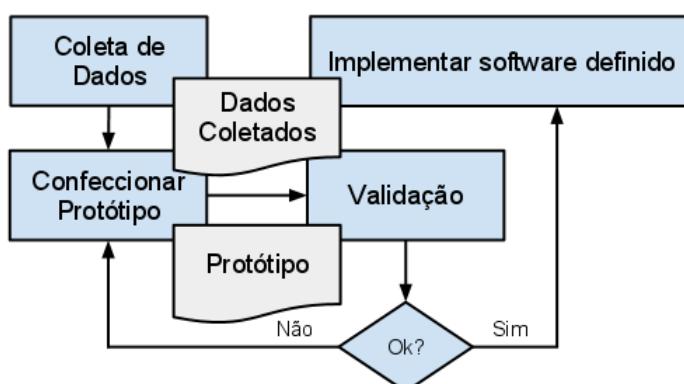


Figura 16 Processo de prototipação iterativa. Com base nos dados previamente coletados e interpretados, o processo consiste em reformular o protótipo confeccionado até que este seja aceito pelos usuários, quando então é implementado. Os dados coletados durante uma validação são utilizados na reformulação do protótipo.

É importante lembrar que a área de IHC é multidisciplinar e utiliza conhecimentos de diversas áreas como Psicologia, Sociologia, Antropologia, Design, Ergonomia, Linguística e Semiótica para um melhor conhecimento dos usuários, por isso se recomenda equipes com profissionais de diversas áreas no desenvolvimento de soluções. Até mesmo equipes compostas de profissionais da mesma área de conhecimento são melhores do que equipes de um único membro (BARBOSA e SILVA, 2010), pois as experiências e criatividade de cada profissional envolvido na pesquisa de usabilidade torna a solução mais rica. Infelizmente não foi possível contar com uma equipe multidisciplinar para a realização deste trabalho. Por esse motivo e pela limitação de tempo, que resultou numa quantidade pequena de reuniões com os colaboradores, é reconhecida uma possível deficiência quanto às interpretações dos dados e definições de ideias, mas que pode ser suprida em outras condições e oportunidades.

As considerações a respeito das anotações realizadas durante as avaliações com protótipo podem ser encontradas no APÊNDICE A - Dados Coletados.

Protótipo Baseado em Cenários

Um dos protótipos construídos, como mostra a **Figura 17**, é um modelo de baixa fidelidade em formato de apresentação digital, que simula cenários de utilização hipotéticos e fornece uma representação das funcionalidades propostas. Cada quadro da apresentação representa um passo do procedimento de utilização das funcionalidades. Esse protótipo possibilitou que os usuários opinassem a respeito das sugestões propostas sem a necessidade de sugerir novas funcionalidades, uma vez que nem todos os entrevistados se sentiram seguros para sugerir novas ideias.

The screenshot shows a presentation slide with the following content:

- Arquivo Questões**: The title bar.
- Barquinho.mid**: The main section title.
- Padrões Rítmicos**: A table showing rhythm patterns:

	A---P---B-p-+---	A---P---B-p-+I--	B---P---B-p-+I-	B---P---B-p-+I-	A---P---B-p-+---
--	------------------	------------------	-----------------	-----------------	------------------
- Acordes**: A table showing chords:

Acordes	Abm7	Fm6	C7(9)	Fm6	Abm7
---------	------	-----	-------	-----	------
- Informações**: A box containing analysis results:

Padrão "P1" + Acorde "Abm7" ocorrem conjuntamente 4 vezes.
Padrão "P2" + Acorde "Edim" ocorrem conjuntamente 2 vezes.

>> Detectando sequencias...
Sequência "A-B" localizada 2 vezes em 0:00:0, 0:00:03
Sequência "C-D" localizada 3 vezes em 0:00:02, ...

Figura 17 Protótipo baseado em cenários em formato de apresentação digital

Protótipo Funcional

Um outro protótipo, que pode ser visto na **Figura 18**, também foi apresentado, este menos abrangente porém funcional. Este protótipo foi construído para que fosse testada a arquitetura da aplicação, discutida na seção 4.2 deste documento.

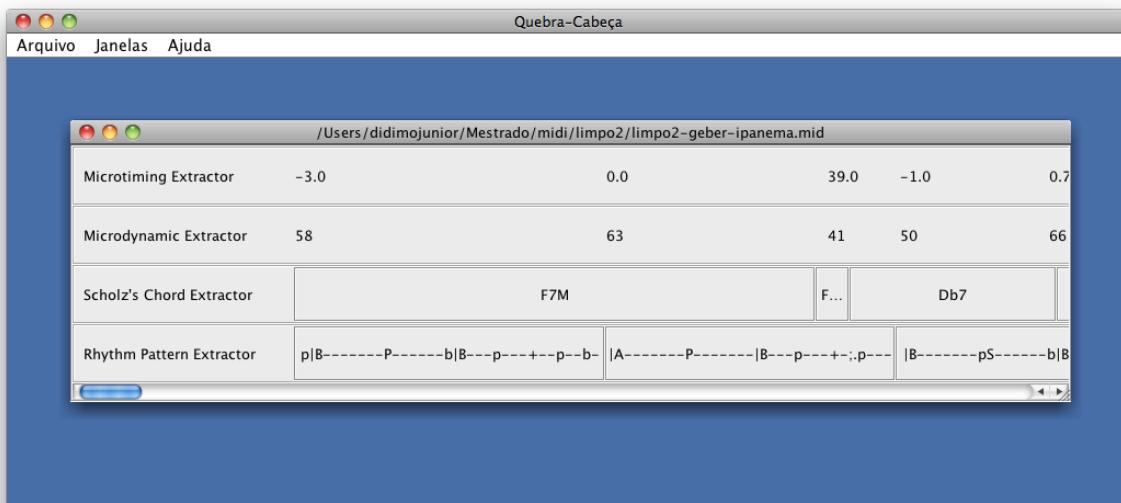


Figura 18 Protótipo funcional desenvolvido para testar a arquitetura proposta

4.1.7 O Produto

Como resultado do processo de concepção utilizado, foi definido um conjunto de requisitos funcionais e casos de uso que definem o produto. Um rascunho de sua interface gráfica pode ser visto na **Figura 19**. Os requisitos podem ser encontrados no APÊNDICE C – Requisitos Funcionais, enquanto os casos de uso (**Figura 20**) que representam essas funcionalidades estão agrupados e descritos de forma contínua⁵ para, de forma mais natural, oferecer fácil comunicação com os usuários. Esses casos de uso podem ser encontrados no APÊNDICE D - Casos de Uso. É importante deixar claro que os requisitos funcionais e casos de uso definidos são referentes ao ambiente da aplicação principal. Especificações para cada foco de estudo não fazem parte do escopo deste trabalho nesse momento, pois precisariam de um estudo mais detalhado e dedicado a este propósito.

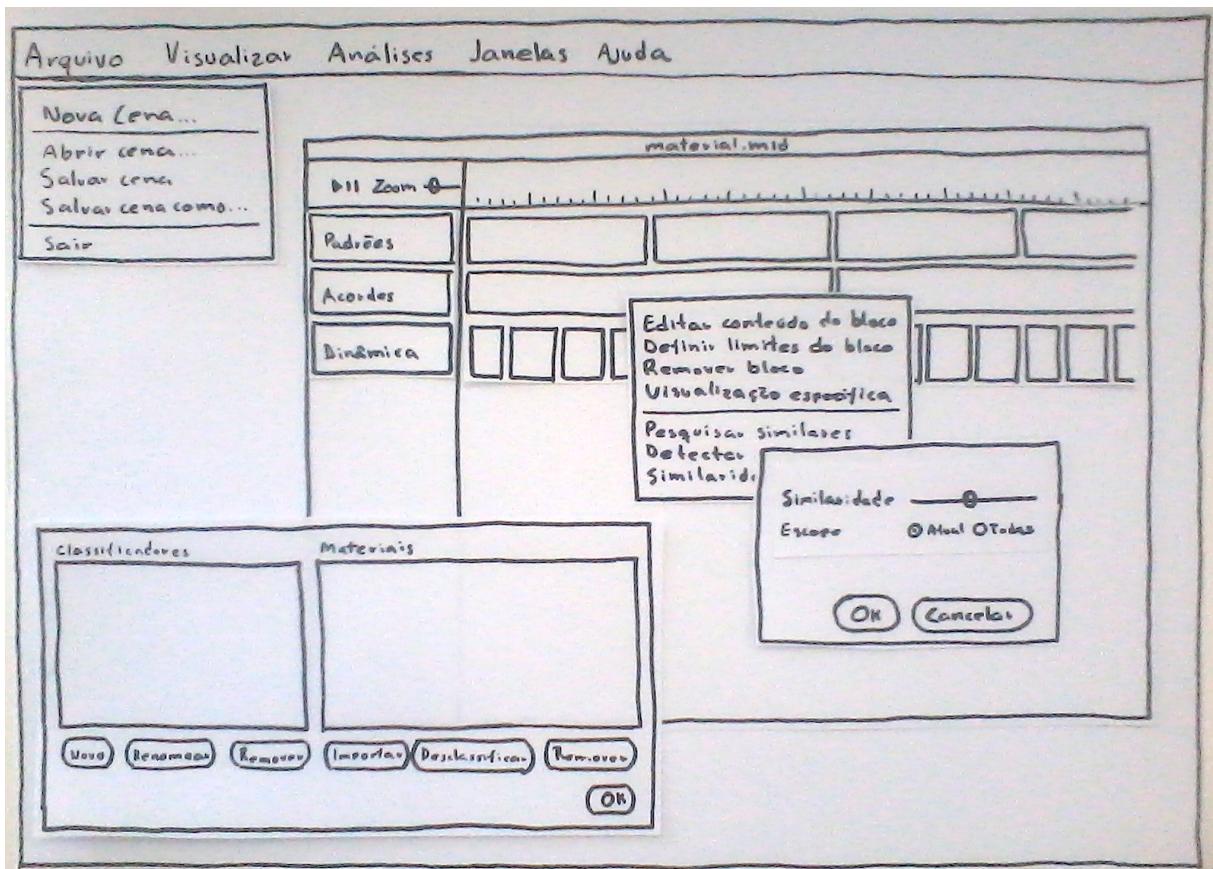


Figura 19 Protótipo final em papel. A imagem mostra uma cena (*timeline*) obtida a partir do menu "Arquivo, Nova cena...". Também estão presentes a janela do gerenciador de materiais (no canto

⁵ A descrição continua de casos de uso assume o formato de uma narrativa, diferente da descrição passo-a-passo convencional.

inferior esquerdo) e o menu de contexto de um bloco ativando a caixa de diálogo para entrada de parâmetros para a pesquisa de blocos similares.

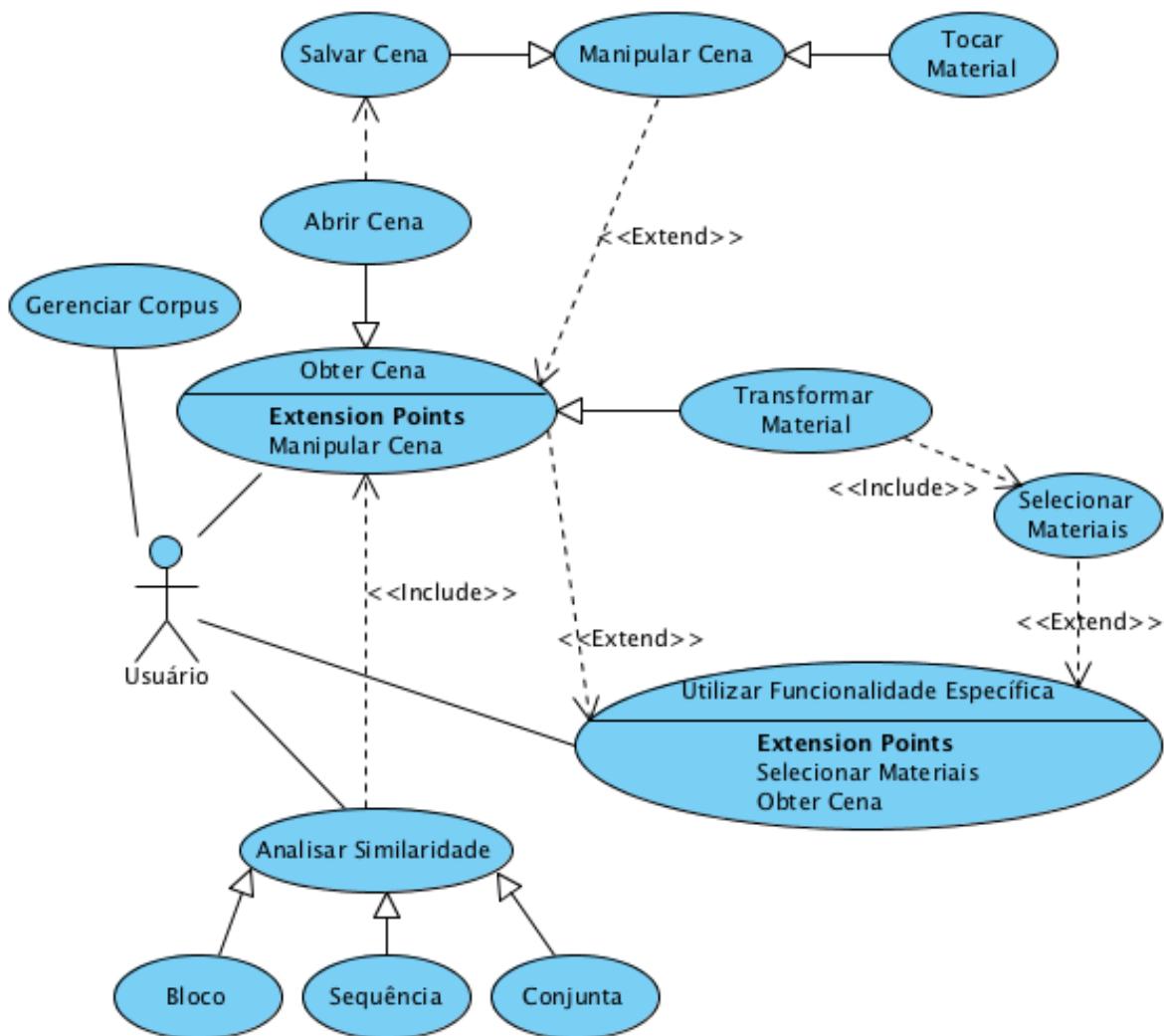


Figura 20 Diagrama de casos de uso das principais funcionalidades

4.2 Projeto da Arquitetura do Software

Arquitetura de software é a estrutura de um sistema, seus elementos, as propriedades externamente visíveis e relações entre eles (SEI, 2011). O desenvolvimento de um software com a capacidade de suportar as características do produto proposto envolve a análise do comportamento dos trabalhos atuais do projeto, assim como suas estruturas e necessidades. O objetivo disso é identificar um modelo de arquitetura de software adequado para o tipo de aplicação a ser desenvolvida e, do ponto de vista estrutural, resolver os problemas de integração e manutenção existentes.

Considerando que os programas analisados não são grandes mas utilizam técnicas avançadas e algoritmos complexos, a estratégia utilizada para entender o funcionamento dos programas foi do tipo *top-down* pois, a princípio, o interesse está no entendimento do comportamento e não em extrair uma arquitetura das aplicações escritas. Seria mais complicado tentar entender o código-fonte antes de entender o funcionamento geral pois, apesar de alguma documentação estar disponível, seu nível de detalhe, em geral, não proporcionaria uma taxa de avanço satisfatória. A estratégia *top-down* tem a vantagem de permitir abstração da implementação como caixas pretas, concentrar atenção apenas no comportamento, além de normalmente ser mais rápida e adequada a integrações (WEIDERMAN, et al. 1997).

Sobre os trabalhos que precisam ser integrados, pode-se dizer que, da forma como foram concebidos e implementados, cada um deles é responsável por estudar um aspecto diferente dos materiais de entrada e retornar um resultado. O conjunto de resultados do processamento desses trabalhos possibilita ao pesquisador tirar conclusões complexas a respeito de uma realidade ou situação extraídas do material analisado. Simplificando o comportamento de cada trabalho temos os seguintes roteiros:

- Padrões Rítmicos: O usuário fornece um arquivo MIDI como entrada. Esse arquivo é transformado em uma representação textual de sua “batida”, em formato de uma gramática própria. O produto dessa transformação é fragmentado, analisado, identificado como um padrão reconhecido e comparado a outros fragmentos em busca de similaridade, fornecendo um grafo de equipolência como resposta. Nesse caso a limpeza de ruídos no material original foi feita manualmente.
- Análise Harmônica: O usuário fornece um arquivo MIDI como entrada. Esse arquivo passa por um processo de limpeza, fragmentação e classificação. O usuário recebe como resultado uma lista de fragmentos da música associados à acordes.
- Micorrítmico e Microdinâmica: O usuário fornece um (ou vários) arquivo MIDI como entrada. O programa identifica os ataques relativos a cada unidade de tempo e subdivisões e retorna para o usuário uma relação de desvios de tempo (atrasos e antecipações) ou dinâmicas (intensidades) e seus cálculos médios para cada uma dessas situações.

Observando esses comportamentos, podemos dizer que todos os programas recebem arquivos MIDI como entrada, realizam alguma transformação ou processamento e retornam um resultado ou, simplificando ainda mais, recebem uma entrada, realizam um processamento e fornecem uma saída, ou seja, possuem um comportamento em comum que pode ser padronizado, como mostra a **Figura 21**.

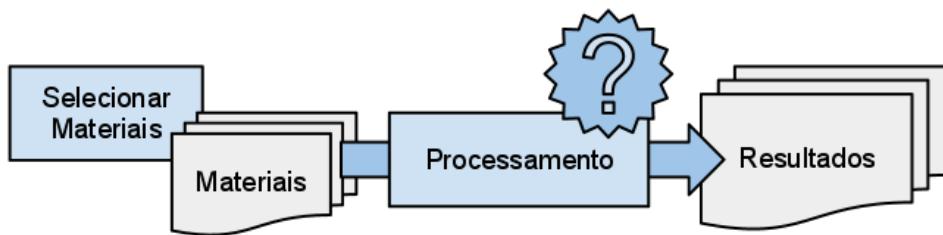


Figura 21 Comportamento padrão dos trabalhos. Os programas recebem materiais como entrada, realizam seus processamentos e devolvem resultados.

Mesmo com um comportamento geral padronizado, o ambiente deve oferecer flexibilidade suficiente para que seja possível estudar e expressar elementos tão distintos. Além disso, esses trabalhos refletem apenas a situação atual de necessidades, o que não significa que permaneça da mesma forma ao longo do tempo. Pelo contrário. É muito provável que novas necessidades de estudo apareçam a medida que se descobrem novos conhecimentos. Para essa situação de flexibilidade futura foi registrada uma necessidade em DC03 e DC01.

Diante do exposto, foi constatado que uma arquitetura baseada em *plugins* seria bastante adequada para ser a base do *Bleem Bloom*. Uma arquitetura baseada em *plugins* consiste basicamente de uma aplicação principal, chamada de host, a qual se adicionam programas menores, denominados *plugins*, de utilidades específicas e com objetivo de extensão da aplicação principal (CHATLEY, EISENBACH e MAGEE, 2003). Existe ainda o *plugin loader*, que é responsável por carregar os *plugins* disponíveis para que possam ser utilizados. Esses elementos podem ser vistos na **Figura 22**. Dessa forma existe a possibilidade de expansão com novas funcionalidades, antes desconhecidas, em tempo de execução após o *software* ter sido distribuído. Modularização de grandes sistemas para reduzir sua complexidade e permitir que terceiros desenvolvam *plugins* com base apenas nas interfaces definidas são outras vantagens (MAYER, MELZER e SCHWEIGGERT, 2003).

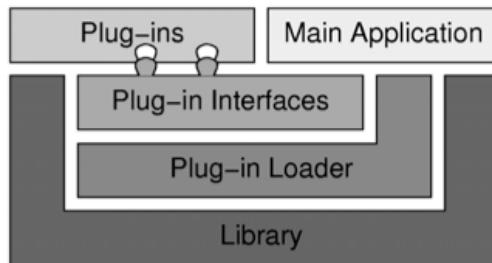


Figura 22 As camadas de uma aplicação baseada em *plugins*. Retirado de (MAYER, MELZER e SCHWEIGGERT, 2003).

Para ter certeza de que o conceito de *plugins* é adequado para a construção do *Bleem Bloom*, foi elaborado um protótipo funcional, já citado na seção 4.1.5, onde cada foco de estudo teve seus trabalhos considerados e convertidos em um *plugin*. Nesse modelo a aplicação principal assumiu o papel do *host* e do *plugin loader*. Para adaptação provisória dos trabalhos em *plugins*, foi utilizada a técnica de *wrapping*. Tal técnica, neste caso, consiste em reutilizar as estruturas originais dos trabalhos encapsuladas dentro de *plugins*, sem a necessidade de realizar modificações internas, como mostra a **Figura 23**. Apenas pequenas adaptações para compatibilização com o formato de parâmetros e resultados do novo ambiente foram implementadas. Isso permite o reuso dos componentes já confiáveis e bem testados (BISBAL, et al. 1999) em uma nova situação de integração.

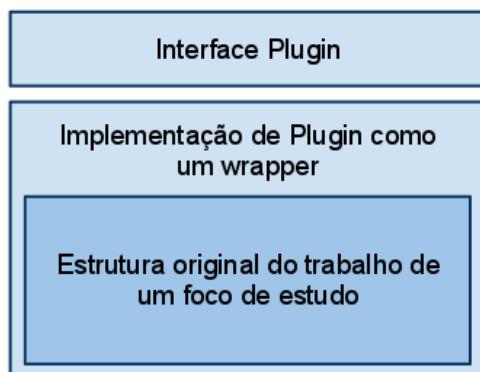


Figura 23 O *plugin* como um *wrapper*

O modelo de arquitetura baseado em *plugins* possibilitou que novos focos de estudo fossem adicionados ao ambiente diante de tal necessidade, característica que faz parte do conjunto de dados coletados explicados na seção 4.1.3. Tal modelo também permitiu que as implementações dos focos de estudo pudessem ser isoladas da implementação do ambiente, o que simplifica e facilita suas

manutenções e o desenvolvimento de futuros *plugins*. Ao mesmo tempo em que proporcionou fácil integração dos trabalhos atuais ao ambiente e tornou o comportamento geral de transformação padronizado, como explicado anteriormente nesta seção, o modelo também permite a implementação das funcionalidades específicas de cada *plugin*, tornando-se flexível. Tudo isso mostrou que uma arquitetura baseada em *plugins* era adequada à realidade do projeto. Os principais elementos da arquitetura de *Bleem Bloom*, que podem ser vistos graficamente na **Figura 24**, ficam assim definidos:

- A interface gráfica, por onde os elementos estruturais são manipulados e os dados podem ser visualizados;
- O programa principal do ambiente assumindo o papel de *host*, onde é realizada a distribuição das atividades e carregamento dos *plugins*;
- Os *plugins* propriamente ditos, que representam os focos de estudos e realizam todos os processamentos relativos a eles;
- As ferramentas, bibliotecas e elementos auxiliares.

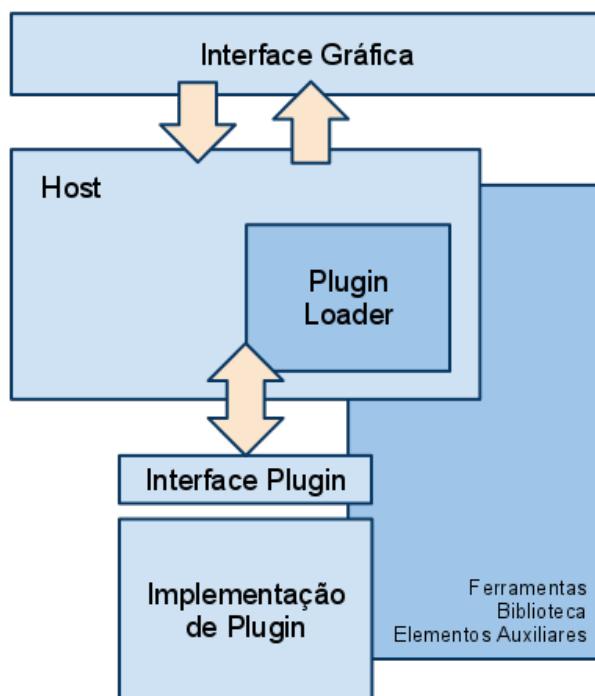


Figura 24 Arquitetura simplificada

5 IMPLEMENTAÇÃO E EXPERIMENTOS

Neste capítulo serão apresentados os critérios para as escolhas tecnológicas adotadas, o processo de desenvolvimento e o detalhamento da implementação dos principais componentes estruturais do ambiente, assim como as técnicas utilizadas. Também serão apresentados os experimentos práticos para avaliação da usabilidade do produto realizados com os usuários colaboradores.

5.1 Detalhes Sobre a Implementação

Esta seção se dedica a apresentar detalhes sobre a implementação da estrutura interna da aplicação.

5.1.1 Escolhas Tecnológicas (linguagem, bibliotecas e toolkits)

Tanto para a construção da interface gráfica quanto para a dos objetos estruturais de um *software*, é necessária a escolha da linguagem de programação mais adequada, assim como qualquer biblioteca que venha a ser utilizada. A linguagem deve ser capaz de expressar as estruturas necessárias para a implementação da aplicação.

A escolha da linguagem de programação errada pode impactar negativamente o desenvolvimento de uma aplicação em muitos pontos, por isso é importante ter em mente as necessidades técnicas do *software* e condições de desenvolvimento para identificar as qualidades que a linguagem precisa ter. Critérios como escalabilidade, facilidade de aprendizado, velocidade de desenvolvimento, manutenibilidade, portabilidade e custo total de propriedade também devem ser considerados (Krieger 2009), como explicados a seguir.

Escalabilidade é uma qualidade que diz respeito a capacidade do *software* crescer, e foi considerada neste trabalho, pois uma das principais características de *Bleem Bloom* é o suporte a *plugins* que aumentam suas possibilidades, além da possibilidade de evolução do próprio ambiente em versões futuras.

A facilidade de aprendizado diz respeito ao esforço necessário para que uma linguagem seja dominada pelo programador, e impacta diretamente na velocidade

de desenvolvimento, que significa o tempo que uma aplicação leva para ser implementada. Manutenabilidade, que se refere à facilidade com que manutenções são feitas em um *software*, também é importante. Essas características também precisaram ser consideradas, pois o tempo para a conclusão deste trabalho foi limitado.

Foi importante considerar também a portabilidade, que se refere à possibilidade do programa ser executado em vários sistemas operacionais, pois é comum encontrar muitos usuários de computador que utilizam a máquina com propósitos musicais utilizando o Mac OS, ao mesmo tempo que o Windows ainda é o sistema operacional mais utilizado.

Muitas ferramentas em *software*, incluindo as de desenvolvimento de *software*, precisam de licenças para que possam ser utilizadas. Isso caracteriza o custo total de propriedade, que pode custar caro. Por se tratar de um projeto sem investimentos financeiros, o custo de licença para utilização da linguagem de programação e de ferramentas de desenvolvimento precisou ser o mais reduzido possível.

Bleem Bloom se enquadra bem como uma aplicação *desktop*, isto é, executada diretamente na máquina do usuário, pois um ambiente web não traria benefícios, uma vez que *Bleem Bloom* não acessa informações de servidores na internet, mas arquivos que se encontram no computador do usuário. Além disso, aplicações *desktop* possibilitam o desenvolvimento de interfaces gráficas mais avançadas, que é uma característica importante do *Bleem Bloom*.

Considerando os critérios citados, C++, Java e Python são exemplos de linguagens de programação que poderiam ser utilizadas para a implementação de *Bleem Bloom*, mas existem considerações importantes a serem feitas:

- É de senso comum que C++ não é uma linguagem de fácil aprendizagem, além de exigir muitos cuidados por parte do programador. Apesar de Python não ser uma linguagem difícil, o impacto inicial de aprendizagem de outra linguagem seria consideravelmente maior, além de não possuir uma biblioteca padrão para a construção de interfaces gráficas.
- Java é uma linguagem orientada a objetos de fácil aprendizagem e bastante conhecida e utilizada nos meios acadêmico e comercial. Isso é conveniente quando se deseja que mais pessoas participem do projeto. Java também

possui uma documentação detalhada disponível e uma biblioteca de classes bastante amadurecida que, inclusive, oferece recursos para trabalhar com áudio e MIDI no pacote javax.sound.

- Existem IDE's (Integrated Development Environment) de qualidade disponíveis para Java. Essas ferramentas oferecem ao programador recursos que dão a vantagem da produtividade, acelerando a velocidade de desenvolvimento. Com essas ferramentas também é possível criar interfaces gráficas visualmente. A utilização de Java foi adequada dada a natureza deste trabalho como um projeto, pois nem a linguagem nem as ferramentas necessárias trazem custos aos desenvolvedores ou usuários.
- Todos os trabalhos realizados para o projeto foram construídos em Java. Desenvolver o *Bleem Bloom* nessa linguagem traria uma grande vantagem de compatibilidade, tornando os processos de evolução e integração desses trabalhos menos problemáticos. Em caso de extrema necessidade, a utilização de JNI (Java Native Interface) permite a integração de Java com bibliotecas escritas para um sistema operacional específico.

Pelos motivos citados, optou-se pela utilização da linguagem de programação Java.

Apesar de existirem *frameworks* Java que facilitam o desenvolvimento de aplicações baseadas em *plugins* como Java Plug-in Framework (JPF) Project, Java Simple Plugin Framework (JSPF) e jin-plugin - Simple Plugin Framework for Java and PHP, optou-se por desenvolver o *software* proposto sem a utilização dessas bibliotecas. O motivo é que esses *frameworks* precisam ser genéricos e oferecer recursos suficientes para suprir as mais diversas necessidades e possibilidades de implementações de *plugins*, o que pode fazer com que sejam complexos demais para uma situação muito específica. Isso pode trazer dificuldades aos desenvolvedores que venham a implementar novos *plugins*, pois precisariam ter familiaridade com um *framework* adicional além dos elementos estruturais específicos do ambiente. Isso entraria em conflito com um dos objetivos de *Bleem Bloom*, que é ser fácil tanto para os usuários quanto para os desenvolvedores, favorecendo usabilidade e desenvolvimento.

Um outro ponto muito importante do *Bleem Bloom* é sua interface gráfica. Por isso é necessário que seja decidido adequadamente os padrões de

desenvolvimento, técnicas e bibliotecas utilizadas. Existem vários *toolkits* para construção de interfaces gráficas em Java, cada um com suas características (FEIGENBAUM, 2006):

- AWT (Abstract Window Toolkit) é o *toolkit* gráfico original de Java e por isso não precisa ser instalado. AWT utiliza rotinas nativas para criar seus componentes, por isso uma mesma aplicação pode parecer diferente quando executada em plataformas diversas, mas com ganho em performance e estabilidade. AWT oferece o conjunto reduzido de componentes comuns a todos os ambientes Java. Com AWT também não é necessária a finalização dos componentes, pois isso é feito automaticamente.
- Swing foi criado para fornecer um conjunto mais sofisticado de componentes para construção de interfaces gráficas, suprindo os componentes ausentes em AWT. Swing é parte da JFC (Java Foundation Classes) como *toolkit* gráfico primário, e não utiliza renderização de componentes nativos, e sim a API (Application Programming Interface) Java 2D para desenhar seus objetos, por isso uma aplicação terá a mesma aparência mesmo quando executada em sistemas operacionais diferentes. Apesar do aprendizado não ser tão simples, existe um vasto conteúdo de referências e documentação disponível na internet. Uma desvantagem é que Swing pode ser complexo para situações comuns e simples. Swing também possui a finalização automática de seus componentes.
- SWT (Standard Widget Toolkit) é uma alternativa para AWT e Swing, que tenta ter as vantagens oferecidas por ambos e ser livre de suas desvantagens. SWT costuma ter bom desempenho e possui uma mistura dos componentes comuns de AWT para a maioria das situações e a emulação como Swing. SWT precisa ser instalado, o que pode complicar para usuários e desenvolvedores de plataformas diferentes, e pode ser mais difícil de ser dominado. Em SWT os componentes containers precisam ser criados antes dos filhos e normalmente os componentes não podem ter seus pais modificados. Além disso os componentes precisar ser explicitamente finalizados. Essas características reduzem a flexibilidade de SWT.

Por oferecer uma vasta paleta de componentes avançados e independentes de plataforma sem precisar de uma biblioteca extra, contar com uma grande coleção de material de referência e documentação, ser padrão integrado no JFC e existir uma ferramenta de construção visual IDE Net Beans, o toolkit Swing foi adotado na implementação deste trabalho.

5.1.2 Processo

Foi adotado um processo de desenvolvimento simplificado que, para fins deste trabalho, foi chamado de Recursivo. Suas atividades, que podem ser vistas graficamente na **Figura 25**, estão descritas abaixo:

1. **Planejamento:** Nesta etapa o conteúdo do ciclo a ser implementado é planejado, objetivando a garantia de que terá as características necessárias para seu funcionamento adequado no projeto. Abrange atividades como levantamento de requisitos, análise, planejamento de cronograma e documentação.
2. **Implementação:** Essa é a etapa onde o conteúdo planejado para o ciclo atual é de fato implementado.
3. **Teste:** Esse é o momento em que o conteúdo construído no ciclo atual será testado. Também poderá ser realizado o controle de versão sempre que um elemento implementado for aprovado pelos testes.

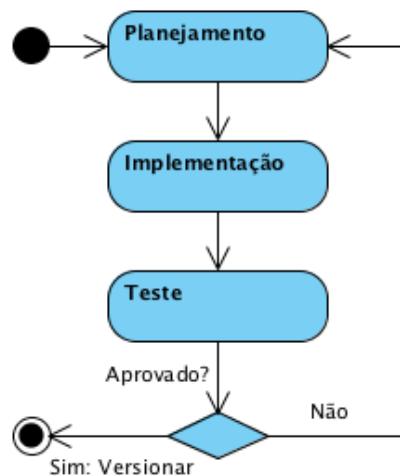


Figura 25 As atividades de um ciclo

Uma etapa de implementação pode também ser composta e conter mais de um ciclo, como mostra a **Figura 26**. No caso de uma etapa de implementação ser composta, a etapa de planejamento deve ser responsável por definir e coordenar o progresso dos ciclos paralelos. A etapa de teste de cada ciclo paralelo deve realizar testes individuais, enquanto a etapa de teste do ciclo de nível mais alto, isto é, o que engloba os ciclos paralelos, também deve se preocupar em realizar testes de integração dos elementos implementados ao restante do projeto.

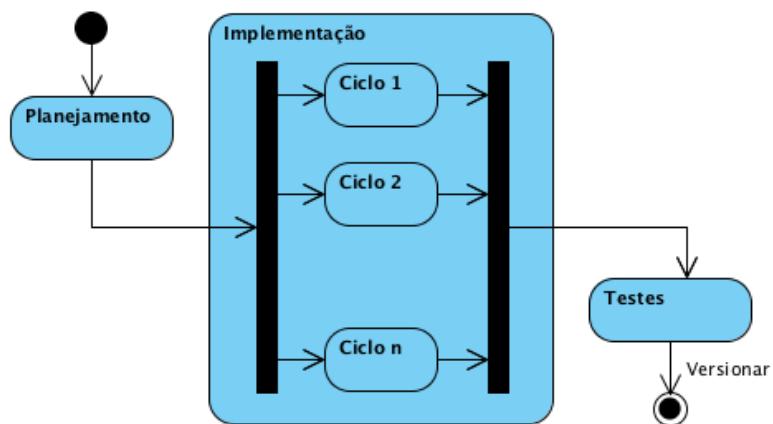


Figura 26 Ciclo composto por ciclos paralelos

O processo de prototipação cíclico utilizado no capítulo 4 também reflete o comportamento do ciclo aqui descrito. Fazendo uma comparação, a etapa de planejamento seria onde dados são coletados para que um protótipo seja pensado, a etapa de implementação está relacionada a confecção do protótipo, enquanto que a etapa de testes seria a validação com os usuários.

5.1.3 Detalhamento

As próximas seções se dedicam a explicar a organização das classes em pacotes e suas respectivas funções no projeto do ambiente, além de mostrar como esses elementos interagem entre si. Elementos menos relevantes para o entendimento da mecânica geral do projeto poderão ser consultados na documentação javadoc.

Pacotes

As classes do ambiente estão agrupadas pelo papel que exercem no projeto, e organizadas nos seguinte pacotes:

- **br.ufpe.cin.dvaj.msc.data:** Contém basicamente classes que representam dados.
- **br.ufpe.cin.dvaj.msc.environment:** É onde estão as classes e recursos relacionados à configuração do ambiente.
- **br.ufpe.cin.dvaj.msc.exception:** Contém classes que representam as situações de exceção que podem ser levantadas durante o funcionamento do ambiente.
- **br.ufpe.cin.dvaj.msc.gui:** Contém classes que representam janelas, caixas de diálogo, classes auxiliares que controlam seus funcionamentos e outros elementos relacionados a interface gráfica.
- **br.ufpe.cin.dvaj.msc.host:** É o pacote que abriga o *host* e as classes estruturais da mecânica de funcionamento essencial do ambiente.
- **br.ufpe.cin.dvaj.msc.plugin:** Abriga interfaces e classes abstratas que devem ser implementadas para a criação de um novo *plugin*.
- **br.ufpe.cin.dvaj.msc.tools:** Possui classes que representam ferramentas disponíveis para utilização pelos *plugins* desenvolvidos e úteis aos usuários.
- **br.ufpe.cin.dvaj.msc.util:** Possui classes de utilidade geral.

O pacote de distribuição bleembloom-pluginkit.jar contém a seleção de classes e interfaces necessárias para o desenvolvimento de novos *plugins*. É o que futuros desenvolvedores deverão adicionar a seus projetos.

As Classes e Suas Funções

Aqui serão apresentadas as principais classes do ambiente e suas funções. Nos casos mais importantes serão apresentadas as interações entre os objetos para explicar a mecânica funcional do ambiente.

Principais Elementos

A classe *Host* representa o host de uma aplicação com arquitetura de *plugin* e implementa toda a sua mecânica de funcionamento. Ela é quem reconhece (ver **Figura 27**) e gerencia os *plugins* através de seu *plugin loader*, além de controlar todas as suas manipulações. Qualquer requisição de processamento passa por uma instância dessa classe, que é responsável por despacha-la para o *plugin* adequado.

Descriptor é um dos principais elementos do ambiente, pois é a partir dele que são mantidos os mapeamentos dos *plugins*. Trata-se de uma interface que descreve objetos que tem o objetivo de identificar uma implementação de um *plugin* em diversas situações durante o funcionamento do ambiente. Seus métodos basicamente retornam informações a respeito do *plugin*.

Plugin é a interface que define um *plugin* propriamente dito. A implementação dela é o centro de processamento de um foco de estudo e possui todas as características necessárias para que o *plugin* funcione no ambiente. É a partir dele que o *host* tem acesso a seu *Descriptor*. As principais atividades do Processor é a transformação, realizada pela execução do método *transform()*, e a execução de funcionalidades específicas realizada pelo método *execute()*. Essa classe também fornece componentes utilizados pelo ambiente para manipulação de seus dados. Uma exigência é que qualquer implementação de *plugin* possua um construtor público sem parâmetros, para que possa ser instanciado pelo *plugin loader*.

Script é uma classe abstrata que representa uma funcionalidade específica de um *plugin*. Sempre que o usuário solicita sua utilização, o *host* executa seu *script*, trazendo os resultados esperados. Um *plugin* deve fornecer tantos scripts quantas funcionalidades específicas ele possuir.

UseCaseController é uma classe que centraliza os roteiros operacionais dos casos de uso e controla os objetos da dados através dos componentes de interface para que uma atividade útil seja realizada pelo usuário. Cada roteiro de usabilidade é basicamente um método desta classe.

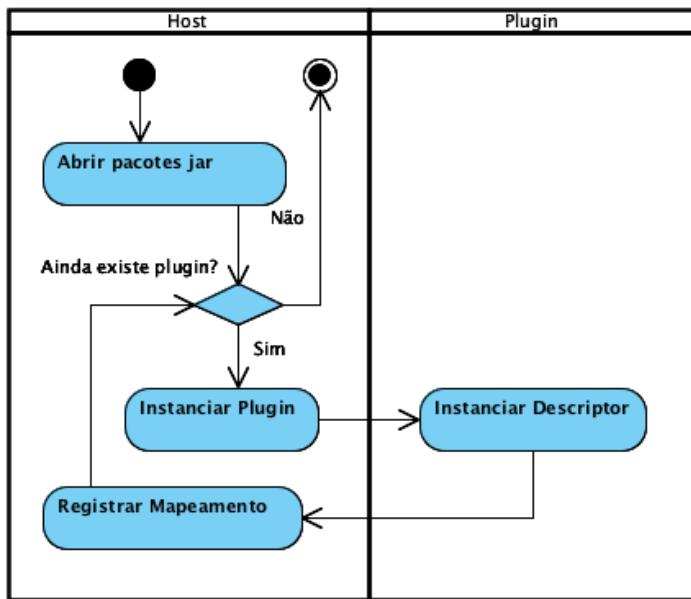


Figura 27 Processo de mapeamento das classes dos *plugins* e *Descriptor* realizado pelos *class loader* do *host*.

Objetos de Conteúdo

Existem objetos cuja função é carregar um conteúdo de um *plugin*. Esses objetos, representados pela classe *Content*, carregam, além dos objetos que são o conteúdo propriamente dito, o *Descriptor* do *plugin* ao qual o conteúdo pertence. Então, para que algo seja processado por um *plugin*, um objeto *Parameters*, que é subclasse de *Content*, carrega o *Descriptor* desse *plugin* e os parâmetros propriamente ditos. Da mesma forma, todo conteúdo retornado de um processamento vem em um objeto *Results*, que também herda de *Content* e contém o mesmo *Descriptor*.

De forma simplificada, o *host* sempre recebe um objeto *Parameters* como parâmetro, repassa para o *plugin* adequado baseado no *Descriptor* que está embutido, que executa o processamento e retorna para o *host* um objeto *Results* daquele *plugin*, contendo o mesmo *Descriptor* e o conteúdo dos resultados propriamente ditos. A **Figura 28** mostra esse comportamento. O comportamento de escolha do *plugin* que irá processar uma requisição baseado nos objetos passados por parâmetro é inspirado nos patterns *Startegy*, onde algoritmos são objetificados, e *Command*, onde a requisição de um serviço é objetcificada (GAMMA, et al., 1993).

O conteúdo retornado por cada *plugin* é uma lista de objetos *ContentItem*, que é um objeto que possui um conteúdo transformado por intervalo de tempo, ou seja,

representa uma informação temporal que indica que um conteúdo está presente a partir de um instante X até um segundo momento Y de um material. O conteúdo desse objeto pode ser qualquer objeto que o *plugin* reconheça, desde que seja uma subclasse de *Comparable*, pois deve suportar as funcionalidades relativas a similaridades.

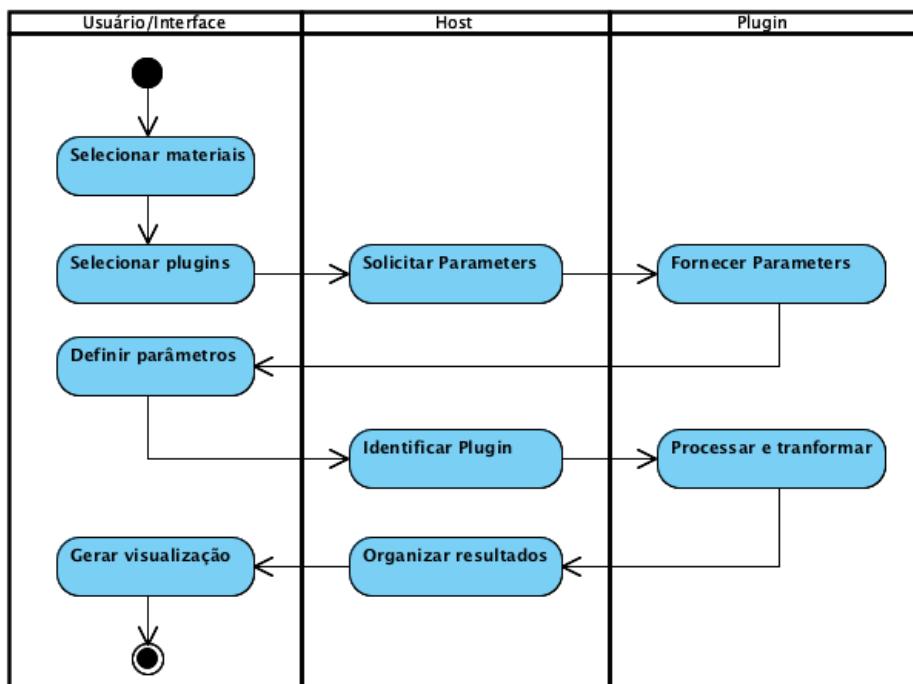


Figura 28 Atividades realizadas durante uma solicitação de transformação

Scene

Um objeto *Scene* representa uma situação após o processamento de uma ou mais transformações para um mesmo arquivo. Essa situação é a base para uma visualização em *timeline*, onde cada *track* é uma coleção de objetos transformados por um *plugin*. Tecnicamente, um objeto *Scene* é uma coleção de objetos *ItemTrack*, que por sua vez é uma coleção de objetos *ContentItem*. A **Figura 29** representa graficamente esses elementos.

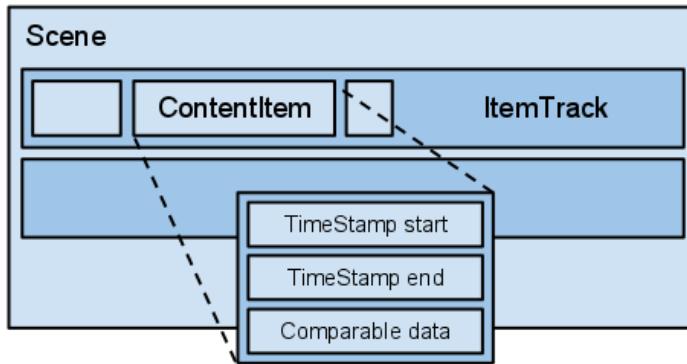


Figura 29 Representação dos objetos que compõem uma cena

Na interface gráfica, as classes *TimelineWindow*, *Track* e *ContentItemBlock* representam respectivamente os objetos *Scene*, *ItemTrack* e *ContentItem*. A visualização padrão do objeto dentro de um *ContentItemBlock* é a representação em *String* fornecida pelo método *toString()* dos objetos *Comparable*.

Ferramentas e Visualização de Dados (Corpus, Console, Progress)

Os elementos que exibem dados funcionam basicamente em pares e seguem o *pattern Observer* (DENZLER e GRUNTZ, 2008). Existe um objeto observável, que são os dados propriamente ditos, e o observador, que é o elemento de interface gráfica que exibe os dados do objeto observável e permite que o usuário faça manipulações sobre eles. Cada objeto observador é registrado em seu respectivo observável, pois sempre que ocorre uma modificação do conteúdo de um observável, este notifica todos os seus observadores para que possam refletir o novo estado de seus dados (DENZLER e GRUNTZ, 2008). Este mecanismo é mostrado graficamente na **Figura 30**. *Corpus*, *Console* e a visualização de progresso são ferramentas que possuem tais características de funcionamento.

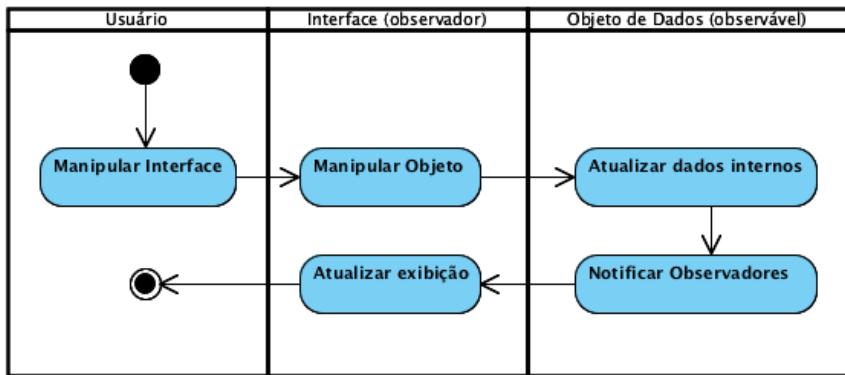


Figura 30 Atualização da interface disparada pela atualização de objetos de dados

A ferramenta referente à biblioteca de materiais para análise é composta pelas seguintes classes: *Corpus*, que representa a coleção, seus dados e suas operações; *CorpusManagerWindow*, que é o componente visual por onde o usuário manipula esses dados; e *TagSelectorAccessory*, que é um componente visual que pode ser inserido em caixas de diálogo para permitir a seleção de arquivos por classificador. Além dessas, existem classes auxiliares que são modelos de como os dados se comportam e são apresentados nos componentes.

O *Console* é uma ferramenta utilizada pelos *plugins* para que possam expressar informações textuais aos usuários. Seu funcionamento é baseado em duas classes: *OutputConsole*, que são as operações e conteúdo acumulado, e; *OutputConsoleWindow*, que é o componente visual onde as informações são exibidas.

Para visualização do progresso dos processamentos existem as classes *ProgressInformation* e *ProgressWindow*. Os dados de um objeto *ProgressInformation* são atualizados pelos *plugins* durante suas execuções e refletidos na *ProgressWindow*, pois este é seu observador.

Existem ainda outras classes de menor relevância no funcionamento da aplicação, e são basicamente elementos auxiliares. Todas as classes implementadas possuem informações explicativas em formato javadoc que serão publicadas ao final da implementação total deste trabalho.

5.2 Experimentos

Nesta seção serão apresentados os métodos e resultados dos primeiros experimentos realizados com usuários interagindo com uma versão funcional,

objetivando avaliar características de qualidade de uso de *Bleem Bloom* com a utilização de métricas discutidas a seguir.

5.2.1 Características e Funcionalidades Avaliadas

Os experimentos foram realizados para avaliar a qualidade de *Bleem Bloom* quanto sua usabilidade. Por isso foram feitas avaliações no subconjunto de funcionalidades implementadas até o momento através da medição de algumas características. Essas medidas foram fornecidas pelos usuários validadores e também obtidas a partir da observação de suas interações com o produto.

As características consideradas foram:

- **Facilidade de Uso:** Indica se as funcionalidades são de fácil utilização.
- **Intuitividade:** Indica se a interface gráfica da aplicação é intuitiva suficiente para que os usuários possam deduzir para que servem seus componentes e como utiliza-los para alcançar um objetivo.
- **Eficácia:** Característica relacionada à quantidade de erros necessários para que o usuário realize uma tarefa corretamente pela primeira vez.
- **Eficiência:** Característica que indica se o software permite os usuários realizarem tarefas no menor tempo possível.

As funcionalidades avaliadas foram:

- **Manipulação do corpus:** Todas as operações que podem ser realizadas em tags (criar, renomear e remover) e arquivos (importar, desclassificar e remover) disponíveis para manipular a coleção de arquivos do corpus.
- **Manipulação de cena:** Todas as operações que podem ser realizadas em cenas, desde a criação até a manipulação de blocos no *timeline* (criar novo bloco, editar conteúdo, remover e definir limites). A utilização da pesquisa de blocos similares também foi validada.

Roteiros de utilização de funcionalidades específicas não foram considerados por não terem sido implementados até o momento da realização dos experimentos aqui descritos e por não terem sido definidas por esta pesquisa. Procedimentos

comuns como abrir arquivo e salvar também foram descartados da validação por não serem de caráter exclusivo de *Bleem Bloom*.

5.2.2 Método de Avaliação

Os usuários avaliadores foram orientados a atribuir notas à facilidade de uso e intuitividade de cada funcionalidade avaliada. Devido a possibilidade das intensidades dos sentimentos em relação a utilização da aplicação ser algo muito pessoal e variar bastante entre um usuário e outro, foi estipulada uma distribuição de valores e conceitos simétricos, com as mesmas quantidades de regiões positivas e negativas, além de uma região neutra. O objetivo disso foi manter as intensidades sincronizadas entre os usuários avaliadores. A escala ficou definida da seguinte maneira:

- de 0 à 1,9 para transmitir uma opinião extremamente negativa (-2);
- de 2 à 3,9 para transmitir uma opinião negativa (-1);
- de 4 à 5,9 para transmitir uma opinião neutra (0);
- de 6 à 7,9 para transmitir uma opinião positiva (+1);
- de 8 à 10 para transmitir uma opinião extremamente positiva (+2).

Os participantes foram informados basicamente da existência das funcionalidades avaliadas. O objetivo disso foi fazer os usuários descobrirem sozinhos como realizar as tarefas para que pudessem avaliar com mais precisão a intuitividade da interface gráfica da aplicação.

Para cada tarefa realizada, os usuários fizeram suas avaliações da facilidade e intuitividade, enquanto a própria aplicação registrava os tempos de interação. Os valores obtidos foram comparados com os de um usuário que já conhece a aplicação a fundo. Ficou por conta do entrevistador registrar a quantidade de erros cometidos pelos usuários.

A eficácia de cada funcionalidade foi medida através do registro da quantidade de erros cometidos pelos usuários até sua correta utilização pela primeira vez. Esses erros podem ser a utilização de um componente de interface que não esteja relacionado com a funcionalidade em avaliação ou a execução do procedimento em uma ordem que não resulte na conclusão do objetivo corretamente, e foram registrados pelo entrevistador. Os índices de erros indicam a

quantidade média de tentativas necessárias por usuário para conclusão da tarefa, e foram calculados como o total de erros ocorridos em cada funcionalidade dividido pela quantidade de usuários. Funcionalidades com grandes índices de erro provavelmente podem melhorar.

Para a medição da eficiência, foram inseridas instruções no código-fonte de *Bleem Bloom* para registrar o tempo utilizado pelos usuários avaliadores durante a realização das tarefas. Com isso foi possível calcular os percentuais de melhora de tempo necessário para equiparação dos usuários avaliadores com um usuário experiente. Considerando Te como o tempo do usuário experiente e Ta como o tempo dos usuários avaliadores, o percentual de melhora Pm foi calculado como:

$$Pm = 100 - \frac{100 * Te}{Ta}$$

O cálculo de Pm foi feito, para cada funcionalidade, em três situações de Ta : no **pior caso**, onde foi considerado o tempo do usuário mais lento e obtidos valores mais altos; no **melhor caso**, onde foi considerado o tempo do usuário mais rápido e obtidos valores mais baixos; e no **caso médio**, onde foi considerada a média de tempo calculada a partir do somatório dos tempos dos usuários avaliadores dividido pela quantidade total de usuários, e indica a melhora média necessária para equiparação.

Com o objetivo de conhecer opiniões além do que os experimentos se propuseram a dizer, ao final de cada etapa foram feitos os seguintes questionamentos aos usuários:

- As operações propostas são suficientes suficientes?
- Os procedimentos de utilização são adequados?
- O que pode melhorar?

5.2.3 Resultados

As funcionalidades foram declaradas, em geral, suficientes e fáceis de serem utilizadas, porém algumas delas poderiam ser mais autoexplicativas. A **Tabela 2** a seguir mostra, para cada funcionalidade avaliada, as médias obtidas para facilidade, intuitividade e os percentuais de melhora de tempo necessários para equiparação dos usuários avaliadores com um usuário experiente, assim como os índices de erro por usuário de cada funcionalidade.

Funcionalidade	Facilidade	Intuitividade	Melhora Necessária (%)			Erros
			Melhor	Média	Pior	
Criar TAG	9,18	8,18	66,66	87,87	93,93	1,75
Renomear TAG	9,8	8,58	0	91,79	97,16	0
Remover TAG	9,8	8,58	33,33	89,61	94,59	0
Classificar Arquivo	8,3	7,08	59,26	85,90	88,04	0,75
Desclassificar Arquivo	9,8	9,8	0	86,66	94,93	0
Remover Arquivo	9,4	9,4	+40,00	88,44	94,50	0
Nova Cena	9,6	9,46	62,50	88,96	93,39	0,08
Novo Bloco	9,4	8,18	+23,07	69,41	85,22	1,25
Editar Bloco	9,8	10	+40,00	79,59	93,33	0,5
Definir Limites de Bloco	9,6	9,8	73,33	93,98	97,63	0,25
Remover Bloco	9,8	10	0	70,00	84,21	0
Localizar Blocos Similares	9,8	9,8	0	66,26	75,86	0,25

Tabela 2 Resultado dos experimentos

Em alguns casos, mesmo sem um contato prévio com a aplicação, os usuários avaliadores obtiveram tempos menores que um usuário experiente em situação normal⁶. Nesses casos os valores aparecem com um sinal de positivo (+) à frente, e indicam o quanto mais rápido foram os usuários avaliadores em relação ao usuário experiente. O cálculo desse percentual de desempenho positivo (Dp) foi feito da seguinte maneira:

$$Dp = 100 - \frac{100 * Ta}{Te}$$

Como na maioria das situações, um dos maiores impactos para os usuários foi a realização de uma atividade desconhecida, que em média ficaram com um percentual de melhora necessário para equiparação de 91,4% nos piores casos, porém foram registradas reduções de até 85% do tempo gasto na utilização de uma funcionalidade na segunda vez em que foi realizada, o que mostra uma grande facilidade de aprendizado por parte dos usuários. Isso, assim como as médias na **Tabela 2**, mostra que a aplicação é satisfatoriamente fácil de aprender e utilizar, mas um pouco menos intuitiva apesar das notas também satisfatórias, e reforça a necessidade da publicação do manual do usuário. Também deve ser considerado que parte do tempo de interação entre os usuários e a aplicação registrado se deve

⁶ Situação normal significa um usuário realizando os procedimentos sem pressa mas já sabendo todos os valores utilizados, a localização dos arquivos e conhecendo o procedimento intimamente.

ao fato de que os usuários levaram muito tempo para decidir valores de parâmetros exigidos durante os procedimentos, que poderia ser desconsiderado em experimentos com métricas mais precisas por não representar um tempo específico da funcionalidade. Foi sugerido pelos usuários um manual também em vídeo, onde a visualização da execução dos procedimentos ajuda ainda mais no aprendizado.

Um outro motivo que reforçou a necessidade do manual do usuário foi a identificação de que as funcionalidades que obtiveram menor avaliação para intuitividade foram as que tiveram mais divergências de opiniões a respeito das descrições utilizadas em seus componentes de interface gráfica, inclusive a alta média de erros registradas para a tarefa “Criar TAG” está relacionada com a dificuldade de localização do gerenciador do corpus. Alguns usuários comentaram “não poderia ser mais adequado”, enquanto outros “não imaginei que seria isso” para a descrição textual de um mesmo elemento da interface gráfica. Muitas vezes não importava a descrição utilizada, bastando para o usuário apenas a explicação da utilidade daquilo.

A maioria dos usuários não apresentou o costume de verificar recursos como *tooltip texts*⁷, disponíveis em quase toda interface gráfica de *Bleem Bloom*, o que faz concluir que explorar mais textos expostos diretamente na interface gráfica torna a aplicação mais autoexplicativa.

A utilização de barras de ferramentas com botões na janela principal da aplicação pode transmitir melhor ao usuário a ideia do que o produto pode fazer, pois costuma ser mais intuitivo deduzir que existem 5 coisas diferentes que podem ser feitas ao visualizar 5 botões do que procurar por essas funcionalidade em menus.

Observando a interação dos usuários com *Bleem Bloom*, assim como suas sugestões, ficou claro que prover várias formas de utilizar uma mesma funcionalidade pode maximizar o potencial de usabilidade do produto, pois alguns usuários consideram intuitivas formas de utilização derivadas de costumes adquiridos em experiências com outros softwares, o que são condições muito pessoais. No lugar da utilização de menus de contexto, por exemplo, o clique duplo ou a tecla *enter* podem ser ótimos atalhos para edição de blocos no *timeline*, assim como a tecla *delete* pode ser um atalho óbvio para exclusão de blocos selecionados.

⁷ *Tooltip texts*, também chamados de *hints*, são textos flutuantes que fornecem dicas sobre um elemento da interface gráfica da aplicação quando o ponteiro do *mouse* é posicionado sobre ele.

6 CONCLUSÃO

Neste capítulo serão apresentadas as conclusões a respeito deste trabalho e os ganhos obtidos para o projeto 1P1V, assim como apontar as próximas direções para trabalhos futuros.

6.1 Objetivos e Contribuições

A concepção do produto proposto considerou os problemas de usabilidade e as necessidades de um público-alvo específico, visando construir uma ferramenta adequada e de fácil utilização. A pesquisa, as validações dos protótipos e os experimentos realizados com representantes de usuários do público-alvo mostraram que as principais ideias de usabilidade e funcionalidades elaboradas foram aprovadas, indicando que o caminho seguido converge para o sucesso do trabalho.

A nova interface gráfica baseada em *timeline* integra ainda mais os trabalhos do projeto, possibilitando que várias dimensões de análises sejam visualizadas conjuntamente, o que torna possível observações de maneiras que antes não eram possíveis, facilitando a descoberta de novos conhecimentos. As funcionalidades de detecção de similaridades a partir da comparação de objetos do *timeline* e a seleção de material para processamento por TAGs classificadoras, que refletem situações mais amplas, são características que tornam o trabalho original.

Durante o desenvolvimento do *Bleem Bloom* foram considerados os problemas estruturais de integração e manutenção do projeto. O modelo de arquitetura baseado em *plugins* é uma característica que torna o trabalho flexível, pois proporciona ao ambiente a capacidade de ser facilmente expandido através da implementação de novos *plugins*. Além de reduzir a complexidade de um grande projeto em módulos menores, tornando sua manutenção mais simples, isso também facilita o desenvolvimento distribuído de novos *plugins*, pois possíveis colaboradores não precisariam se preocupar com a aplicação principal, reduzindo conflitos em modificações.

O objetivo do desenvolvimento de um novo produto foi estimular a utilização prática dos trabalhos científicos, tirando-os dos laboratórios, e contribuir com o avanço e evolução do projeto musicológico “Um País, Um Violão”. Espera-se que as

novas funcionalidades, interface e estrutura interna ofereçam uma contribuição significativa para os trabalhos musicológicos, ampliando as possibilidades do projeto com novas formas de análise e expressão de resultados.

6.2 Trabalhos Futuros

Apesar da situação de várias funcionalidades do produto apresentado estar atualmente em funcionamento prático, este não pôde ter sua implementação concluída por limitações de tempo, o que também impossibilitou um maior número de reuniões e validações com os colaboradores. Isso fez com que prioridades precisassem ser atribuídas a cada elemento do ambiente. A implementação dos elementos incompletos, cujos baixos níveis de prioridade atribuídos são justificados a seguir, fica registrada como trabalho futuro.

- RF03.00 e UC03, referentes às funcionalidades específicas de cada foco de estudo, são elementos que exigem um estudo dedicado e mais aprofundado para cada situação, o que não seria possível neste trabalho que dá prioridade aos elementos gerais do ambiente. A não implementação das funcionalidades específicas, que ainda podem ser utilizadas no ambiente de Sena (2008), não impacta nos recursos gerais do ambiente, que são o grande diferencial deste trabalho.
- A não implementação de RF04.01, referente à barra de progresso de atividades, mesmo tornando a interface de *Bleem Bloom* menos comunicativa em termos de *feedback* ao usuário, não impede que o software seja utilizado.
- Aos recursos de detecção de sequências (UC04.02) e similaridade conjunta (UC04.03) foi dada menor prioridade pelo fato de que a similaridade de blocos, que é base para as duas, já se encontra implementada. Além disso, a complexidade de UC04.03 poderia reduzir o tempo que foi dedicado para a implementação de outras funcionalidade que torna, neste primeiro momento, o conjunto de funcionalidades atualmente disponíveis mais diversificado.
- A ausência de UC05.05, referente ao *player* acoplado ao *timeline*, apesar de dificultar o trabalho do pesquisador, não impede que descobertas sejam feitas a partir da análise visual dos materiais transformados em um *timeline*.

- A ausência da visualização específica dos blocos (UC05.06), apesar de bastante conveniente, não impede a utilização da ferramenta, que ainda conta com a visualização padrão de conteúdo de blocos.
- UC07, que trata da organização das janelas abertas, também recebeu baixa prioridade, uma vez que não foi uma solicitação direta dos colaboradores entrevistados. Mesmo assim ainda foi implementada a organização de janelas em cascata.
- A confecção e publicação da documentação do usuário (UC08) também teve sua prioridade rebaixada, pois de nada adiantaria a documentação sem a total implementação das funcionalidades do ambiente.

A detecção de sequências de objetos, o player e a barra de progresso são exemplos de recursos que já se encontram em estado de implementação avançado, faltando apenas suas adaptações às estruturas definitivas de objetos do *Bleem Bloom*. Após a implementação dos elementos pendentes, será gerada e publicada a documentação javadoc e um guia sobre como desenvolver novos *plugins* para o ambiente. Esses artefatos serão de grande importância aos desenvolvedores que venham a colaborar com o projeto.

Além da implementação dos elementos de menor prioridade citados, é desejável uma evolução mais detalhada dos trabalhos integrados na nova arquitetura, pois a integração atual destes trabalhos visa apenas mostrar as possibilidade de funcionamento do ambiente e auxiliar em seu desenvolvimento e testes. Tal esforço será de grande importância, pois os trabalhos podem sofrer melhorias significativas. Um exemplo disso é que alguns dos trabalhos adicionam informações nos arquivos contidos no corpus para, por exemplo, identificar marcações de tempo. Essas informações, que são eventos MIDI, são necessárias em suas análises mas não desejáveis em caráter permanente, pois podem interferir na reprodução e perda do conteúdo original desses arquivos.

Uma consideração importante e que é comum à todos os trabalhos seria a possibilidade de exportação de dados em XML, pois poderiam ser adaptados para utilização em outras ferramentas onde fossem possíveis outras formas de extrair informações. Ferramentas que utilizam inteligência artificial são bastante adequadas para isso, como o Weka.

Bleem Bloom também trata da integração de soluções, por isso também deve ser considerado como trabalho futuro um estudo dedicado a cada foco de estudo integrado, pois estes também estão sujeitos a melhorias e identificação de novas necessidades e funcionalidades específicas. Ficam registradas como sugestões os seguintes elementos:

Padrões Rítmicos, por Lima (2007).

- A exibição do grafo de equipolência explicado no capítulo 2 - que mostra os níveis de similaridade entre cada padrão rítmico localizado nos materiais - como uma funcionalidade específica é importante pois elimina a necessidade de utilização das barras de rolagem no *timeline* e exibe as relações entre cada padrão diretamente, além de preservar a análise e forma de visualização original do trabalho. Uma possível melhoria nesse grafo seria a exibição do padrão rítmico em seus nós.
- A janela de edição do conteúdo de seus blocos poderia ser semelhante a janela de transformação de materiais em articulações do trabalho original, pois cada símbolo seria manipulado de forma mais restrita e segura, assim como definem as regras da gramática - descritas no trabalho de Lima (2007) - que expressa as batidas rítmicas.
- Reconhecer batidas recorrentes como padrões sem precisar recorrer a uma base de pesquisa tornaria o trabalho independente da identificação manual de padrões e, consequentemente, mais poderoso. Uma base de padrões poderia ser mantida para utilização em uma feramente de síntese de acompanhamentos rítmicos. Neste caso poderia existir um botão "Adicionar à base" na janela de edição de conteúdo dos blocos.
- Uma forma de visualização específica para os conteúdos de seus blocos poderia ser a notação musical (partitura), pois proporcionaria maior conforto aos usuários. A visualização padrão de articulações expressas textualmente é importante no contexto funcional da aplicação, mas não é a mais familiar para músicos e musicólogos.

Análise Harmônica, por Scholz (2008).

- O objeto conteúdo de seus blocos poderia conter informações adicionais como o tipo de acorde (maior, menor, diminuto, com sétima, inversão, etc), cada nota formadora do acorde independentemente, o valor adquirido na função de utilidade durante a classificação e a regra de resolução harmônica disparada, caso aplicável. Esses elementos são descritos em detalhes na dissertação de Scholz (2008). Com isso a forma de comparação entre esses objetos poderia ser mais detalhada e eficiente nas funcionalidades que envolvem similaridade, além de proporcionar maior nível de detalhe aos pesquisadores.
- Uma possibilidade de visualização específica para o conteúdo de seus blocos poderia ser o desenho da cifra no braço do violão, pois é uma alternativa bastante amigável para usuários do meio musical.

Microrritmo e Microdinâmica, por Silvestre (2009) e Holanda (2010).

- Uma forma de visualização específica para dinâmica poderia ser uma barra vertical, onde sua altura representaria a intensidade do ataque aplicado às cordas do violão. No caso de desvios de tempo, uma barra vertical poderia marcar o que seria o momento exato, enquanto o evento seria desenhado com pequenos desvios para o lado esquerdo (no caso de antecipação) ou para o lado direito (no caso de atraso). A distância do evento para o momento central seria a tamanho do desvio. Essas representações seriam muito adequadas, pois são semelhantes a eventos MIDI em ferramentas musicais, como mostram os similares em SM07 no capítulo 4.
- Os gráficos dos resultados de suas análises, como apresentados em suas dissertações, são elementos que se adaptariam muito bem como funcionalidade específicas. Esses resultados, que são essenciais nesses trabalhos, seriam extraídos a partir de uma janela de análise selecionada (tempo, compasso ou frase) e também poderiam ser adicionados a uma base para utilização em uma ferramenta de síntese de dinâmica e desvios de tempo.

Apesar de protótipos terem sido examinados pelos usuários colaboradores e de experimentos parciais terem sido realizados, também deve ser considerado como

trabalho futuro a realização dos experimentos finais com todas as funcionalidades implementadas e um método de medição de eficiência mais preciso, pois nos experimentos realizados não foi possível desconsiderar o tempo desperdiçado com interações irrelevantes como, por exemplo, o tempo que o usuário leva para decidir um novo nome para uma TAG que está sendo renomeada. Só assim será possível uma avaliação mais realista dos resultados práticos proporcionados pelo ambiente.

Ainda que não se encontre em condições finais de uso, a utilização de *Bleem Bloom*, mesmo que com propósitos experimentais e de validação, será de grande importância na continuação do projeto e desenvolvimento das próximas versões, pois trata-se de um trabalho contínuo. O ciclo de vida de um software não se encerra após sua implementação ser concluída, pois essa é a parte mais mecânica do trabalho e que muitas vezes é limitada por condições que não permitem suas conclusões, enquanto as definições e utilização é que são sua essência fundamental.

REFERENCIAS BIBLIOGRÁFICAS

- GOUYON, F. "Microtiming in — Samba de Rod — Preliminary experiments with polyphonic audio." Proceedings of the 11th Brazilian Symposium on Computer Music, São Paulo, 2007.
- W3SCHOOLS. *W3Schools*. 2011. <http://www.w3schools.com/xml/> (accessed 2011 иил 17-Maio).
- WEIDERMAN, Nelson H., John K. BERGEY, Dennis B. SMITH, and Scott R. TILLEY. "Approaches To Legacy System Evolution." Carnegie Mellon University, Pittsburgh., 1997.
- VIGDER, Mark R., W. Morven GENTLEMAN, and John DEAN. "COTS Software Integration: State of the art." National Research Council of Canada - Software Engineering Group, 1996.
- ALMEIDA, Maurício Barcellos. "Uma introdução ao XML, sua utilização na internet e alguns conceitos complementares." Ci. Inf. v. 31, n. 2, Brasília, 2002, p. 5-13.
- ARAÚJO, Érica. "Sistemas Legados: Uma abordagem de uso por parte das empresas." Faculdade de Tecnologia da Zona Leste, São Paulo, 2009.
- BALABAN, M. et al. *Understanding Music with AI: Perspectives on Music Cognition*. Cambridge, MA: AAAI Press e MIT Press, 1992.
- BARBOSA, S.D.J., M.S. SILVEIRA, M.G. PAULA, and K. BREITMAN. "Supporting a Shared Undestanding of Communication-Oriented Concerns in Human-Computer Interaction: A Lexicon-Based Approach." Proceedings of EHCI-DSVIS'04: The 9th IFIP Working COnference on Engineering for Human-COMputer Interaction and the 11th International Workshop on Design, Specification and Verification of Interactive Systems, 2004, pp. 271-288.
- BARBOSA, Simone Diniz Junqueira, and Bruno Santana da SILVA. "Interação Humano-Computador." Elsevier, Rio de Janeiro, 2010.
- BEYER, H., and K. HOLTZBLATT. "Contextual Design." Interactions Magazine January/February, 1999.
- BISBAL, Jesús, Deirdre Lawless, Bing Wu, and Jane Grimson. "A Brief Review of Problems, Solutions and Research Issues." Computer Science Department, Trinity College, Dublin, Ireland, 1999.
- BOOTH, David, et al. "Web Services Architecture. W3C Working Group Note 11 February 2004." W3C, 2004.
- Bray, Tim, Jean Paoli, and C. M. Sperberg-McQueen. "Extensible Markup Language (XML) 1.0. W3C Recommendation 10 February 1998." W3C, 1999.

- BRESSON, J., C AGON, and G. ASSAYAG. “OpenMusic 5: A Cross-Platform Release of the Computer-Assisted Composition Environment.” In 10th Brazilian Symposium on Computer Music, Belo Horizonte, 2005.
- BRESSON, Jean. *OpenMusic*. 2011. <http://repmus.ircam.fr/openmusic/home> (accessed 2011 йил 1-Julho).
- BRODIE, M., and M. STONEBRAKER. “Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach.” Morgan Kaufmann Publishers, 1995.
- CUTHBERT, Michael Scott, and Christopher ARIZA. *Music21 Documentation Quick Start: Getting Started with music21*. 2011b. <http://mit.edu/music21/doc/html/quickStart.html> (accessed 2011 йил 1-Julho).
- CUTHBERT, Michael Scott, and Christopher ARIZA. “music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data.” Massachusetts Institute of Technology. International Society for Music Information Retrieval Conference ISMIR 2010, 2010.
- . *Music21: A toolkit for computer-aided musicology*. 2011a. 2011a. <http://mit.edu/music21/> (accessed 2011 йил 1-Julho).
- CAKEWAL. SONAR X1. 2011. <http://www.cakewalk.com/Products/SONAR/Two-Worlds-Collide/feature.aspx/Recording-in-the-studio> (accessed 2011 йил 1-Julho).
- CAMBOUROPOULOS, E., e G. WIDMER. *Automatic motivic analysis via melodic clustering*. Journal of New Music Research, 2000.
- CARVALHO, Sérgio Teixeira. “Um Design Patterns para a Configuração de Arquiteturas de Software.” Universidade Federeal Fluminense, Rio de Janeiro, 2001.
- CARSOLL, J.M. “Making Use: Scenario-Based Design of Human-Computer Interactions.” 2000.
- CHATLEY, Robert, Susan EISENBACH, and Jeff MAGEE. “Painless Plugins.” Department of Computing Imperial College, London, 2003.
- Comella-Dorda, Santigago, Kurt Wallnau, Robert C. Seacord, and John Robert. “A survey of Legacy System Modernization Approaches.” Carnegie Mellon University, 2000.
- COOPER, A., R. REIMANN, and D CRONIN. “About Face 3: The Essentials of Interaction Design.” John Wiley & Sons, New York, 2007.
- CROCKFORD, D. *The application/json Media Type for JavaScript Object Notation (JSON)*. 2006. <https://tools.ietf.org/html/rfc4627> (accessed 2011 йил 1-Julho).

da Silva, Edna Lúcia, and Estera Muszkat Menezes. "Metodologia de Pesquisa e Elaboração de Dissertação. 3^a edição." Universidade Federal de Santa Catarina, Florianópolis, 2001.

DE SOUZA, C. S., and C. F. LEITÃO. "Semiotic Engineering Methods for Scientific Research in HCI. In: J. M. Carroll (ed.) Synthesis Lecture on Human-Centred Informatics." Morgan & Claypool Publishers, Princeton, 2009.

DENZLER, Christoph, and Dominik GRUNTZ. "Design Patterns: Between Programming and Software Design." 30th international conference on Software engineering, 2008.

DESAIN, P., e H. HONING. "The formation of rhythmic categories and metric priming." Perception, 2003. 32(3):341–365.

DIXON, S., F. GOUYON, e G. WIDMER. "Towards characterization of music via rhythmic patterns." *Proc. of the 5th International Conference on Music Information Retrieval (ISMIR'04)*. Barcelona, 2004. 509–516.

FEIGENBAUM, Barry. *SWT, Swing or AWT: Which is right for you? What to consider when choosing a GUI tool kit for new applications*. 2006.
<http://www.ibm.com/developerworks/grid/library/os-swingswt/> (accessed 2011 иил 1-Julho).

FERRE, X., N. JURISTO, H. WINDL, and L. CONSTANTINE. "Usability basics for software developers." Software, IEEE , vol.18, no.1, 2001, pp.22-29.

Flora S. Tsai, Wenchou Han, Junwei Xu, Hock Chuan Chua. "Design and development of a mobile peer-to-peer social networking application." *Expert Systems with Applications* Volume 36, Issue 8, October 2009, 2009, p11077-11087.

GAMMA, Erich, Richard HELM, Ralph E. JOHNSON, and John M. VLISSIDES. "Design Patterns: Abstraction and Reuse of Object-Oriented Design." Springer-Verlag, London, 1993, 406-431.

GARLAN, David, and Mary SHAWY. "An Introduction To Software Architecture." Carnegie Mellon University, 1994.

HURON, David. *A Brief Tour*. 1999. <http://humdrum.org/Humdrum/guide01.html> (accessed 2011 иил 1-Julho).

- . *Humdrum FAQ*. 2001. <http://www.musiccog.ohio-state.edu/Humdrum/FAQ.html> (accessed 2011 иил 1-Julho).
- . *Representing Music Using **kern (I)*. 1999. <http://humdrum.org/Humdrum/guide02.html> (accessed 2011 иил 1-Julho).

HAIR JR, Joseph F., Mary WOLFINBARGER, David J. ORTINAU, and Robert P. BUSH. "Fundamentos de Pesquisa de Marketing." Bookman, 2010.

- Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. "The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1." 2009.
- HIX, D., and H. HARTSON. "Developing User Interfaces: Ensuring Usability Through Product And Process." John Wiley & Sons, New York, 1993.
- HOLANDA, Raphael. "Expressividade Musical na Bossa Nova: Análise e Síntese de Microandamento e Microdinâmica." Centro de Informática - Universidade Federal de Pernambuco, 2010.
- Holman, G. Ken. *What is XSLT?* 2000.
<http://www.xml.com/pub/a/2000/08/holman/index.html> (accessed 2011 йил 17-Maio).
- Johnson, Ralph, Erich Gamma, John Vlissides, and Richard Helm. "Design Patterns: Elements of Reusable Object Oriented Software." Addison Wesley, 1997.
- JSON. JSON. 2011. <http://www.json.org/> (accessed 2011 йил 1-Julho).
- KAZMAN, Rick & CARRIÈRE, S. Jeremy. "Playing Detective: Reconstructing Software Architecture from Available Evidence." Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.
- KNOPKE, Ian. "PerlHumdrum and PerlLilypond Toolkits for Symbolic Music Information Retrieval." International Conference of Music Information Retrieval, 2008.
- KORNSTADT, Andreas. "The JRing System for Computer-Assisted Musicological Analysis." Arbeitsbereich Softwaretechnik (SWT), Fachbereich Informatik, Universität Hamburg. International Symposium on Music Information Retrieval, Hamburg, 2001.
- KRUGER, C., and N. CROSS. "Solution-driven versus problem-driven design: strategies and outcomes." Design Studies 27, 2006, pp. 527-548.
- Krieger, Paulo. *Fatores para a Escolha de uma Linguagem de Programação*. 2009.
<http://www.baguete.com.br/columnistas/colunas/51/paulo-krieser/07/05/2009/fatores-para-a-escolha-de-uma-linguagem-de-programaca> (accessed 2011 йил 1-Julho).
- LAWSON, B. "How Designers Think: The Design Process Demystified 4a edição." Architectural Press, Oxford, 2006.
- LIMA, Ernesto. "DESCOBERTA AUTOMATICA DE CONHECIMENTO EM INTERPRETAÇÕES MUSICAIS: O CASO DO ACOMPANHAMENTO RÍTMICO AO VIOLÃO." Centro de Informática - Universidade Federal de Pernambuco, 2007.
- LOWGREN, J. & STOLTERMAN, E. "Thoughtful Interaction Design: A design Perspective on Information Technology." The MIT Press, Cambridge, 2004.

- NAVEDA, L, F GOUYON, C GUEDES, and M LEMAN. "Multidimensional microtiming in Samba music." In Proceedings of the 12th Brazilian Symposium on Computer Music, Recife, 2009.
- Nielsen, J. "Usability Engineering." Academic Press, New York, 1993.
- Muschamp, Paul. "An introduction to Web Services." BT Technology Journal, Springer Netherlands, 2004, p9-18.
- MAYER, Johannes, Ingo MELZER, and Franz SCHWEIGERT. "Lightweight Plug-In-Based Application Development. Objects, Components, Architectures, Services, and Applications for a Networked World." Lecture Notes in Computer Science Vol 2591, Berlin / Heidelberg, 2003, p87-102.
- Mayhew, D. "The Usability Engineering Lifecycle: A practitioner's handbook for user interface design." Morgan Kaufmann, San Francisco, 1999.
- Madruga, Marcos, Thaís Batista, and Luiz Affonso Guedes. "Uma Arquitetura P2P Baseada na Hierarquia do Endereçamento IP." Universidade Federal do Rio Grande do Norte. Simpósio Brasileiro de Redes de Computadores, 2006.
- MCGUINESS, A. "Microtiming Deviations in Groove." Australian National University, 2005.
- PALMER, Caroline. "Music Performance." In: LIMA, Ernesto T. 'DESCOBERTA AUTOMÁTICA DE CONHECIMENTOS EM INTERPRETAÇÕES MUSICAIS: O CASO DO ACOMPANHAMENTO RÍTMICO AO VIOLÃO'., 1997.
- Papazoglou, M.P., P. Traverso, S. Dustdar, and F. Leymann. "Service-Oriented Computing: State of the Art and Research Challenges." Computer , vol.40, no.11, 2007, pp.38-45.
- PARDO, B., e W. BIRMINGHAM. "Automated partitioning of tonal music." Electrical Engineering and Computer Science Department, University of Michigan, 1999.
- PRATES, R.O., and S.D. BARBOSA. "Avaliação de Interfaces de Usuário – Conceitos e Métodos." In: Anais do XXVIII. Congresso da Sociedade Brasileira de Computação, JAI'2003 (CD-ROM), Campinas, 2003.
- SCHOLZ, Ricardo. "“CHOCONUT: Um processo para reconhecimento de acordes em sequências capturadas por violões MIDI”." Centro de Informática - Universidade Federal de Pernambuco, 2008.
- SEI. *Architecture*. 2011. <http://www.sei.cmu.edu/architecture/start/glossary/> (accessed 2011 1-Julho).
- SENA, Valmir. "Ferramenta Integrada de Aquisição e Análise do Violão Brasileiro." Centro de Informática - Universidade Federal de Pernambuco, 2008.
- SILVESTRE, Fúlvio. "DESCOBERTA AUTOMÁTICA DE CONHECIMENTO EM INTERPRETAÇÕES MUSICAIS: MICROANDAMENTO e MICRORITMICA." Centro de Informática - Universidade Federal de Pernambuco, 2009.

- SILBERSCHATZ, Abraham, Peter GALVIN, and Greg GAGNE. "Sistemas Operacionais: Conceitos e Aplicações". Editora Campus, Rio de Janeiro, 2000.
- SHAFRANOVICH, Y. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. 2005. <https://www.tools.ietf.org/html/rfc4180> (accessed 2011 йил 1-Julho).
- SLOBODA, J. A. *Individual differences in music performance*. Trends in Cognitive Sciences, 2000.
- SOARES, Felipe S. Furtado, et al. "Adoção de SCRUM em uma Fábrica de Desenvolvimento Distribuído de Software." Centro de Informática – Universidade Federal de Pernambuco (UFPE), Recife, 2007.
- SOMMERVILLE, Ian. "Engenharia de Software. 6^a ed." Addison Wesley, São Paulo, 2004.
- SORENSEN, A., and A. BROWN. *An introduction to JMusic*. 2011. <http://jmusic.ci.qut.edu.au/jmtutorial/t2.html> (accessed 2011 йил 1-Julho).
- SORENSEN, A., and A. R. BROWN. "Introducing jMusic." In Brown, A.R. and Wilding, R., InterFACES: Proceedings of The Australasian Computer Music Conference, Brisbane, 2000, pp. 68-76.
- SORENSEN, Andrew, and Andrew BROWN. *JMusic Tutorial: Lets make a noise*. 2011. <http://jmusic.ci.qut.edu.au/jmtutorial/x104.html> (accessed 2011 йил 1-Julho).
- REZENDE, Denise Alcides. "Engenharia de Software e Sistemas de Informação. 3 ed." Brasport, Rio de Janeiro, 2005.
- REYNOLDS, Mathew, Richard BLAIR, Jonathan CROSSLAND, and Thearon WILLIS. "Beginning Visual Basic .NET." Pearson Education do Brasil, São Paulo, 2002.
- ROADS, C. et al. *The Computer Music Tutorial*. Cambridge, MA: The MIT Press, 1996.
- ROLAND. "TD-4 Owner's Manual." Roland, 2009.
- ROLAND. "TD-9 Owner's Manual." Roland, 2008.
- ROSSON, M.B., and J.M. CARROLL. "Usability Engineering: Scenario-Based Development of Human-Computer Interaction." Morgan Kaufmann Publishers, San Francisco, 2002.

APÊNDICE A - DADOS COLETADOS

Dados coletados durante entrevistas:

- DC01 (E2): Necessidade de integrar os projetos atuais em um ambiente único.
- DC02 (E2): Permitir trabalhar também com arquivos de formato diferente de MIDI.
- DC03 (E2): Possibilidade de estender o ambiente com novos focos de estudo.
- DC04 (E2): Possibilidade de tocar uma execução gravada ou trechos dela.
- DC05 (E2): Escolher entre tocar um material utilizando síntese de microdinâmica e/ou microrítmo, ou de forma flat (sem dinâmica nem desvios de tempo).
- DC06 (E2): Utilização de *softwares* diversos para auxílio ao ensino.
- DC07 (E2): Observador de respiração, pois esta gera tensão e dificulta na concentração do intérprete, ocasionando erros.
- DC08 (E2): Observador de dedilhado.
- DC09 (E3a): Diferenças e semelhanças entre executantes quanto à dimensões de estudo como, por exemplo: Padrões rítmicos, desvios de tempo e variações de dinâmica.
- DC10 (E3a): Revelar características relativas a épocas quanto à focos de estudo como, por exemplo: Padrões rítmicos, desvios de tempo e variações de dinâmica.
- DC11 (E3a): Saber em que momentos ou condições ocorrem um tipo específico de situação ou evento.
- DC12 (E3b): Utilizar uma interface gráfica e não precisar utilizar manualmente comandos complicados.
- DC13 (E3b): O *software* deve ser fácil de ser manipulado para que responda questões a respeito de uma dimensão de estudo.
- DC14 (E3b): Possibilidade de seleção de um "cenário/situação" para análise: "Quero estudar a microdinâmica de um intérprete específico... ou de uma década".

- DC15 (E3c): Régua de tempo ajudam na visualização de dados onde o momento ou duração são relevantes.
- DC16 (E3c): Ao longo de uma gravação, rótulos indicando o significado dos trechos facilitam análise do material.
- DC17 (E3c): Representação ou classificação de objetos estruturais e resultados utilizando cores proporciona uma assimilação fácil e eficiente para refletir frequências, grupos de objetos, etc. Outros elementos expressivos como números, formas geométricas ou texturas também podem ser utilizados com o mesmo propósito.
- DC18 (E3c): Gráficos são úteis para representar dados numéricos.
- DC19 (E3c): Notação musical é mais natural para o público alvo.
- DC20 (E2): Comparação entre partitura original e o que de fato foi tocado.
- DC21 (E3a): A precisão do executante.
- DC22 (E3b): Selecionar uma obra (em partitura) e iniciar a execução.
- DC23 (E3c): Exibir marcações, na partitura, de notas erradas e imprecisões de dinâmica e de tempo.
- DC24 (E2): Leitor de musculatura.
- DC25 (E3a): Saber a precisão e origem de falhas na dinâmica aplicada durante uma execução.
- DC26 (E3b): Selecionar a gravação da execução para ser comparada com a execução ideal (partitura).
- DC27 (E3c): Exibição simultânea da dinâmica ideal (partitura) e da realizada, para comparação baseada na informação midi e/ou eletrodos.
- DC28 (E2): Comparação entre intérpretes.
- DC29 (E3a): Quanto do mestre o aluno herda?
- DC30 (E3b): Selecionar execuções gravadas do professor e do respectivo aluno para comparação.
- DC31 (E3c): Frases, padrões de dinâmica, desvios de tempo (DEDUZIDO).
- DC32 (E2): Treinador de leitura de partitura.
- DC33 (E3a): Capacidade de leitura do aluno (DEDUZIDO).
- DC34 (E3b): Selecionar a obra, a quantidade de compassos para serem exibidos por vez e o tempo de exibição desses trechos.

- DC35 (E3c): Exibir em notação musical (partitura), com opção de metrônomo, a quantidade de compassos escolhida por vez.
- DC36 (E2): Indicador da precisão de um músico quanto a manter o andamento durante execuções sem utilização de metrônomo.
- DC37 (E3a): Saber onde um executante acelera e desacelera durante uma execução, pois mudanças de andamento que não retornam a velocidade original caracterizam erro.
- DC38 (E3b): Selecionar uma gravação de execução e escolher a opção para utilizar a funcionalidade.
- DC39 (E3c): Exibir, ao longo da execução, onde seriam as batidas das unidades de tempo. As variações entre as distâncias de um tempo até outro indicam desvios de tempo.

Dados coletados nas validações dos protótipos:

- DC40 (P): Formato básico de apresentação em *timeline* aceita. O protótipo fornece boa visualização das informações em uma disposição familiar no meio musical;
- DC41 (P): Funcionalidades sugeridas como similaridade de blocos, detecção de sequências, análise conjunta e funcionalidades específicas agrupadas em menu aceitas, pois parecem ser boas maneiras de utilização;
- DC42 (P): Representações das informações podem ser mais próximas da notação musical convencional, assim como uma visualização mais vertical, semelhante a páginas de partitura, são mais familiares para os músicos;
- DC43 (P): A existência de extratores de elementos melódicos ofereceria mais possibilidades de análise e maior variedade de conclusões quando analisado conjuntamente a outros resultados;
- DC44 (P): É importante a possibilidade de analisar vários tipos de arquivos (mid, wav, mp3, etc);
- DC45 (P): Possibilidade de, ao longo do tempo, rotular trechos do material, dando significado a eles.

- DC46 (P): Na janela de Seleção de Classificadores, permitir seleção de arquivos de dois modos: União dos conjuntos, que retorna uma união do conjunto de arquivos de cada classificador independentemente, e; Interseção dos conjuntos, que seleciona os arquivos que são classificados, ao mesmo tempo, por todos os classificadores selecionados, e;
- DC47 (P): Alguns integrantes do grupo focal sentiram dificuldade em compreender as mudanças de tonalidades de cores ao selecionar um dos blocos similares como elemento raiz de comparação.
- DC48 (P): O título do menu “Funcionalidades Específicas” poderia ser algo menor: “Funções” ou “Plugins”.
- DC49 (P): As opções dentro do menu “Arquivo” poderiam ser escritas como “Nova cena...”, “Abrir cena...”, “Salvar cena” e “Salvar cena como...”.
- DC50 (P): As opções dentro do menu “Visualizar” poderiam ser escritas da seguinte forma: “Relatório de saída” em vez de “Console de saída”; “Classificador de arquivos” ou “Gerenciador de arquivos” em vez de “Gerenciador de corpus”. A palavra “corpus” foi considerada muito técnica.
- DC51 (P): Considerar cores diferentes para cada *track* de *plugin*. No caso do rotulador (*plugin* que permite o usuário rotular livremente trechos da música no *timeline*), considerar a possibilidade de utilizar cores diferentes para cada bloco.
- DC52 (P): Na janela de entrada de parâmetros do *plugin* de padrões rítmicos, deixar explícito o significado de “Fator de quantificação” e “Batidas por compasso”.
- DC53 (P): Sugestão de uma janela comparativa para praticar, que mostra as diferenças entre um material e o que o usuário tocou.
- DC54 (P): Ao utilizar o *player* em uma cena aberta, destacar os blocos que estão sendo tocados no momento.
- DC55 (P): Visualização do que está sendo tocado pelo *player* em um braço de violão.
- DC56 (P): Opção de utilizar o *player* com ou sem metrônomo.
- DC57 (P): Intensificar diferenças na tonalidade de cor em funcionalidades que envolvem similaridade e modificações nas cores dos blocos.
- DC58 (P): Possibilidade de aplicar zoom no *timeline*.

- DC59 (P): Incluir dicas em todos os componentes possíveis da interface.

Análise das anotações de avaliação com protótipos

Os dados coletados em avaliações de usabilidade através dos protótipos foram estudados e a eles dadas as seguintes interpretações:

- Os dados DC40 e DC41 a respeito das ideias iniciais são *feedbacks* positivos e confirmam o rumo do projeto.
- As anotações do dado DC42 serão convertidas em uma funcionalidade que permite que o conteúdo de um bloco possa ser visualizado como notação musical ou de uma forma mais específica, caso disponível.
- O dado DC43 futuramente poderá ser convertido em um *plugin* com capacidade de extrair ou detectar melodias ou frases, oferecendo os benefícios apontados.
- DC44 está registrada nos dados coletados como DC04, no capítulo 4.1.3, e apenas não pôde ser observado nos protótipos pelo fato de nenhum dos trabalhos atualmente desenvolvidos serem projetados para suportar formatos diferentes de MIDI. Por esse motivo a implementação do reconhecimento de outros formatos teve menor prioridade.
- DC45 também está presente no conjunto de dados coletados como DC16, no capítulo 4.1.3, mas foi inicialmente interpretado com imprecisão. Este dado poderá ser convertido em um *plugin* adicional que retorna um *track* de resultados vazio, onde poderão ser adicionados objetos ao longo do *timeline* para representar trechos e estes serem rotulados.
- O dado DC46 foi acatado e adicionado como parte do projeto.
- DC47 resultou na simplificação do destque de cores da funcionalidade sobre similaridade, por não ter sido considerada complicada e não despertado interesse relevante para os entrevistados da forma como proposto originalmente. Agora, para para mudar a referencia principal de comparação, deve ser realizada uma nova busca.
- Sobre DC48 foi considerado utilizar temporariamente o título "Análises".
- DC49 foi acatado para o produto.

- Sobre DC50 o título "Relatório de saída" foi aceito, enquanto "Gerenciador de materiais" foi considerado para o corpus, pois "Gerenciador de arquivos" é muito genérico quando considerado que “materiais” se refere ao conteúdo específico do projeto.
- A sugestão de DC51, que fala sobre cores no *timeline*, será parcialmente adotada, pois aplicar cores nos blocos do *timeline* implica conflito com as funcionalidades relacionadas a similaridades, e podem dificultar a visualização das tonalidades de cores para usuários daltônicos.
- DC52, que fala sobre explicações a respeito de parâmetros, foi aceito mas considerado para outro momento. A princípio o desenvolvedor de cada *plugin* é responsável pela documentação de uso e comunicabilidade em elementos específicos de seus trabalhos.
- O propósito de DC53 não está diretamente relacionado ao escopo do projeto.
- O dado DC54, sobre destacar objetos sendo reproduzidos, foi acatado para fazer parte do ambiente.
- Será estudada uma forma de aplicar DC55, sobre visualização em braço de violão, em uma próxima versão.
- O dado DC56, sobre utilizar ou não um metrônomo durante reprodução do material, será estudado para aplicação em próximas versões pois, a princípio, existem conflitos com a estrutura dos materiais que já possuem a batida do metrônomo em sua estrutura ou utilizam dados adicionais inseridos pelos programas dos trabalhos. Podem ser consideradas modificações nos algoritmos originais dos trabalhos.
- O dado DC57 foi acatado e será implementado no ambiente. Será utilizada uma função não linear para definir a escala de cor das similaridades.
- O dado DC58, sobre aplicar *zoom* a uma cena, foi acatado para o ambiente.
- O dado DC59, a respeito de dicas nos componentes de interface, foi acatado para ser implementado no ambiente.

APÊNDICE B - REDUÇÃO EM CATEGORIAS

Para cada categoria produzida estão listadas as expressões recorrentes, o número de ocorrências e as indicações dos dados onde foram localizadas.

CT01 Foco de Estudo: Categoria que resume os trabalhos já realizados para o projeto 1P1V e a outros que venham a ser realizados futuramente por necessidade. Cada trabalho possui seu tema básico e estuda características específicas da música. Um foco está relacionado a um tema específico, e não diretamente ao trabalho desenvolvido.

- projetos atuais (1): DC01
- dimensão(ões) de estudo (4): DC03, DC09, DC10, DC13

CT02 Dimensão de Análise: Palavras ou expressões que são relacionados aos resultados, elementos específicos de cada trabalho, suas análises e conclusões.

- dinâmica (7): DC05, DC09, DC10, DC23, DC25, DC27, DC31
- desvios de tempo (5): DC05, DC09, DC10, DC31, DC39
- tempo (3): DC15, DC23, DC34
- padrões (3): DC09, DC10, DC31
- rítmicos (2): DC09, DC10
- MD/microdinâmica (2): DC05, DC14
- variações (3): DC09, DC10, DC39
- metrônomo (2): DC35, DC36
- andamento (2): DC36, DC37

CT03 Exibição e Representação: O conteúdo deste agrupamento está imerso no contexto de interface gráfica ou ligadas a possibilidades de representações das informações ou conteúdo das obras que fazem parte do corpus.

- exibir (3): DC23, DC35, DC39
- exibição (2): DC27, DC34
- exibidos (1): DC34
- durante (3): DC25, DC36, DC37

- ao longo (2): DC16, DC39
- dados (2): DC15, DC18
- possibilidade (3): DC03, DC04, DC14
- notação musical (2): DC19, DC35
- partitura (7): DC20, DC22, DC23, DC26, DC27, DC32, DC37
- ideal (2): DC26, DC27
- original (2): DC20, DC37
- precisão (3): DC21, DC25, DC36

CT04 Intérprete: Se refere a uma pessoa que interpretou uma das obras que foi gravada e adicionada ao corpus do projeto. A identificação e individualidade desses intérpretes são extremamente importantes para o estudo de microrrítmo e microdinâmica. A categoria também pode ser dirigida a futuros colaboradores ou em situações hipotéticas, para descrever procedimentos de usabilidade de funcionalidade.

- intérprete(s) (2): DC07, DC14, DC28
- executante(s) (3): DC09, DC21, DC37

CT05 Material: Categoria que representa referências aos arquivos contidos no corpus do projeto, que são interpretações de obras musicais e estão disponíveis para análise.

- execução(ões) (8): DC04, DC22, DC25, DC26, DC37, DC38, DC39, DC30, DC36
- material (2): DC05, DC16
- gravação (3): DC16, DC26, DC38
- obras (2): DC22, DC34
- midi (2): DC02, DC27
- trecho(s) (3): DC04, DC16, DC34

CT06 Ação Funcional: Indica ações com uma utilidade esperada, podendo ser aplicadas aos materiais disponíveis no corpus. Pode ser relacionada à formas de usabilidade ou mesmo, em algumas situações, funcionalidades propriamente ditas.

- escolher (2): DC05, DC38

- utilizar (3): DC12, DC38.
- selecionar (5): DC22, DC26, DC30, DC34, DC38
- análise (2): DC14, DC16
- tocar (2): DC04, DC05
- comparação (4): DC20, DC27, DC28, DC30

CT07 Quadro: Categoria que se refere ao conceito de refletir uma realidade ou situação, que pode ser uma combinação de variáveis.

- situação (2): DC11, DC14
- condições (1): DC11
- cenário (1): DC14

APÊNDICE C – REQUISITOS FUNCIONAIS

RF01.00	Nova Cena (transformação de arquivos em <i>timeline</i>)
Realizar transformação de materiais em dados para análise. Gerar visualização como um <i>timeline</i> , onde cada <i>track</i> contém os objetos de conteúdo de um foco de estudo, aqui chamados de blocos. O ambiente pode exibir várias janelas de transformações ao mesmo tempo, sendo uma para cada arquivo. Essas janelas são chamadas de cenas.	
RF01.01	Manipulação de Cena
Permitir realização de modificações nos blocos do <i>timeline</i> . As operações são excluir, editar e definir limites, além de criar um novo bloco.	
RF01.02	Salvar/Abrir Cena
Permitir que uma cena obtida através de transformação seja salva para que não seja necessário realizar sempre uma nova transformação, bastando apenas abrir o arquivo.	
RF01.03	Tocar
Permitir que o arquivo musical de uma cena, ou apenas trechos dele, seja tocado.	
RF01.04	Visualização Específica
Exibir o conteúdo de um bloco (de um <i>track</i> no <i>timeline</i>) de forma detalhada, em uma visualização específica do contexto do foco de estudo de origem.	
RF02.00	Análise de Similares
Possibilitar a análise de similaridades no conteúdo das cenas abertas, individualmente ou coletivamente. A base desse recurso é a comparação de objetos de um mesmo tipo para que seja detectado o grau de similaridade entre eles.	
RF02.01	Similaridade de Blocos
Destacar, em tons de uma cor, os blocos semelhantes a um outro bloco selecionado. A intensidade da cor revelará o grau de similaridade entre os objetos: Quanto mais forte, maior a similaridade.	
RF02.02	Similaridade de Sequências
Detectar sequências de blocos semelhantes. Cada sequência detectada terá sua própria cor	

para diferenciar uma sequência de outra.

RF02.03	Similaridade Conjunta
----------------	------------------------------

Detectar situações recorrentes de combinações de blocos em *tracks* diferentes que ocorram em um mesmo momento.

RF03.00	Funcionalidades Específicas
----------------	------------------------------------

Executar análises específica de cada foco de estudo para gerar resultados mais detalhados e responder questões mais específicas a respeito de um assunto.

RF04.00	Recursos e Ferramentas Auxiliares
----------------	--

Oferecer ganho em usabilidade e *feedback* ao usuário.

RF04.01	Barra de Progresso
----------------	---------------------------

Informar ao usuário o percentual de processamento realizado da atividade atual e permitir que a operação sendo realizada seja cancelada.

RF04.02	Informações
----------------	--------------------

Fornecer aos usuários informações detalhadas em formato textual e linguagem natural durante a utilização das funcionalidades, processamentos e sobre a interface gráfica.

RF04.03	Corpus e TAGs Classificadoras
----------------	--------------------------------------

Permitir armazenamento e organização da coleção de arquivos (corpus) do projeto musicológico. Possibilita ao usuário a classificação dos arquivos para que sejam mais facilmente selecionadas para processamento por meio de uma palavra classificadora.

APÊNDICE D - CASOS DE USO

UC01	Gerenciar Corpus
Pré-condição:	
Usuário escolhe a funcionalidade para gerenciar o corpus do projeto no menu "Visualizar->Gerenciador de materiais" e o programa exibe uma janela de manipulação. As opções de manipulação do corpus são: Criar, renomear e excluir classificador; Vincular, excluir e desvincular arquivo da <i>tag</i> .	

UC01.01	Criar TAG Classificadora
Pré-condição:	UC01 (gerenciador de corpus aberto)
Usuário clica no botão "Criar" e digita o nome do classificador no diálogo que será aberto. Para concluir, o usuário deve clicar em no botão "Ok".	

UC01.02	Renomear TAG Classificadora
Pré-condição:	UC01, existe ao menos uma <i>tag</i>
Usuário seleciona uma <i>tag</i> na lista e clica no botão "Renomear". Uma caixa de diálogo será aberta, onde o usuário deverá digitar o nome do classificador e clicar no botão "Ok" para concluir.	

UC01.03	Remover TAG Classificadora
Pré-condição:	UC01, existe ao menos uma <i>tag</i>
Usuário seleciona uma <i>tag</i> na lista e clica no botão "Remover" (abaixo da lista de classificadores). Esta operação não apaga os arquivos fisicamente os arquivos classificados pela <i>tag</i> removida.	

UC01.04	Importar Arquivos
Pré-condição:	UC01, existe ao menos uma <i>tag</i>
Usuário seleciona uma <i>tag</i> na lista e clica no botão "Importar". Será exibida uma caixa de diálogo para seleção de arquivos, onde o usuário deverá selecionar os arquivos desejados e clicar no botão "Ok".	

UC01.05	Desclassificar Arquivo
Pré-condição:	UC01, existe ao menos um arquivo
Usuário seleciona uma opção na lista de <i>tags</i> classificadoras para que seja exibida sua lista	

de arquivos. O usuário deve agora selecionar os arquivos que deseja desclassificar e clicar no botão “Desclassificar”.

UC01.06	Remover Arquivo
Pré-condição:	UC01, existe ao menos um arquivo
Usuário seleciona uma opção na lista de <i>tags</i> classificadoras para que seja exibida sua lista de arquivos. O usuário deve agora selecionar os arquivos que deseja remover e clicar no botão “Remover”. Esta operação não apaga os arquivos fisicamente.	

UC02	Obter Cena
Pré-condição:	
A obtenção de cenas pode ser realizada de duas maneiras: Por criação de uma nova cena, onde é realizada uma transformação de arquivo em <i>timeline</i> ; Por abertura de uma cena salva. Ao final do procedimento a cena é exibida em uma janela de <i>timeline</i> .	

UC02.01	Nova Cena
Pré-condição:	
Usuário seleciona a opção “Nova cena” no menu “Arquivo”. Será exibida uma caixa de diálogo que permite a seleção de arquivos, onde o usuário seleciona individualmente os arquivos que deseja transformar ou em grupo, selecionando <i>tags</i> . Após clicar no botão “Ok”, será exibido um diálogo para que sejam selecionadas as transformações desejadas. O usuário deve agora clicar no botão “Configurar” para selecionar os parâmetros de cada transformação e clicar no botão “Ok”.	

UC02.02	Abrir Cena
Pré-condição:	Uma cena salva
Usuário seleciona a opção "Abrir cena..." no menu "Arquivo". Será exibida uma caixa de diálogo onde o usuário deve selecionar o arquivo da cena salva e clicar no botão "Ok".	

UC03	Utilizar Funcionalidade Específica
Pré-condição:	
Esta descrição reflete apenas a forma de utilizar uma funcionalidade específica, pois seu roteiro depende de cada foco de estudo e da implementação por seu <i>plugin</i> . O usuário clica no menu “Análises” para que seja exibida uma lista de focos de estudos disponíveis (<i>plugins</i>), escolhe uma funcionalidade específica de um dos focos de estudo e realiza os	

passos necessários para conclusão de seu procedimento. Em seguida os resultados são exibidos.

UC04	Análise de Similaridades
Pré-condição:	Uma cena aberta
Usuário escolhe um dos seguintes tipos de similaridades, a partir de menu de contexto: Bloco, onde o usuário escolhe o grau de similaridade e o programa destaca os blocos similares ao selecionado; Sequência, onde o programa detecta sequências similares de blocos de objetos ou procura sequências similares à sequência selecionada, utilizando um grau de similaridade escolhido pelo usuário; Conjunta, onde o usuário seleciona um bloco ou sequência de blocos e o programa retorna situações semelhantes relacionadas a outros tipos de blocos, baseado em um grau de similaridade escolhido pelo usuário.	

UC04.01	Similaridade de Blocos
Pré-condição:	Uma cena aberta
O usuário seleciona a opção "Pesquisar similares" no menu de contexto (<i>popup</i>) de um bloco. Será exibida uma caixa de diálogo onde o usuário escolhe o grau de similaridade exigido e define o escopo da busca entre "Janela atual" e "Todas abertas". O usuário deve clicar no botão "Ok" para finalizar.	

UC04.02	Similaridade de Sequências
Pré-condição:	Uma cena aberta
<p>Alternativa 1: O usuário seleciona a opção "Detectar sequências" no menu de contexto (<i>popup</i>) de um bloco ou espaço vazio do <i>track</i> desejado. Será exibida uma caixa de diálogo onde o usuário escolhe o grau de similaridade exigido e define o escopo da busca entre "Janela atual" e "Todas abertas". O usuário deve clicar no botão "Ok" para finalizar.</p> <p>Alternativa 2: O usuário seleciona vários blocos em sequência dentro de um mesmo <i>track</i> segurando a tecla "Shift" ao clicar e escolhe a opção "Pesquisar similares" no menu de contexto de um dos blocos selecionados. Será exibida uma caixa de diálogo onde o usuário escolhe o grau de similaridade exigido e define o escopo da busca entre "Janela atual" e "Todas abertas". O usuário deve clicar no botão "Ok" para finalizar.</p>	

UC04.03	Similaridade Conjunta
Pré-condição:	Uma cena aberta
O usuário seleciona a opção "Similaridade conjunta" no menu de contexto (<i>popup</i>) de um bloco. Será exibida uma caixa de diálogo onde o usuário escolhe o grau de similaridade	

exigido e define o escopo da busca entre "Janela atual" e "Todas abertas". O usuário deve clicar no botão "Ok" para finalizar.

UC05	Manipular Cena
Pré-condição:	Uma cena aberta
Usuário abre o menu de contexto de um bloco e escolhe uma das opções de manipulação.	

UC05.01	Editar Conteúdo de Bloco
Pré-condição:	UC05
Usuário clica na opção "Editar conteúdo do bloco". Será exibida uma caixa de diálogo onde o usuário deve realizar as modificações desejadas e clicar no botão "Ok" para concluir. As possibilidades de modificações dependem do <i>track</i> ao qual o bloco pertence.	

UC05.02	Definir Limites de Bloco
Pré-condição:	UC05
Usuário clica na opção "Definir limites do bloco". Será exibida uma caixa de diálogo onde o usuário deve definir o momento inicial e final do bloco. O usuário deve clicar no botão "Ok" para concluir.	

UC05.03	Remover Bloco
Pré-condição:	UC05
Usuário clica na opção "Remover bloco". Será exibida uma mensagem de confirmação onde o usuário deverá clicar no botão "Sim" para realizar a operação.	

UC05.04	Novo Bloco
Pré-condição:	Uma cena aberta
Usuário abre o menu de contexto de um espaço vazio em um <i>track</i> e clica na opção "Novo bloco de conteúdo". Será exibida uma caixa de diálogo (a mesma de UC05.01) onde o usuário deve inserir o conteúdo desejado e clicar no botão "Ok". Será exibida uma caixa de diálogo (a mesma de UC05.02) onde o usuário deve definir o momento inicial e final do bloco e clicar no botão "Ok".	

UC05.05	Tocar
Pré-condição:	Uma cena aberta
Alternativa 1: O usuário executa dois cliques no bloco que deseja ouvir enquanto segura a	

tecla "Shift". O trecho do arquivo entre os momentos de limites do bloco será tocado.

Alternativa 2: O usuário clica no botão "Tocar/Pausa" no *timeline*. O arquivo será tocado do momento atual até o seu final, caso o botão "Tocar/Pausa" não seja clicado novamente.

UC05.06	Visualização Específica do Conteúdo de Bloco
Pré-condição:	UC05
Usuário clica na opção "Visualização específica". Será exibida uma janela contendo as informações desejadas. O usuário deve clicar no botão "Ok" para fechar a janela.	

UC05.07	Aplicar zoom
Pré-condição:	Uma cena aberta
O usuário desliza a barra de zoom no <i>timeline</i> para a esquerda, caso queira reduzir, ou para a direita, caso queira ampliar.	

UC06	Visualizar Janela de Informações
Pré-condição:	
Usuário escolhe a opção "Relatório de saída" no menu "Visualizar". Será exibida uma janela para exibição de mensagens textuais sobre o uso das funcionalidades.	

UC07	Organizar janelas
Pré-condição:	Cenas abertas
O usuário clica no menu "Janelas" e escolhe uma opção de organização entre "Organizar em cascata", "Dividir horizontalmente" e "Dividir Verticalmente".	

UC08	Obter Ajuda
Pré-condição:	
O usuário clica no menu "Ajuda" e escolhe a opção "Documentação" para exibir o manual de instruções do ambiente.	