



Pós-Graduação em Ciência da Computação

“Sistemas Interativos de Tempo Real para
Processamento Audiovisual Integrado”

Por

Jarbas Jácome de Oliveira Júnior

Dissertação de Mestrado



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

RECIFE, AGOSTO DE 2007



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

JARBAS JÁCOME DE OLIVEIRA JÚNIOR

**“Sistemas Interativos de Tempo Real para
Processamento Audiovisual Integrado”**

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

ORIENTADOR: PROF. PHD SÍLVIO MEIRA
CO-ORIENTADOR: PROF. PHD GEBER RAMALHO

RECIFE, AGOSTO DE 2007

Oliveira Júnior, Jarbas Jácome de
Sistemas interativos de tempo real para
processamento audiovisual integrado / Jarbas
Jácome de Oliveira Júnior. - Recife: O autor, 2007.
xvii, 126 p. : il., fig. tab.

Dissertação (mestrado) - Universidade Federal
de Pernambuco. CIn. Ciência da Computação, 2007

Inclui bibliografia e anexos.

1. Sistemas multimídia. 2. Sistemas interativos de
tempo real. 3. Processamento de áudio. 4.
Processamento de vídeo. 5. Interface gráfica. 6. Caixa
aberta. 7. Pure Data. I. Título.

006.7

CDD (22.ed.)

MEI2008-010

Aos meus pais e irmãos

Agradecimentos

Aos profs. Sílvio Meira e Geber Ramalho pela confiança, orientação e suporte;

Aos profs. examinadores Sílvio Melo e Etienne Delacroix pelas sugestões e extensão do prazo de entrega;

Ao C.E.S.A.R (Centro de Estudos e Sistemas Avançados do Recife) e Prêmio Rumos Itaú Cultural Arte Cibernética pelo suporte financeiro;

A Mavi Pugliese, Márcio Dahia, João Paulo Rolim (Nix), H. D. Mabuse, Paulinho (Seu Pereira), Gabriel Furtado, Igor Medeiros, Gabriel “Salsaman” Finch, Pedro Luna (Peloseco), Yellow, Hugh Sung, Pixel, Mathieu Bouchard, Tim Blechmann, Laurie Spiegel, Miller Puckette, Randy Jones, Bart van der Ploeg, Fred Collopy, Jodele, Spetto, Pedro Zaz, Alexandre D’albergaria, 1mpar, Roger S., Timba, Erms, Palmieri Ricardo, Glerm, Dado França, Ruth Slinger e toda a turma da VJBR, Pure Deposito, Re:combo, Pd-list e VJForums pelas contribuições diretas a essa pesquisa;

A Carlos Amarelo, Dani Azevedo e Aslan Cabral (Alvinho) pelo estímulo nos últimos meses da escrita;

A Paula Torreão e toda a equipe com quem trabalhei no C.E.S.A.R pelo grande apoio no início do mestrado;

A Gandhi, Dedeco, Diogo, Capiau, Mazza, Clylton Galamba, enfim, todos os amigos que fiz no CIn-UFPE e que sempre me ajudaram desde a graduação;

À família Negroove por todo o apoio e compreensão;

A Nessinha e nossa família pelo amor e carinho.

Sistemas Interativos de Tempo Real para Processamento Audiovisual Integrado

Autor: Jarbas Jácome de Oliveira Júnior

Orientador: Prof. PH.D. Sílvio Meira

Co-orientador: Prof. PH.D. Geber Ramalho

RESUMO

A popularização das artes mediadas por computador e, em especial, do visual-jóquei (VJ), estimula o desenvolvimento de novos sistemas multimídia de tempo real. Esta demanda vem sendo atendida por programas específicos para o controle de exibição de vídeos e efeitos como Resolume, ArKaos e diversas variantes; e com linguagens de programação visual como o Pd/GEM, Max/MSP/Jitter e diversas variantes. Entretanto, existe um compromisso entre usabilidade (facilidade de uso) e expressividade (limite técnico para se criar novos efeitos e funcionalidades) que ainda não é bem resolvido nestes sistemas e se torna problemático para muitos usuários como confirmado a partir de entrevistas respondidas por VJs.

Este trabalho tem duas contribuições principais. A primeira é um estudo analítico e original das tecnologias existentes, incluindo desafios de arquitetura para processamento audiovisual em tempo real. A segunda contribuição é o desenvolvimento do sistema ViMus, que introduz o conceito de “interface gráfica de caixa aberta” na tentativa de amenizar o problema de compromisso entre expressividade e usabilidade, fazendo uso de interfaces análogas às dos sistemas mais simples, porém pouco expressivos, como Resolume e oferecendo, gradativa e intuitivamente ao usuário, o acesso a uma máquina de processamento análoga à dos sistemas mais expressivos, porém mais difíceis de se aprender a usar, como o Pd/GEM.

Palavras-chave: sistemas interativos de tempo real, processamento audiovisual, visual-jóquei, VJ, Pure Data, Pd, GEM, processamento de vídeo, expressividade, usabilidade, interface gráfica, caixa aberta, multimídia, ViMus, música visual, *color music*

Real-time Interactive Systems for Integrated Audiovisual Processing

Author: Jarbas Jácome de Oliveira Júnior

Advisor: Prof. PH.D. Sílvio Meira

Co-advisor: Prof. PH.D. Geber Ramalho

ABSTRACT

Computer arts and, in particular, visual-jockey (VJ) recent interest growth stimulates new developments in real-time multimedia systems. This demand has been supplied by video and effect oriented programs like Resolume, ArKaos and many variants; and by visual programming languages like Pd/GEM, Max/MSP/Jitter and many variants. However, a trade-off between usability (ease of use) and expressiveness (technical limits to create new effects and new functionalities) is not solved in these systems, as verified through VJs interviews.

The work presented in this thesis has two main contributions. The first one is an analytical and original study of existing technologies, including challenges of software architecture for real-time audiovisual processing. The second contribution is the development of the ViMus system, which introduces the concept of “open box graphic interface” in attempt to brighten up the trade-off problem between expressiveness and usability. To achieve that, ViMus makes use of analogous interfaces from simplest systems, like Resolume, and offers gradual access to an analogous processing machine of more expressive systems like Pd/GEM.

Keywords: real-time interactive systems, audiovisual processing, visual-jockey, VJ, Pure Data, Pd, GEM, video processing, expressiveness, usability, graphic interface, open box, multimedia, ViMus, visual music, color music

Sumário

1.	Introdução	19
1.1	Apresentação.....	19
1.2	Motivação	21
1.3	Objetivos	21
1.4	Estrutura da dissertação	22
2.	Trabalhos relacionados	23
2.1	Contextualização histórica	23
2.2	Classificação dos <i>softwares</i> atuais	26
3.	Sistemas Orientados a Amostras de Vídeo e Efeitos	29
3.1	Apresentação.....	29
3.2	História.....	29
3.3	Considerações sobre interface humano-máquina.....	32
3.4	Análise funcional	39
4.	Sistemas Orientados a Fluxogramas	43
4.1	Apresentação.....	43
4.2	História.....	47
4.3	Considerações sobre arquitetura	58
4.3.1	Introdução	58
4.3.2	Acoplamento entre interface gráfica e máquina de processamento.....	59
4.3.3	Processamento de áudio e controle	61
4.3.4	Processamento de vídeo	65
4.4	Considerações sobre interface humano-máquina.....	72
5.	Usabilidade x Expressividade	83
5.1	Considerações sobre expressividade.....	83
5.2	Considerações sobre usabilidade	86
5.3	Conclusões	88
6.	ViMus	91
6.1	Apresentação.....	91
6.2	Tecnologias de infra-estrutura	91
6.3	Interface gráfica de caixa aberta	93

6.4	Arquitetura	96
6.4.1	Arquitetura geral	96
6.4.2	Arquitetura do módulo de interface de caixa aberta	98
6.4.3	Arquitetura da máquina de processamento	100
6.5	Processo de desenvolvimento	101
6.6	Resultados	102
6.6.1	Eficácia e eficiência da máquina de processamento	102
6.6.2	Usabilidade e expressividade da interface de caixa aberta	105
7.	Conclusões	109
7.1	Considerações finais	109
7.2	Trabalhos futuros	111
8.	Referências.....	113
	Anexo A: Questionário sobre <i>softwares</i> para VJs	123
	Anexo B: Questionário de Avaliação da Interface de Caixa Aberta	125
	Anexo C: Notações de diagramas	126

Lista de figuras

Figura 1-1 Exemplo de aplicação do ViMus em uma instalação artística: (a) é a configuração da instalação, (b) e (c) são possíveis interações com um espectador.....	19
Figura 1-2. Exemplo de aplicação do ViMus em um concerto musical.	20
Figura 2-1 (a) Lanterna mágica e os espetaculares lanternistas viajantes: (b) e (c).....	23
Figura 2-2 <i>Color organs</i> de Bainbridge Bishop (esquerda) e Alexander Wallace Rimington (direita).....	24
Figura 2-3 Foto da obra interativa "Recollections" de Ed Tannenbaum (1981).....	25
Figura 2-4 Telas de alguns <i>softwares</i> para VJ: os mais amigáveis (a) Resolume, (b) Arkaos e (c) Modul8; os mais poderosos (d) PD/Gem, (e) Max/Msp/Jitter e (f) vvvv.....	28
Figura 3-1 Bit Bopper (1988): sendo apresentado em programa de TV (esq. cima), interface gráfica (esq. baixo) e <i>rack</i> móvel (direita)	30
Figura 3-2 Exemplo de <i>mixer</i> A/B de áudio.	33
Figura 3-3 Fotos de exemplares de <i>mixers</i> A/B de vídeo específicos para VJs.....	34
Figura 3-4 Tela do AVmixer Pro da empresa Neuromixer: metáfora do <i>mixer</i> A/B	35
Figura 3-5 ArKaos VJ: metáfora do teclado de computador.	36
Figura 3-6 ArKaos VJ: metáfora do teclado musical.....	36
Figura 3-7 Resolume: metáfora do teclado de computador associado à de um <i>mixer</i> de 3 canais.	38
Figura 3-8 Modul8: metáfora de uma mesa de som de vários canais, associada à de um mixer A/B verticalizado (o canal A está posicionado acima do B e não ao lado).....	39
Figura 4-1 Foto de um <i>patch</i> (conjunto de ligações entre os módulos através dos cabos) montado em um sintetizador Moog.	44
Figura 4-2 Exemplo de um fluxograma (<i>patch</i>) em Pure Data.....	45
Figura 4-3 Exemplo de variação do estado interno de um objeto em um fluxograma do Pd.	46
Figura 4-4 Max Mathews	48
Figura 4-5 Laurie Spiegel trabalhando no GROOVE (foto de Emmanuel Ghent) e quatro imagens geradas pelo VAMPIRE.....	50
Figura 4-6 Linha do tempo do paradigma Max e sistemas relacionados.....	55
Figura 4-7 Linha do tempo dos principais Sistemas Orientados a Fluxograma.....	58
Figura 4-8 Arquitetura geral do Pd.....	60
Figura 4-9 Arquitetura geral do Desire Data	61
Figura 4-10 Exemplo de mensagens de controle para tocar notas musicais no Pure Data, utilizando saída MIDI.	62
Figura 4-11 Exemplo de fluxograma do Pd com mensagens de sinal.	63
Figura 4-12 Esquema representando um ciclo DSP (PUCKETTE, 1991b).....	64
Figura 4-13 Um fluxograma do GEM nas primeiras versões, de 1996 (ZMÖLNIG, 2004a).....	67
Figura 4-14 O fluxograma hierárquico do GEM, de 1997 até 2003. (ZMÖLNIG, 2004a)	68
Figura 4-15 Exemplo de 3 fluxogramas abertos no Pd.....	73

Figura 4-16 Abstrações de fluxogramas no Pd.....	75
Figura 4-17 Objetos de interface no Pd (1 ^a . quadro), Max (2 ^a . quadro) e vvvv (3 ^a . quadro).	76
Figura 4-18 Fluxograma de exemplo do Isadora semelhante a um SOAVE do tipo mixer A/B.	77
Figura 4-19 Exemplo de <i>patch</i> e <i>subpatch</i> com a opção de <i>graph-on-parent</i> desligada.	78
Figura 4-20 Mesmo exemplo com a opção de <i>graph-on-parent</i> do <i>subpatch</i> ligada.	78
Figura 4-21 Fluxograma "pai" quando a opção <i>graph-on-parent</i> do filho está ligada e sua janela fechada: os objetos de interface (ex.: <i>slide</i>) ficam visíveis no pai.	79
Figura 4-22 Fluxograma construído com Max/MSP/Jitter semelhante a um SOAVE <i>mixer A/B</i>	80
Figura 4-23 À esquerda interface de Pelletier e à direita interface tradicional de objetos e linhas (PELLETIER, 2005).....	81
Figura 5-1 Comparação dos principais softwares de VJ em termos de usabilidade e expressividade.....	89
Figura 6-1 ViMus: exemplo de rotação da caixa aberta.	94
Figura 6-2 ViMus: exemplo de " <i>zoom in</i> " na caixa aberta.	95
Figura 6-3 Esquema representando arquitetura geral do ViMus.	97
Figura 6-4 Classe abstrata <i>Machine</i> e possíveis implementações.....	97
Figura 6-5 Arquitetura do módulo de interface de caixa aberta do ViMus.	99
Figura 6-6 Estrutura dinâmica dos objetos em tempo de execução.	99
Figura 6-7 Diagrama de classes da máquina de processamento.	101
Figura 6-8 Exemplos de imagens geradas pela máquina do ViMus.	104
Figura 6-9 Posicionamento do ViMus com um cubo de painel de controle inspirado na interface de alguns SOAVEs.	106
Figura 7-1 Arquitetura do ViMus reutilizando máquina do Pd.	112

Lista de tabelas

Tabela 1 Alguns dos principais SOAVEs surgidos nos últimos anos (em ordem cronológica)	32
Tabela 2 Tabela de possíveis configurações de funcionamento da <i>threads</i> no Max/MSP	70
Tabela 3 Respostas ao questionário sobre a usabilidade da interface de caixa aberta do ViMus em comparação com a interface dos SOFs (baseada em janelas).	108
Tabela 4 Respostas ao questionário sobre softwares para VJs.	124

1. Introdução

1.1 Apresentação

Imagine que se queira montar uma instalação artística em que o espectador encontra uma TV ligada em algum canal aberto, exibindo uma novela, jornal ou outro programa qualquer que esteja sendo transmitido naquele horário. Ao lado da TV existe um microfone posicionado estratégicamente como que convidando o espectador a experimentar falar algo ou produzir um barulho qualquer para ver o que acontece. Imagine que quando o microfone captura algum som de intensidade maior, a imagem que está passando na TV sofre um efeito especial como saturação das cores ou deformação da imagem em três dimensões. Quanto mais alto a pessoa gritar, por exemplo, maior é a intensidade desse efeito na imagem (Figura 1-1).

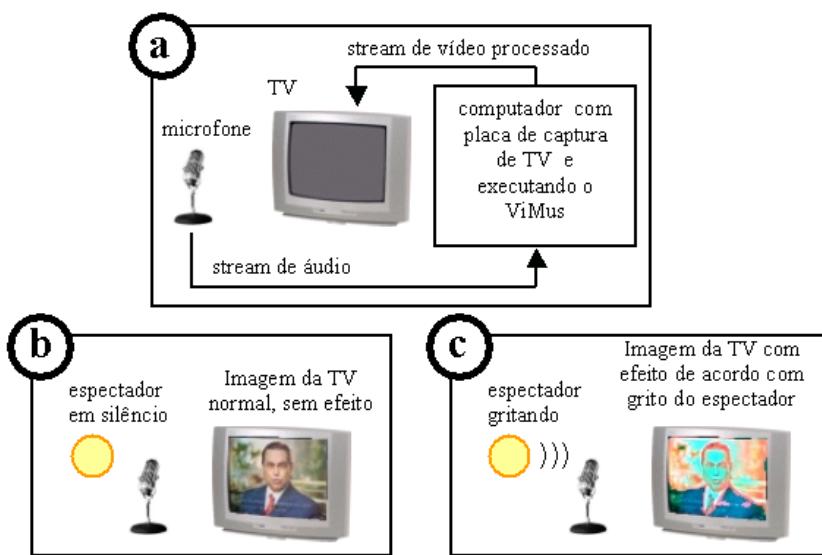


Figura 1-1 Exemplo de aplicação do ViMus em uma instalação artística: (a) é a configuração da instalação, (b) e (c) são possíveis interações com um espectador.

Imagine um outro cenário: um concerto musical em que além da iluminação tradicional existam projetores direcionados ao palco exibindo vídeos com efeitos sintetizados de acordo com o áudio produzido por alguns dos instrumentos (Figura 1-2). Imagine, por exemplo, que cada batida do caixa da bateria provoque um brilho

curto na imagem projetada; ou que o aumento de intensidade do som da guitarra faz com que a camisa branca do guitarrista fique avermelhada; ou ainda que a intensidade do som dos aplausos do público seja usada para controlar automaticamente a intensidade de um efeito de transição de um vídeo que estava sendo exibido para a imagem de uma câmera filmando o próprio público naquele momento.



Figura 1-2. Exemplo de aplicação do ViMus em um concerto musical.

Os objetos de estudo desta pesquisa são as tecnologias destinadas ao processo de criação, construção e execução de obras audiovisuais interativas como descritas nos cenários acima. Estudamos as tecnologias de *software* interativo para processamento de vídeo e áudio de forma integrada, ou seja, aquelas que permitem que a análise de áudio possa ser usada como parâmetro de um efeito de vídeo, por exemplo. Por interativo, entenda-se como a qualidade do *software* permitir a intervenção do artista e/ou espectador com a obra não apenas no momento em que está sendo criada, mas também no momento em que está sendo exibida (ou executada, no caso de obras performáticas).

Além disso, apresentamos o processo e o resultado do desenvolvimento de uma novo programa de computador chamado “ViMus”, cujo nome consiste em uma fusão das palavras “visual” e “música”. O ViMus é fruto de alguns anos de experimentação de técnicas para programação de sistemas de tempo real para processamento de vídeo, síntese de gráficos 2D/3D, análise de áudio, entre outras funcionalidades. Além dessas funcionalidades, este programa apresenta um novo conceito de interface, ao qual demos o nome de “interface gráfica de caixa aberta”,

que tenta amenizar alguns problemas existentes nas interfaces gráficas dos programas atuais.

1.2 Motivação

Existem vários trabalhos acadêmicos e iniciativas tanto comerciais quanto das comunidades de *software* livre, que objetivam o desenvolvimento de ferramentas para este domínio de aplicação. Entretanto, ainda assim, esta área apresenta muitas questões abertas e está em pleno desenvolvimento. As informações sobre o tema ainda estão dispersas e parece não haver muito acúmulo de conhecimento de um projeto para outro.

Outro fato evidente é a distância entre as tecnologias que já existem e a população de artistas que poderia utilizá-las, assim como o quase absoluto desconhecimento do grande público em geral. Uma das possíveis explicações estritamente técnicas para isto é que apesar de existir um número demasiado de tecnologias de *software* para o domínio de aplicação apresentado, as ferramentas mais poderosas em termos de possibilidades de criação de efeitos interativos são muito difíceis de se utilizar, enquanto as mais fáceis são limitadas a um número determinado de efeitos “pré-fabricados” e pouco personalizáveis.

1.3 Objetivos

Esta pesquisa possui dois objetivos primários. O primeiro consiste em construir uma narrativa que apresente as várias tecnologias de *softwares* existentes, contextualizando-as historicamente, formalizando o conhecimento e facilitando o trabalho de novos pesquisadores. Esta apresentação deve ser crítica, isto é, deve avaliar positiva ou negativamente os aspectos técnicos de cada ferramenta, incluindo suas interfaces gráficas do usuário, funcionalidades e arquitetura (quando possível). Um segundo objetivo é a investigação de fatores que dificultam o acesso e popularização das tecnologias estudadas e uma proposta de amenização destes problemas através de uma nova abordagem para a interface gráfica para este tipo de ferramenta.

A aplicação prática dos conhecimentos adquiridos no processo de construção da narrativa para o primeiro objetivo resultou no desenvolvimento da máquina de processamento do ViMus. O estudo de caso da nova abordagem criada para o segundo objetivo resultou no desenvolvimento da interface gráfica do ViMus.

O ViMus funciona como uma “camada de interface” mais amigável, personalizável mas ainda capaz de oferecer ao usuário a alta expressividade das tecnologias mais poderosas. O programa é, por tanto, um meta-sistema que tenta reunir características boas dos vários sistemas existentes. Por ser um projeto de código aberto, esperamos que a própria comunidade também se aproprie do desenvolvimento de novas interfaces, módulos e aplicações, personalizadas de acordo com os diversos interesses de cada usuário.

1.4 Estrutura da dissertação

O capítulo corrente apresenta uma introdução ao tema desta dissertação, a motivação para o desenvolvimento da mesma e sua principal finalidade.

O Capítulo 2 contextualiza esta pesquisa na história das tecnologias para apresentações audiovisuais artísticas. Além disso, introduz e classifica as principais tecnologias de *software* disponíveis atualmente para este domínio de aplicação.

Os capítulos 3 e 4 apresentam com maior profundidade cada uma das duas classes de sistemas interativos de tempo real para processamento audiovisual integrado: os SOAVEs (Sistemas Orientados a Amostras de Vídeos e Efeitos) e os SOFs (Sistemas Orientados a Fluxograma).

O Capítulo 5 apresenta uma definição mais aprofundada do problema “usabilidade x expressividade”. Neste capítulo apresentamos a existência de uma graduação crescente de expressividade e decrescente de usabilidade dos SOAVEs mais simples para os SOFs mais poderosos.

O Capítulo 6 descreve o processo de desenvolvimento, arquitetura, interface gráfica de caixa aberta e os resultados obtidos com o ViMus.

Finalmente, o Capítulo 7 traz as considerações finais sobre o trabalho desenvolvido e discute possíveis caminhos que podem ser seguidos a partir desta dissertação.

2. Trabalhos relacionados

2.1 Contextualização histórica

A utilização de técnicas de controle da luz para apresentações audiovisuais com fins artísticos é milenar. Como exemplo, podemos citar os fogos de artifício e o teatro de sombras; ambos são espetáculos audiovisuais nos quais também existe a possibilidade técnica do improviso, isto é, a criação no momento em que está ocorrendo a apresentação.

Um marco importante no percurso deste tipo de arte foi a invenção da câmera obscura e, por conseqüência, a “lanterna mágica” – ancestral do projetor de *slide* e projetor de cinema. Os “lanternistas viajantes” (THE BILL DOUGLAS CENTRE, 2002) eram artistas que viajavam o mundo fazendo apresentações nas quais projetavam imagens usando uma lanterna mágica portátil enquanto tocavam instrumentos musicais para acompanhar as histórias que contavam (Figura 2-1).



Figura 2-1 (a) Lanterna mágica e os espetaculares lanternistas viajantes: (b) e (c).

Outro ponto importante na história das tecnologias para apresentação audiovisual é a modalidade artística conhecida como “*color music*” que busca correspondência entre notas musicais e cores. Apesar de ser considerada por Thomas

Wilfred em (WILFRED, 1947) uma “perda de tempo” e má interpretação dos textos de Aristóteles referente ao assunto, os defensores da *color music* desenvolveram os primeiros teclados de cores (“*color organs*”) (Figura 2-2). O argumento no qual Wilfred baseia sua crítica é sólido e importante para esta pesquisa: a *color music* utiliza a cor e o movimento (mudança de cor), mas ignora um terceiro e fundamental elemento que é a forma.

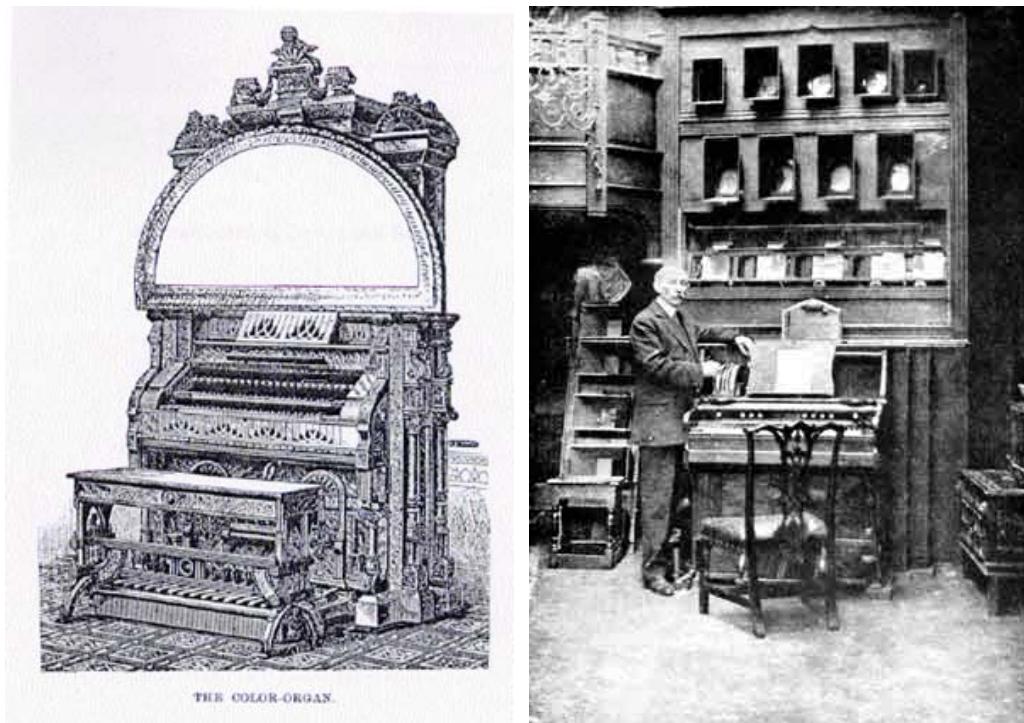


Figura 2-2 *Color organs* de Bainbridge Bishop (esquerda) e Alexander Wallace Rimington (direita)

Como apresentado em (BASBAUM, 2002) a música visual (“*visual music*”) é um território da arte específico para trabalhos que aspirem a uma sinestesia não-hierárquica, isto é, um “complementaridade”, entre música e imagem (não apenas cor e movimento, mas também a forma). A teoria da complementaridade foi proposta por John Whitney (WHITNEY, 1980) depois de vários anos produzindo obras audiovisuais sob influência dos filmes de Oskar Fischinger, quadros de Kandinsky e estudando as novas possibilidades oferecidas pelo computador digital.

As pesquisas em música visual estimulam o desenvolvimento de ferramentas que diminuam o desafio tecnológico a ser enfrentado pelo artista. Apesar da relativa carência de referências acadêmicas na área de ciência da computação sobre música

visual, parece estar havendo uma tendência de valorização do tema. No final de 2005, o assunto do *Computer Music Journal* foi *Visual Music*, no qual foram publicados cinco artigos relevantes para esta pesquisa: “Computer Music Video: A Composer’s Perspective” (RUDI, 2005), “Creating Visual Music in Jitter: Approaches and Techniques” (JONES & NEVILE, 2005), “Foundations of a Visual Music” (EVANS, 2005), “Interactive Visual Music: A Personal Perspective” (DANNENBERG, 2005) e “Digital Harmony of Sound and Light” (ALVES, 2005).

Um dos primeiros artistas a experimentar as novas possibilidades da computação digital especificamente para o processamento de vídeo em tempo real foi Ed Tannenbaum. Antes mesmo da popularização dos computadores pessoais, Tannenbaum construiu um dos primeiros sintetizadores digitais de vídeo através de uma readaptação da placa de um computador Apple II. Em 1981, o artista iniciou uma trajetória de exposições de sua obra interativa “Recollections” (TANNENBAUM, 2007), que consistia em um sistema de captura da imagem do espectador cujos movimentos deixam rastros coloridos (Figura 2-3).



Figura 2-3 Foto da obra interativa "Recollections" de Ed Tannenbaum (1981)

Além da arte interativa, a computação digital contribuiu para o desenvolvimento e recente popularização da modalidade artística protagonizada pelos chamados VJs (visual-jóqueis). A sigla VJ tem origem na abreviação de “vídeo-jóquei” (do inglês, “video-jockey”) e hoje tem pelo menos dois significados bem

distintos: um para designar os apresentadores de programas de TV para exibição de videoclipes musicais; e outro para designar os artistas que manipulam projeções de vídeos em eventos, seja acompanhando uma banda eletro/acústica, DJs (discotecários), seja “re-mixando” em tempo real o áudio dos próprios vídeos ou ainda simplesmente sem utilizar áudio, no caso de um espetáculo apenas visual. Neste segundo significado, costuma-se definir VJ como a abreviação de “visual-jóquei”, por ser mais abrangente que “vídeo-jóquei”.

Este último significado é o que nos interessa para esta dissertação, ou seja, utilizaremos a sigla VJ, como abreviação de “visual-jóquei”, para englobar as várias categorias de “VJs” no sentido de artistas audiovisuais. Isto inclui os que manipulam imagens projetadas apenas utilizando amostras de vídeo, os que utilizam apenas efeitos visuais, os que manipulam imagem e áudio simultaneamente (chamados DVJs, junção de DJ e VJ), e afins. A crescente importância desses artistas pode ser observada nas grandes comunidades da *Internet* como a VJForums e VJCentral que possuem milhares de membros e recebem dezenas de milhares de acessos diários.

A atividade do visual-jóquei se divide em aquisição e/ou produção de material visual, preparação da apresentação e execução da apresentação. O computador pode ser usado em qualquer uma dessas três etapas. Entretanto os sistemas de tempo real são utilizados principalmente na etapa de preparação e execução da obra.

Por tanto, a música visual, a arte interativa e a atividade do visual-jóquei são exemplos importantes de domínios de aplicação para os sistemas interativos de tempo real para processamento audiovisual integrado. No contexto desta pesquisa estudamos as principais tecnologias de *software* desenvolvidas para estas modalidades artísticas.

2.2 Classificação dos softwares atuais

Existem centenas de *softwares* destinados aos domínios de aplicação estudados nestas pesquisas. Para a maior parte destes programas, algumas informações podem ser encontradas em páginas especializadas como VJCentral (VJCENTRAL, 2006) e Audiovisualizers (EAGAN, 2006). Para os *softwares* de origem ligada a trabalhos acadêmicos como Pure Data/GEM e Max/MSP/Jitter, mais informações podem ser obtidas a partir de publicações científicas na área de computação musical.

Analizando a história das pesquisas acadêmicas nesta área ao lado da história das tecnologias surgidas no mercado, fica evidente a existência de duas diferentes abordagens para o problema de sistemas multimídia de tempo real. De um lado estão os acadêmicos, muitas vezes envolvidos em pesquisas de música eletroacústica, que criaram ferramentas complexas de se usar, inicialmente destinadas à experimentação de novas possibilidades musicais e que nos últimos anos passaram a possibilitar a interação com processamento de vídeo. De outro lado, existem os artistas visuais, amantes da vídeo-arte e da música eletrônica que criaram *softwares* fáceis de se usar, a fim de expandir seus limites técnicos de projeção de vídeos em eventos populares.

Assim, estas tecnologias de *software* podem ser classificadas em pelo menos dois grandes grupos. Existem de um lado os *softwares* mais complexos para o usuário final, orientados a linguagens de programação visual. A programação nestas linguagens consiste em posicionar nós ou módulos de processamento de mídia (áudio ou vídeo) cujas entradas e saídas devem ser interligadas pelo usuário a fim de se criar um fluxo de processamento ao qual a mídia é submetida. Podemos citar como exemplos de programas deste tipo o *Pure Data/GEM* (PUCKETTE, 2006b), *Max/MSP/Jitter* (ZICARELI, 2006) e *vvvv* (OSCHATZ *et al.*, 2006). Esta classe de *software* será aqui chamada de Sistema Orientado a Fluxograma (SOF).

De outro lado existem os *softwares* mais simples de se usar, orientados principalmente a disparo de amostras de vídeo como *Resolume* (KONIG & PLOEG, 2006), *Arkaos* (HINIC, 2006) e *Modul8* (SCHIMID & EDELSTEIN, 2006), que não permitem ao usuário a mesma flexibilidade e poder, mas são extremamente populares. Esta classe de *software* será aqui chamada de Sistema Orientado a Amostras de Vídeo e Efeitos (SOAVEs).

Ao analisarmos estes principais *softwares* em uma pesquisa preliminar, observamos a existência de um problema que na verdade é comum em tecnologias operadas diretamente por um ser humano: quanto mais funcionalidades e pluralidade de uso a tecnologia apresenta, pior é sua curva de aprendizado e sua usabilidade em geral (NIELSEN, 1994). Os programas mais simples como Resolume têm uma excelente curva de aprendizado, porém oferecem pouca flexibilidade para se criar novos tipos de efeitos, processamento e interação entre as mídias. Entretanto, programas como Pure Data (ou simplesmente Pd) e Max/MSP são extremamente expressivos, permitindo a construção de praticamente qualquer aplicação multimídia que se possa imaginar desde que seja modelável matematicamente. Porém, estes

softwares possuem uma interface com usuário mais complexa e que geralmente assusta os iniciantes (Figura 2-4).

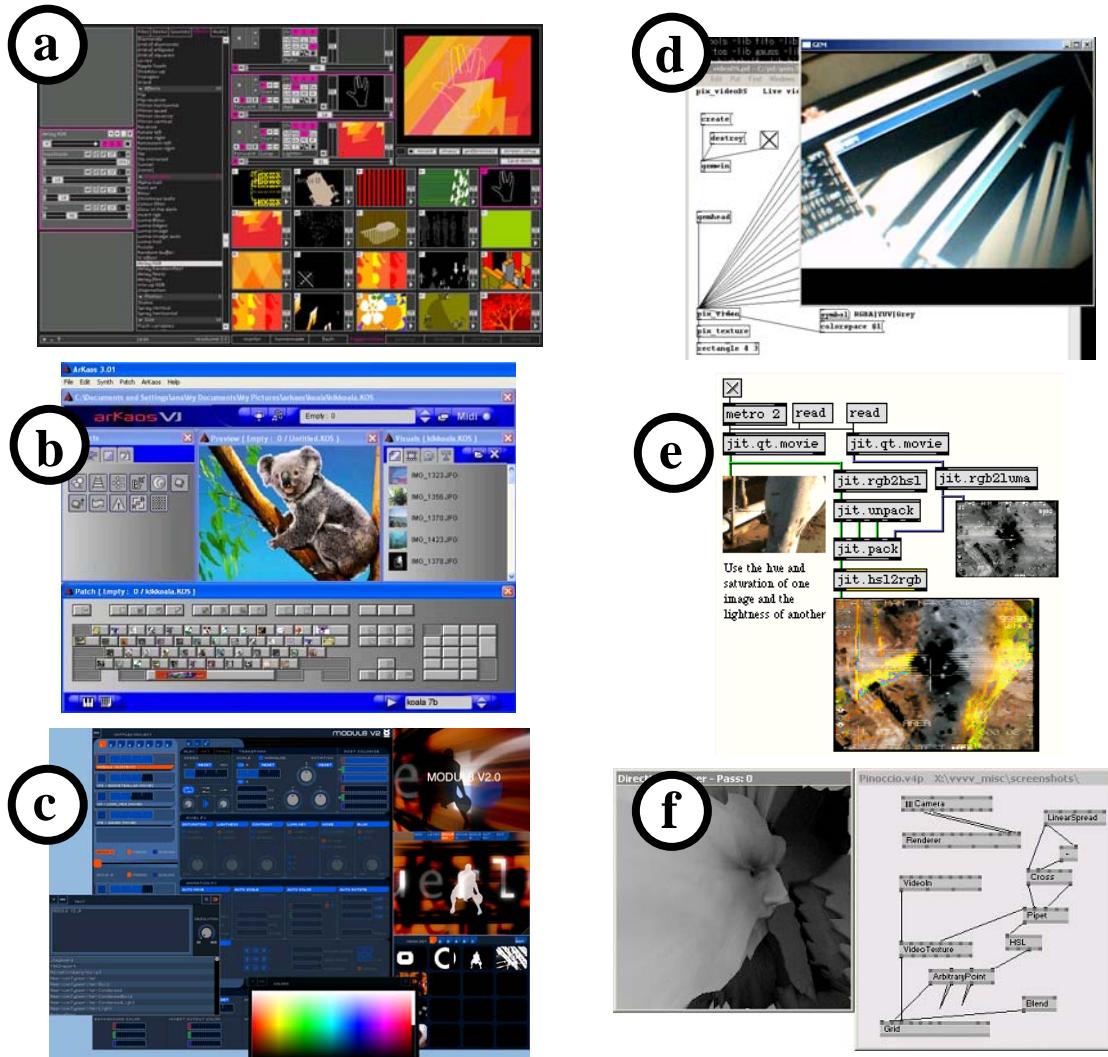


Figura 2-4 Telas de alguns *softwares* para VJ: os mais amigáveis (a) Resolume, (b) Arkaos e (c) Modul8; os mais poderosos (d) PD/Gem, (e) Max/Msp/Jitter e (f) vvvv.

Este problema, chamado nesta pesquisa de “usabilidade x expressividade”, será extensivamente discutido no Capítulo 5. Antes disso, apresentaremos estudos de vários aspectos de cada uma dessas duas classes de programas: primeiramente os SOAVEs no Capítulo 3 e em seguida os SOFs no Capítulo 4.

3. Sistemas Orientados a Amostras de Vídeo e Efeitos

3.1 Apresentação

Podemos classificar como SOAVE a grande maioria dos *softwares* para VJs e apresentações audiovisuais em tempo real em geral que estão disponíveis atualmente. Os SOAVEs apresentam um modo de funcionamento que prioriza o controle de execução de amostras de vídeo (de arquivo, ou capturado em tempo real) e de aplicação de efeitos de vídeo nestas amostras.

3.2 História

Os SOAVEs da forma como conhecemos atualmente surgiram no final da década de 90 com o desenvolvimento dos primeiros *softwares* específicos para a atividade do visual-jóquei como o ArKaos VJ, Image/ine, Aestesis, VJamm e MotionDive. Estes *softwares* reúnem diversas funcionalidades incluindo o disparo de amostras de vídeo (*vídeo triggering*), efeitos em vídeo, síntese de imagens 2D/3D e controle por análise de áudio.

Entretanto, estes programas atuais herdam técnicas criadas em diversas ferramentas que já utilizavam tecnologia de processamento digital de imagem desde o fim da década de 70 e início da década de 80, como o já citado projeto “*Recollections*” de Ed Tannenbaum. Com a revolução dos computadores pessoais, houve uma primeira explosão de ferramentas para processamento de imagem que já possibilitava apresentações audiovisuais criadas em tempo real. Mas ainda assim, era necessário *hardware* específico para este tipo de *software*.

Como uma das primeiras tecnologias do tipo, podemos citar o Fairlight Computer Video Instrument (CVI) desenvolvido no início da década de 80, pelos engenheiros australianos Peter Vogel e Kim Ryrie. Lançado em 1984, o CVI foi um dos primeiros sintetizadores digitais de vídeo disponíveis no mercado e possibilitava a aplicação em tempo real da maioria dos efeitos de vídeo que conhecemos hoje (EAGAN, 2005).

O BiT BOPPER (1988) (O'WONDER:: CLASSIC BiT BOPPERTM, 2007) foi lançado em 1988 e usado por diversas empresas de TV como a BBC de Londres. Este equipamento já era capaz de produzir gráficos 2D e 3D sincronizados por áudio, além de efeitos em amostras de vídeo (Figura 3-1).



Figura 3-1 Bit Bopper (1988): sendo apresentado em programa de TV (esq. cima), interface gráfica (esq. baixo) e rack móvel (direita)

Os *softwares* Performer e INVISION foram desenvolvidos entre 1988 e 1989 pela empresa Elan Design para o computador Commodore Amiga 500. O INVISION já era capaz de aplicar efeitos em vídeo capturado em tempo real com a placa de vídeo Amiga LIVE Board (JACOBS, 1994).

Pouco tempo depois, funcionários da Elan Design entraram no time da NewTek e contribuíram para o desenvolvimento do Video Toaster criado por Brad Carvey e Steve Kell e lançado em 1990. O Vídeo Toaster era o conjunto de um *software* e uma placa de captura com várias entradas de vídeo que funcionava como um poderoso *mixer* de vídeo, processamento de vídeo em tempo real, modelagem e animação 3D.

Outro pioneiro na área foi o programador Jeff Minter, fundador da empresa de jogos Llamasoft, que desenvolveu uma série de programas para apresentações visuais

ao vivo: Psychedelia (1984), Colourspace (1985), Trip-a-Tron (1987) e VLM (1990) (MINTER, 2007). Estes programas não eram capazes de analisar áudio, mas já permitiam a criação de gráficos de forma interativa em tempo real.

Podemos citar ainda como precursores das técnicas utilizadas na maioria dos SOAVEs atuais os seguintes programas: Fractint, para geração de fractais em tempo real desenvolvido pelo Stone Soup Group desde 1988; VuJak (1992) desenvolvido por Brian Kane, Lisa Eisenpresser e Jay Haynes provavelmente o primeiro *software* a disparar amostras de vídeo (usado pelo grupo EBN - Emergency Broadcast Network, um marco na história dos grupos audiovisuais); Cthuga (1993), desenvolvido pelo australiano Kevin 'Zaph' Burfitt; Videotracker, desenvolvido por Peter van Campen e lançado em 1994 para plataforma Amiga; Bomb (1995), desenvolvido por Scott Draves; e dezenas de outros.

Apesar de serem apresentações não-interativas, os programas produzidos pelas várias comunidades de programadores da chamada *Demoscene* (SCENE, 2007) também utilizam técnica de síntese audiovisual em tempo real e também são fontes de inspiração e influência para muitos SOAVEs. A *Demoscene* consiste em uma subcultura de grupos de programadores, artistas visuais e músicos que criam pequenos programas de demonstração (*demos*) de suas habilidades e promovem eventos para exibição de suas criações.

Finalmente, a partir da metade da década de 90, os novos modelos de computador pessoal IBM PC que já havia dominado o mercado, começaram a oferecer as mesmas capacidades multimídia semelhantes a que os computadores Amiga ofereciam desde os anos 80, mas dessa vez com maior velocidade de processamento. Isso possibilitou o surgimento dos primeiros SOAVEs da forma como conhecemos hoje, isto é, sistemas que possibilitam um conjunto de várias funcionalidades para manipulação de vídeo em tempo real orientados a amostras de vídeo e efeitos. A Tabela 1 apresenta em ordem cronológica alguns dos principais SOAVEs desenvolvidos nos últimos anos.

Tabela 1 Alguns dos principais SOAVEs surgidos nos últimos anos (em ordem cronológica).

Programa	Ano	Autor	País
ArKaos VJ (HINIC, 2006), inicialmente chamado “X<>Pose It”	1996	Marco Hinic e Jean-Charles Tramasure (HERACLES, 2001)	Bélgica
Image/ine, atualmente chamado ImX (DEMEYER, 2006)	1997	Tom Demeyer com a colaboração da vídeo-artista Steina Vasulka	Holanda
Aestesis	1998	Renan Jegouzo	França
VJamm	1998	Russell Blakeborough e Matt Black	Inglaterra
VRStudio	1999	VJSpetto	Brasil
Motion Dive	1999	Digitalstage	Japão
Composite Station	1999	SGRA	Japão
Videodelic (UI Software: Videodelic , 2007)	2000	Eric Wenger	EUA
Resolume	2001	Edwin de Koning e Bart van der Ploeg	Holanda
Meimi	2001	SEERA	Japão
EffecTV	2001	Kentaro Fukuchi (FUKUCHI <i>et al.</i> , 2004)	Japão
LiVES	2002	Gabriel "Salsaman" Finch	Inglaterra
Modul8	2004	Yves Schimid e David Hodgetts	Suíça

3.3 Considerações sobre interface humano-máquina

Nesta sessão apresentaremos alguns aspectos relativos à interface gráfica do usuário dos SOAVEs. Este estudo é importante para a compreensão do modo de operação destes sistemas. Além disso, a interface gráfica dos SOAVEs é o fator determinante para sua facilidade de uso, eficiência e grande popularidade entre os visual-jóqueis. Por tanto seu entendimento é fundamental para esta pesquisa.

Um dos aspectos que gostaríamos de considerar sobre a interface dos SOAVEs é a escolha de seus criadores pelas metáforas visuais a serem adotadas. Como é comum no desenvolvimento de *software*, geralmente o desenvolvedor da interface escolhe alguns objetos do mundo físico para serem representados graficamente em duas dimensões na tela do computador. Por exemplo, a folha de papel em branco é uma metáfora usada em editores de texto, editores de imagem e páginas eletrônicas da

Internet. Da mesma forma, desenhos representando botões e *sliders* (controles deslizantes) são usados nos mais diversos tipos de programas.

Por razões históricas e culturais, a atividade do visual-jóquei esteve e ainda está muitas vezes associada à atividade do DJ (discotecário) e às apresentações musicais em geral. Por tanto, não é por coincidência que muitas das metáforas escolhidas pelos desenvolvedores de *softwares* para VJs tenham origem nos equipamentos de DJs ou músicos em geral.

Um exemplo disso é o modelo de *mixer A/B*, isto é, um módulo que recebe como entrada dois sinais (sinal A e sinal B) – de áudio no caso dos DJs e de vídeo no caso dos VJs – e é capaz de juntar (mixar) os dois sinais em apenas um sinal de saída. O dispositivo principal de um *mixer* é um controle deslizante que determina qual dos dois sinais terá mais presença na saída: quando o controle está mais próximo de A, o sinal A ficará mais visível (ou audível no caso dos DJs) que o sinal B. A Figura 3-2 é a foto de um *mixer A/B* de áudio comumente utilizado por DJs. Note o controle deslizante vertical próximo ao lado inferior do painel de controle.



Figura 3-2 Exemplo de mixer A/B de áudio.

O conceito de *mixer* como um aparelho eletrônico para sinal de vídeo é usado pelos VJs antes mesmo de existir *softwares* específicos para a atividade. Este tipo de aparelho atualmente é muito usado, principalmente por VJs que não utilizam *softwares* e cujo equipamento consiste em dois DVJs (tocador especial de DVDs para apresentações ao vivo) e um *mixer* de vídeo A/B. A Figura 3-3 apresenta fotos de *mixers A/B* de vídeo.



Figura 3-3 Fotos de exemplares de mixers A/B de vídeo específicos para VJs.

A maioria dos *softwares* para VJ utiliza o conceito do *mixer*. Entretanto, alguns têm sua interface gráfica do usuário totalmente baseada na representação gráfica de um *mixer A/B*. Por tanto, à primeira vista estas interfaces já são familiares para um usuário acostumado a ver e saber como funciona o equipamento dos DJs, que geralmente consiste em dois toca-discos (ou tocadores de CDs) e um *mixer A/B*. Este é o caso do AVmixer da Neuromixer, por exemplo, um típico SOAVE cuja a interface gráfica do usuário é baseada em um *mixer A/B* (veja Figura 3-4).



Figura 3-4 Tela do AVMixer Pro da empresa Neuromixer: metáfora do mixer A/B.

Entretanto, devido a alta popularidade do Resolume e do ArKaos, a metáfora mais conhecida usada em *softwares* para VJs é provavelmente a do teclado de computador. A interface do ArKaos possui quatro partes principais como pode ser visto na Figura 3-5: uma janela de ferramentas de efeitos (canto superior esquerdo), uma janela de mídias (canto superior direito), uma janela de *preview* (ao centro) e uma janela contendo o desenho de um teclado de computador que ocupa toda a parte de baixo da interface.

O funcionamento do ArKaos é muito simples: os efeitos e as mídias (arquivos de vídeos, vídeo capturado em tempo real, imagens, etc.) são associados às teclas, quando o usuário aperta uma tecla o programa executa a mídia e o efeito que está associado àquela tecla. Para associar um efeito ou uma mídia a uma tecla basta clicar em um efeito na janela de efeitos (ou uma mídia na janela de mídias) e arrastá-lo para a tecla desejada.

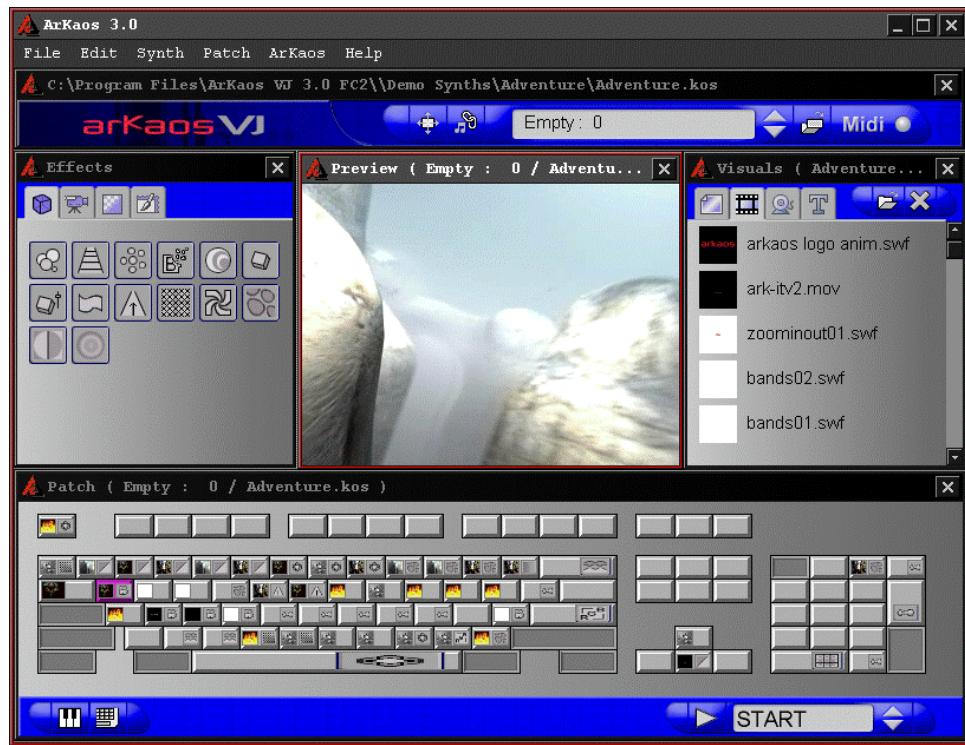


Figura 3-5 ArKaos VJ: metáfora do teclado de computador.

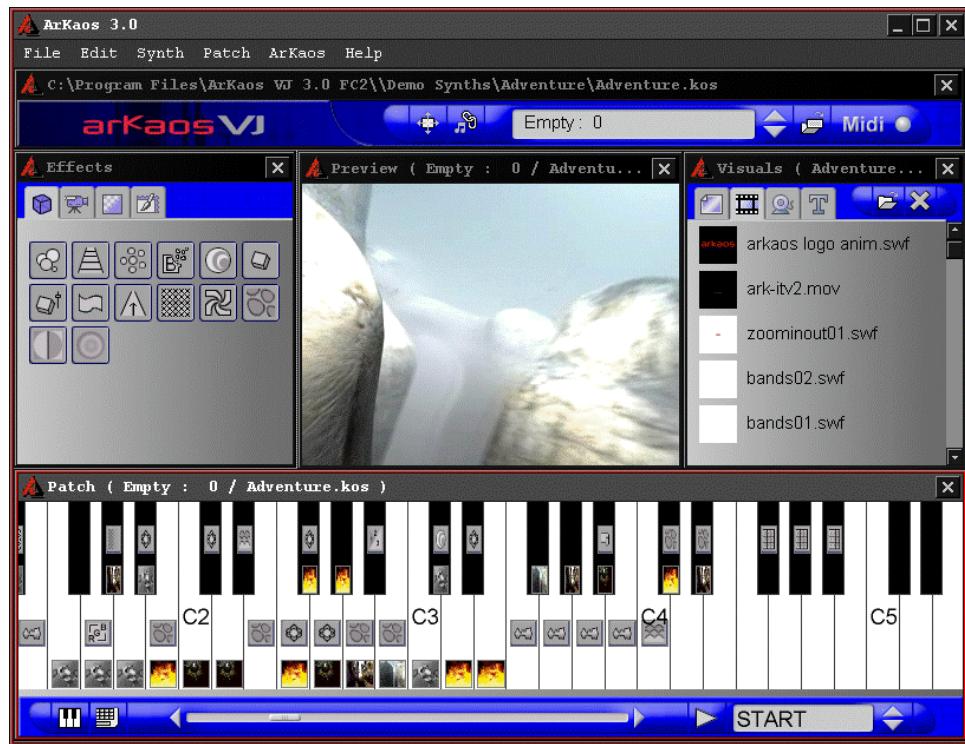


Figura 3-6 ArKaos VJ: metáfora do teclado musical.

Apesar de ser prático para usuários que não possuem equipamentos especiais como teclados MIDI, o uso do teclado de computador no momento da apresentação é

evitado pela maioria dos VJs profissionais. Esse é o caso do grupo Media Sana (**Media Sana**, 2007), por exemplo, que prefere utilizar teclados musicais para controlar o ArKaos através do protocolo MIDI. Para estes casos, ArKaos possui uma outra opção de metáfora, a do teclado musical como pode ser visto na Figura 3-6.

Já o Resolume também utiliza a metáfora do teclado de computador, porém, não de forma tão explícita e direta como o ArKaos. Como pode ser visto na Figura 3-7, a interface gráfica do Resolume apresenta uma matriz de retângulos aos quais podem ser associados os vídeos e os efeitos. Diferentemente do ArKaos, essa matriz não é representada por um desenho semelhante ao de um teclado. No Resolume, a analogia ao teclado existe somente devido à ordem em que estão dispostos os retângulos, que é correspondente ao padrão de teclas “QWERT” dos teclados de computadores.

Entretanto, além do teclado de computador o Resolume também utiliza o conceito de *mixer*. Neste caso, o *mixer* é de três canais, em vez do A/B (dois canais) tradicional. Cada canal recebe a denominação de camada, como utilizada nos *softwares* de edição de imagens e de vídeo. Por tanto, trata-se de um sistema um pouco mais sofisticado que o ArKaos e AVmixer, e permite uma maior flexibilidade. Em compensação, a compreensão da interface não é tão imediata, e por isso o Resolume é um pouco mais difícil de se aprender. Esta constatação pôde ser confirmada em uma pesquisa com VJs apresentada no capítulo 5.

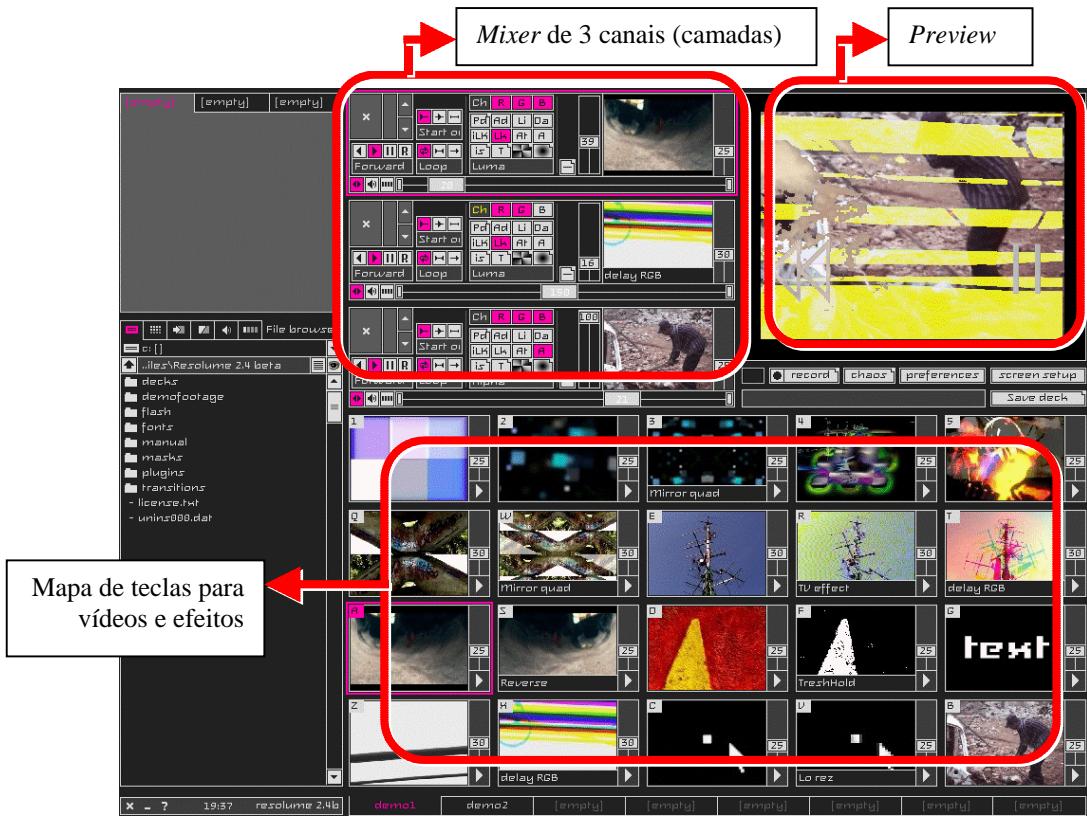


Figura 3-7 Resolume: metáfora do teclado de computador associado à de um *mixer* de 3 canais.

Apesar da inquestionável popularidade do Resolume e ArKaos, o Modul8 é um outro SOAVE que adquiriu uma grande aceitação entre os VJs profissionais devido ao seu alto desempenho e flexibilidade. Entretanto sua interface possui um nível de sofisticação um pouco mais elevado que o Resolume.

Como pode ser observado na Figura 3-8, a interface gráfica do Modul8 apresenta uma referência tanto ao *mixer* A/B quanto à idéia de *mixers* de vários canais. Na verdade o sistema consiste em dois *mixers* de 5 canais ligados a um *mixer* A/B. Como explicitado na pág. 8 de seu manual (SCHIMID & HODGETTS, 2007), o Modul8 é baseado em um paradigma de camadas, existindo, por tanto, 10 camadas (5 de cada *mixer* primário) que podem ser agrupadas separadamente com a utilização do *mixer* A/B.

Além disso, o Modul8 também apresenta uma associação entre as teclas numéricas e uma matriz de retângulos representando as mídias. No entanto, diferentemente do ArKaos ou Resolume, não há nenhuma referência ao teclado do computador; simplesmente as teclas de “1” a “9” são associadas às nove primeiras mídias do banco de mídias.

Além dos *mixers*, do banco de mídias e das telas de *preview*, o Modul8 ainda apresenta um grande painel de controle (maior elemento da interface) com o qual o usuário pode alterar diversos parâmetros da camada que está selecionada no momento, indicada por um contorno laranja. O Modul8 também oferece ao usuário a possibilidade de criação de seus próprios módulos de interface.



Figura 3-8 Modul8: metáfora de uma mesa de som de vários canais, associada à de um *mixer A/B* verticalizado (o canal A está posicionado acima do B e não ao lado).

3.4 Análise funcional

Na sessão anterior analisamos os aspectos de interface gráfica dos SOAVEs que contribuem para sua usabilidade. Nesta sessão, apresentaremos uma análise funcional de um SOAVE. O objetivo dessa análise é descobrir e evidenciar os limites de funcionalidades que existem no Resolume e nos SOAVEs em geral.

Como explicado em (BOMFIM, 1995), a técnica de Análise de Funções é utilizada para se comparar produtos de uma mesma família através de seus valores de uso, expressos em graus de utilidade, identificando aquele que oferece ao usuário maior valor utilitário. O procedimento completo para a utilização da técnica inclui a determinação das funções, sub-funções e sub-funções primárias (aqueles que não têm mais subdivisões) do produto analisado e a ponderação das sub-funções do produto. Neste trabalho executamos apenas a primeira etapa, ou seja, a determinação das funções, sub-funções e sub-funções primárias, pois nosso objetivo é apenas identificar e enumerar as funções e as possibilidades de associação de funções de processamento de um SOAVE no contexto da atividade do visual-jóquei.

Como resultado deste procedimento no Resolute, obtemos a seguinte árvore de funções, sub-funções e sub-funções primárias:

1. exibir vídeos
 - 1.1. carregar e exibir vídeos de arquivo
 - 1.1.1. carregar e exibir imagem de arquivo
 - 1.1.1.1. montar vídeo a partir de imagens carregadas
 - 1.1.2. controlar vídeo
 - 1.1.2.1. pausar
 - 1.1.2.2. inverter sentido
 - 1.1.2.3. tocar quadros aleatoriamente
 - 1.1.2.4. aumentar/diminuir velocidade do vídeo
 - 1.1.2.5. definir começo e fim do vídeo (corte)
 - 1.1.2.6. definir modo de exibição:
 - 1.1.2.6.1. exibir apenas uma vez
 - 1.1.2.6.2. repetir indefinidamente (loop)
 - 1.1.2.6.3. pra-frente-pra-trás indefinidamente
 - 1.1.2.7. controlar vídeo por automação**
 - 1.2. capturar vídeo (webcam, placa de captura)
 - 1.3. exibir vídeo de fontes freeframes
 - 1.4. exibir vídeo a partir de streaming de outros computadores rodando Resolute
 - 1.5. alocar vídeos em canais
 - 1.5.1. ativar canais (toca vídeo do canal) por comando de controle
 - 1.6. sintetizar vídeo
 - 1.6.1. mixar vídeos
 - 1.6.1.1. alocar canais de vídeo em camadas sobrepostas
 - 1.6.1.1.1. associar canal a camada
 - 1.6.1.1.1.1. selecionar camada (F1, F2, F3)

- 1.6.1.1.2. limpar camada (desassociar de canal)
- 1.6.1.2. modos de sobreposição de camadas de vídeos
 - 1.6.1.2.1. selecionar modo (transparência, chroma key, etc)
 - 1.6.1.2.2. controlar parâmetros de sobreposição por **automação****
- 1.6.2. aplicar efeitos de vídeo
 - 1.6.2.1. por canal (apenas um efeito)
 - 1.6.2.2. na saída global (até 3 efeitos simultâneos encadeados)
 - 1.6.2.3. controlar parâmetros de efeitos de vídeo por **automação****
 - 1.6.2.4. selecionar modos de sobreposição de efeito sob vídeo original (transparência, chroma key, etc)
 - 1.6.2.5. aplicar efeitos freeframe
- 1.7. mapear comandos de controle em dispositivos de entrada
 - 1.7.1. teclado
 - 1.7.1.1. mapear teclas a comandos de controle
 - 1.7.2. mouse
 - 1.7.2.1. mapear clicks em determinada área a um comando de controle
 - 1.7.2.2. mapear posição do mouse a slides
 - 1.7.3. controlador MIDI
 - 1.7.3.1. mapear comandos MIDI a comandos de controle, ou qualquer tecla de comando
 - 1.7.4. **automação:**
 - 1.7.4.1. entrada de áudio
 - 1.7.4.1.1. FFT
 - 1.7.4.2. relógio do sistema
 - 1.7.4.2.1. BPM
 - 1.7.4.2.1.1. manual com *tap*
 - 1.7.4.2.1.2. por relógio MIDI
 - 1.7.4.3. posição (x, y) do mouse

Uma observação fundamental a ser feita a partir dessa análise, é a possibilidade que o Resolume oferece de associação das saídas de alguns sub-programas (sub-funções primárias) como parâmetros de entrada para outras sub-funções. O exemplo mais importante é o processamento de áudio, 1.7.4.1.1. FFT (sub-função primária da função de automação), que pode ter suas saídas associadas a outras três sub-funções: 1.1.2.7. controlar reprodução do vídeo, 1.6.1.2.2. controlar parâmetros de sobreposição e 1.6.2.3. controlar parâmetros de efeitos em vídeo.

Colocando de forma mais clara, isso significa que existem três possibilidades de uso da análise de áudio no Resolume: controlar reprodução de vídeo, ou seja, quanto maior é a intensidade do som capturado, quadros mais perto do fim do vídeo

serão exibidos; controlar parâmetros de sobreposição, ou seja, quanto maior é a intensidade do som capturado, menor será a transparência de uma determinada camada; e finalmente, quanto maior a intensidade do som, maior será a intensidade de um determinado efeito em vídeo.

Entretanto, se o usuário quiser criar outros tipos de associações entre as funções de análise de áudio e o processamento/execução de vídeos está limitado à falta de possibilidade de personalização do Resolume. Este problema é geral entre os SOAVEs e será mais profundamente discutido no Capítulo 5.

4. Sistemas Orientados a Fluxogramas

4.1 Apresentação

Pure Data/GEM (DANKS, 1997) e Max/MSP/Jitter (JONES & NEVILE, 2005) são os mais conhecidos *softwares* de uma classe de sistemas modulares para fins de processamento integrado de áudio e vídeo em tempo real, que aqui chamaremos de Sistemas Orientados a Fluxogramas (SOF). Uma das principais características desse tipo de programa é a grande quantidade de possibilidades de aplicações devido à sua natureza modular, extremamente flexível e dinâmica. Entre suas diversas aplicações podemos citar as apresentações de visual-jóqueis e as instalações artísticas interativas.

Apesar de, historicamente, o conceito deste tipo de *software* estar fortemente associado à computação musical, nesta pesquisa estudaremos os SOFs que possibilitam o processamento modular não apenas de áudio mas também de vídeo. Muitas vezes, o suporte a vídeo é oferecido por uma extensão do sistema que inicialmente foi pensado para processamento de áudio. Por exemplo: Jitter é a extensão para processamento de vídeo do Max/MSP (*software* para áudio); da mesma forma o GEM é a extensão para processamento de vídeo do Pure Data (outro *software* para áudio).

O princípio de operação e funcionamento desses sistemas é semelhante aos antigos sintetizadores modulares analógicos utilizados como instrumentos musicais nos quais os módulos são conectados através de cabos (*patch-chords*) que ligam a saída de um módulo à entrada de outro. Essas conexões determinam um fluxo do sinal de áudio e uma seqüência de processamento ao qual este sinal é submetido até chegar à saída (caixa de som). Uma configuração ou arranjo desses módulos através de conexões (veja a Figura 4-1) recebe o nome de “*patch*” nos países de língua inglesa. O uso do termo “*patch*” é derivado da experiência dos primeiros sistemas de telefonia nos quais a telefonista ligava manualmente um terminal a outro conectando fios (“*patch-chords*”). Alterando-se as ligações de um *patch* modifica-se a seqüência de processamento do sinal mudando assim o resultado final da síntese.

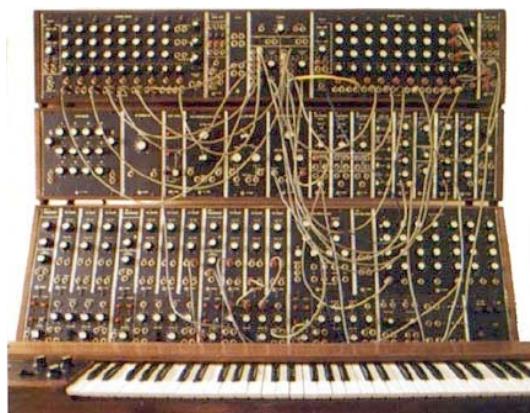


Figura 4-1 Foto de um *patch* (conjunto de ligações entre os módulos através dos cabos) montado em um sintetizador Moog.

Analogamente, os SOFs apresentam vários módulos de processamento, muitas vezes chamados de objetos, representados por desenhos de caixas, com conexões de entrada e saída, além de linhas que representam o “caminho” dos dados (áudio ou vídeo) a serem transmitidos de um módulo a outro, até chegar à saída (Figura 4-2). A arquitetura e a interface gráfica dos SOFs apresentam a mesma natureza modular dos antigos sistemas de telefonia e dos sintetizadores analógicos, sendo por tanto natural que hoje em dia se utilize o termo “*patch*” para se referir a uma configuração de caixas e linhas nestes programas.

Contudo, vale ressaltar que segundo o cientista da computação musical Jean-Baptiste Barrière, a metáfora dos “*patch-chords*” utilizada em *softwares* não foi inspirada nos sintetizadores analógicos e sim o contrário: Robert Moog se inspirou no trabalho de Max Mathews (pioneiro da computação musical e o primeiro programador a utilizar essa metáfora em *software*) para desenvolver seu famoso sintetizador analógico Moog (DELAHUNTA, 2002b).

Neste texto traduziremos “*patch*” para “fluxograma”, termo sugerido por Guilherme Soares na lista de discussão de desenvolvedores brasileiros de Pure Data, chamada Puredeposito (JÁCOME *et al.*, 2007). As palavras derivadas de “*patch*” também serão traduzidas. Por exemplo, o verbo “*to patch*” – que significa construir “*patches*” em programas como Pure Data e Max – será “fluxogramar” e “*patching languages*” (PUCKETTE, 1996) será “linguagens de fluxogramação”.

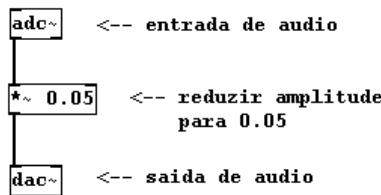


Figura 4-2 Exemplo de um fluxograma (*patch*) em Pure Data.

Em geral, os SOFs podem ser vistos como linguagens de programação no sentido de que o usuário pode construir aplicações (programas de computador), definindo o fluxo de dados e de execução através da construção de fluxogramas em um ambiente gráfico utilizando uma linguagem visual. Os programas construídos geralmente podem ser alterados em tempo de execução. Isso justifica o fato desses sistemas serem descritos como “ambientes de programação visual em tempo real para processamento de áudio e gráficos” (PUCKETTE, 2006a), “linguagens de programação visual” em (DANNENBERG, 2004) e (PELLETIER, 2005), e “linguagens de computação musical para tempo real” (WANG & COOK, 2004).

Programação visual é aquela em que mais de uma dimensão carrega semântica (BURNETT, 1999). Nas linguagens de programação textuais tradicionais, apesar de serem escritas e apresentadas em duas dimensões, uma dimensão já é suficiente para expressar um programa, pois a semântica de um código nestas linguagens é determinada pela ordem em que se apresenta cada símbolo (*token*). Isto é, poderíamos escrever um programa completo em apenas uma linha, entretanto utilizamos a quebra de linha para facilitar a leitura humana do código (com exceção das linguagens em que a quebra de linha significa “próxima instrução” e nesse caso a dimensão vertical carrega uma semântica, mesmo que limitada).

Já nas linguagens de programação visual (*visual programming languages*, ou *VPLs*), a sintaxe inclui símbolos que são expressões visuais e consistem em objetos ou relações multidimensionais cuja organização no plano bidimensional pode mudar o significado do código (BURNETT, 1999). Os diagramas e ícones são exemplos de expressões visuais utilizados nas *VPLs*. Um fluxograma é um tipo de diagrama em que a ligação entre os objetos representa um caminho de informação com um sentido fixo.

Devido à semelhança conceitual, é comum considerar erroneamente certas linguagens de programação visual como sendo também linguagens de fluxo de dados (*dataflow languages*), o que nem sempre é verdade. Como introduzido em (WHITING & PASCOE, 1994), linguagens de fluxo de dados derivam da idéia de um grafo de fluxo de dados, ou seja, a modelagem de um programa como um conjunto de nós, que representam operadores, interconectados por um conjunto de arcos que “carregam” dados de um nó a outro. Neste paradigma cada operador é livre para executar assim que o dado chega. A interpretação desse tipo de linguagem é naturalmente concorrente, pois teoricamente diferentes nós podem executar suas operações no mesmo instante que outros nós dependendo apenas de que os dados já estejam disponíveis em cada um.

Uma das principais características das linguagens de fluxo de dados é que o estado interno de cada nó (ou de cada objeto) é invariável. Este não é o caso de Pure Data, Max/MSP e a maioria das *VPLs* que apresentaremos neste texto. Estas são linguagens de programação visual, porém não são linguagens de fluxo de dados, pois seus objetos (nós) podem ter os estados alterados durante a execução do programa. Como exemplificado na Figura 4-3, objetos do Pure Data e Max/MSP, por exemplo, podem ter seu estado interno modificado em tempo de execução. Na Figura 4-3 o objeto de operação de multiplicação de amplitude de sinal representado pelo símbolo $*\sim$, pode ter seu multiplicador (inicialmente 0,05) alterado durante a execução do fluxograma. A barra de rolagem pode ser movida pelo usuário através do *mouse*. Ao se deslocar a barra, o objeto barra de rolagem envia uma mensagem para o objeto $*\sim$, contendo o novo valor a ser multiplicado pela amplitude do sinal de entrada.

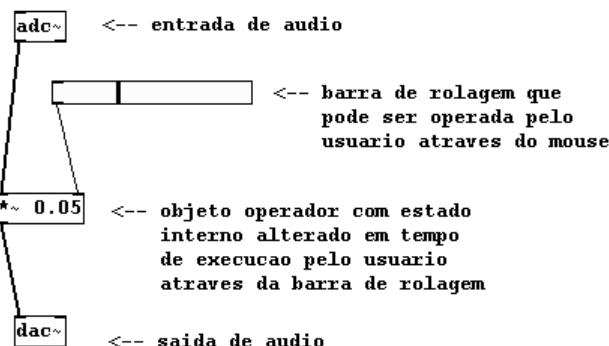


Figura 4-3 Exemplo de variação do estado interno de um objeto em um fluxograma do Pd.

Existe uma grande variedade de SOFs com diferentes graus de complexidade e poder de expressão. Os mais populares são os derivados dos sistemas de processamento de áudio Pure Data e Max/MSP. A extensão de processamento de vídeo mais conhecida do Pure Data é a GEM (DANKS, 1997), entretanto existem outras extensões como PDP (SCHOUTEN, 2007), PiDiPi (DEGOYON, 2007), Framestein (VEHVILÄINEN & TWINS, 2007) e GridFlow (BOUCHARD, 2007). Da mesma forma, Jitter (JONES & NEVILE, 2005) é a mais conhecida extensão para vídeo do Max/MSP, mas não a única; surgiu depois de NATO 0+55 (BERNSTEIN, 2000) que hoje já não é mais atualizada.

Existem ainda outros SOFs mais específicos para processamento de vídeo como EyesWeb (CAMURRI *et al.*, 2004), Isadora (CONIGLIO, 2006), vvvv (MESO, 2005), Quartz Composer (APPLE COMPUTER, 2007), Visual Jockey (HÖNGER *et al.*, 2006), GePhex (BAYER & SEIDEL, 2005) e Salvation (HARRISON, 2006) que apresentam uma crescente comunidade de usuários e peculiaridades importantes que serão discutidas neste texto. Além desses, alguns projetos acadêmicos como “Sonnet+Imager” (COLLOPY, 1999) e Aura (DANNENBERG, 2005) renderam publicações na área, mas ainda não foram distribuídos.

Podemos citar ainda linguagens como SuperCollider (MCCARTNEY, 1996), CSoundAV (MALDONADO, 2003) (extensão do Csound para processamento de vídeo) e Processing (FRY & REAS, 2005) que possuem área de aplicação e arquitetura semelhante aos sistemas anteriores, entretanto operados através de linguagens de programação textual em vez de visual. Finalmente, vale lembrar que existem ainda SOFs apenas para processamento de áudio como Reaktor (NATIVE INSTRUMENTS, 2007), CPS (GORISSE, 2007), AudioMulch (BENCINA, 2007) e Bidule (BEAULIEU *et al.*, 2007). O Reaktor, criado na segunda metade da década de 90, obteve grande sucesso comercial e contribuiu para a difusão do conceito de fluxogramação entre praticantes de música eletrônica.

4.2 História

A história dos SOFs se confunde com a história do programa de computador conhecido como Max, um dos primeiros a utilizar o conceito de fluxograma e processamento digital de sinal programável em tempo real através de unidades de

processamento inter-conectáveis. O autor das primeiras versões do Max, Miller Puckette, conta a história de dezessete anos desde a criação do programa, estudava no MIT (*Massachusetts Institute of Technology*), início dos anos 80, até suas ramificações atuais, no artigo “Max at Seventeen” (PUCKETTE, 2002). Miller esclarece que três programas atuais – Max/MSP, jMax e Pd (Pure Data) – podem ser considerados implementações derivadas de um paradigma comum ao qual chama “paradigma Max”.

O nome Max foi dado às primeiras versões desse programa em homenagem a Max Mathews (Figura 4-4), cientista pioneiro da computação musical e autor do programa RTSKED (MATHEWS, 1981), que foi influência fundamental para concepção do programa de Puckette.



Figura 4-4 Max Mathews

O paradigma Max pode ser descrito como uma classe de sistemas que permitem uma forma de se combinar dinamicamente blocos de construção pré-programados (módulos de processamento ou objetos) em configurações (fluxogramas) úteis para apresentações musicais, utilizando assim um computador como instrumento para a síntese interativa de música em tempo real. Isso inclui um protocolo para escalonamento de controle e processamento de sinal, uma abordagem para modularização e comunicação entre esses componentes, uma representação gráfica dos fluxogramas e ainda um editor para alterá-los em tempo real.

Miller Puckette ressalta a importância dos ambientes de pesquisa que o estimularam para a criação e desenvolvimento do Max que teve sua origem na atmosfera rica e criativa do *MIT Experimental Music Studio* do início dos anos 80, que depois viria a ser parte do famoso *MIT Media Lab* (MIT, 2007). A partir da metade dos anos 80 até o início dos anos 90, o Max adquiriu sua forma moderna

graças ao estimulante grupo de pesquisadores, músicos e compositores do IRCAM (*Institut de Recherche et Coordination Acoustique/Musique*) (IRCAM, 2007), em Paris.

O paradigma Max e sua primeira implementação completa foi quase toda desenvolvida no período entre 1980 e 1990, a partir de uma grande variedade de influências. A influência mais importante foi provavelmente o programa RTSKED de Max Mathews. O RTSKED tentou solucionar o problema de escalonamento em tempo real de operações de controle para um sintetizador polifônico, possibilitando o uso do computador como um instrumento musical.

Antes do RTSKED, Max Mathews havia desenvolvido as primeiras tentativas de geração de música em computador, nos laboratórios da Bell Telephone (que atualmente se chama AT&T). Entre essas tentativas está a série de linguagens Music-N: Music I (1957), Music II (1958), Music III (1960), Music IV (1962) e Music V (1968). O Music I, por exemplo, calculou em 1957 uma composição de 17 segundos de Newman Guttman, sendo a primeira música gerada por um computador que se tem conhecimento (CHADABE, 2000). Nestes primeiros anos, os compositores chegavam a ter de esperar dias até que fossem calculadas as operações de síntese para ouvirem uma composição pronta. Como princípio de funcionamento, a música era programada com o uso de “cartões de nota” (“*note cards*”) que explicitamente determinavam uma ação instantânea (disparo de uma nota) e o momento em que ela deveria ocorrer.

Paralelamente, através do desenvolvimento do sistema GROOVE (SPIEGEL, 1998), Mathews e seu grupo também experimentaram o uso de computador para se tocar música em tempo real. GROOVE era um computador híbrido (digital e analógico), pois utilizava sintetizadores analógicos de áudio para contornar as limitações de velocidade de processamento daquele tempo. Este sistema, que ocupava uma sala inteira, possuía vários dispositivos de entrada (botões, teclado de órgão, teclado alfanumérico, joysticks, entre outros) e vários dispositivos de saída (conversores digital-analógico usados para controle de voltagem, relês, impressora e discos rígidos) conectados a um computador DDP-24 programável através de FORTRAN IV ou linguagem de máquina.

Como princípio de funcionamento, o programador modelava a síntese utilizando funções contínuas de mudança no tempo. Periodicamente, a cada 10 milissegundos, as entradas eram lidas, as computações efetuadas e os parâmetros de controle perceptíveis (controle de voltagem dos dispositivos analógicos que

produziam o som) eram atualizados. Não havia embutido o conceito de eventos, nem a noção de nota. Para usar esses conceitos o programador deveria programá-los. Por tanto, o que o computador calculava não era uma seqüência de amostras de amplitude sobre o tempo representando uma onda sonora como fazem os computadores atuais. Isso seria impossível de se calcular em tempo real devido à limitação de velocidade de processamento daquela época. Em vez disso, GROOVE tratava o som como funções contínuas no tempo graças à sua parte analógica, como os sintetizadores analógicos modulares de sua época (muito utilizado por bandas de *rock* progressivo dos anos 60). A diferença é que o sistema era capaz de executar algoritmos que determinavam como essas funções seriam associadas aos parâmetros de controle desses dispositivos de saída durante a execução.

Além de áudio o GROOVE também chegou a ser utilizado para geração de imagem através do projeto VAMPIRE (*Vídeo And Music Program for Interactive Realtime Exploration/Experimentation*) entre os anos de 1974 e 1976. A cientista Laurie Spiegel conta sua experiência em (SPIEGEL, 1998) no desenvolvimento deste projeto que provavelmente foi a primeira tentativa de se integrar num mesmo sistema computacional o processamento interativo de som e imagem em tempo real (Figura 4-5). No VAMPIRE, em vez de sintetizadores analógicos de áudio, os dispositivos controlados pelo GROOVE eram sintetizadores analógicos de vídeo como o Rutt-Etra (AUDIOVISUALIZERS, 2006).

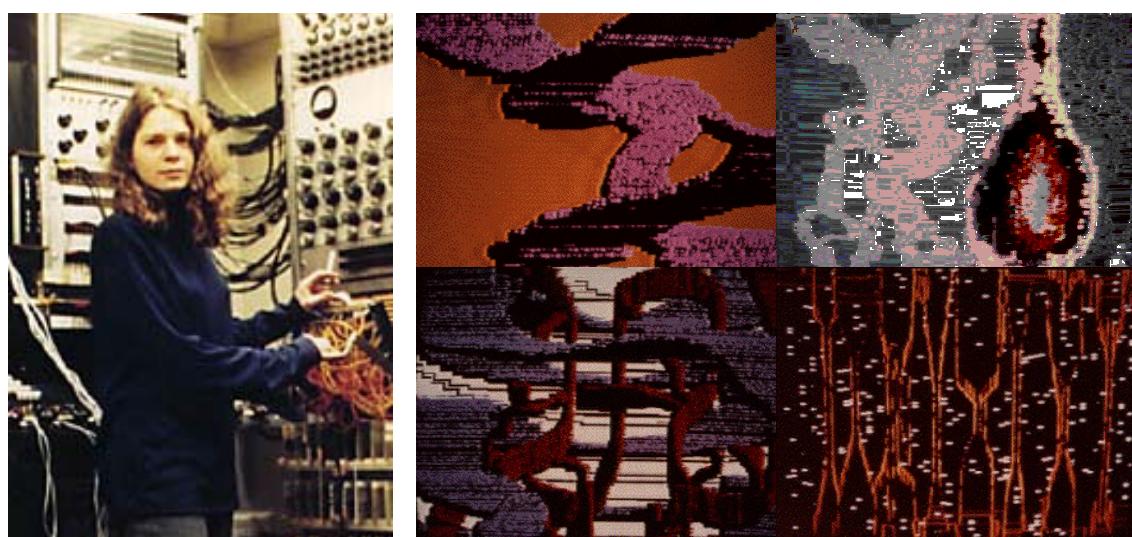


Figura 4-5 Laurie Spiegel trabalhando no GROOVE (foto de Emmanuel Ghent) e quatro imagens geradas pelo VAMPIRE

Enquanto o GROOVE utilizava sinais de controle de voltagem calculados periodicamente, o RTSKED ofereceu uma noção de controle diferente, em que eventos esporádicos causavam mudanças de estado nos sintetizadores. A diferença para os sistemas Music-N é que, assim como o GROOVE, o RTSKED era destinado à execução em tempo real, ou seja, deveria reagir a ações disparadas de forma arbitrária pelo usuário. Assim, em vez de se modelar a apresentação musical como uma seqüência de “cartões de nota” cujo momento de execução já era pré-determinado, o RTSKED pode ser visto como uma coleção de tarefas sendo executadas em paralelo. Por exemplo, cada tarefa poderia ser responsável por uma voz específica de uma música polifônica.

Assim, o RTSKED introduziu o importante avanço de se separar o problema de controle de sistemas tempo real em diferentes tarefas. Essa separação permite que o usuário controle a seqüência de execução do programa selecionando qual a próxima tarefa a ser disparada. Essa noção de controle em tarefas separadas pode ser claramente vista no paradigma Max que dispara cada tarefa (representada pelos objetos) através do envio de mensagens. Isso permite a modelagem de programas (fluxogramas) que funcionam como instrumentos musicais.

Outra grande influência para Miller Puckette foi seu professor Berry Vercoe (que por sua vez foi aluno de Max Mathews). Vercoe, é conhecido como autor do Csound, um poderoso e flexível sistema para síntese musical que tem origem no Music IV, que foi ponto de partida para uma série de melhoramentos e experiências com síntese de áudio. O sistema que começou se chamando Music 360, posteriormente foi renomeado para Music 11 e finalmente em 1985 recebeu o nome Csound como é conhecido atualmente (VERCOE, 2007). O grande avanço do Csound em relação aos primeiros Music-N está em sua estrutura de controle altamente refinada para fins de síntese.

Inspirando-se na estrutura de controle para tempo real do RTSKED e na máquina de síntese do Music 11 (como era chamado o Csound na época), Miller iniciou em 1982 o desenvolvimento do Music500, primeiro programa a instanciar muito do que viria a ser o paradigma Max. O sistema não possuía interface gráfica ainda. O controle e a síntese eram separados, sendo o controle uma coleção de processos para manipulação dos parâmetros de controle e a “orquestra” (síntese de áudio) uma coleção de processos para calcular os sinais.

Em 1985, Miller foi convidado para trabalhar com Vercoe no IRCAM, onde teve acesso ao processador e sintetizador de áudio mais poderoso do mundo na época, o 4X. Ao portar o Music500 para o 4X, o cientista manteve apenas a estrutura de controle e mudou o nome do sistema para “Max” em referência à grande influência do RTSKED, de Max Mathews. A parte de síntese do Music500 não precisou ser portada e foi descartada, pois o próprio 4X já possuía uma linguagem de fluxograma própria com a qual os processos de síntese poderiam ser programados.

Acompanhando a revolução dos computadores pessoais, em 1987, o Max foi reescrito em C para um Macintosh e dessa vez já com uma interface gráfica para a construção de fluxogramas. A idéia de interface gráfica utilizando esta metáfora dos *patches* não era nova, em 1980 já havia sido utilizada por Richard Steiger e Roger Hale no sistema Oedit (PUCKETTE, 2002). Ainda em 1987, foi escrito o primeiro “patch” para o Max chamado *Pluton* que funcionava em um Macintosh (onde era construído o fluxograma através de uma interface gráfica e executadas as operações de controle) que se comunicava através do protocolo MIDI com o 4X (onde eram executadas as operações de síntese e processamento).

Essa versão do Macintosh foi continuada por David Zicarelli e paralelamente, Puckette reescreveu o Max para uma nova plataforma, um novo computador chamado *IRCAM Signal Processing Workstation (ISPW)* que substituiu o 4X na instituição. Dessa vez, foi possível implementar novamente dentro do próprio programa um conjunto de objetos para processamento de sinal de áudio além dos objetos de controle. Assim, os processos de controle e de processamento e síntese de áudio voltaram a coabitar o mesmo ambiente como nas primeiras versões do Music500. Entretanto a interface gráfica teve de ser separada da máquina de processamento devido à restrição do *ISPW* que necessitava de um terminal *NeXT* para ser operado graficamente pelo usuário. Assim, a parte da máquina de processamento ao qual Puckette chamou de *FTS (Faster Than Sound)* (PUCKETTE, 1991b) ficava no *ISPW* rodando em multiprocessadores e a interface gráfica ficava no *NeXT*.

Em 1991 o Max/FTS começou a ser portado para outras plataformas e até a metade dos anos 90 foi a ferramenta padrão para desenvolvimento no IRCAM, sendo apelidada de “Audio Max” (ZICARELLI, 2005). Joseph Francis e Paul Foley portaram o programa para um Unix com o sistema de janelas *X Window*. Essa versão foi retrabalhada extensivamente por um grupo liderado por François Déchelle e hoje é distribuída gratuitamente e em código aberto pelo *IRCAM* com o nome de *jMax*

(DÉCHELLE, 2006). O nome jMax foi dado em alusão ao fato de sua interface gráfica ser implementada utilizando a linguagem de programação Java (SUN MICROSYSTEMS, 2007).

Em 1994, de volta aos EUA, Puckette iniciou o desenvolvimento de um novo programa visando vários melhoramentos em relação ao Max. Assim, surgiu o Pure Data (PUCKETTE, 1996) que foi distribuído como código aberto desde o início. Os objetos de processamento de sinal do Pure Data foram inspirados nos do Max/FTS e foram tomados como ponto de partida para que David Zicarelli desenvolvesse o “MSP”, que é a parte de processamento e síntese de áudio do atual Max/MSP (ZICARELLI, 2006). Desde 1997, Zicarelli comercializa através de sua empresa Cycling74, esse sistema que é a continuação da versão de 1987 do Max para Macintosh. O Max/MSP já foi utilizado por importantes nomes da música popular como a banda Radiohead e o produtor de música eletrônica Aphex Twin.

Entre os melhoramentos que Puckette imaginava para o Pure Data estava a reestruturação do sistema para possibilitar mais facilmente a integração de processamento de áudio com outras mídias como, por exemplo, vídeo. A primeira iniciativa neste sentido foi de Mark Danks em 1995, com o desenvolvimento do GEM (*Graphic Environment for Multimidia*) (DANKS, 1997). O GEM é um conjunto de objetos para Pd que podem gerar gráficos 2D e 3D através de OpenGL (WOO *et al.*, 2003), uma API (*Application Program Interface*) para desenvolvimento de aplicações gráficas compatível com diversas plataformas e linguagens de programação. Posteriormente, uma extensão semelhante chamada DIPS (MATSUDA & RAI, 2000) também foi desenvolvida para o jMax, no IRCAM.

Ainda no final dos anos 90, também foi desenvolvida uma extensão para o Max/MSP para fins de processamento de vídeo chamada NATO 0.55 (BERNSTEIN, 2000) que se tornou bastante conhecida no meio da arte digital. O NATO 0.55 era um conjunto de objetos que permitiam processar vídeos do formato *Quicktime* (APPLE COMPUTER, 2001) de forma integrada ao ambiente de processamento de áudio do Max/MSP. Atualmente, esse pacote não é mais atualizado e pouco se sabe sobre a sua autoria. Nas listas de discussão sobre arte digital a autora do sistema se identificava como Netochka Nezvanova (MIESZKOWSKI, 2002).

Em 2002, a empresa Cycling74 lançou o Jitter, seu próprio conjunto de objetos para processamento de matrizes, inicialmente desenvolvido por Joshua Kit Clayton. O Jitter possibilita a integração do Max/MSP com vídeo e OpenGL. Randy Jones

apresenta este pacote de objetos e suas possibilidades de aplicação para *visual music* em (JONES & NEVILE, 2005).

Como exemplos de aplicações que foram construídas utilizando o pacote Max/MSP/Jitter podemos citar o AVmixer (CUI, 2007), *software* popular entre os VJs iniciantes; o VuJak lançado em 1992 e provavelmente o primeiro *software* a disparar amostras de vídeo, como citado na Sessão 3.2; e o o8, a versão escrita em Max do Orpheus (RITTER, 2005) desenvolvido por Don Ritter, um dos pioneiros da arte interativa. O Orpheus foi utilizado em praticamente todas as instalações e *performances* de Don Ritter incluindo um dos primeiros experimentos de processamento de vídeo em tempo real controlado por um instrumento musical. Neste experimento do fim da década de 80, George Lewis, renomado trombonista e cientista da computação musical, controlava o processamento de vídeo através de sua improvisação ao trombone. Quase dez anos depois, o mesmo George Lewis participou do *Global Visual Music Project* (PUCKETTE *et al.*, 2006), em uma apresentação semelhante, na qual o trombonista e um baterista controlavam o processamento de vídeo com a execução musical de seus instrumentos, desta vez para demonstrar as possibilidades do Pure Data/GEM (PUCKETTE, 1997). Em 2003, o Orpheus foi reescrito em Max/MSP/Jitter e renomeado para o8 que desde então é o *software* utilizado nas instalações de Don Ritter. Tanto o Orpheus quanto o o8 são *softwares* de uso pessoal de Don Ritter e não foram publicados.

Além do Jitter, existe a versão do SoftVNS (ROKEBY, 2007b) de David Rokeby distribuída como pacote de objetos do Max para processamento de vídeo em tempo real. Entre as especialidades do SoftVNS está a localização de objetos em movimento (*video tracking*) e por isso é bastante utilizado por artistas de mídias interativas como o próprio David Rokeby (ROKEBY, 2007a). Em 1986, o autor expôs pela primeira vez a instalação interativa “Very Nervous System”, em que um programa de computador interpretava os movimentos de uma pessoa a partir de imagens capturadas por uma câmera e gerava em tempo real uma seqüência musical de acordo com algumas características desses movimentos. Atualmente, após se tornar um pacote de objetos do Max/MSP, o SoftVNS (nome dado ao programa utilizado na instalação descrita) pode ser utilizado em conjunto com o Jitter, sendo assim um importante complemento ao pacote Max/MSP/Jitter.

A Figura 4-6 representa uma linha do tempo dos principais sistemas que instanciaram ou influenciaram o paradigma Max.

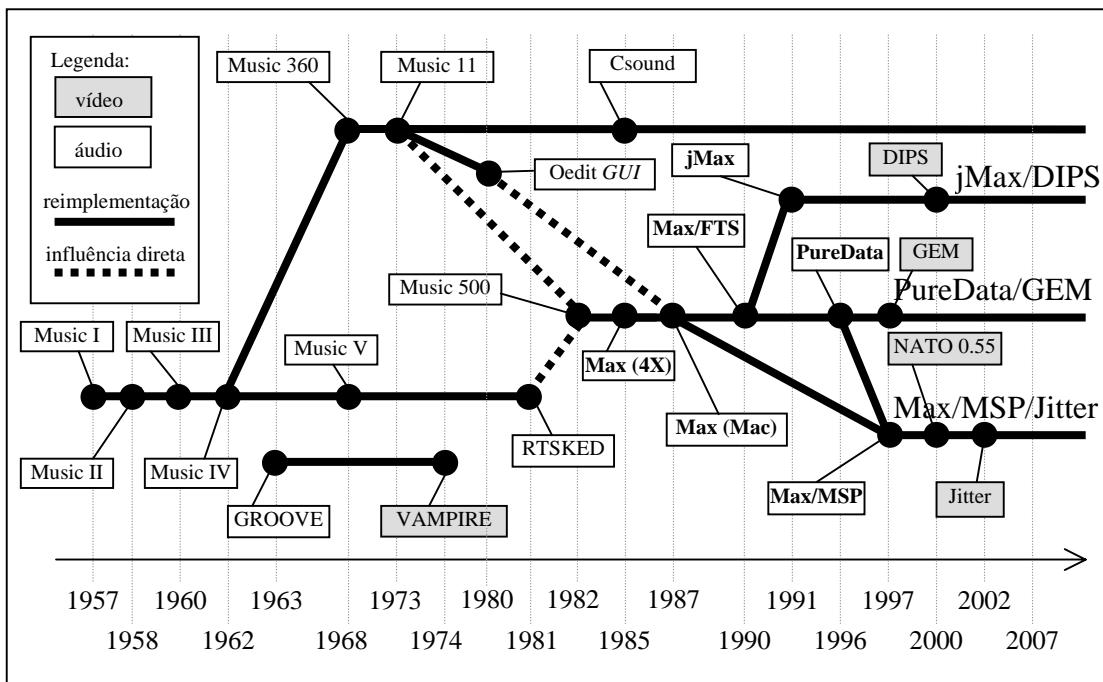


Figura 4-6 Linha do tempo do paradigma Max e sistemas relacionados.

Paralelamente ao desenvolvimento dos *softwares* do paradigma Max, outros programas semelhantes foram surgindo e deram origem a sistemas que estão se tornando populares nos últimos anos como é o caso do Isadora (CONIGLIO, 2006) e outros sistemas apresentados a seguir.

A história do *software* Isadora desenvolvido por Mark Coniglio também teve início na década de 80 (DELAHUNTA, 2002a). Coniglio é um dos fundadores do grupo nova-iorquino Troika Ranch (RANCH, 2007) que integra dança, teatro e mídias digitais interativas, e foi aluno do cientista da computação musical Morton Subotnick, no *California Institute of the Arts (CalArts)* onde iniciou em 1986 o desenvolvimento de um *software* para ser usado na apresentação de Subotnick chamada *Hungers* (1987) apresentada no festival Ars Electronica 1988 (ARS ELECTRONICA LINZ, 1988). Este programa, que depois passou a se chamar Interactor, foi uma versão de Coniglio implementada diretamente em código de um fluxograma que Subotnick havia desenvolvido no MIT utilizando o programa Hookup de David Levitt. O Hookup foi o primeiro programa que utilizava linguagem de fluxogramação com o qual Mark Coniglio teve contato, e por tanto a principal inspiração para o que viria a ser a interface do Isadora.

Em 1996, durante uma residência do grupo Troika Ranch no centro holandês para fomentação de arte digital STEIM (STEIM, 2007), Coniglio conheceu o Image/ine (DEMEYER, 2006), programa do artista Tom Demeyer para processamento de vídeo em tempo real, e passou a utilizá-lo nas apresentações do grupo. Após alguns anos usando o Interactor para o processamento MIDI em conjunto com o Image/ine para processamento de vídeo, Coniglio decidiu implementar um novo *software* com as duas funções integradas para suprir o problema de instabilidade e a interface gráfica limitada do Image/ine. Assim foi desenvolvido o *software* Isadora (nome dado em homenagem a Isadora Duncan, pioneira da dança moderna) com interface gráfica inspirada no Hookup, ou seja, utilizando a metáfora dos sintetizadores modulares.

O programa vvvv (OSCHATZ *et al.*, 2006) é um SOF que foi desenvolvido desde o início com o objetivo de ser usado para criar fluxogramas de processamento de vídeo em tempo real. O sistema de autoria de Sebastian Oschatz e Max Wolf teve início em 1997 como um *software* interno para uso em projetos de sua então recém fundada empresa de *design* Meso, de Frankfurt. No final de 2002, ou seja, após cinco anos de uso e desenvolvimento, a Meso lançou o sistema gratuitamente (apenas para uso não comercial). Desde então sua popularidade vem crescendo e atualmente é um dos principais SOFs disponíveis para VJs e artistas multimídia em geral.

O Visual Jockey (HÖNGER *et al.*, 2006) é um dos SOFs mais conhecidos entre os VJs. O autor do sistema, Josua Hönger, foi um participante de comunidades de *Demoscene* (SCENE, 2007), grupos de programadores, artistas visuais e músicos que criam pequenos programas de demonstração (*demos*) de suas habilidades e promovem campeonatos e eventos para exibição de suas criações. Em 1999, Josua Hönger iniciou o desenvolvimento de um *software* com o objetivo de fazer uma ferramenta para a criação de *demos* através de uma linguagem de fluxo de dados. O nome *visual jockey* foi dado sem o conhecimento da existência da sub-cultura dos VJs, entretanto, o *vJo* (abreviatura do nome do *software* Visual Jockey) acabou sendo mais utilizado por VJs do que por *demosceners* (como são chamados os programadores de *demos*).

O GePheX (BAYER & SEIDEL, 2005) é uma outra opção de SOF semelhante ao Visual Jockey, porém publicado como código aberto, sob licença *GPL* (FREE SOFTWARE FOUNDATION, 1991). O *software* começou a ser desenvolvido em

2001 por Phillip Promesberger, Martin Bayer e Georg Seidel, tendo sua primeira versão estável lançada no final de 2003.

O Quartz Composer (APPLE COMPUTER, 2007) é outro exemplo de sistema modular surgido nos últimos anos que apresenta uma crescente comunidade de usuários. Esta tecnologia se tornou bastante popular entre os VJs mais experientes e usuários de computadores Macintosh. Originalmente o *software* se chamava PixelShox Studio (LATOUR, 2003), desenvolvido por Pierre-Oliver Latour em 2002 e 2003 (KIRN, 2007). O sistema foi sendo melhorado por Latour, então funcionário da Apple, para ser lançado em 2005 junto com o sistema operacional Mac OS X Tiger, como um exemplo de aplicação das tecnologias CoreImage (parte gráfica do Tiger).

Além destes *softwares* utilizados por crescentes comunidades de usuários, existem outros que ainda não chegaram a ser publicados, entretanto foram apresentados em importantes artigos científicos. Deste grupo merece destaque o Aura (DANNENBERG, 2004), de Roger Dannenberg, apresentado pela primeira vez em 1995 e o Sonnet+Imager, de Fred Collopy apresentado pela primeira vez em 1999 através do artigo “Visual Music in a Visual Programming Language” (COLLOPY, 1999).

A Figura 4-7 apresenta uma linha do tempo dos principais SOFs, desde suas origens, ramificações, até suas versões atuais, incluindo o paradigma Max. Analisando a história dos SOFs de áudio e de vídeo e observando a Figura 4-7, podemos perceber que a década de 80 foi marcada pelo surgimento dos primeiros SOFs para processamento de áudio (como o Max) e a década de 90 pela popularização e surgimento de dezenas de variações (Pure Data, Reaktor, etc). Da mesma forma a década de 90 foi marcada pelo surgimento dos SOFs para processamento de vídeo (como o GEM) e a primeira década deste século está sendo marcada pela popularização e surgimento de dezenas de variações (NATO 0.55, Jitter, vvvv, Isadora, etc.).

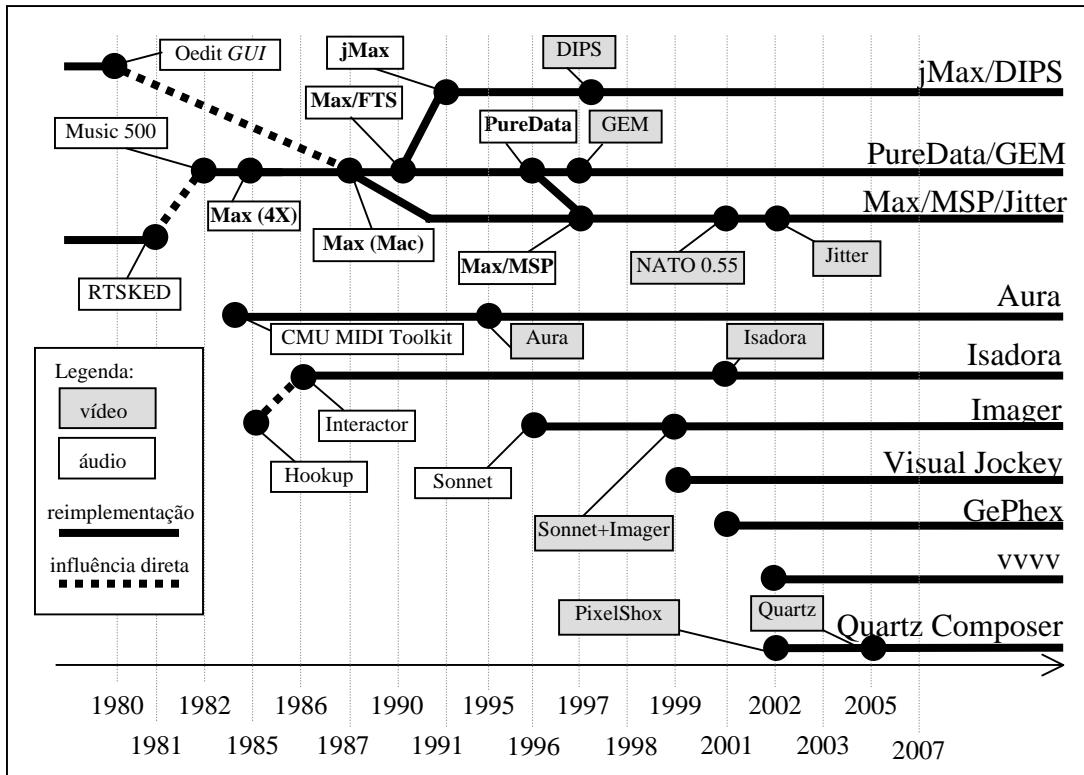


Figura 4-7 Linha do tempo dos principais Sistemas Orientados a Fluxograma.

4.3 Considerações sobre arquitetura

4.3.1 Introdução

Nesta sessão serão discutidas algumas questões relacionadas a arquiteturas de sistemas orientados a fluxogramas. Esta discussão é importante como um registro de um conhecimento ainda pouco documentado e se faz necessária para facilitar o entendimento das escolhas de arquitetura do ViMus.

A maior quantidade de documentação e registros acadêmicos e não acadêmicos relacionados à arquitetura de SOFs pode ser encontrada em artigos sobre sistemas do paradigma Max, como (PUCKETTE, 1991b), (PUCKETTE, 1991a), (PUCKETTE, 2002), (ZMÖLNIG, 2004b), (ZMÖLNIG, 2004a), (JONES & NEVILLE, 2005), (CLAYTON, 2004) e (ZICARELLI, 2005). Por tanto utilizaremos a arquitetura geral dos sistemas do paradigma Max (com foco no Pure Data, por ser um programa de código aberto) como ponto de partida para a compreensão do assunto.

Segundo uma definição moderna de (BASS *et al.*, 2003), arquitetura de *software* de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que compreende os elementos do *software*, as propriedades externamente visíveis destes elementos, e a relação entre os mesmos. As “propriedades externamente visíveis” se referem aos serviços providos e outras suposições que podem ser feitas de um elemento sem precisar entender sua estrutura e funcionamento interno. O comportamento e a dinâmica de interação entre os elementos também fazem parte da arquitetura de um *software*.

O elemento fundamental do paradigma Max (assim como qualquer SOF) é o fluxograma, tanto do ponto de vista do usuário final como das pessoas que desenvolvem o sistema. Por tanto, um SOF é projetado para permitir a edição (construção e modificação) e a execução em tempo real de um fluxograma. Assim, um SOF deve possuir pelo menos um editor e um executor de fluxogramas, os vários módulos de processamento (a serem instanciados no fluxograma) e um protocolo de comunicação entre os módulos (tipo de mensagens, conexões). Estes conceitos podem ser implementados em diferentes componentes de *software* ou implicitamente em grandes estruturas monolíticas, dependendo do SOF.

Naturalmente, para o usuário editar graficamente o fluxograma e visualizar o resultado de sua execução faz-se necessária uma *GUI* (do inglês *Graphic User Interface*). Por tanto, em uma visão mais geral do sistema, podemos dividi-lo em máquina de processamento e *GUI*, mesmo que essa separação não esteja explícita no código do SOF. A máquina de processamento é a parte do programa que modifica e executa o fluxograma. A *GUI* simplesmente exibe o fluxograma, o resultado e estados de sua execução (ex.: um vídeo processado) e captura eventos do usuário. Esses eventos desencadeiam comandos de edição do fluxograma que são enviados para a máquina.

4.3.2 Acoplamento entre interface gráfica e máquina de processamento

Uma questão básica referente à arquitetura de um SOF é o grau de acoplamento entre a máquina e a *GUI*. A vantagem de uma maior separação entre essas duas entidades é que a máquina fica independente do tipo de *GUI*, que por sua vez pode ser facilmente substituída por outra interface qualquer (gráfica ou não). Uma desvantagem dessa separação é que o desempenho pode ser comprometido com a

adição de mais camadas de abstração ou pelo processamento extra, necessário para a troca de mensagens entre os dois módulos através de um protocolo de comunicação.

No Pd essa separação é praticamente inexistente como constatado em (BOUCHARD & LEE, 2005) (veja Figura 4-8). Na verdade essa parte da arquitetura do Pd é pouco documentada, no entanto, observando o código fonte do programa podemos verificar facilmente o alto grau de acoplamento entre a GUI e a máquina. Para isso basta observar que na declaração de um mesmo objeto (*env~*, por exemplo) existem funções de processamento de áudio e comandos de desenho na interface gráfica.

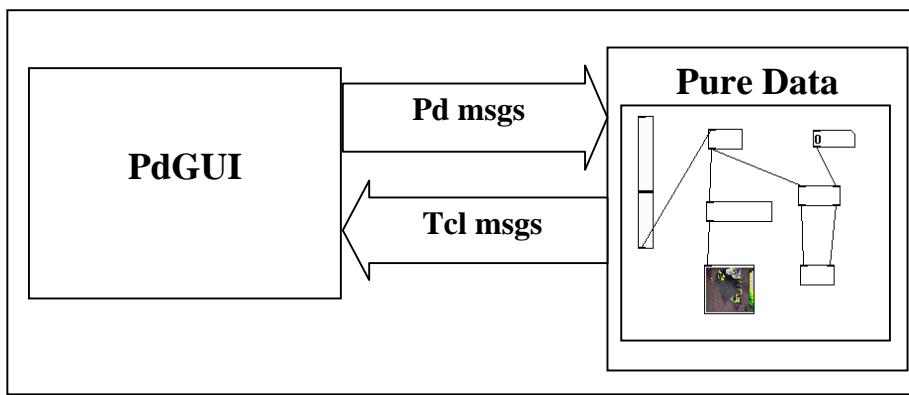


Figura 4-8 Arquitetura geral do Pd

O DesireData (BOUCHARD & LEE, 2005) é uma versão modificada do Pd com uma proposta de separação entre a *GUI* e a máquina (veja). A principal consequência dessa separação é a existência de duas representações de um mesmo fluxograma: uma na *GUI* e outra na máquina. Na primeira, os objetos do fluxograma apresentam métodos relacionados ao desenho das caixas na interface gráfica. Na segunda, os objetos do fluxograma apresentam métodos relacionados ao processamento correspondente àquele objeto.

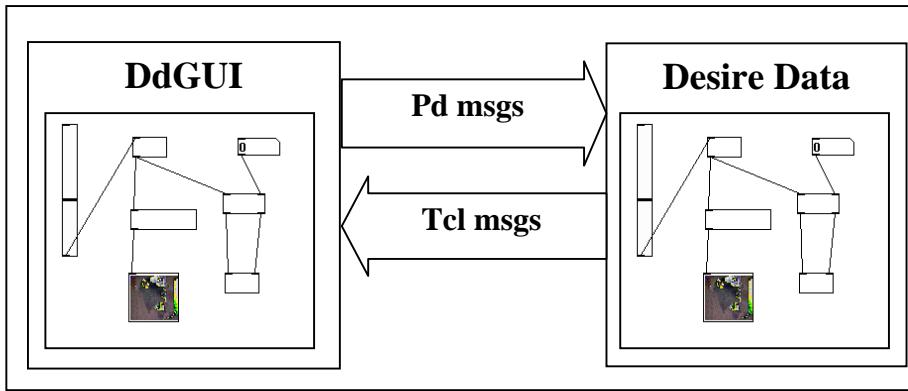


Figura 4-9 Arquitetura geral do Desire Data

4.3.3 Processamento de áudio e controle

Uma questão muito importante em relação à arquitetura de um SOF diz respeito à solução do problema de integrar diferentes tipos de processamento, com diferentes restrições de tempo, em um mesmo sistema de tempo real. Para entendermos quais são esses tipos de processamento e restrições precisamos compreender melhor o conceito e o funcionamento de um fluxograma.

Um fluxograma no paradigma Max (e na maioria dos SOFs) é constituído por objetos e conexões entre esses objetos. Um objeto é um módulo (unidade de processamento) responsável por uma função específica e pode possuir zero ou mais entradas e zero ou mais saídas. Através das entradas os objetos recebem mensagens e através das saídas enviam mensagens. Uma conexão determina que as mensagens de uma saída de um objeto serão enviadas para uma entrada de outro objeto. Entradas e saídas podem ter zero ou mais conexões.

Existem dois tipos básicos de mensagem no paradigma Max: mensagem de controle e mensagem de sinal. Os objetos de controle processam mensagens de controle e os objetos de sinal processam mensagens de sinal. O conceito de mensagem de controle sempre existiu desde as primeiras versões do Max na década de 80 (Music500, Max/4X) e consiste no envio de informações de controle de um objeto para outro. Por exemplo, uma mensagem de controle pode significar um “comando” para que um objeto inicie sua operação (como por exemplo uma mensagem “bang” no Pd) ou simplesmente um número que será usado como parâmetro para a função de processamento do objeto (veja Figura 4-10).

Atualmente, nas versões modernas do Pure Data e Max/MSP, uma mensagem de controle é composta por átomos. Um átomo pode ser um inteiro, ponto-flutuante ou um símbolo alfanumérico (por exemplo, “bang”). Na maioria das vezes que uma mensagem de controle é disparada (geralmente por um comando do usuário como um clique de *mouse*, por exemplo) ela desencadeia uma seqüência de novas mensagens de controle, como exemplificado na Figura 4-10.

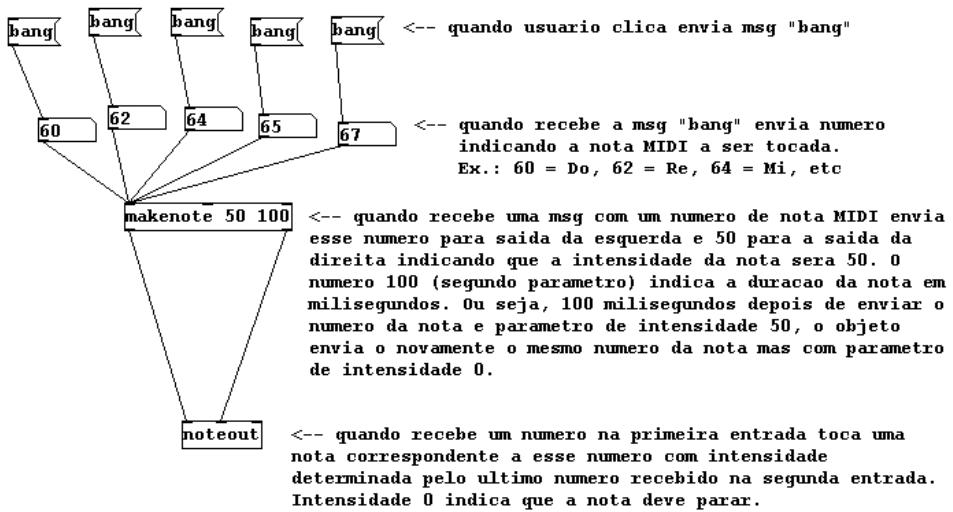


Figura 4-10 Exemplo de mensagens de controle para tocar notas musicais no Pure Data, utilizando saída MIDI.

Uma mensagem de sinal representa um fluxo constante de amostras de sinal de áudio. O conceito de mensagem de sinal, apesar de existir no Music500 (1982), não foi usado no Max/4X (1985) pelo fato do 4X já possuir uma interface própria para construção da síntese de áudio. Entretanto, no Max/FTS (1990) a idéia foi novamente implementada dentro do programa. Diferentemente da mensagem de controle que geralmente, é eventual e esporádica, a mensagem de sinal representa um *stream* (sucessão de dados que vão sendo disponibilizados com o passar do tempo) de áudio de um objeto para outro (veja Figura 4-11). Em cada saída de um objeto de sinal, a mensagem é representada na prática por um vetor a ser preenchido por esse objeto e lido pelo objeto que está ligado a esta saída.

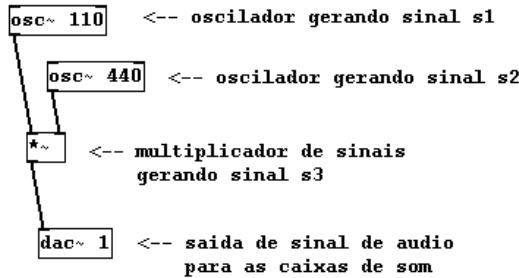


Figura 4-11 Exemplo de fluxograma do Pd com mensagens de sinal.

Quanto a restrição de tempo, uma mensagem de controle e todas as outras mensagens por ela desencadeadas devem ser processadas o mais rápido possível. Já uma mensagem de sinal deve ser processada periodicamente a um intervalo de tempo exato para que a taxa de amostragem do áudio (número de amostras por segundo) seja sempre respeitada. A coexistência desses dois mecanismos de troca e processamento de mensagens tão distintos numa mesma estrutura representa um desafio no projeto de um SOF.

O Pd foi desenvolvido com base no Max/FTS (PUCKETTE, 1991b) e o Max/MSP foi desenvolvido com base no Pd. Por tanto a compreensão do FTS, que significa *Faster Then Sound*, implica na compreensão do funcionamento destas implementações atuais do paradigma Max.

O Max/FTS solucionou o problema do processamento de mensagens de controle e sinal partindo da restrição de tempo da mensagem de sinal. Isto por que, enquanto a árvore de execução de uma mensagem de controle deve ser processada o mais rápido possível, as mensagens de sinal devem ser processadas obrigatoriamente em determinados instantes de intervalos fixos e precisos para que a fluência do áudio não seja prejudicada. Assim, o FTS determina que a um certo intervalo de tempo, um certo número de amostras passa por toda a seqüência de objetos de processamento de sinal do fluxograma. Chamaremos essa passagem de “ciclo *DSP*” (originalmente “*DSP duty cicle*”) (PUCKETTE, 1991b); o valor do tempo de duração de um ciclo *DSP* é chamado “*DSP tick*”. Entre um *DSP tick* e outro, as mensagens de controle são processadas pelos objetos de controle.

Um ciclo *DSP* é exemplificado na Figura 4-12 onde S1, S2 e S3 são os vetores que representam os sinais (*signals*, na parte de baixo da figura), sendo S1 e S2 as

saídas de objetos geradores de sinal e S3 a saída de um objeto que efetua uma operação de multiplicação de sinal (representado por um “x”).

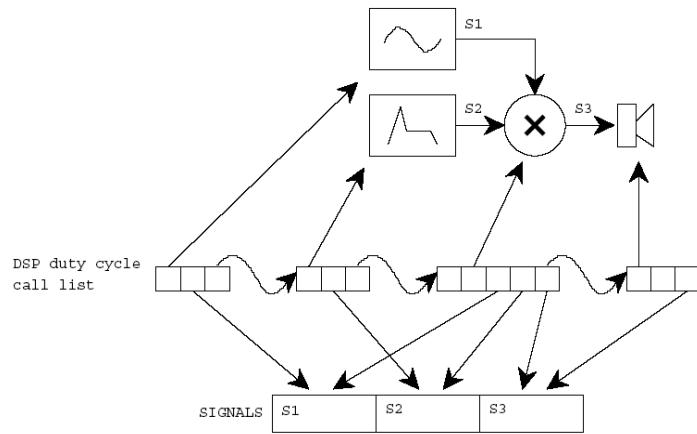


Figura 4-12 Esquema representando um ciclo DSP (PUCKETTE, 1991b)

Cada objeto tem sua ação executada quando chega a sua vez de acordo com a ordem em que está posicionada na lista de operações de processamento de sinal (“*DSP duty cycle call list*”). Esta ordem é muito importante, pois existe obviamente uma dependência temporal entre as operações de objetos de sinal: a multiplicação de sinais (que gera S3) não pode ser calculada antes que S1 e S2 já tenham sido gerados e estejam disponíveis. Essa ordem de execução de um ciclo *DSP* é calculada através de um algoritmo de ordenação topológica do grafo ordenado que representa o fluxograma. Esse algoritmo de ordenação é executado apenas quando ocorre alguma mudança no grafo, ou seja, uma nova conexão é feita ou apagada, ou ainda um objeto é adicionado ou apagado pelo usuário.

A cada *DSP tick*, um ciclo completo de processamento de sinal é executado. Todas as operações de sinal são executadas, desde o cálculo das saídas dos primeiros objetos, que não possuem conexão de entrada de sinal, como osciladores; até as operações dos últimos objetos, que não possuem conexão de saída de sinal, como o objeto que escreve na saída de áudio da placa de som. É processado um vetor para cada conexão de entrada e gerado um vetor para cada conexão de saída de cada objeto de sinal.

Por isso é necessário que um ciclo *DSP* seja chamado a cada intervalo de P segundos, sendo P igual ao número de amostras por vetor dividido pela taxa de

amostragem. Pois assim, a taxa de amostragem na saída será atendida, não comprometendo a continuidade do *stream* de áudio. No caso do Pd e Max/MSP, por padrão, o tamanho do vetor é de 64 amostras e a taxa de amostragem é de 44100 Hz. Por tanto a cada 1,45 milissegundos ($64/44100 \times 1000$), um ciclo *DSP* é processado.

O período de um ciclo *DSP* é a unidade fundamental de tempo. Existe um “tempo lógico” global que incrementa uma unidade a cada ciclo, ou seja, a cada *DSP tick*. O termo “tempo lógico” aqui é usado como em (FARINES *et al.*, 2000): “Tempo lógico: definido a partir das relações de precedência entre eventos, o que permite o estabelecimento de ordens causais sobre um conjunto de eventos.”

Uma mensagem de controle está sempre associada a um tempo lógico medido em quantidade de *DSP ticks* correspondente ao número de *DSP ticks* ocorridos desde o início da contagem até o instante em que foi criada. As mensagens de controle podem ser originadas de: ações do usuário (como apertar um botão), preenchimento de *buffer* de dispositivos de entrada e saída (MIDI, por exemplo) ou um evento indicando um tempo esgotado (para um objeto de *delay*, por exemplo). A cada *DSP tick*, todas as mensagens de controle agendadas para aquele instante serão enviadas. Só depois disso, as operações de DSP são executadas e o tempo lógico é incrementado finalizando o ciclo de processamento.

4.3.4 Processamento de vídeo

Com a criação dos pacotes de objetos para processamento de vídeo e gráficos vetoriais como o GEM (DANKS, 1997) para o Pd e o Jitter (JONES & NEVILE, 2005) para o Max/MSP, novos tipos de dados são enviados e processados pelos objetos. As duas extensões utilizaram diferentes abordagens para o processamento de vídeo no contexto da infra-estrutura de fluxograma desses sistemas.

GEM (*Graphics Environment for Multimedia*) é uma biblioteca para o Pure Data que permite processamento gráfico baseado em *OpenGL*. Isso torna possível o uso de primitivas gráficas para desenhos 2D e 3D como retângulo, círculo, cubo, esfera, etc. Além disso, *GEM* permite a possibilidade de uso de texturas *bitmap*, como imagens e vídeos. Tudo isso no contexto do Pure Data, ou seja, viabilizando a síntese de gráficos 2D e 3D integrada à síntese, processamento e análise de áudio.

Em 2001, Mark Danks deixou de participar ativamente do desenvolvimento do *GEM*, que passou então a ser mantido por Johannes Zmölning do IEM (*Institute of*

Electronic Music and Acoustics), Graz, Áustria. Apesar de estar disponível por vários anos, *GEM* sofreu várias modificações significativas antes do artigo (ZMÖLNIG, 2004b). Este artigo descreve essas mudanças feitas na arquitetura do *GEM*, que permitiram o uso do “grafo de renderização dinâmico”. O grafo de renderização é a estrutura de objetos a ser renderizada, incluindo suas transformações e informações de texturas, iluminação, etc. Por questões de desempenho, o *GEM* utilizava grafos de renderização estáticos, que não podiam ser alterados durante a renderização. Essa abordagem não era apropriada, pois a máquina de renderização tinha de ser reiniciada toda vez que um novo objeto era adicionado ao grafo. Com as mudanças, o construtor de grafo agora trabalha com grafos de renderização dinâmicos graças ao uso da máquina de troca de mensagens do próprio Pd.

A máquina de renderização do *GEM* possui um certo grau de complexidade devido ao fato de usar uma abordagem hierárquica que difere significativamente do modelo nativo do *Pd* de fluxo de sinal. Em (ZMÖLNIG, 2004a), Zmölnig tenta explicar a arquitetura do *GEM* partindo de como o mesmo funcionava em suas primeiras versões em meados dos anos 90 e o que foi mudando em sua essência com o passar dos anos.

A maior razão das mudanças no *GEM* foram por consequência da dificuldade de se integrar um “sistema de modelagem 3D” dentro da estrutura de fluxo de sinal do *Pd*. Linguagens como *Pd* possuem uma estrutura linear de processamento: um sinal é gerado numa fonte, segue passando por vários objetos que o modificam, e finalmente é enviado a uma saída de som. Este tipo de estrutura de troca de mensagens se aplica perfeitamente ao processamento de vídeo, porém não é tão apropriado para gráficos vetoriais que são geralmente melhor descritos por uma estrutura hierárquica.

Nas primeiras implementações do *GEM*, Mark Danks (DANKS, 1996) optou por tentar utilizar a estrutura linear de fluxo de sinal do *Pd* da seguinte forma: um objeto fonte gera uma figura geométrica que é enviada através de mensagens de controle para objetos que a transformam, adicionando modificadores como por exemplo, rotação e cores; essas informações vão sendo acumuladas na mensagem (chamada de *gemList*) cada vez que passa em um desses objetos até chegar no objeto “renderizador” que finalmente executa essa “lista” de ações (veja Figura 4-13). Esse modelo é simples e intuitivo, porém não é o modo como a máquina *OpenGL* opera e isso causa um custo de desempenho demasiado.

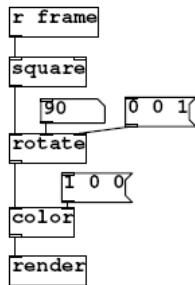
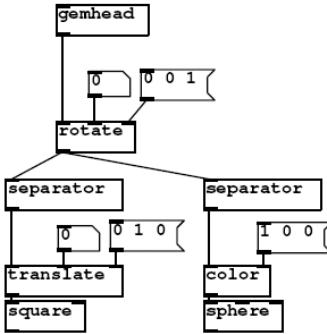


Figura 4-13 Um fluxograma do GEM nas primeiras versões, de 1996 (ZMÖLNIG, 2004a)

Em 1997, o *GEM* foi portado do Max para o Pd e sua arquitetura foi totalmente remodelada passando de um modelo “*bottom-up*” para um modelo “*top-down*”, que corresponde melhor ao funcionamento da máquina *OpenGL*. Pois *OpenGL* é um sistema baseado em estados, de modo que os vários modificadores como rotação, mudam um estado representado por uma matriz de transformação (ROGERS & ADAMS, 1990) sem precisar saber o que será desenhado. Apenas nas últimas etapas do processo de exibição da máquina *OpenGL* que o estado é aplicado aos vértices. Por tanto, é mais apropriado que o objeto que representa a figura geométrica apareça no final da cadeia de objetos, pois é o último elemento a ser processado.

Adequada a esse modelo, a versão de 1997 do *GEM* possui, para cada cadeia de renderização, um objeto chamado *gemhead* que representa um contexto válido de exibição. O objeto *gemhead* é chamado a cada ciclo de renderização e envia a matriz de transformação que representa o estado *OpenGL*. Cada objeto conectado modifica a matriz adicionando uma transformação. Por exemplo, no fluxograma da Figura 4-14, o objeto *gemhead* envia a matriz para o objeto *rotate* que adiciona uma transformação de rotação e passa a matriz resultante adiante para o próximo objeto conectado. A seqüência de modificações continua a cada nó da cadeia até que o objeto que “emite” os vértices (como *square* ou *sphere*), desenha seus vértices transformados de acordo com a matriz final. Esse processo se repete a cada ciclo de renderização.



**Figura 4-14 O fluxograma hierárquico do GEM,
de 1997 até 2003. (ZMÖLNIG, 2004a)**

Entretanto, esta arquitetura do *GEM* ainda apresentava alguns problemas por utilizar um grafo de renderização estático, representado pelo objeto *GemDAG* (*DAG* é a abreviação de “*directed acyclic graph*”). Apesar da vantagem de se permitir o uso das eficientes *displaylists* (WOO *et al.*, 2003), listas pré-compiladas de comandos *OpenGL*, o uso de grafo de renderização estático apresenta grandes desvantagens. O principal problema é a impossibilidade de modificar o grafo depois de iniciada a renderização.

Para resolver este problema, em 2003, a arquitetura do *GEM* foi novamente alterada. A mudança consistiu em eliminar o objeto *GemDAG* e utilizar o próprio sistema de mensagens do Pd para chamada da função de renderização de cada objeto. Assim, em vez de cada objeto registrar previamente sua função de renderização em um objeto *GemDAG*, agora cada objeto tem sua função chamada dinamicamente quando recebem uma mensagem do objeto anterior na cadeia de objetos.

Um ciclo de renderização ocorre a um intervalo fixo determinado pela taxa de quadros por segundo que por padrão é 20, mas que pode ser alterada pelo usuário. No final de um ciclo de renderização, um novo ciclo é agendado para acontecer 1/20 segundos depois (caso a taxa seja 20). Isto significa que, na prática, a taxa de quadros por segunda jamais é respeitada, pois sempre ocorre um atraso correspondente ao tempo necessário para completar um ciclo de renderização. No *DSP tick* (tempo lógico) em que esses 1/20 segundos tiverem esgotados, o próximo ciclo de renderização é disparado. Caso a renderização demore mais que o intervalo de um *DSP tick*, o processamento de áudio é prejudicado, pois sofre um atraso que acaba sendo percebido como um corte de sinal no áudio.

Enquanto o *GEM* tem uma abordagem com foco principal em oferecer as funcionalidades de *OpenGL*, o Jitter (JONES & NEVILE, 2005) soluciona o problema de processamento e síntese gráfica de uma forma mais geral. Pois, além de oferecer uma interface a *OpenGL* para uso dentro dos fluxogramas, este pacote permite a manipulação de dados multidimensionais no contexto do ambiente de programação do Max/MSP.

Assim, com a introdução do Jitter, um novo tipo de dado foi acrescentado ao Max/MSP: a matriz multidimensional. As matrizes do Jitter podem armazenar quatro tipos de dados: *char* (8 bits sem sinal), *int* (32 bits), *float32* (32 bits), *float64* (64 bits). As matrizes passam de um objeto a outro por referência dentro de mensagens compostas pelo símbolo *jit_matrix* seguido do nome da matriz. As matrizes podem ser utilizadas para representar imagens, vídeos, vértices de um objeto geométrico 3D, assim como qualquer outro tipo de informação vetorial, inclusive sinal de áudio.

Apesar do GEM não oferecer um mecanismo específico para manipulação de matrizes existem outros pacotes de objetos do Pure Data que oferecem essa funcionalidade como o GridFlow (BOUCHARD, 2007) de Mathieu Bouchard e o PDP (SCHOUTEN, 2007) de Tom Schouten, que podem ser usados em conjunto com o GEM.

Devido ao fato de percebermos variações de tempo através da audição muito mais precisamente que através da visão, as restrições de tempo para o processamento de áudio são muito maiores que as restrições de tempo para processamento visual. Assim, o Jitter foi projetado para que sua taxa de quadros por segundo se adapte ao poder de processamento disponível na máquina após a computação do áudio e de outros eventos de alta prioridade ter sido feita.

Isto é possível por que o Max/MSP adota um esquema de *threads* diferente do Pd. Enquanto o Pd possui apenas uma *thread* para todo o processamento do fluxograma, as versões atuais do Max/MSP adotam um esquema de três *threads*. A chamada *main thread* é de baixa prioridade e é responsável por operações curtas e custosas como processamento de mensagens e ações disparadas por objetos de interface de usuário. A *scheduler thread*, de alta prioridade, é responsável por eventos que têm restrição de tempo como comandos MIDI ou mensagens vindas de objetos do tipo metrônomo. E finalmente a *perform thread* de altíssima prioridade, executa as operações de processamento de sinais, que não podem atrasar para que o sinal de áudio não seja interrompido.

Apesar da ordem de prioridade, quando o Jitter roda em sistemas operacionais de multitarefas preemptivas (como é o caso do Windows XP e do Mac Os X) qualquer *thread* pode interromper outra. Isso pode causar situações confusas para o usuário, pois objetos do fluxograma podem ser interrompidos no meio de um cálculo. Por isso o Max/MSP oferece opções para mudar o modo de funcionamento das *threads*. A opção *overdrive* quando desligada, elimina a *scheduler thread* e as tarefas de alta e baixa prioridade são executadas conjuntamente na *main thread*. Já a opção *Scheduler In Audio Interrupt (SIAI)* quando ligada faz com que as tarefas de alta prioridade sejam executadas após cada *DSP tick*, ou seja, cada vez que os processamentos de sinal são executados. Por tanto, quando a opção *SIAI* está ligada, as tarefas de alta prioridade ficam sendo executadas pela *perform thread*. A Tabela 2 mostra as possíveis configurações dessas duas opções e o efeito correspondente no funcionamento das *threads*.

Tabela 2 Tabela de possíveis configurações de funcionamento da *threads* no Max/MSP

<i>overdrive</i>	<i>SIAI</i>	Efeito no funcionamento das <i>threads</i>
ligado	desligado	<i>main thread</i> (baixa prioridade), <i>scheduler thread</i> (alta prioridade) e <i>perform thread</i> (altíssima prioridade).
ligado	ligado	<i>main thread</i> e <i>perform thread</i> (executando tarefas de alta prioridade, que normalmente seriam executadas pela <i>scheduler thread</i>)
desligado	ligado	<i>perform thread</i> (executando tarefas da <i>main thread</i> e da <i>scheduler thread</i>), ou seja o sistema fica com apenas uma <i>thread</i> única para mensagens de alta e baixa prioridade e processamento de sinais, como ocorre no Pure Data.
desligado	desligado	<i>main thread</i> (executando tarefas de alta prioridade, que normalmente são executadas pela <i>scheduler thread</i>) e <i>perform thread</i>

Os objetos do Jitter geralmente são disparados a partir da *main thread*, tentando operar na melhor taxa de quadros por segundo possível utilizando o poder de processamento que estiver disponível. Isso significa que, diferentemente do Pd/GEM, mesmo que o tempo de um ciclo de renderização ultrapasse o tempo de um *DSP tick*, o áudio não é prejudicado, pois a *thread* de renderização (*main thread*) é interrompida pela *thread* de processamento de áudio (*perform thread*) que então executa o ciclo

DSP. A não ser que a opção *overdrive* esteja desligada e *SIAI* esteja ligada (veja Tabela 2).

Além desta otimização oferecida pelo esquema de *multithreads* do Max/MSP, para fins de aumento de desempenho, o Jitter ainda cria internamente várias outras pequenas *threads* específicas para operação paralela de matrizes muito grandes (JONES & NEVILE, 2005).

A plataforma de *software* Aura (DANNENBERG, 2004) também utiliza a idéia de *threads* com prioridades fixas para a solução do problema de integração de processamento de áudio, vídeo e controle num sistema de tempo real. Para isso, Roger Dannenberg separou os objetos em três *threads* (às quais dá o nome de “zonas”) de acordo com sua finalidade e restrição de tempo: uma *thread* de maior prioridade para os processamentos de áudio que possuem a maior restrição de tempo (*hard-deadline*); outra para processamento de mensagens de controle (como MIDI, por exemplo) que possuem uma restrição de tempo um pouco menos rígida (*soft-deadline*); e uma terceira para o processamento de vídeo cuja latência é a mais tolerável de todas.

Segundo (FARINES *et al.*, 2000) um sistema tempo real tenta solucionar um problema de concorrência podendo adotar duas abordagens: assíncrona e síncrona. A abordagem assíncrona trata a percepção de eventos independentes numa ordem arbitrária e não simultânea, considerando realmente o tempo físico em que ocorrem. Já na abordagem síncrona o tempo não é tratado de forma explícita, sendo visto com granularidade suficientemente grossa para que eventos próximos sejam considerados simultâneos, ou seja, ocorridos no mesmo instante no tempo lógico. Dessa forma os atrasos causados pelo tempo de processamento de uma instrução são abstraídos na abordagem síncrona, pois são menores que o período de incremento do tempo lógico, que é sempre calculado para ser suficientemente grande para que isso seja verdade.

Assim, podemos concluir que o Pd é um sistema que utiliza uma abordagem síncrona caracterizada pelo escalonamento de tarefas semelhante ao “Executor Cíclico” (“Cyclic Executive”), padrão de projeto para tempo real como definido em (DOUGLASS, 2000). Ou seja, o Pd simplesmente executa a cada laço principal, as seguintes tarefas numa seqüência pré-definida: processa as mensagens de controle e executa o ciclo *DSP*. As mensagens de controle incluem entradas do usuário, renderização de um quadro do GEM, mensagens MIDI, etc. Isso significa que, mesmo que a renderização de um quadro demore mais que 1,45 milissegundos (tempo padrão

de um *DSP tick*), o ciclo *DSP* não é executado até que a renderização do quadro termine, causando um corte no sinal de áudio.

Já o Max/MSP/Jitter e Aura são sistemas de tempo real assíncronos e utilizam o padrão de projeto para tempo real “Multitarefas Preemptivas” (“*Preemptive Multitasking*”) no modo de prioridade estática como definido em “*Real-time Design Patterns*” (DOUGLASS, 2000), ou “Escalonamento de Prioridade Estática” (“*Static Priority Scheduling*”) como definido em “*Design Patterns for Real-Time Computer Music Systems*” (DANNENBERG & BENCINA, 2005). Este padrão determina que tarefas com maior restrição de tempo possam interromper outras com menor restrição de tempo. Por exemplo, o processamento de sinal de áudio (ciclo *DSP*) deve ser executado obrigatoriamente a um determinado intervalo de tempo fixo. Caso chegue o instante em que o ciclo *DSP* deva ocorrer e uma mensagem de controle para renderização de um quadro do Jitter esteja em execução, a renderização é interrompida e o ciclo *DSP* é executado.

Entre as vantagens da abordagem utilizada pelo Pd está a garantia de um determinismo que facilita a depuração de erros. Além disso, o “Executor Cíclico” é extremamente simples de ser implementado e depurado. Por outro lado a abordagem utilizada pelo Max/MSP apresenta melhor otimização do tempo, tira proveito da existência de mais de um processador (multiprocessadores), pois, nesse caso, duas *threads* podem ser executadas simultaneamente.

4.4 Considerações sobre interface humano-máquina

Nesta sessão apresentaremos algumas questões relativas à interface humano-máquina dos SOFs. Esta discussão é importante para a compreensão das escolhas de interface dos SOFs, e o entendimento do porquê de algumas dessas características também terem sido adotadas no ViMus.

A interface gráfica de um SOF utiliza basicamente a metáfora de um papel em branco onde o usuário desenha um fluxograma representando esquematicamente a cadeia de processamentos de áudio e vídeo que ele deseja criar e interagir em tempo real. Detalhes sobre como o fluxograma é representado e editado variam de um SOF para outro, entretanto o aspecto geral da interface é quase sempre o mesmo: uma janela contendo a imagem do fluxograma e alguns menus de ferramentas para sua

edição. Geralmente é permitida a execução de vários fluxogramas simultaneamente e para cada fluxograma aberto existe uma janela com as mesmas funcionalidades, menus e barras de ferramentas, além de uma janela principal da aplicação (veja a Figura 4-15).

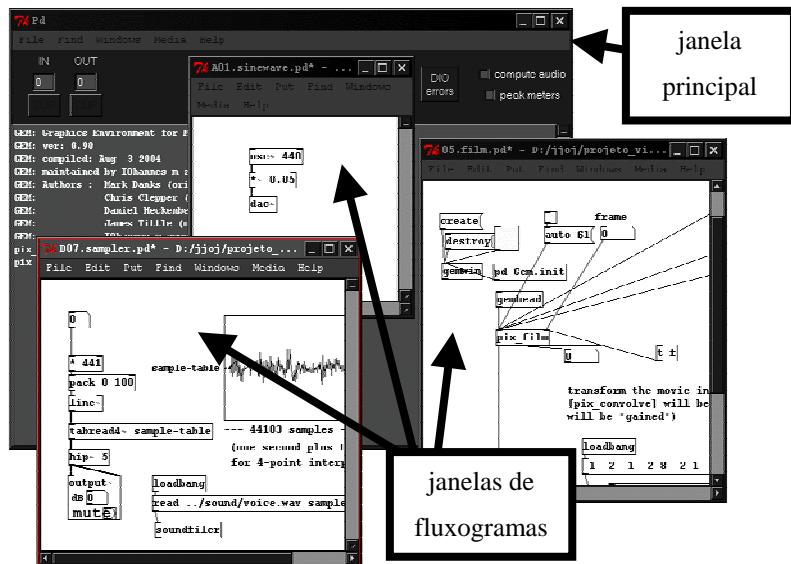


Figura 4-15 Exemplo de 3 fluxogramas abertos no Pd

Basicamente os objetos dos fluxogramas são representados por desenhos de caixas, suas entradas e saídas por desenhos de pinos e as conexões por linhas. Um ponto essencial a ser observado para se fazer a leitura correta de um fluxograma é saber a direção e sentido do fluxo de mensagens. Quando a direção do fluxo é vertical no sentido de cima para baixo, os pinos superiores dos objetos são sempre entradas e os pinos inferiores são sempre saídas (ex.: Pd, Max, vvvv, Visual Jockey). Quando a direção do fluxo é horizontal no sentido da esquerda para a direita, os pinos à esquerda dos objetos são sempre entradas e os pinos à direita são sempre saídas (ex.: Isadora, Quartz Composer, GePhex, Sonnet).

Os objetos podem ser identificados de duas formas: através de um ícone (ex.: o desenho de uma caixa de som representando a saída de áudio) ou através de um texto (ex.: “adc~” que é a abreviação de “analog-digital converter” representando a entrada de áudio). Entre os sistemas que utilizam ícones podemos citar Isadora, GePhex,

Visual Jockey e entre os sistemas que utilizam texto podemos citar o Pd, Max, vvvv e Quartz Composer.

Em “Max at Seventeen” (PUCKETTE, 2002), Miller Puckette apresenta três razões para a escolha de utilizar texto em vez de ícones para identificar os objetos. A primeira é que, de qualquer forma o usuário precisará de uma interface de texto para especificar os parâmetros de criação do objeto. A segunda é que geralmente existem dezenas de objetos disponíveis e esse número pode crescer sem limites já que o usuário pode criar seus próprios objetos. A partir de dez ícones distintos pode-se tornar difícil lembrar qual ícone corresponde a qual funcionalidade; já o texto funciona melhor mnemonicamente. A terceira razão é que a representação textual apresenta mais informações por centímetro quadrado de tela do que ícones.

Os sistemas do paradigma Max diferenciam os objetos de controle dos objetos de sinal através de um “~” adicionado à direita do nome de todos os objetos de sinal (que por isso são apelidados de “*tilde objects*”). Além disso, as conexões que transportam mensagens de sinal são desenhadas por linhas mais grossas que as conexões que transportam mensagens de controle. Essa diferenciação não é feita pela maioria dos outros SOFs como vvvv, Isadora, Visual Jockey, GePhex e Quartz Composer. Já o Max/MSP/Jitter, além de usar linhas mais grossas, de pontos alternados e de cor amarela para representar conexões de mensagens de sinal de áudio, usa linhas grossas e verdes para representar conexões de mensagens que transportam matrizes (vídeo, gráficos).

Miller Puckette optou por não deixar explícito nas caixas o que cada entrada recebe e cada saída retorna, através de um texto em cada pino de entrada e saída. Provavelmente esta escolha foi feita para deixar a imagem do fluxograma mais “limpa”, entretanto isso diminui a transparência e dificulta a compreensão do que está ocorrendo. O Isadora, além de mostrar o tipo ou, no caso de um número, o valor que está passando pelo pino, também muda a cor da conexão de vermelho para verde no momento em que alguma mensagem está passando, para indicar que naquele momento está ocorrendo um fluxo de mensagens ali.

Um conceito importante e comum em SOFs é a possibilidade de abstração de fluxogramas, ou seja, o usuário pode criar um fluxograma e utilizá-lo como um novo objeto que pode ser usado dentro de outro fluxograma. Este mecanismo facilita a leitura e reutilização de código (considerando fluxogramas como sendo códigos, nesse caso, de linguagens de programação visual).

Na maioria dos SOFs (incluindo Pd, Max e Isadora), quando um fluxograma possui um objeto que é uma abstração de outro fluxograma (criado pelo usuário ou não) ele pode ser “expandido” em uma nova janela para que seja possível visualizá-lo e editá-lo. No Pd, por exemplo, isso é possível utilizando *subpatches* (quando o fluxograma abstraído é salvo no mesmo arquivo do fluxograma “pai”, veja Figura 4-16) ou *abstractions* (quando o fluxograma abstraído é salvo em um arquivo separado).

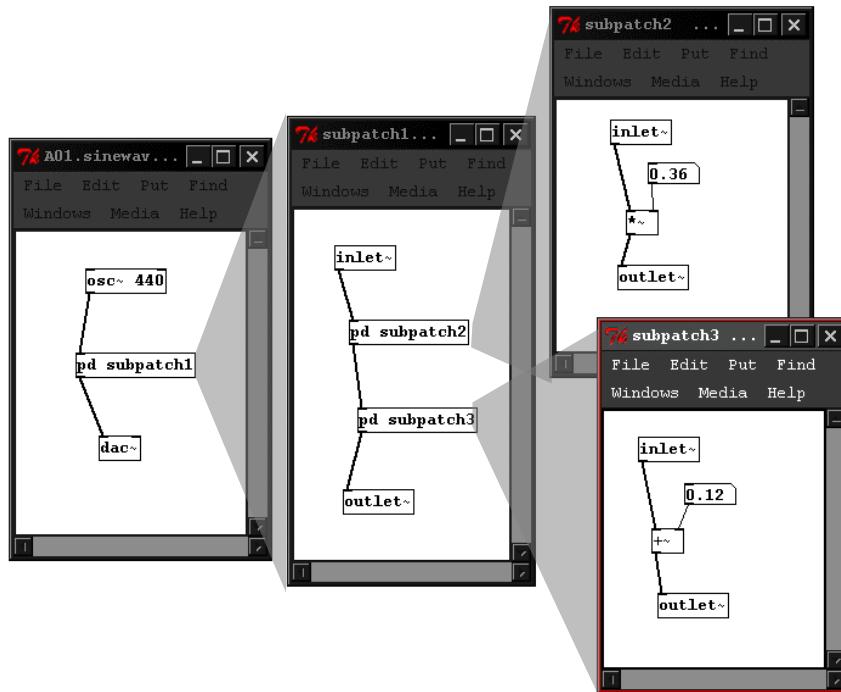


Figura 4-16 Abstrações de fluxogramas no Pd

Existe um tipo especial de objeto responsável por receber entradas do usuário quando o fluxograma estiver sendo executado. No Pd, esses objetos são chamados “caixas de interface” (“*interface boxes*”), no Max/MSP “objetos de interface” (“*interface objects*”) e no vvvv, “*IOBoxes*”. Exemplos comuns desse tipo de objeto são: botões, barras deslizantes (*sliders*) e caixa de verificação (*checkbox*) (veja a Figura 4-17). Esses objetos possibilitam a construção de uma interface entre o fluxograma e o usuário, permitindo uma forma de interação mais fácil e prática do que modificar diretamente o esquema de conexões, ou adicionar e apagar novos objetos no momento de uma apresentação.

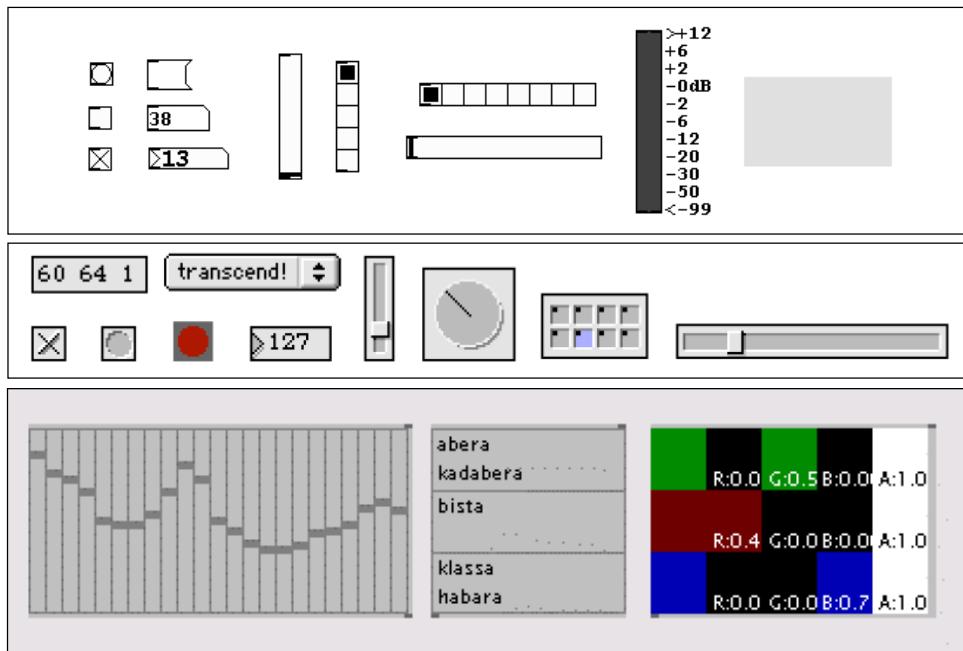


Figura 4-17 Objetos de interface no Pd (1^a. quadro), Max (2^a. quadro) e vvvv (3^a. quadro).

Os objetos de interface capturam as entradas do usuário e imediatamente enviam essas informações em forma de mensagens de controle para seus pinos de saída que podem estar ligados a quaisquer outros objetos do fluxograma. Por exemplo, sempre que o usuário arrasta o botão da barra de rolamento de um objeto “*vslide*” este envia uma mensagem de controle para seu pino de saída informando o novo valor.

O Isadora oferece um modo de visualização chamado “painele de controle” que é separado da visualização do fluxograma. No modo painel de controle é possível montar uma interface gráfica para o fluxograma utilizando os objetos de interface. Este mecanismo possibilita a criação de uma pequena aplicação cujos fluxogramas são completamente abstraídos pelo usuário. A Figura 4-18 apresenta o painel de um fluxograma construído no Isadora. Este fluxograma é funcionalmente um SOAVE simples, com alguns efeitos, e que utiliza a metáfora do *mixer A/B* (apresentada na sessão 3.3).



Figura 4-18 Fluxograma de exemplo do Isadora semelhante a um SOAVE do tipo mixer A/B.

Os sistemas do paradigma Max, assim como o Salvation, oferecem essa mesma possibilidade, porém de uma maneira mais flexível e sofisticada. Estes sistemas permitem que os objetos de interface de abstrações de fluxogramas apareçam diretamente nos objetos que representam essas abstrações. Com isso é possível construir níveis de abstração de fluxogramas. No Pd e Max/MSP essa funcionalidade é chamada “*graph-on-parents*”.

As figuras a seguir apresentam um exemplo de uso do *graph-on-parent* no Pd. A Figura 4-19 apresenta um fluxograma (esquerda) e um *subpatch* (direita) aberto em outra janela. A Figura 4-20 apresenta o mesmo fluxograma e *subpatch*, porém com a opção *graph-on-parent* ligada na janela do *subpatch*. Quando ligamos esta opção, um objeto *graph* (retângulo vermelho) aparece na tela. Esse retângulo representa os limites do “painel de controle” do *subpatch* a ser exibido no fluxograma pai (esquerda). Qualquer objeto de interface colocado dentro dos limites deste retângulo vermelho aparece automaticamente no fluxograma pai. A Figura 4-21 apresenta o resultado final da operação, isto é, quando fechamos a janela do *subpatch* e o objeto de interface *slide* aparece no fluxograma pai.

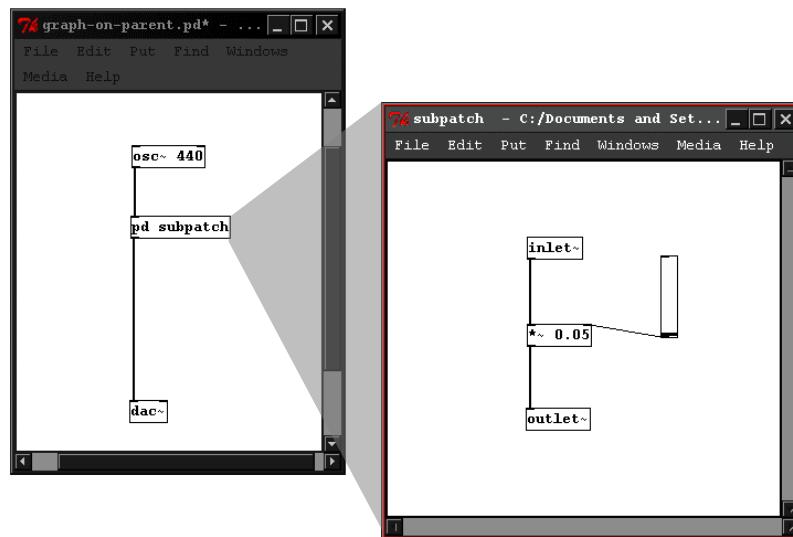


Figura 4-19 Exemplo de *patch* e *subpatch* com a opção de *graph-on-parent* desligada.

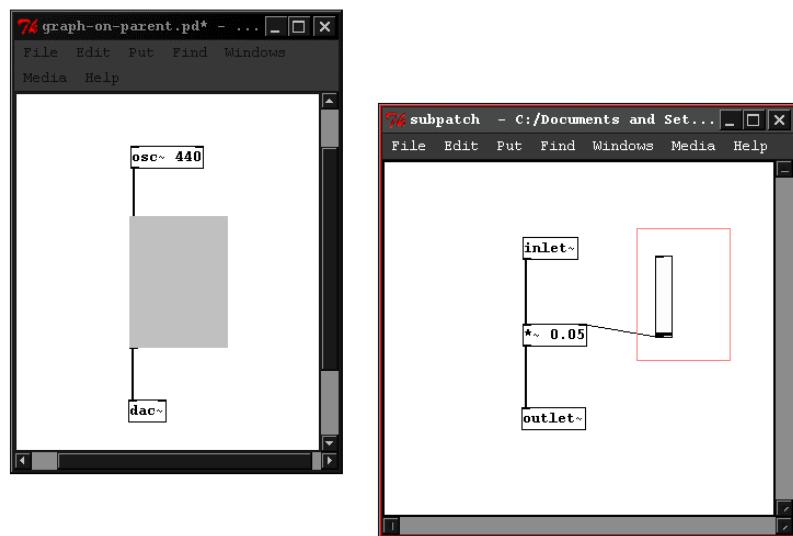


Figura 4-20 Mesmo exemplo com a opção de *graph-on-parent* do *subpatch* ligada.

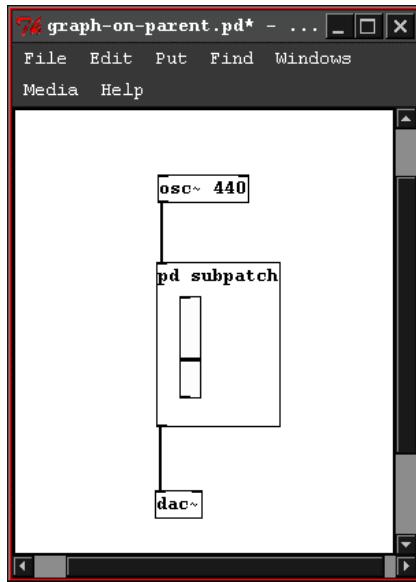


Figura 4-21 Fluxograma "pai" quando a opção *graph-on-parent* do filho está ligada e sua janela fechada: os objetos de interface (ex.: *slide*) ficam visíveis no pai.

O mecanismo de “*graph-on-parent*” é extremamente importante para construção de interfaces gráficas mais amigáveis para os fluxogramas, justamente pelo fato de que uma abstração de fluxograma pode “esconder” seus detalhes, mas sem deixar de exibir botões, *sliders* e outros objetos de interface.

A Figura 4-22 apresenta uma tela do fluxograma Dervish (GOLDBERG, 2005), um SOAVE que, assim como o AVmixer, foi desenvolvido no Max/MSP/Jitter e utiliza o mecanismo de *graph-on-parent*. Note que os diversos módulos exibem seus botões e *sliders*, mesmo que seus fluxogramas (*sub-patches*) não estejam expandidos, ou seja, no fluxograma “pai”, apenas os objetos de interface ficam visíveis.

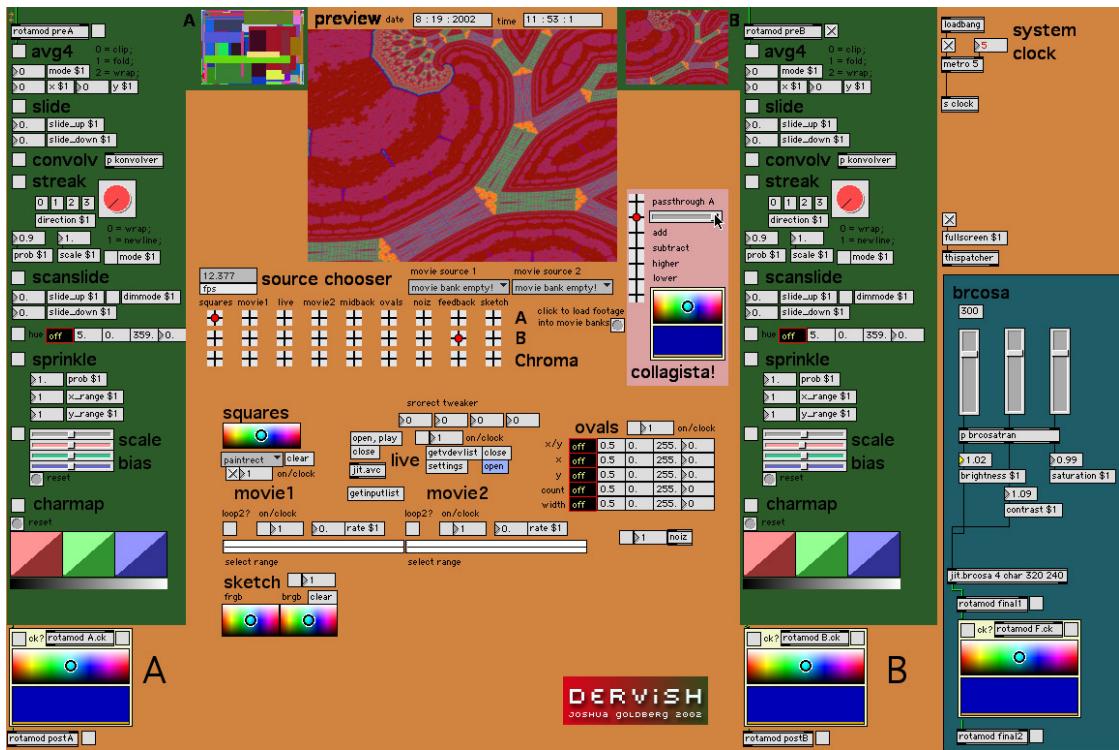


Figura 4-22 Fluxograma construído com Max/MSP/Jitter semelhante a um SOAVE mixer A/B.

Existem ainda outras possibilidades de exibição de um fluxograma sem necessariamente utilizar o desenho de caixas e linhas. Uma delas é a utilizada pelo AVS (*Advanced Visualization Studio*) um *plugin* para visualização do tocador de arquivos MP3 chamado Winamp (NULLSOFT, 2007). O AVS permite um “cascateamento” de efeitos visuais, ou seja, a criação de uma árvore de efeitos cujos nós são simplesmente representados por texto em uma lista ordenada e tabulada de acordo com a hierarquia da árvore. Esta estrutura é semelhante à interface gráfica de gerenciadores de arquivo comuns em sistemas operacionais. As vantagens dessa representação é a facilidade de navegação e economia de espaço na tela.

Outro exemplo de representação alternativa é a proposta por Jean-Marc Pelletier em “A Graphical Interface for Real-Time Signal Routing” (PELLETIER, 2005). A idéia é representar as conexões entre os objetos através de diagramas semelhantes aos usados para representação de conjuntos em matemática. Assim, se uma caixa C está contida na caixa D, significa que a saída de C está ligada à entrada de D. Os objetos de entrada (ou seja, objetos geradores, que não possuem pinos de entrada) são representados por pontos e o objeto de saída é representado pela caixa que contém todos os demais.

A grande vantagem dessa representação é a possibilidade de alterar as conexões de forma extremamente rápida em tempo real, apenas movendo livremente as caixas pelo diagrama. Por exemplo, no diagrama da esquerda na Figura 4-23, para desconectar o objeto B de D e conectar-lo ao objeto C, bastaria simplesmente mover o objeto B para dentro da caixa C. Na interface de fluxograma tradicional, seria necessário clicar na conexão de B para D, apagá-la, clicar no objeto B, arrastar o cursor para o objeto C e finalmente soltar para que a nova conexão fosse estabelecida.

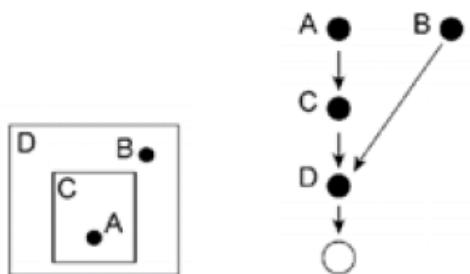


Figura 4-23 À esquerda interface de Pelletier e à direita interface tradicional de objetos e linhas (PELLETIER, 2005)

A desvantagem da abordagem do AVS e de Pelletier, é que elas são mais apropriadas para fluxogramas de sistemas de digramas de fluxos de dados (*dataflow*), pois não permitem a representação de diferentes tipos de conexões entre os objetos.

Apresentadas essas considerações sobre a interface humano-máquina dos SOFs, podemos notar que apesar da complexidade de operação inerente a este tipo de sistema, existem justificativas coerentes para muitas de suas escolhas de interface. A perda de usabilidade com a introdução da funcionalidade de edição de fluxogramas é realmente inevitável. Entretanto, existem alternativas para se amenizar esta perda.

5. Usabilidade x Expressividade

5.1 Considerações sobre expressividade

Definiremos expressividade de um sistema interativo de tempo real para processamento audiovisual integrado como uma estimativa de seu poder para solucionar problemas envolvendo o controle de processamento de mídias. Este poder é determinado por funcionalidades, pontos de função perceptíveis pelo usuário, que permitem sua intervenção direta no cálculo das saídas. Essas funcionalidades são executadas por um conjunto de subprogramas (sub-rotinas, funções ou métodos) cujas entradas e saídas podem ser dados de controle (parâmetros para funções de processamento, eventos) e dados de mídia (áudio, vídeo ou imagem).

A quantidade de funcionalidades que consistem em subprogramas que recebem e geram apenas dados de mídia (ex. efeitos de vídeo), contribui para o aumento da expressividade. Por exemplo, se adicionarmos mais opções de efeitos de vídeo a um SOAVE, ele oferecerá mais opções de solução para o problema de sintetizar efeitos em vídeo.

Já funcionalidades que permitem ao usuário a possibilidade de escolher como será o fluxo de processamento das mídias e das mensagens de controle, contribuem muito mais para o aumento da expressividade. Pois, a opção de associar saídas de um subprograma a entradas de outro, possibilita a solução de um número maior de problemas. Por exemplo, a funcionalidade de camadas sobrepostas presente na maioria dos SOAVEs, permite que o usuário utilize diferentes fontes de vídeo como entrada para as diferentes camadas. Outro exemplo é a opção que o Resolume oferece ao usuário de poder utilizar a saída da análise de áudio (FFT) para controlar automaticamente a execução de um vídeo, parâmetros de efeitos ou o grau de transparência de uma camada de vídeo.

Ainda assim, a expressividade dos SOAVEs é bastante limitada, pois as possibilidades de escolhas de como será o fluxo de processamento das mídias e das mensagens de controle no Resolume, são pré-determinadas e fixas. Isto é, o caminho de fluxo da informação já está estabelecido. O que o usuário pode fazer é apenas permitir ou não a passagem das mensagens por aquele caminho. Por exemplo, no caso

da funcionalidade de camadas sobrepostas (*mixer*), o número de camadas é fixo e imutável. Além disso, o usuário não pode ligar a saída de vídeo resultante da mistura das camadas à entrada de vídeo de um segundo *mixer*, por exemplo.

Existem, no entanto, SOAVEs mais avançados como o Modul8, que permite a construção de novos módulos de interface. Essa funcionalidade possibilita a criação de novos fluxos de processamento, pois é possível associar as saídas de subprogramas já existentes no Modul8 às entradas dos módulos, que por sua vez podem ter saídas de seus elementos de interface associadas às entradas de outros subprogramas do sistema. Além disso, o Modul8 permite o uso de *scripts* para criação de novas funcionalidades. Porém, tanto os módulos como os scripts atuam majoritariamente no fluxo de mensagens de controle e não permitem manipulações mais avançadas das mídias, como processamento a nível de *pixels* ou processamento de áudio.

Os SOFs, por sua vez, oferecem uma linguagem de programação visual e nesse caso a expressividade pode ser naturalmente associada ao conceito de poder de expressão de linguagens de programação cuja formalização matemática vem sendo pesquisada há alguns anos (FELLEISEN, 1990) e (MATSUSHITA, 1998). Naturalmente, a expressividade de um sistema que oferece uma linguagem de programação (seja, visual ou textual) é potencialmente muito maior que um sistema que apenas permite algumas mudanças contáveis de fluxo de processamento como o Resolume ou ArKaos.

Para os SOFs, podemos considerar que a expressividade é determinada pelo poder de expressão de sua linguagem de programação como o faz Miller Puckette em “Max at Seventeen” (PUCKETTE, 2002):

“Max é mais orientado a processos do que a dados. (Não existe a noção de uma partitura musical, por exemplo.) Se nós pensarmos em um fluxograma do Max como uma coleção de caixas interconectadas por linhas, a expressividade do Max vem de suas interconexões e aparatos de intercomunicação, enquanto o conteúdo das próprias caixas é geralmente invisível para o usuário.”

De maneiras diferentes Felleisen e Matsushita colocam que duas propriedades conjuntamente determinam se uma linguagem de programação tem mais poder expressivo que outra: sua capacidade computacional (CHOMSKY, 1956), isto é, a quantidade de algoritmos que podem ser escritos; e a concisão, isto é, quão concisos

são os textos que representam esses algoritmos. Esses dois fatores podem ser observados em SOFs para sabermos quando um é mais expressivo que outro.

Sistemas como Visual Jockey e GePhex, oferecem uma linguagem visual interessante porém limitada em termos computacionais. A principal limitação desses sistemas é o fato de que existem somente objetos (nós) capazes de processar e gerar como resultados apenas dados de mídia, não existindo nós para o processamento de dados de controle. Isto é, a linguagem não oferece construtores que possibilitem a expressão de instruções de controle de fluxo mais avançadas como laços e condicionais.

Já as linguagens do paradigma Max são provavelmente os SOFs mais expressivos que existem, pois possibilitam a representação de praticamente qualquer algoritmo que possa ser escrito em uma linguagem universal de alto-nível como C, por exemplo. Por isso, diferentemente do Isadora e Visual Jockey, o Max/MSP/Jitter e o Pd/GEM possuem objetos mais básicos, capazes de executar operações mais atômicas, em um nível mais baixo de abstração.

Miller Puckette argumenta que os objetos do Max não são feitos para serem musicalmente inteligentes, mas sim transparentes, ou seja, seu funcionamento é tão simples que se torna claro e facilmente comprehensível. “A falta de ‘inteligência’ nesse sentido não limita a expressividade de uma linguagem visual. Em vez disso, a expressividade é incrementada pela ‘concretude’ (como oposto de abstração), objetividade, clareza e associação direta.” (PUCKETTE, 2002)

Apesar de ser possível fazer essas observações a respeito dos SOFs e SOAVEs, é extremamente difícil medir com exatidão a expressividade desses sistemas. O cálculo exato do número de subprogramas, multiplicado pelo número de entradas e saídas de cada subprograma, multiplicado pelo número de possibilidades de ligações que podem ser feitas entre essas entradas e saídas ainda não seria suficiente, pois não leva em consideração o aspecto qualitativo de cada subprograma e cada parâmetro de entrada. Isto é, este cálculo não leva em consideração o impacto de cada subprograma, ou de cada parâmetro de entrada, na capacidade de resolver um número maior de problemas.

Ainda assim, analisando vários fatores em conjunto, incluindo análise funcional, poder computacional (poder expressivo no caso dos SOFs) fica evidente a existência de uma graduação crescente de expressividade entre os SOAVEs mais simples como Resolume, até os SOFs mais complexos como Pd/GEM.

5.2 Considerações sobre usabilidade

De acordo com a definição do padrão ISO 9241-11 (1998), usabilidade é a medida para avaliar o quanto um produto pode ser usado por usuários específicos para atingir objetivos específicos com efetividade, eficiência e satisfação em um determinado contexto de uso (USERFOCUS, 2007). A palavra efetividade (tradução de *effectiveness*) está sendo empregada nesta definição no sentido de “adequado ao propósito de uso”. A facilidade de aprendizagem (*learnability*) e a facilidade de memorização (*memorability*) são outros fatores apontados por Jacob Nielsen como determinantes para a usabilidade de uma interface gráfica (NIELSEN, 2003).

Essas definições deixam claro que para se avaliar a usabilidade de um *software* devemos levar em consideração o contexto e propósito de uso. A princípio, SOFs e SOAVEs são destinados a atividades diferentes. SOFs são destinados à atividade de construção e modificação de diversas aplicações multimídia para tempo real. SOAVEs são destinados à atividade de VJing que se divide em: montagem de uma apresentação (tempo de preparação) e controle e manipulação das mídias (tempo de apresentação).

Entretanto, devido à limitação de expressividade dos SOAVEs, como apresentado na sessão anterior, muitos VJs sentem a necessidade de utilizar SOFs para criar seus próprios efeitos e mecanismos de interação entre as mídias. Por isso avaliamos alguns aspectos da usabilidade dos SOFs no contexto de uso dos VJs e apenas confirmamos a clara evidência de que os SOFs apresentam sérios problemas de usabilidade quando utilizados para esse propósito. Os problemas mais evidentes são: a dificuldade de aprendizagem e a falta de eficiência.

Na verdade a dificuldade de aprendizagem é inerente à atividade de programação. Então, até independentemente do contexto de uso, os SOFs são geralmente mais difíceis de se aprender a usar. Isto vale tanto para VJs, como para artistas de novas mídias em geral que têm interesse em utilizar o computador como meio para construir suas obras de arte interativa mas não são programadores. Existe uma barreira cultural que desestimula a entrada do artista em um ambiente computacional no qual ele é o programador. Isto pode ser verificado com os resultados de uma pesquisa com VJs (Anexo A: **Questionário sobre softwares para**

VJs) na qual todos os entrevistados apontaram os SOFs como os sistemas mais difíceis de se aprender.

Ao iniciar o Pd e qualquer SOF em geral, o usuário vê apenas uma página em branco, diferentemente da maioria dos *softwares* que as pessoas não-especialistas estão acostumadas a utilizar. Essa escolha é justificada por Miller Puckette com o argumento de que o paradigma Max usa uma abordagem de software *design* na qual a aplicação possa ser totalmente personalizável. O objetivo é que o programa seja adaptável à realidade do usuário e não o contrário, como é o caso da maioria dos *softwares* comerciais. Segundo Puckette, essa abordagem vai de encontro ao que os vendedores de *softwares* comerciais geralmente procuram fazer: impor um contexto ao usuário além de formatos de arquivos proprietários, funcionalidades dependentes de um determinado sistema operacional ou *hardware*, e outras restrições.

Já a falta de eficiência dos SOFs no contexto de uso do VJ existe por que estes sistemas geralmente consideram que o fluxo de processamento será modificado pelo usuário principalmente no momento de preparação da apresentação e não durante. Durante a apresentação, espera-se que o usuário apenas controle sua aplicação através dos objetos de interface (botões, *sliders*) e não ligando, desligando, removendo e adicionando objetos ao fluxograma.

Porém, existem alguns SOFs como Visual Jockey e GePhex que, além de serem mais fáceis de usar por apresentarem objetos de um nível de abstração mais alto que os do paradigma Max, foram projetados justamente para possibilitar uma interação de tempo real mais eficiente. No Visual Jockey, por exemplo, existe uma funcionalidade que permite a substituição imediata de um objeto por uma árvore de outros objetos copiados de outra região do fluxograma em tempo de execução. Existem ainda outras propostas de interfaces interessantes para essa finalidade como a de Pelletier (PELLETIER, 2005), discutida na sessão 4.4, ou ainda outras mais radicais como FMOL e ReacTable (KALTENBRUNNER *et al.*, 2004).

Outros SOFs como Isadora e Salvation não são tão adequados para modificação eficiente de fluxograma em tempo de apresentação porém também são mais fáceis de usar que o Pd/GEM e Max/MSP/Jitter. O Isadora, como apresentado na sessão 4.2, foi criado para facilitar o trabalho de artistas da área de teatro e dança que queiram experimentar interação multimídia em tempo real sem precisar programar da maneira tradicional. Ainda assim, como escrito por Mark Coniglio no capítulo de introdução do manual do próprio programa, um dos maiores desafios para o usuário é

que ao iniciar o Isadora, aparece na tela apenas a página em branco (CONIGLIO, 2006).

Entre os SOAVEs também existe uma graduação de usabilidade que pode ser verificada pelos resultados da pesquisa com os VJs (Anexo A: **Questionário sobre softwares para VJs**). De acordo com os entrevistados, existe uma graduação de diminuição de facilidade de uso da esquerda para direita na seguinte seqüência de SOAVEs: ArKaos, Resolume e Modul8. Este resultado já era esperado, pois esta mesma ordem corresponde a uma diminuição de número de funcionalidades e elementos de interface gráfica, nitidamente perceptível. Além disso, algumas escolhas de metáforas de interface contribuem para uma diferença significativa na facilidade de compreensão de como operar o sistema. O ArKaos, por exemplo, apresenta um desenho esquemático do teclado semelhante a um teclado real de computador, e isso faz com que o usuário entenda imediatamente a associação das amostras de vídeo com as teclas.

5.3 Conclusões

Apresentadas as considerações sobre expressividade e usabilidade, observamos que, apesar de existir uma graduação crescente de expressividade entre dos SOAVEs mais simples aos SOFs mais complexos, existe também uma graduação decrescente de usabilidade. A Figura 5-1 representa um gráfico onde estão dispostos em um plano “usabilidade x expressividade”, os sistemas analisados.

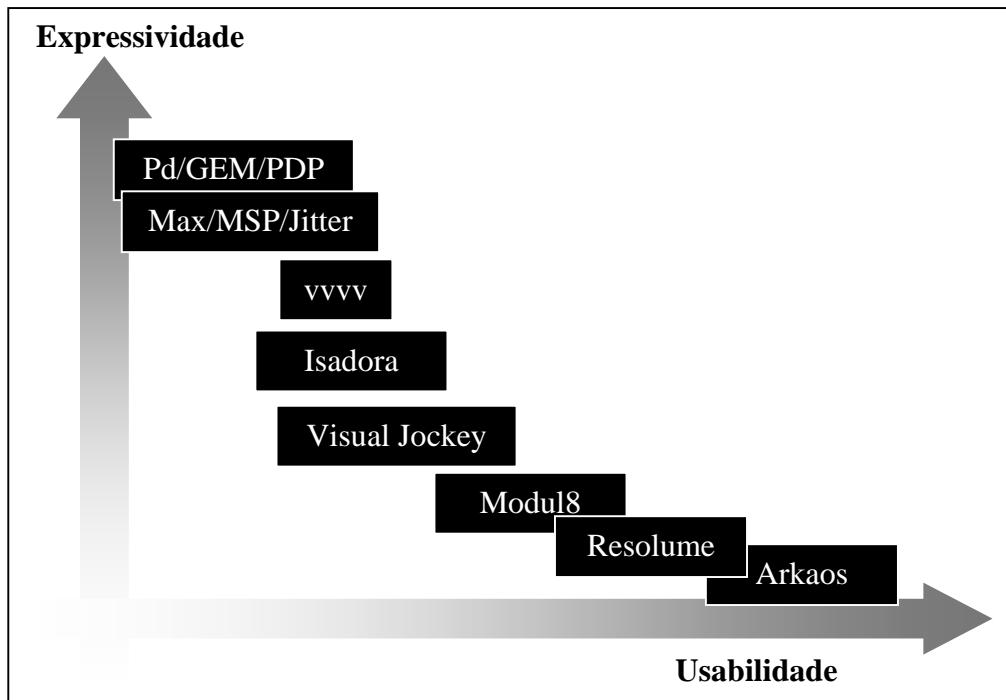


Figura 5-1 Comparação dos principais softwares de VJ em termos de usabilidade e expressividade.

Conseqüências deste problema de compromisso entre usabilidade e expressividade puderam ser observadas a partir de declarações de alguns VJs nas entrevistas. Uma delas é a utilização de SOAVEs em conjunto com SOFs como por exemplo: ArKaos + Quartz Composer ou Resolume + Max/MSP/Jitter. Por exemplo, o VJ Alexandre D'albergaria declarou utilizar o ArKaos para suas apresentações, porém para contornar os limites de funcionalidade do mesmo, utiliza arquivos do formato QuickTime manipulados em tempo real por fluxogramas construídos no QuartzComposer.

6. ViMus

6.1 Apresentação

Após o estudo dos sistemas interativos de tempo real para processamento audiovisual integrado, demos continuidade ao processo de desenvolvimento de um novo programa chamado ViMus (fusão das palavras “visual” e “música”). Os principais objetivos desta etapa da pesquisa foram: validar e aprimorar os conhecimentos adquiridos com o estudo dos sistemas existentes e buscar soluções de arquitetura e interface gráfica para a questão de compromisso entre usabilidade e expressividade discutida no capítulo anterior.

Neste capítulo apresentaremos este novo sistema através da exposição das tecnologias utilizadas para sua construção, da interface gráfica de caixa aberta, da arquitetura, de seu processo de desenvolvimento e de uma avaliação dos resultados obtidos.

6.2 Tecnologias de infra-estrutura

Entre as contribuições deste trabalho está o estudo de uma variedade de tecnologias associadas ao desenvolvimento de programas para processamento de áudio e de imagem em tempo real. Apresentaremos a seguir as tecnologias que utilizamos para desenho gráfico, sistema de janelas, captura de áudio e captura de vídeo.

Uma das primeiras e mais importantes decisões de projeto foi a escolha da *API* (*Application Program Interface*) gráfica a ser utilizada. Entre as opções mais adotadas para este tipo de aplicação estão OpenGL e Direct3D. OpenGL foi criada em 1992 pela Silicon Graphics, é independente de plataforma e sua especificação é mantida por um consórcio de diversas empresas que formam o *OpenGL Architecture Review Board* (OPENGL, 2007). A segunda foi criada em 1995 pela Microsoft e faz parte do DirectX (MICROSOFT, 2007a), um conjunto de *APIs* para criação de jogos e outras

aplicações multimídia de alto desempenho e funciona apenas para aplicativos que sejam executados no sistema operacional Windows.

Para a maioria dos domínios de aplicação de sistemas gráficos, OpenGL é considerada a *API* padrão. Entretanto, para a área específica de jogos ou outras aplicações de tempo real para entretenimento que exijam um alto desempenho, existe uma competição entre OpenGL e DirectX. Por exemplo, em nosso domínio de aplicação, o GEM para Pd e o Jitter para Max/MSP utilizam OpenGL como *API* gráfica, mas o vvvv utiliza Direct3D. Esta disputa entre as duas *APIs*, principalmente entre os desenvolvedores de jogos, rendeu longos debates em fóruns e listas de discussão desde a segunda metade da década de 90 (HSIEH, 1997) até os dias de hoje (WIKIPEDIA, 2007).

O ViMus foi pensado para ser uma aplicação de código aberto que tenha o maior alcance e acesso possível, incluindo usuários de Linux e outras plataformas. Partindo desse princípio a escolha de OpenGL seria óbvia devido a sua portabilidade. Entretanto, a grande popularização de Direct3D entre os desenvolvedores de jogos estabeleceu um senso comum de que programas escritos em Direct3D tem um maior desempenho que programas escritos em OpenGL para plataforma Windows (que até o momento ainda é a mais popular).

Isto geralmente acontece quando comparamos as duas *APIs* em computadores que não possuem placas de vídeo com aceleração de processamento gráfico. Porém, essa proposição deixa de ser verdade em computadores que possuem essa aceleração de *hardware*. Neste caso, o desempenho das *APIs* depende da implementação feita pelas empresas que desenvolvem as placas. Algumas placas apresentam melhor desempenho na execução de programas desenvolvidos em OpenGL e outras em Direct3D. Entretanto, neste caso, esta diferença de desempenho é mínima e dependente de muitos fatores que estão fora do controle do desenvolvedor da aplicação e por tanto, é pouco relevante para a escolha da *API* (ROY, 2002).

A opção por *OpenGL* contribuiu para a decisão de adotar o *GLUT* (*OpenGL Utility Toolkit*) (KILGARD, 1996) como sistema de janelas e de tratamento de eventos de teclado e *mouse*. A vantagem de utilizar *GLUT* é que o mesmo consiste em uma extensão de *OpenGL* com funcionalidades específicas para se trabalhar com esta *API*. Além disso, assim como *OpenGL*, *GLUT* também é independente de plataforma.

Entretanto, este sistema de janelas não é mais atualizado pelo autor e possui algumas limitações como a obrigatoriedade da aplicação ter que utilizar o laço infinito

iniciado pela função `glutMainLoop()`, que nunca retorna. Isto dificulta que a integração de *GLUT* em um programa que precisa ter o controle de seu próprio laço de eventos (como é obviamente o nosso caso). Existem algumas alternativas para esse problema como a correção de Rob Fletcher (FLETCHER, 2002).

Por razões históricas e culturais do ambiente acadêmico em que este projeto foi desenvolvido e antes disso, pela experiência do autor, a primeira versão do ViMus foi desenvolvida para o sistema operacional Windows. As *APIs* para captura de áudio e vídeo escolhidas para o ViMus são específicas para esta plataforma, já que ainda existiam poucas opções de *APIs* estáveis independentes de plataforma para estas finalidades. Para a captura de áudio foi utilizada a *Windows Media API* (MICROSOFT, 2007b) e para a captura de vídeo *DirectShow*, que assim como *Direct3D* é uma das *APIs* que faz parte do *DirectX*.

Esta sessão é complementada pelo Capítulo 3 da monografia relativa ao trabalho de graduação que deu origem a este projeto (JÁCOME, 2005). Neste texto estão registrados conhecimentos específicos do uso dessas tecnologias que podem ser úteis a novos pesquisadores da área.

6.3 Interface gráfica de caixa aberta

O poder de expressão do ViMus é oferecido ao usuário de forma gradativa e natural. Para isso a arquitetura e interface gráfica do usuário evidenciam dois conceitos já existentes nos SOFs: abstração de fluxogramas e flexibilidade de interface. Como apresentado na sessão 4.4, uma abstração de fluxograma consiste em um objeto especial que contém um fluxograma. Flexibilidade de interface consiste na possibilidade de composição e modificação de interfaces gráficas para os fluxogramas, através dos objetos de interface (*sliders*, botões, etc.).

Além disso, o ViMus também faz uso do conceito de *graph-on-parent*. Como explicado na sessão 4.4, *graph-on-parent* é uma opção que determina se objetos de interface de uma abstração de fluxograma podem ser visualizados dentro do objeto que representa esta abstração no fluxograma “pai”.

Uma das diferenças fundamentais entre a interface gráfica do ViMus e dos demais SOFs é a introdução de um novo conceito de interface gráfica ao qual demos o nome de “interface gráfica de caixa aberta” em referência à expressão “caixa preta” de

sentido oposto. A idéia fundamental da interface de caixa aberta é a representação dos componentes do *software* como caixas tridimensionais com alguns lados transparentes, de modo a permitir a visualização de seus componentes internos, ou seja, de seu fluxograma.

Assim, os fluxogramas são representados utilizando-se a metáfora de caixas tridimensionais abertas em vez de janelas como nos SOFs tradicionais. Uma caixa, assim como uma janela em Pd, contém um fluxograma cujos objetos podem ser outras caixas (abstrações de fluxogramas). Uma caixa possui apenas três lados opacos, os lados restantes são transparentes, permitindo a visualização de seu interior, ou seja, do fluxograma. O lado frontal da caixa é chamado “painele de controle” e exibe elementos de interface gráfica para controle do fluxograma incluindo o painel de controle de caixas “filhas”. O lado superior exibe as entradas e o lado inferior, as saídas.

As figuras a seguir apresentam telas capturadas do ViMus com um exemplo de fluxograma cujo painel de controle foi inspirado na interface gráfica do Resolume. A Figura 6-1 consiste em uma seqüência de imagens capturadas durante a rotação da caixa aberta para a transição do modo de visão do painel de controle para o modo de visão do fluxograma. A Figura 6-2 consiste em uma seqüência de imagens capturadas durante um “zoom in” da caixa principal para uma caixa filha.

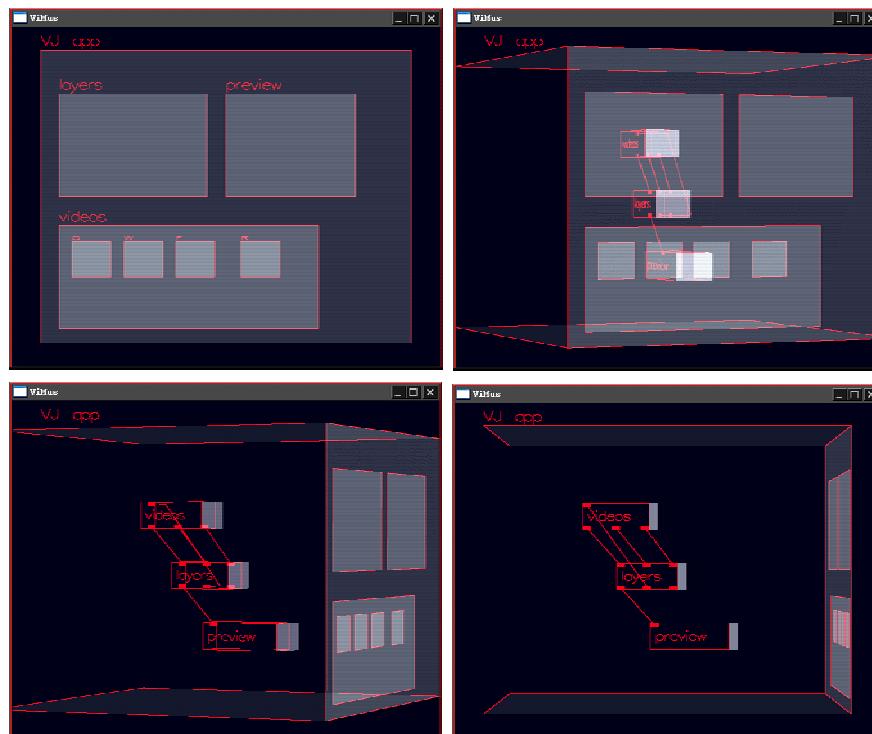


Figura 6-1 ViMus: exemplo de rotação da caixa aberta.

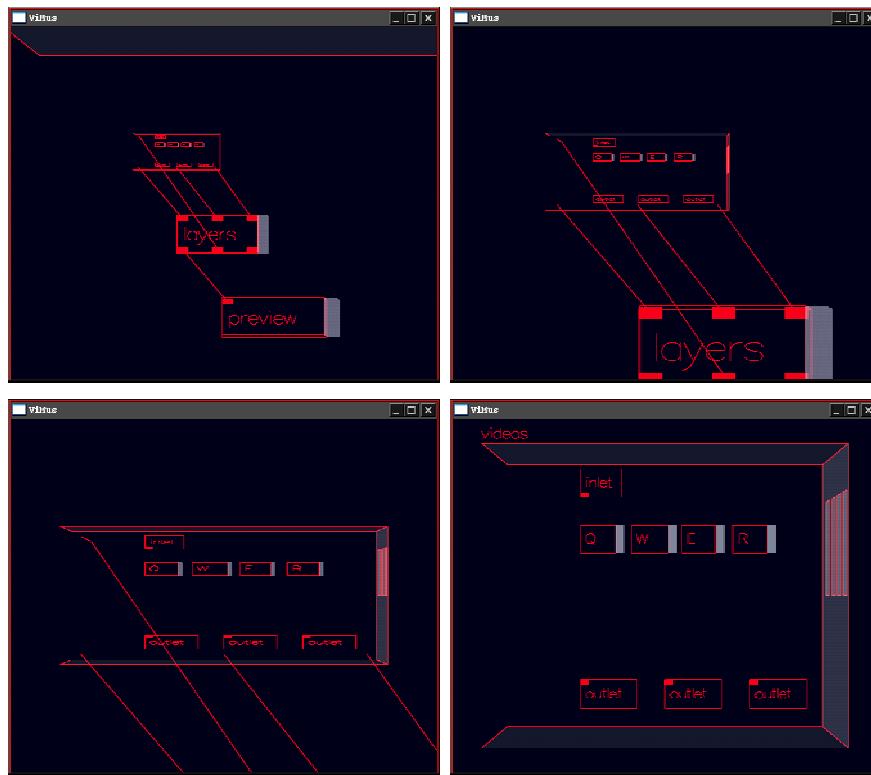


Figura 6-2 ViMus: exemplo de "*zoom in*" na caixa aberta.

O ViMus sempre apresenta pelo menos uma caixa contendo um fluxograma. O programa será distribuído com algumas caixas iniciais extremamente simples inspiradas em SOAVEs já consagrados como Resolume e novas caixas poderão ser criadas. O usuário escolhe a caixa que lhe parecer mais apropriada. Qualquer caixa de abstração de fluxograma pode ser aberta e girada o que permite a visualização de seu fluxograma. Ativando o modo de edição, as conexões podem ser modificadas, objetos removidos e uma paleta de opções de objetos é exibida para que o usuário possa adicioná-los, modificando o fluxograma, isto é, a aplicação, de acordo com suas necessidades.

6.4 Arquitetura

6.4.1 Arquitetura geral

Como vimos na sessão 4.3.2, uma decisão de projeto importante quando estamos desenvolvendo um SOF é o grau de acoplamento entre a *GUI* e a máquina de processamento. Ainda na sessão 4.3.2, observamos que enquanto o Pure Data apresenta um acoplamento alto entre as duas entidades, os desenvolvedores do Desire Data optaram por uma separação mais explícita.

Para o ViMus a independência entre a *GUI* e a máquina é essencial pois permite que o paradigma de caixa aberta possa ser facilmente utilizado como interface gráfica de outras máquinas além da máquina original do ViMus, como Pd ou vvvv. Assim como o Desire Data, a arquitetura geral do ViMus segue basicamente o conceito de cliente-servidor, onde a *GUI* é o cliente e a máquina o servidor e cada um possui sua própria representação do fluxograma. A *GUI* solicita operações de edição como adicionar, remover, conectar e desconectar objetos. A máquina recebe as solicitações da *GUI* e tenta executá-las retornando se a operação foi bem sucedida ou não.

Contudo, é também essencial para o ViMus que a comunicação entre a *GUI* e a máquina seja extremamente eficiente pois a *GUI* deve oferecer o retorno visual em tempo real do que está sendo gerado pela máquina. Por exemplo, um objeto de interface responsável por exibir a pré-visualização (*preview*) de um arquivo de vídeo precisa receber vários quadros por segundo da máquina de processamento. A interface de caixa aberta foi projetada para que o usuário tenha a opção de visualizar o painel de controle de cada caixa, desde que o mesmo esteja visível, independentemente se a caixa que contenha esse painel esteja dentro de outra caixa. Isso significa que precisamos de um canal de retorno muito eficiente e versátil para que a interface gráfica receba as mídias processadas pela máquina.

A solução que adotamos foi manter na *GUI* as referências dos endereços de memória onde os *buffers* de vídeo estão sendo preenchidos na máquina (Figura 6-3). Essa escolha aumenta a eficiência, mas em compensação, também aumenta a dependência entre os dois módulos. Isso dificulta que a máquina e a *GUI* estejam

localizados em computadores diferentes e ainda a substituição de uma máquina por outra.

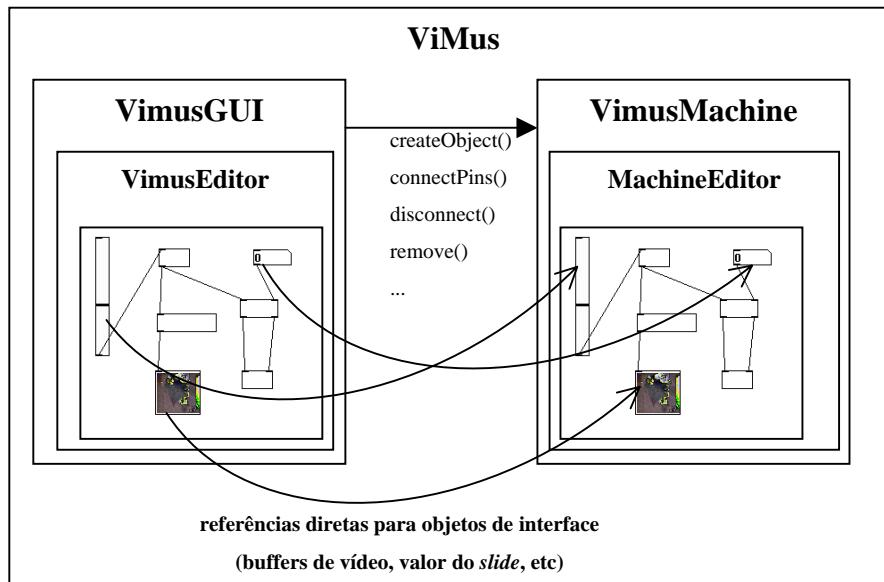


Figura 6-3 Esquema representando arquitetura geral do ViMus.

Para amenizar as dificuldades causadas pela dependência entre a *GUI* e a máquina, optamos por criar uma classe abstrata *Machine*. Esta classe abstrai para a *VimusGUI*, os detalhes de implementação da máquina de processamento. A Figura 6-4 apresenta exemplos de outras possíveis implementações de *Machine*.

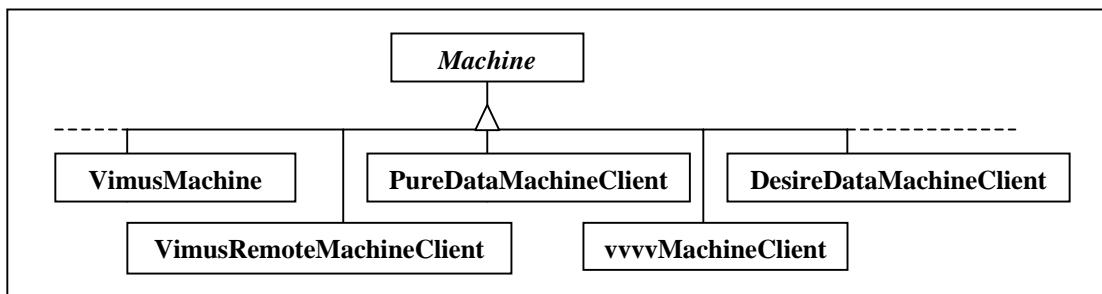


Figura 6-4 Classe abstrata *Machine* e possíveis implementações.

Para este trabalho, implementamos somente uma dessas várias possibilidades de máquinas de processamento: a *VimusMachine*, que foi construída a partir dos vários protótipos implementados em C++ e OpenGL. Nas próximas sessões,

apresentamos a arquitetura do módulo de interface gráfica do ViMus e, em seguida, a VimusMachine mais detalhadamente.

6.4.2 Arquitetura do módulo de interface de caixa aberta

Assim como a interface gráfica de qualquer SOF, a *GUI* do ViMus tem o objetivo principal de permitir a visualização e edição do fluxograma que está sendo executado. Por tanto, podemos identificar facilmente algumas entidades essenciais na arquitetura desse módulo: um editor (VimusEditor) que encapsula os comandos de criação, conexão e remoção de objetos do fluxograma; e o próprio fluxograma (VimusPatch) que contém os vários objetos (VimusObject).

Os objetos do fluxograma podem ser de dois tipos e são representados por duas classes que implementam VimusObject: objeto normal (VimusNormalObject) e objeto de interface (VimusInterfaceObject). O primeiro representa aqueles objetos cujo desenho consiste em uma caixa, com os pinos e um texto dentro. O segundo representa os objetos de desenho variado como *sliders*, botões, etc.

Como exposto na sessão 6.3, a interface do ViMus substitui a metáfora de janelas por uma metáfora de caixas tridimensionais para a representação da figura que contém o desenho do fluxograma. Assim, o objeto que representa a abstração de fluxograma é um caso especial de um objeto normal, que quando expandido é exibido como uma figura tridimensional correspondente a um paralelepípedo com apenas três lados visíveis. Em referência ao seu aspecto visual esta entidade é representada pela classe de nome VimusCubeObject.

Com essa mudança de paradigma visual, passamos a pensar em um fluxograma agora não mais como um desenho bidimensional dentro de uma janela e sim como a estrutura interna de um VimusCubeObject. Por tanto, todos os fluxogramas, inclusive o fluxograma de maior nível de abstração estão contidos em um VimusCubeObject.

Finalmente, para tornar o código independente do sistema de janela, encapsulamos as funções do GLUT em VimusGUI. Assim, podemos resumir a estrutura de classes da *GUI* do Vimus com o diagrama da Figura 6-5.

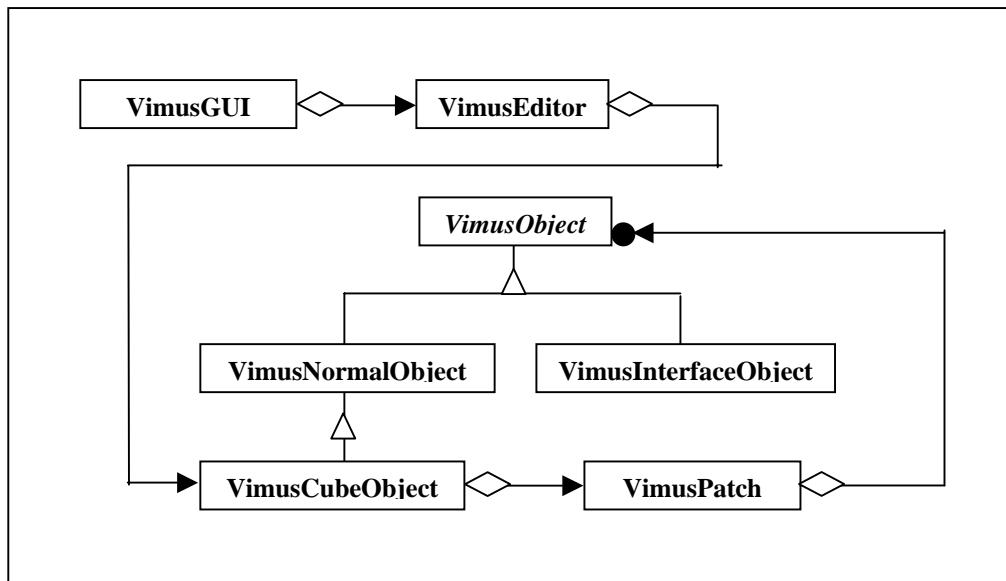


Figura 6-5 Arquitetura do módulo de interface de caixa aberta do ViMus.

A Figura 6-6 apresenta um exemplo de configuração dos objetos em tempo de execução correspondente ao mesmo fluxograma da Figura 6-1.

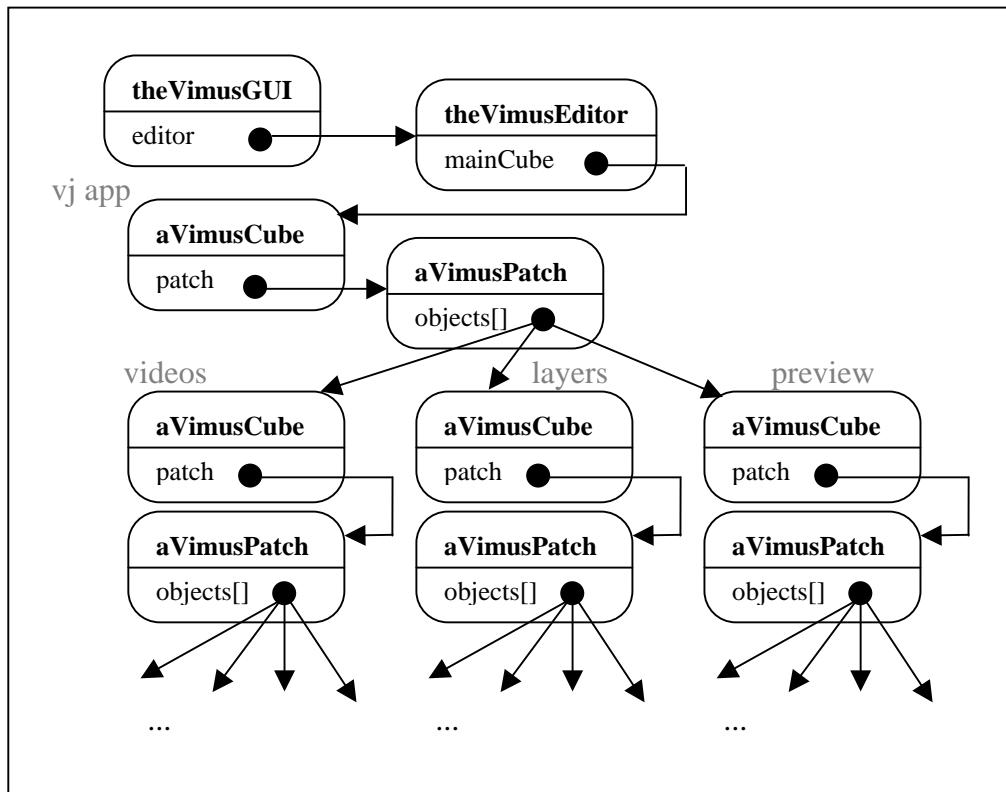


Figura 6-6 Estrutura dinâmica dos objetos em tempo de execução.

6.4.3 Arquitetura da máquina de processamento

A máquina de processamento implementada para o ViMus pode ter sua arquitetura compreendida a partir do diagrama de classes simplificado na Figura 6-7. As principais entidades estão representadas pelas classes VimusMachine (implementação de Machine), MachineEditor, MachineExecutor e as classes que representam o fluxograma (MachinePatch e MachineObject e suas implementações).

Machine é uma fachada (*Facade*), isto é, ela provê uma interface unificada para um conjunto de interfaces (MachineEditor e MachineExecutor) de um subsistema (máquina de processamento), como descrito no padrão de projeto *Facade* em (GAMMA *et al.*, 1995).

MachineEditor é responsável pelas funções de edição do fluxograma enquanto MachineExecutor é responsável pelas funções de execução. No modo de edição, os cliques do *mouse* e pressionamento de teclas são processados como comandos de edição e enviados para o MachineEditor. Já no modo de execução esses eventos são enviados diretamente para o MachineExecutor que, por sua vez, os envia para cada objeto de interface (*sliders*, botões, etc).

Assim como o Pd, optamos por um modelo de sistema de tempo real síncrono utilizando o conceito de “Executor Cíclico” – “*Cyclic Executive*” (DOUGLASS, 2000). Entretanto, para facilitar ainda mais a implementação, a GUI e a máquina do ViMus estão na mesma *thread*, diferentemente do Pd que possui uma *thread* para a GUI e outra para a máquina (PUCKETTE, 1996). Assim podemos utilizar o próprio laço da interface (laço principal do GLUT) para chamar as funções de processamento da máquina através do método update() da classe Machine (Figura 6-7).

O laço principal de funcionamento do ViMus se dá da seguinte forma. Enquanto nenhum evento é disparado (clique de mouse, tecla pressionada, etc) a VimusGUI chama o método update() da máquina de processamento e depois desenha a interface. O método update(), por sua vez chama os métodos de processamento de vídeo e áudio. O processamento de vídeo e de áudio consiste na execução ordenada de cada objeto do grafo de objetos de vídeo e grafo de objetos de áudio.

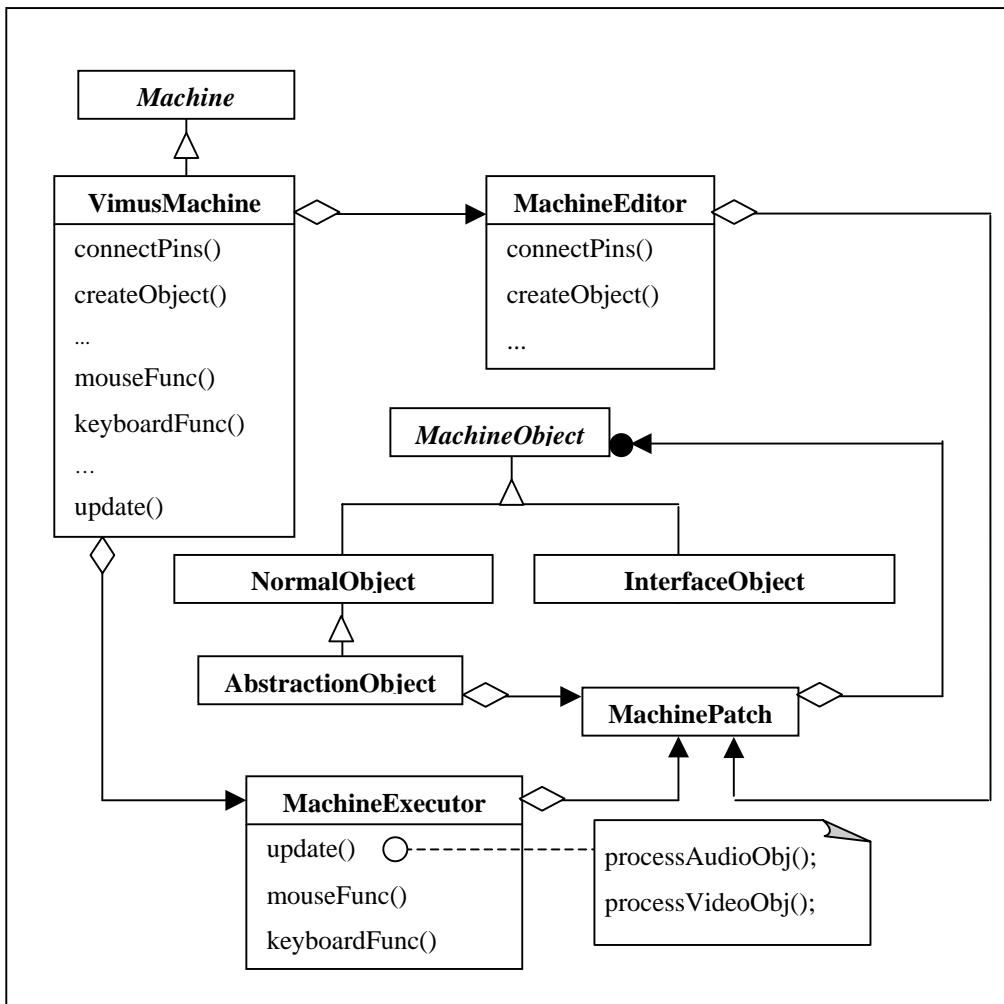


Figura 6-7 Diagrama de classes da máquina de processamento.

6.5 Processo de desenvolvimento

O ViMus foi desenvolvido em basicamente três grandes etapas: uma primeira para a máquina de processamento, uma segunda para a interface gráfica do usuário e uma terceira para a integração das duas partes.

O desenvolvimento da máquina de processamento se deu através das seguintes sub-etapas: definição de requisitos, estudo e escolha das tecnologias de infra-estrutura, implementação de protótipos funcionais e independentes (ex.: exibição de vídeo capturado utilizando OpenGL), testes dos protótipos, modelagem, implementação da máquina integrando os módulos (antes separados e agora reescritos

utilizando orientação a objetos de acordo com a arquitetura definida na modelagem) e testes.

O desenvolvimento da interface gráfica se deu através das seguintes sub-etapas: análise de requisitos, estudo de usabilidade e expressividade dos sistemas existentes (discutido ao longo dos capítulos 3, 4 e 5), concepção da interface de caixa aberta, modelagem, implementação e testes.

A integração da interface gráfica com a máquina de processamento se deu através das seguintes sub-etapas: remodelagem da máquina, implementação e testes. A remodelagem da máquina de processamento foi necessária para possibilitar a construção de fluxogramas para possibilitar o funcionamento da interface de caixa aberta.

6.6 Resultados

Nesta sessão apresentaremos alguns dos resultados obtidos com o ViMus. Nossos principais objetivos ao desenvolvermos o ViMus para esta pesquisa foram: a sedimentação e aplicação do conhecimento adquirido com o estudo dos sistemas já existentes através da criação de uma nova ferramenta; e oferecer uma proposta viável de solução para o problema “expressividade x usabilidade”. Aqui discutiremos o quanto perto chegamos destes objetivos.

6.6.1 Eficácia e eficiência da máquina de processamento

Pudemos verificar se o conhecimento adquirido com o estudo das tecnologias existentes foi bem aplicado na prática através da avaliação da eficácia e eficiência da máquina de processamento implementada.

Tivemos a oportunidade de avaliar a eficácia da máquina com a demonstração do *software* para diversos VJs, artistas performáticos e grande público em geral, sempre com um retorno de opiniões positivas e estimulantes. Além de centenas de demonstrações individuais ou para pequenos grupos, o sistema despertou interesse de artistas que chegaram a utilizar em algumas de suas apresentações em eventos abertos ao público.

Como exemplos desses eventos, podemos citar: uma das apresentações de visual-jóquei no Porto Musical (PORTOMUSICAL, 2007) do ano de 2006, pelo grupo Re:combo (RE:COMBO, 2007); e exposição para experimentação livre do *software* no evento Futuríveis, como parte da programação do Festival de Inverno de Garanhuns, em 2007.

Já a eficiência da máquina de processamento pôde ser avaliada por medidas de seu desempenho. Como nosso foco inicial foi a atividade do visual-jóquei e, por tanto, o processamento de vídeo em tempo real, adotamos a métrica de desempenho como sendo a taxa do número de quadros por segundo ou *fps* (*frames-per-second*). Em um computador com processador de 2.0 GHz de *clock* e uma placa de vídeo com aceleração, obtemos, geralmente, taxas de 20 a 50 *fps*, para executar efeitos que incluem processamento de áudio (como *FFT*) capturado em tempo real, processamento de imagem (como filtros diversos), utilização de vídeo (capturado em tempo real) como textura em figuras geométricas tridimensionais, como cubos e superfícies *NURBs* (ROGERS & ADAMS, 1990).

As figuras a seguir mostram alguns exemplos de imagens geradas a partir da máquina de processamento do ViMus, através de análise de áudio em tempo real.

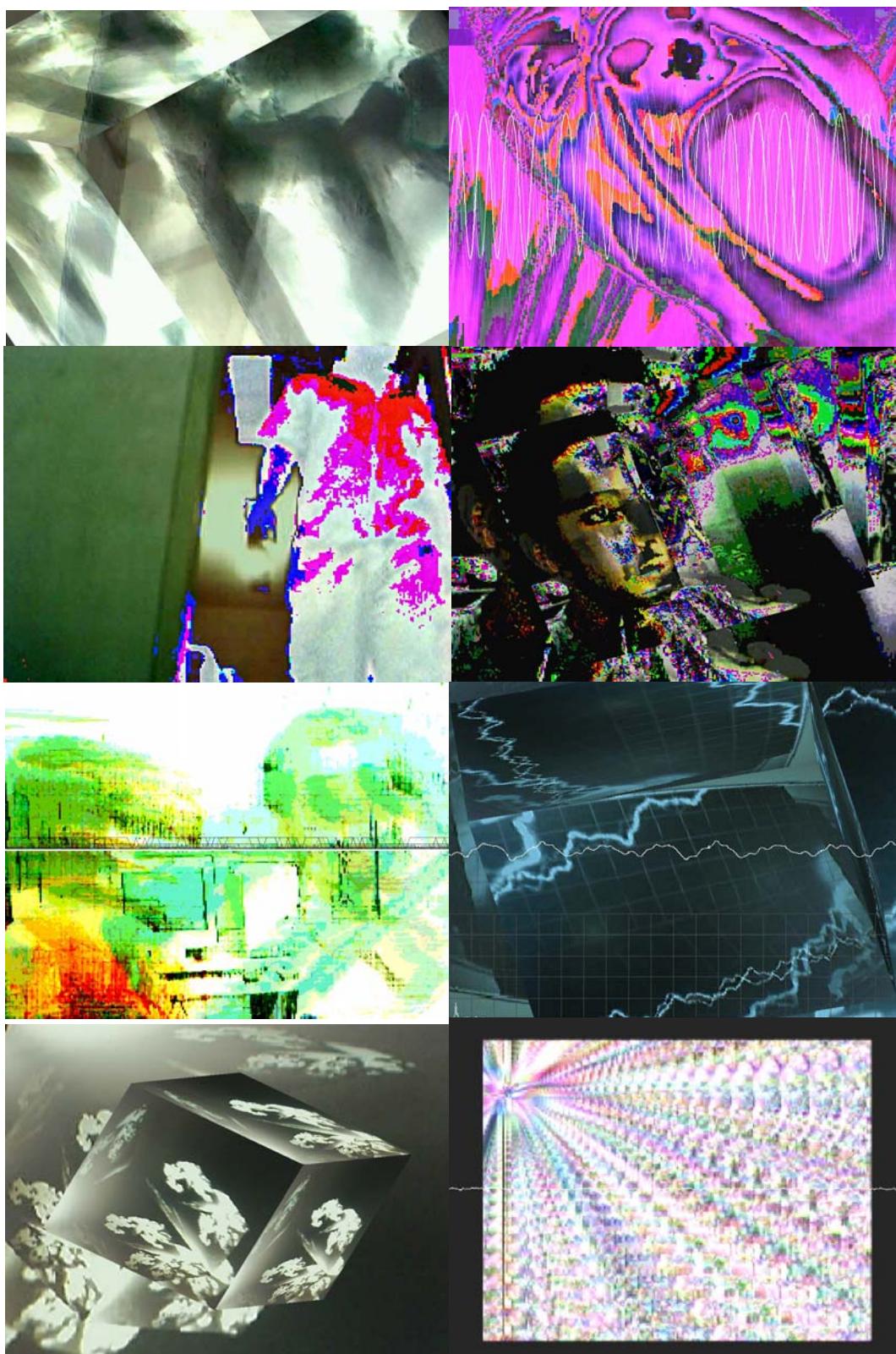


Figura 6-8 Exemplos de imagens geradas pela máquina do ViMus.

6.6.2 Usabilidade e expressividade da interface de caixa aberta

A segunda fase de desenvolvimento do ViMus foi orientada pela busca de uma solução para o problema de compromisso entre usabilidade e expressividade entre os vários sistemas de tempo real apresentados. Como exposto na sessão 6.3, chegamos ao conceito da “interface de caixa aberta” associado à idéia de abstrações de fluxograma e “*graph-on-parent*” dos SOFs como Pd e Max/MSP.

A interface gráfica do usuário desenvolvida para o ViMus permite os mesmos tipos de manipulação do fluxograma que as interfaces do Pd e Max/MSP e por tanto, potencialmente, o ViMus é tão expressivo quanto estes outros SOFs. Na prática, isso ainda não é possível simplesmente porque a máquina atual do ViMus implementa apenas uma pequena fração de operações sobre áudio e vídeo. Entretanto, substituindo a máquina do ViMus pela máquina do Pd, teríamos o mesmo poder de expressão que o Pd.

Se nós podemos fazer os mesmos tipos de operações no fluxograma, porque então o ViMus seria menos difícil de se aprender a usar que o Pd? Isto é, partindo da visão de que o ViMus também é um SOF e que, como qualquer outro SOF, para utilizar todo o seu potencial o usuário precisa ser um programador, o ViMus deveria ter o mesmo grau de dificuldade. Porém, uma primeira diferença do ViMus para os outros SOFs é que a visão inicial que o usuário tem do sistema é a do painel de controle da caixa principal, e não de seu fluxograma. Por exemplo, considerando a atividade de visual-jóquei, a caixa principal pode ter seu painel de controle inspirado em SOAVEs como Resolume e, neste caso, esta caixa apresenta uma usabilidade tão boa quanto um SOAVE (veja a Figura 6-9).

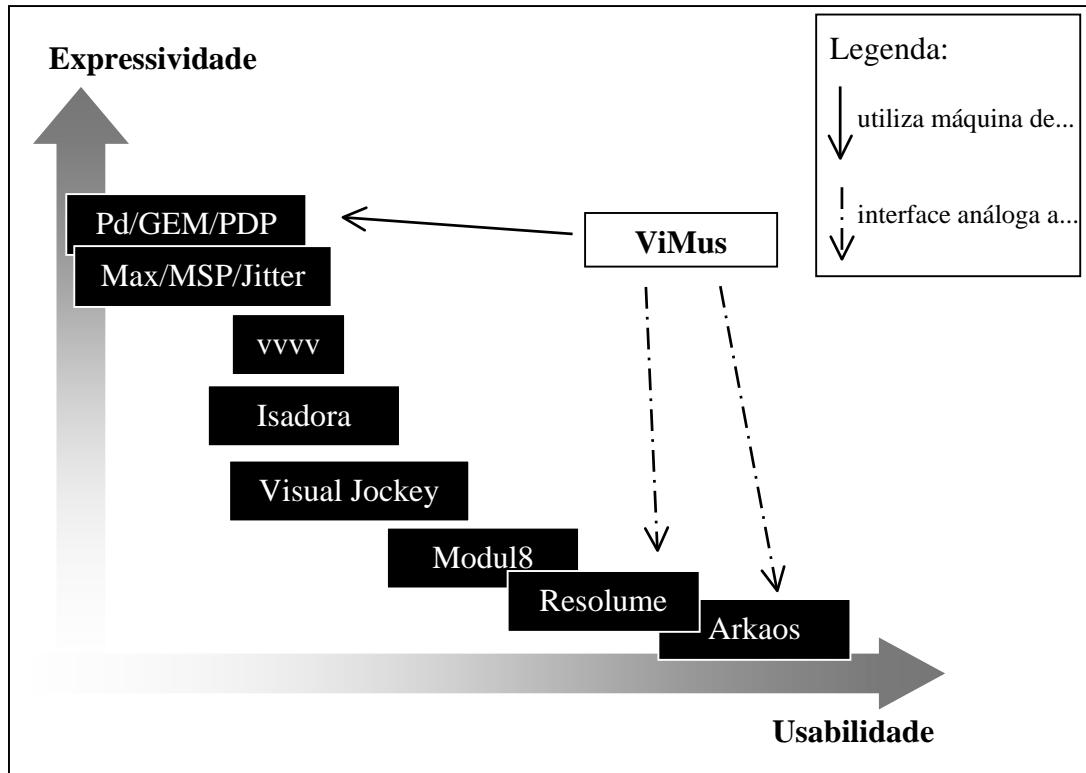


Figura 6-9 Posicionamento do ViMus com um cubo de painel de controle inspirado na interface de alguns SOAVEs.

Ainda assim, a maioria dos SOFs também permitem a criação de interfaces gráficas semelhantes a dos SOAVEs mais simples, através da funcionalidade “*graph-on-parent*” como Pd e Max/MSP, ou através do modo de painel de controle, como Isadora. Um exemplo disso é a existência de *softwares* comerciais como o AVmixer da Neuromixer, que é basicamente um fluxograma desenvolvido no Max/MSP/Jitter. O AVmixer não permite a visualização e edição de seu fluxograma simplesmente porque neste caso o fluxograma é o próprio código do *software* (que no caso do Neuromixer não é aberto, por ser um *software* proprietário).

Apesar de não ser o caso do AVmixer, é obviamente possível se desenvolver um SOAVE usando o Max/MSP/Jitter, Pd/GEM/PDP, Isadora e outros SOFs, e ainda assim permitir que o usuário navegue pelo fluxograma e possa editá-lo. Por exemplo, como vimos na sessão 4.4, a Figura 4-18 apresenta o painel de um fluxograma construído no Isadora que é um SOAVE simples, com alguns efeitos que utilizam a metáfora do *mixer A/B*. Já a Figura 4-22 apresenta uma tela do fluxograma Dervish (GOLDBERG, 2005), um SOAVE que, assim como o AVmixer, foi desenvolvido no

Max/MSP/Jitter, porém seu código (fluxograma) é aberto e o, por tanto, o usuário pode visualizá-lo e alterá-lo.

Entretanto, a principal diferença do ViMus para os SOFs existentes é a forma como é representado visualmente o fluxograma e a relação do mesmo com sua interface gráfica do usuário, isto é, com o seu “painel de controle”. No Pd e Max/MSP o fluxograma é representado como um desenho bidimensional em uma janela. E a relação entre o fluxograma em si e sua abstração de fluxograma correspondente é percebida com a noção de hierarquia de janelas, de modo que uma janela “pai” exibe a abstração de fluxograma com o painel de controle dentro; e a janela “filho” exibe o fluxograma correspondente àquela abstração.

Já no ViMus, a metáfora de janelas foi substituída pela metáfora das caixas tridimensionais com três lados transparentes. Dessa vez o fluxograma é visto pelo usuário como um elemento que está realmente dentro da caixa que o representa. Essa simples mudança de analogia torna a representação mental do objeto (fluxograma) mais parecida com a visão que temos dos objetos computacionais que possuem representação no mundo físico do nosso dia-a-dia como celulares (aparelhos telefônicos móveis), computadores, etc. As máquinas no mundo físico são como caixas dentro das quais está contida a maquinaria que a faz funcionar, isto é, que recebe as entradas do usuário ou de outro sistema e retorna a saída para o usuário ou para outro sistema. Da mesma forma que ocorre com as caixas do ViMus, no mundo físico as entradas e saídas para o usuário ocorrem através de um “painel de controle” como o visor e o teclado de um celular.

Nossa hipótese é que esse novo tipo de interface gráfica facilita a compreensão do funcionamento e modo de operação do SOF. Entretanto, essa hipótese envolve aspectos subjetivos que necessitaram sua confirmação através de uma pesquisa com um questionário (Anexo B: **Questionário de Avaliação da Interface de Caixa Aberta**) respondido por VJs que já tiveram a oportunidade de experimentar alguns SOFs ou pelo menos sabem como eles funcionam. O questionário foi desenvolvido com base nas definições de usabilidade de Jacob Nielsen, e pede para que os usuários comparem a interface de caixa aberta do ViMus com a interface tradicional dos SOFs. As respostas eram dadas após uma experimentação da interface e os resultados foram satisfatórios, como pode ser observado na Tabela 3, a seguir.

Tabela 3 Respostas ao questionário sobre a usabilidade da interface de caixa aberta do ViMus em comparação com a interface dos SOFs (baseada em janelas).

	Questão 1	Questão 2	Questão 3	Questão 4
Usuário 1	c) Já experimentei	f) Mais fácil no ViMus	f) Mais eficiente no ViMus	g) Muito mais prazeroso no ViMus
Usuário 2	c) Já experimentei	f) Mais fácil no ViMus	f) Mais eficiente no ViMus	g) Muito mais prazeroso no ViMus
Usuário 3	e) Usuário intermediário	e) Um pouco mais fácil no ViMus	e) Um pouco mais eficiente no ViMus	g) Muito mais prazeroso no ViMus
Usuário 4	d) Usuário iniciante	e) Um pouco mais eficiente no ViMus	d) Mesma coisa	f) Mais prazeroso no ViMus
Usuário 5	d) Usuário iniciante	f) Mais fácil no ViMus	e) Um pouco mais eficiente no ViMus	f) Mais prazeroso no ViMus

7. Conclusões

7.1 Considerações finais

Esta dissertação apresentou uma pesquisa em sistemas interativos de tempo real para processamento audiovisual integrado. Apesar da existência de vários trabalhos acadêmicos além das iniciativas estritamente comerciais e das comunidades de *software* livre, esta área de estudo ainda é incipiente e possui muitas questões abertas. Isto pôde ser constatado desde o início da pesquisa, quando logo se observou uma grande quantidade de iniciativas independentes com pouca interação ou acúmulo de experiências de um trabalho para o outro. Isto é, alguns trabalhos com objetivos semelhantes ou que pelo menos abordam o mesmo tema, apresentam pouca inter-referência. Este fato é uma das razões para uma grande dificuldade a ser enfrentada por alguém que queira iniciar uma pesquisa nesta área, pois as informações existem, mas estão dispersas.

Nossa pesquisa contribui para uma amenização deste problema com a tentativa de formalização e agregação de conhecimento na área. Os três primeiros capítulos abarcam uma grande parte dos trabalhos acadêmicos e não-acadêmicos já desenvolvidos. Além disso, apresentamos uma visão original da história e estado da arte da área, relacionando as diferentes iniciativas de diferentes épocas em um mesmo contexto analítico orientado basicamente pela questão de suas possibilidades e limitações técnicas e pelo equilíbrio entre usabilidade e expressividade dos sistemas atuais (com foco nas tecnologias de *software* e na atividade do visual-jóquei).

Este contexto analítico é o que dá unidade à dissertação e foi derivado da constatação de um segundo importante problema nesta área: observamos uma grande distância entre as tecnologias que já existem e a população de artistas que poderia utilizá-las, assim como o grande público em geral. Isto ficou claro desde as demonstrações dos primeiros protótipos para amigos e pessoas mais próximas. Este protótipo do ViMus era capaz de distorcer as cores da imagem capturada em tempo real de uma *webcam* de acordo com a intensidade do áudio capturado por um microfone. O desconhecimento geral das tecnologias de sistemas interativos de tempo real faz com que as pessoas pensem que este tipo de efeito é algo inovador e inédito,

no entanto é um tipo de funcionalidade possível de ser obtida em *softwares* como Pd (de distribuição gratuita) desde os anos 90, para citar um exemplo.

Nossa pesquisa foi conduzida através de um estudo aprofundado de *softwares* como Pd/GEM, assim como pela busca de possíveis razões técnicas para a distância e desinformação sobre este tipo de tecnologia. Descobrimos que alguns *softwares* específicos para a atividade de visual-jóquei de disparar vídeos e efeitos e, por isso, classificados aqui como SOAVEs (Sistemas Orientados a Amostras de Vídeo e Efeitos), são capazes de realizar este tipo de efeito do protótipo do ViMus, ou seja, efeitos em vídeo de tempo real que integram o processamento de diferentes mídias, mapeando resultados de análises de uma mídia em entradas de parâmetros de outra. No entanto, estas funcionalidades são limitadas nestes *softwares*, devido à falta de flexibilidade, isto é, a falta de possibilidades de novos tipos de associações entre as funções existentes para a construção de novas funções.

Diferentemente dos SOAVEs, os programas como Pd, classificados aqui como SOFs (Sistemas Orientados a Fluxograma), são pensados para que o próprio usuário construa a sua ferramenta. Por isso são mais expressivos que os SOAVEs, porém mais difíceis de se aprender a usar. Este problema de compromisso entre usabilidade e expressividade foi discutido em profundidade no capítulo 5.

Finalmente, no capítulo 6, apresentamos o ViMus, resultado de alguns anos de experimentos em processamento audiovisual integrado em tempo real utilizando a linguagem C++ e a API gráfica OpenGL. Após o desenvolvimento do primeiro protótipo, a máquina foi remodelada e reimplementada para possibilitar o seu controle através da manipulação de fluxogramas. Além disso, implementamos e experimentamos o novo conceito de interface de caixa aberta, que aproveita os conceitos de desenho da interface gráfica do Pd substituindo a metáfora de janelas, pela metáfora de caixas tridimensionais com três lados transparentes. Os resultados, tanto da máquina de processamento como do novo conceito de interface de caixa aberta, foram considerados satisfatórios por usuários consultados através de um questionário preenchido após experimentação do *software*.

Além das contribuições já citadas como a formalização e agregação de conhecimento, podemos identificar outras contribuições igualmente importantes como o início do desenvolvimento de uma nova ferramenta de código aberto que poderá ser (em alguns casos já está sendo) utilizada para experimentos em diversas subáreas da ciência da computação: computação gráfica, computação musical, arquitetura de

software, reuso de *software*, sistemas de tempo real, inteligência artificial (durante o desenvolvimento do ViMus foram feitos experimentos de análise de aspectos musicais do áudio de entrada para modificação do vídeo), interface humano-máquina (engenharia de usabilidade para a criação de painéis de controle e objetos de interface mais intuitivos), sistemas distribuídos, otimização de código, entre outras.

7.2 Trabalhos futuros

Apesar de esta pesquisa ter atingido seus objetivos iniciais, podemos identificar a necessidade de um período maior de avaliação do ViMus para que a ferramenta possa ter uma primeira versão distribuída. Uma das possibilidades de melhoramento do ViMus é torná-lo compatível com outras máquinas de processamento de licença aberta de uso como Pd/GEM/GridFlow/PDP. Estes outros sistemas já estão em desenvolvimento há alguns anos e possuem uma comunidade grande e ativa de usuários e desenvolvedores.

Graças à arquitetura modular que desenvolvemos para o ViMus, a substituição, ou até a soma de uma nova máquina de processamento ao sistema não exigirá muito retrabalho. Será necessário apenas o desenvolvimento de uma nova implementação da interface *Machine*, como por exemplo, PdMachineClient (Figura 7-1) que transformaria o ViMus em um cliente (*GUI*) para o Pd. Isto é possível utilizando a opção de execução do Pd em que sua interface gráfica é ocultada (“pd -nogui”). A comunicação entre o ViMus (módulo PdMachineClient) e o Pd, pode ser feita pelo envio de mensagens do Pd por TCP/IP (exatamente como atualmente é feito no próprio Pd para a comunicação entre sua *GUI* e sua máquina).

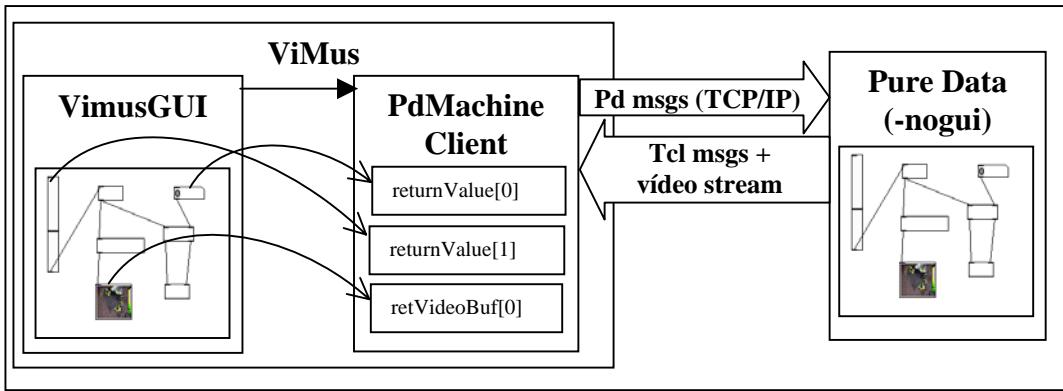


Figura 7-1 Arquitetura do ViMus reutilizando máquina do Pd.

Além disso, apesar do conceito de interface de caixa aberta nos parecer promissor, ainda não é suficiente para que na prática o ViMus possa ser realmente uma ferramenta que diminua a distância entre os artistas não-especialistas e as tecnologias interativas de processamento audiovisual de tempo real. Um próximo passo nessa direção, poderia ser um processo de engenharia de usabilidade baseada em cenários (ROSSON & CARROLL, 2002) para o desenvolvimento de novas aplicações usando o ViMus. Isto é, criamos uma base para a construção de aplicações mais fáceis usando a complexidade dos SOFs, agora devemos criar e estimular a criação dessas aplicações.

Os SOFs em geral são extremamente extensíveis, pois possuem mecanismos que facilitam o processo de criação de novos módulos, como por exemplo, os objetos do tipo *external* do Pd e Max. Este tipo de mecanismo pode ser facilmente implementado no ViMus, graças a sua arquitetura modular. Dessa forma, este sistema poderá ser continuado por programadores em geral e no ambiente acadêmico por estudantes que poderão desenvolver novos módulos para o ViMus como trabalhos para disciplinas de computação gráfica, computação musical, inteligência artificial, sistemas distribuídos e outras.

8. Referências

ALVES, B. Digital Harmony of Sound and Light. **Computer Music Journal**, v.29, n.4, p.45-54. 2005.

APPLE COMPUTER, I. **QuickTime File Format**. Disponível em: <<http://developer.apple.com/documentation/QuickTime/QTFF/index.html>>. Acesso em: 22 abr 2007.

_____. **Quartz Composer Programming Guide**. Disponível em: <<http://developer.apple.com/documentation/GraphicsImaging/Conceptual/QuartzComposer/>>. Acesso em: 8 abr 2007.

ARS ELECTRONICA LINZ, G. **Ars Electronica Katalogartikel**. Linz, Áustria. Disponível em: <http://www.aec.at/en/archives/festival_archive/festival_catalogs/festival_artikel.asp?iProjectID=9078#>. Acesso em: 25 abr 2007.

AUDIOVISUALIZERS. **Rutt-Etra VSynth**. Disponível em: <<http://www.audiovisualizers.com/toolshak/vidsynth/ruttetra/ruttetra.htm>>. Acesso em: 4 abr 2007.

BASBAUM, S. R. **Sinestesia, arte e tecnologia: fundamentos da cromossomia**. São Paulo: Annablume. 2002

BASS, L., et al. **Software Architecture in Practice**. Pittsburgh: Addison-Wesley Professional. 2003. 560 p.

BAYER, M.; SEIDEL, G. **The GePhex Book**. Disponível em: <<http://www.gephex.org/docu/documentation.html>>. Acesso em: 8 abr 2007.

BEAULIEU, S., et al. Plogue Bidule, versão 0.94. Montreal, Canadá: Plogue Art et Techonologie, Inc., 2007. Arquivo de programa baixado em <http://www.plogue.com>

BENCINA, R. AudioMulch, versão 1.0. Melbourne, Australia: AudioMulch, 2007. Arquivo de programa baixado em <http://www.audiomulch.com>

BERNSTEIN, J. **A Discussion Of NATO.0+55+3d Modular**. Disponível em: <http://www.bootsquad.com/old_site/nato/>. Acesso em: 8 jan 2007.

BOMFIM, G. A. **Metodologia para Desenvolvimento de Projetos**. João Pessoa - PB: UFPB/EDITORA UNIVERSITÁRIA. 1995. 64 p.

BOUCHARD, M. **Gridflow**. Disponível em: <<http://gridflow.ca>>. Acesso em: 7 abr 2007.

BOUCHARD, M.; LEE, C. **DesireData**. Disponível em:
 <http://artengine.ca/matju/piksel/DesireData_Piksel05_final.pdf>. Acesso em: 10 abr. 2007.

BURNETT, M. M. **Visual Programming**. Encyclopedia of Electrical and Electronics Engineering. New York, EUA: John Wiley & Sons Inc., 1999.

CAMURRI, A., *et al.* **Toward real-time multimodal processing: EyesWeb 4.0**. AISB 2004 Convention: Motion, Emotion and Cognition. Leeds, UK, 2004.

CHADABE, J. **The Electronic Century Part III: Computers and Analog Synthesizers**. Disponível em:
 <http://emusician.com/mag/emusic_electronic_century_part_2/>. Acesso em: 5 jan 2007.

CHOMSKY, N. Three models for the description of language. **IRE Transactions on Information Theory**, v.2, p.113-124. 1956.

CLAYTON, J. K. Event Priority in Max (Scheduler vs. Queue). Cycling'74. 2004

COLLOPY, F. **Visual Music in a Visual Programming Language**. IEEE Symposium on Visual Music. Tokyo, Japan: IEEE, 1999. 111-118 p.

CONIGLIO, M. Isadora, versão 0.9b38. Nova Iorque, EUA: TroikaTronix, 2006. Arquivo de programa baixado em <http://www.troikatronix.com/>

CUI, J. **Neuromixer**. Los Angeles, EUA. Disponível em:
 <<http://www.neuromixer.com>>. Acesso em: 10 mai. 2007.

DANKS, M. **The Graphics Environment for Max**. International Computer Music Conference. Hong Kong, 1996. 67-70 p.

_____. **Real-time image and video processing in GEM**. International Computer Music Conference. Tessaloniki, Grécia, 1997. 220-223 p.

DANNENBERG, R. B. **Aura II: Making Real-Time Systems Safe for Music**. New Interfaces for Musical Expression. Hamamatsu, Japan, 2004. 132-137 p.

_____. Interactive Visual Music: A Personal Perspective. **Computer Music Journal**, v.29, n.4, p.25-35. 2005.

DANNENBERG, R. B.; BENCINA, R. **Design Patterns for Real-Time Computer Music Systems**. International Computer Music Conference, 2005.

DÉCHELLE, F. **A brief history of MAX**. Paris. Disponível em:
 <http://freesoftware.ircam.fr/article.php3?id_article=5>. Acesso em: 21 abr 2007.

DEGOYON, Y. **PiDiP Is Definitely In Pieces**. Disponível em:
 <<http://ydegoyon.free.fr/pidip.html>>. Acesso em: 7 abr 2007.

DELAHUNTA, S. **Software for Dancers: Isadora Article/ Part I.** Disponível em: <<http://www.s dela.dds.nl/sfd/isadora.html>>. Acesso em: 19 set. 2006.

_____. **Software for Dancers: Isadora Article/ Part II.** Disponível em: <<http://www.s dela.dds.nl/sfd/isadora2.html>>. Acesso em: 26 abr 2007.

DEMEYER, T. **Image/ine for OsX.** Amsterdam, Holanda. Disponível em: <<http://www.image-ine.org>>. Acesso em: 22 abr 2007.

DOUGLASS, B. P. Real-Time Design Patterns. I-Logix. 2000

EAGAN, D. **VSynth FairLight Computer Video Instrument.** Toronto, Canadá. Disponível em: <http://www.audiovisualizers.com/toolshak/vidsynth/fair_cvi/fair_cvi.htm>. Acesso em: 19 ago. 2007.

_____. **Audiovisualizers.** Disponível em: <<http://www.audiovisualizers.com>>. Acesso em: 19 ago. 2007.

EVANS, B. Foundations of a Visual Music. **Computer Music Journal**, v.29, n.4, p.11-24. 2005.

FARINES, J.-M., et al. **Sistemas de Tempo Real.** Florianópolis: Departamento de Automação e Sistemas, UFSC. 2000

FELLEISEN, M. **On the Expressive Power of Programming Languages.** European Symposium on Programming. Copenhagen, Denmark, 1990.

FLETCHER, R. **Some modifications to the standard glut.** York. Disponível em: <<http://www-users.york.ac.uk/~rpf1/glut.html>>. Acesso em: 5 ago. 2007.

FREE SOFTWARE FOUNDATION, I. **GNU General Public License - GNU Project - Free Software Foundation (FSF).** Boston, EUA. Disponível em: <<http://www.gnu.org/copyleft/gpl.html>>. Acesso em: 6 abr 2007.

FRY, B.; REAS, C. Processing, versão 1.0 beta. Cambridge, Massachusetts, 2005. Arquivo de programa baixado em <http://processing.org/>

FUKUCHI, K., et al. **EffecTV: a real-time software video effect processor for entertainment.** International Conference on Entertainment Computing. Eindhoven, Holanda, 2004. 602-605 p.

GAMMA, E., et al. **Design Patterns: Elements of Reusable Object-Oriented Software:** Addison-Wesley Professional. 1995

GOLDBERG, J. **dervish.** Nova Iorque. Disponível em: <<http://goldbergs.com/dervish/>>. Acesso em: 14 de ago. 2007.

GORISSE, N. CPS, versão 1.5. Teteringen: Bonneville, 2007. Arquivo de programa baixado em <http://cps.bonneville.nl/>

HARRISON, T. Salvation, versão 1.0. Harrison Digital Media, 2006. Arquivo de programa baixado em <http://www.harrondigitalmedia.com/>

HERACLES. ARKAOS : Une petite société Carolorégienne qui fait vibrer Jean-Michel Jarre, David Bowie et quelques autres. Charleroi, Bélgica. Disponível em: <http://www.heraclies.be/fr_arkaos.php>. Acesso em: 20 ago. 2007.

HINIC, M. ArKaos VJ, versão 3.6. Waterloo, Bélgica: ArKaos, 2006. Arquivo de programa baixado em <http://www.arkaos.net>

HÖNGER, J., *et al.* Visual Jockey, versão 3.6. Langenthal, Suíça: Visual Light, 2006. Arquivo de programa baixado em <http://www.visualjockey.com>

HSIEH, P. **Direct 3D and OpenGL.** Disponível em: <<http://www.azillionmonkeys.com/windoze/OpenGLvsDirect3D.html>>. Acesso em: 5 ago. 2007.

IRCAM. **WWW Ircam: Accueil.** Paris, França. Disponível em: <<http://www.ircam.fr>>. Acesso em: 19 abr 2007.

JACOBS, S. **Flying Toasters.** Disponível em: <http://www.wired.com/wired/archive/2.05/flying.toasters_pr.html>. Acesso em: 19 ago. 2007.

JÁCOME, J. **Sintetizador de Imagens Metafóricas de Execução Musical.** Recife. Disponível em: <<http://www.cin.ufpe.br/~tg/2004-2/jjoj.pdf>>. Acesso em: 6 ago. 2007.

JÁCOME, J., *et al.* **Tradução de "patch"?** Disponível em: <<https://listas.estudiolivre.org/pipermail/puredeposito/2007-March/000425.html>>.

JONES, R.; NEVILE, B. Creating Visual Music in Jitter: Approaches and Techniques. **Computer Music Journal**, v.29, n.4, p.55-70. 2005.

KALTENBRUNNER, M., *et al.* **Dynamic Patches for Live Musical Performance.** International Conference on New Interfaces for Musical Expression. Hamamatsu, Japan, 2004.

KILGARD, M. J. **The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3.** Disponível em: <<http://www.opengl.org/documentation/specs/glut/spec3/spec3.html>>. Acesso em: 5 ago. 2007.

KIRN, P. **Create Digital Motion: Video Salon Macworld/SF.** Disponível em: <<http://createdigitalmotion.com/2007/01/03/video-salon-macworldsf-free-visual-lounge-open-jam-quartz-composer-jitter-wii-motion-workshops/>>. Acesso em: 6 abr. 2007.

KONIG, E. D.; PLOEG, B. V. D. Resolume, versão 2.4. The Hague, Holanda: Resolume, 2006. Arquivo de programa baixado em <http://www.resolume.com>

LATOUR, P.-O. **PixelShox Technology**. Disponível em: <http://www.pol-online.net/pixelshox_technology/>. Acesso em: 6 abr. 2007.

MALDONADO, G. CSoundAV, versão 0.0432. Roma, 2003. Arquivo de programa baixado em <http://www.csounds.com/maldonado/index.html>

MATHEWS, M. V. **Rtsked, a scheduled performance language for crumar general development system**. International Computer Music Conference: International Computer Music Association, 1981. 286 p.

MATSUDA, S.; RAI, T. **DIPS: the real-time digital image processing objects for Max environment**. International Computer Music Conference. Berlin, 2000.

MATSUSHITA, T. **Expressive Power of Declarative Programming Languages**. 194 f. Phd - Department of Computer Science, University of York, York, 1998.

MCCARTNEY, J. **SuperCollider: a new real time synthesis language**. ICMC. Hong Kong, 1996.

Media Sana. Recife. Disponível em: <<http://www.mediasana.org/>>. Acesso em: 15 ago. 2007.

MESO. **VVVV - A Multipurpose Toolkit**. Frankfurt, Alemanha. Disponível em: <<http://vvvv.meso.net>>. Acesso em: mar. 2005.

MICROSOFT. **DirectX Resource Center**. Disponível em: <<http://msdn2.microsoft.com/pt-br/xna/aa937781.aspx>>. Acesso em: 5 ago. 2007.

_____. **Multimedia Audio**. Disponível em: <<http://msdn2.microsoft.com/en-us/library/ms712572.aspx>>. Acesso em: 6 ago. 2007.

MIESZKOWSKI, K. **The most feared woman on the Internet**. Disponível em: <<http://dir.salon.com/story/tech/feature/2002/03/01/netochka/index.html>>. Acesso em: 8 jan 2007.

MINTER, J. **Welcome do Llamasoft**. London. Disponível em: <<http://www.llamasoft.co.uk/frontpage.php>>. Acesso em: 19 ago. 2007.

MIT. **MIT Media Laboratory**. Cambridge, Massachusetts, EUA. Disponível em: <<http://www.media.mit.edu/>>. Acesso em: 19 abr 2007.

NATIVE INSTRUMENTS, I. **NATIVE INSTRUMENTS: Reaktor 5**. Berlim. Disponível em: <http://www.nativeinstruments.de/index.php?id=reaktor5_us>. Acesso em: 18 abr 2007.

NIELSEN, J. **Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier.** San Francisco, EUA. Disponível em:
[<http://www.useit.com/papers/guerrilla_hci.html>](http://www.useit.com/papers/guerrilla_hci.html). Acesso em: nov. 2006.

_____. **Usability 101: Introduction to Usability.** Disponível em:
[<http://www.useit.com/alertbox/20030825.html>](http://www.useit.com/alertbox/20030825.html). Acesso em: 9 jul. 2007.

NULLSOFT. **Winamp Media Player >> NSDN | Winamp | Writing Plugins | AVS.** Disponível em: [<http://www.winamp.com/nsdn/winamp/plugins/avs/>](http://www.winamp.com/nsdn/winamp/plugins/avs/). Acesso em: 16 mai 2007.

O'WONDER:: CLASSIC BiT BOPPER™. Londres, Inglaterra. Disponível em:
[<http://www.owonder.com/bitbopper/classic/index.php>](http://www.owonder.com/bitbopper/classic/index.php). Acesso em: 19 ago. 2007.

OPENGL. OpenGL Overview. Disponível em:
[<http://www.opengl.org/about/overview/>](http://www.opengl.org/about/overview/). Acesso em: 5 ago. 2007.

OSCHATZ, S., *et al.* VVVV, versão 11.1. Frankfurt, Alemanha: Meso, 2006. Arquivo de programa baixado em <http://vvvv.meso.net>

PELLETIER, J.-M. A Graphical Interface for Real-Time Signal Routing. International Conference on New Interfaces for Musical Expression. Vancouver, Canada, 2005.

PORATOMUSICAL. Porto Musical. Disponível em:
[<http://www.portomusical.com.br/2007/indexport.htm>](http://www.portomusical.com.br/2007/indexport.htm). Acesso em: 7 ago. 2007.

PUCKETE, M., *et al.* Global Visual Music. Disponível em:
[<http://www.visualmusic.org/gvm.htm>](http://www.visualmusic.org/gvm.htm). Acesso em: 11 mai 2007.

PUCKETTE, M. Combining Event and Signal Processing in the MAX Graphical Programming Environment. **Computer Music Journal**, v.15, n.3, p.68-77. 1991a.

_____. FTS: A Real-time Monitor for Multiprocessor Music Synthesis. **Computer Music Journal**, v.15, n.3, p.58-67. 1991b.

PUCKETTE, M. S. Pure Data: another integrated computer music environment. Second Intercollege Computer Music Concerts. Tachikawa, Japan, 1996. 37-41 p.

_____. **Pure Data: Recent Progress.** Third Intercollege Computer Music Festival. Tokyo, Japan, 1997. 1-4 p.

_____. Max at Seventeen. **Computer Music Journal**, v.26, n.4, p.31-43. 2002.

_____. **Pd Documentation.** São Diego, EUA. Disponível em:
[<http://www.crca.ucsd.edu/~msp/Pd_documentation/index.htm>](http://www.crca.ucsd.edu/~msp/Pd_documentation/index.htm). Acesso em: 6 nov. 2006.

_____. Pure Data, versão 0.40. San Diego, EUA, 2006b. Arquivo de programa baixado em <http://www.puredata.org>

RANCH, T. **The Company**. Nova Iorque, EUA. Disponível em:
<http://www.troikaranch.org/company.html>. Acesso em: 25 abr 2007.

RE:COMBO. >>>**Re:Combo**>>>. Recife. Disponível em:
<http://www.recombo.art.br/>. Acesso em: 7 ago. 2007.

RITTER, D. **Orpheus by Don Ritter**. Berlim, Alemanha. Disponível em:
<http://aesthetic-machinery.com/orpheus.html>. Acesso em: 10 mai 2007.

ROGERS, D. F.; ADAMS, J. A. **Mathematical Elements for Computer Graphics**: McGraw-Hill, Inc. 1990. 611 p.

ROKEBY, D. **David Rokeby**. Toronto, Canadá. Disponível em:
<http://homepage.mac.com/davidrokeby/home.html>. Acesso em: 10 mai 2007.

_____. **David Rokeby: softVNS**. Toronto, Canadá. Disponível em:
<http://homepage.mac.com/davidrokeby/softVNS.html>. Acesso em: 10 mai 2007.

ROSSON, M. B.; CARROLL, J. M. **Usability Engineering: Scenario-Based Development of Human-Computer Interaction**. San Francisco: Morgan Kaufmann. 2002. 422 p. (Interactive Technologies)

ROY, P. **Direct3D vs. OpenGL: Which API to Use When, Where, and Why**. Disponível em: <http://www.gamedev.net/reference/articles/article1775.asp>. Acesso em: 5 ago. 2007.

RUDI, J. Computer Music Video: A Composer's Perspective. **Computer Music Journal**, v.29, n.4, p.36-44. 2005.

SCENE. **Scene.org - About the Demoscene**. Disponível em:
<http://www.scene.org/demoscene.php>. Acesso em: 5 abr. 2007.

SCHIMID, Y.; EDELSTEIN, B. Mudul8, versão 2.5. Geneva, Suíça: GarageCUBE, 2006. 1 CD ROM

SCHIMID, Y.; HODGETTS, D. **Modul8 User Manual**. Geneva, Suíça. Disponível em: http://www.garagecube.com/download/Modul8_Documentation.pdf. Acesso em: 15 ago. 2007.

SCHOUTEN, T. **Pure Data Packet**. Disponível em:
<http://zwizwa.fartit.com/pd/pdp/overview.html>. Acesso em: 7 abr 2007.

SPIEGEL, L. Graphical GROOVE: Memorial for the VAMPIRE, a Visual Music System. **Organised Sound**, v.3, n.3, p.187-191. 1998.

STEIM. **S T E I M**. Amsterdam, Holanda. Disponível em:
<http://www.steim.org/steim/>. Acesso em: 25 abr 2007.

SUN MICROSYSTEMS, I. **Java Technology**. Disponível em:
<<http://java.sun.com/>>. Acesso em: 21 abr 2007.

TANNENBAUM, E. **Ed Tannenbaum Home Page**. São Francisco. Disponível em:
<<http://www.et-arts.com/index.html>>. Acesso em: 19 ago. 2007.

THE BILL DOUGLAS CENTRE. **Travelling lanternists**. Disponível em:
<http://www.ex.ac.uk/bdc/young_bdc/lanterns/lantern3.htm>. Acesso em: 18 ago 2007.

UI Software: Videodelic. San Francisco. Disponível em:
<<http://www.uisoftware.com/videodelic/press.html>>. Acesso em: 19 ago. 2007.

USERFOCUS. **ISO 9241: Part 11**. Londres. Disponível em:
<<http://www.userfocus.co.uk/resources/iso9241/part11.html>>. Acesso em: 9 jul 2007.

VEHVILÄINEN, J.; TWINS, P. **Framestein**. Disponível em:
<<http://framestein.org/>>. Acesso em: 7 abr 2007.

VERCOE, B. **History of Csound**. Disponível em:
<<http://www.csounds.com/vercoe/index.html#HISTORY>>. Acesso em: 8 jan 2007.

VJCENTRAL. **VJ Central - Softwares**. Disponível em: <<http://www.vjcentral.com>>. Acesso em: nov. 2006.

WANG, G.; COOK, P. R. **On-the-fly Programming: Using Code as an Expressive Musical Instrument**. International Conference on New Interfaces for Musical Expression. Hamamatsu, Japan, 2004.

WHITING, P. G.; PASCOE, R. S. V. A History of Data-flow Languages. **Annals of the History of Computing**, IEEE, v.16, n.4, p.38-59. 1994.

WHITNEY, J. **Digital Harmony**. Peterborough, EUA: McGraw-Hill. 1980

WIKIPEDIA. **Talk:Comparison of OpenGL and Direct3D**. Disponível em:
<http://en.wikipedia.org/wiki/Talk:Comparison_of_OpenGL_and_Direct3D>. Acesso em: 4 ago. 2007.

WILFRED, T. Light and the Artist. **The Journal of Aesthetics & Criticism**, v.4, n.4. 1947.

WOO, M., et al. **OpenGL Programming Guide**. Massachusetts: Addison-Wesley. 2003

ZICARELI, D. Max/MSP, versão 4.6.2. Los Angeles, EUA: Cycling 74, 2006.
Arquivo de programa baixado em <http://www.cycling74.com>

ZICARELLI, D. MSP Learns to Ride a Bike. Cycling'74. 2005

_____. Max/MSP, versão 4.6.2. Los Angeles, EUA: Cycling 74, 2006. Arquivo de programa baixado em <http://www.cycling74.com>

ZMÖLNIG, M. J. **GEM's rendering engine: a mistery unrevealed**. First Internation Pd Convention. Graz, Austria, 2004a.

_____. **Gem for pd - recent progress**. International Computer Music Conference. University of Miami, CA, USA, 2004b.

Anexo A: Questionário sobre softwares para VJs

1. Há quantos anos você exerce a atividade de VJ?
2. Quantas apresentações de VJ você estima já ter realizado?
3. Quais softwares você utiliza em suas apresentações? (em ordem decrescente de freqüência)
4. Que software você recomendaria para um VJ iniciante?
5. Que software você recomendaria para um VJ experiente?
6. Você está satisfeito com os softwares que você utiliza?
Em caso negativo, marque a alternativa ou especifique a razão principal de sua insatisfação.
 - a. O software que utilizei apresenta limitações de desempenho.
 - b. O software que utilizei apresenta limitações de funcionalidades.
 - c. Outra:
7. Qual seu nível de conhecimento em relação aos seguintes softwares?

a. () ArKaos	
b. () GePhex	0 - Nunca ouvi falar deste software.
c. () Isadora	1 - Já vi ou ouvi falar deste software.
d. () LiVES	2 - Já experimentei este software.
e. () Max/MSP/Jitter	3 - Sou um usuário iniciante deste software.
f. () Modul8	4 - Sou um usuário intermediário deste sw.
g. () Neuromixer	5 - Sou um usuário avançado deste software.
h. () Pd/GEM/PDP	
i. () Quartz Composer	
j. () Resolume	
k. () Visual Jockey	
l. () VJamm	
m. () VRStudio	
n. () vvvv	
8. Da lista de softwares anterior:
 - a. quais os 3 mais fáceis de se aprender a usar (em ordem decrescente)?
 - b. quais os 3 mais difíceis de se aprender a usar (em ordem decrescente)?

	1	2	3	4	5	6	7												8		
							a	b	c	d	e	f	g	h	i	j	k	l	m		
E1	1 ano e 8 meses	15	Res	Res	talvez vvvv	Não. b)	4	2	1	2	1	1	1	3	1	5	2	1	0	2	Res, Ark, LiV, Mod, Gep, Isa, Vjo, vvvv, Pd,Max
E2	5 anos	~75	ArK e Vjo	ArK	depende do estilo	Não. b)	4	3	1	2	1	2	2	2	0	2	4	3	2	1	Ark,Res,VJam,Pd, Isa,Vjo
E3	4 anos e 6 meses	~130	Res, Mod, ArK	Res	Flo, Isa	Não. b)	5	2	2	1	1	4	4	1	1	5	1	2	2	2	VRs, ArK, Res, Neu, VJam, Gep, Isa, vvvv
E4	10 anos	~400	Mod, Neu, Res	VRs	Isa	Não. b)	5	3	5	3	3	5	5	3	3	5	5	5	5	3	-
E5	1 ano e 6 meses	20	Qua + ArK	-	Qua e Isa	Não. b)	4	0	4	0	3	3	2	2	5	3	1	1	1	1	ArK, Mod, Res
E6	3 anos	~125	Res, ArK, vvvv, Max	Res	talvez Res	Sim.	4	1	1	1	4	2	3	2	1	5	3	2	2	4	Res, ArK, Neu, vvvv, Max, Pd
E7	4 anos	~20	ArK,Flash, VLC, ViM	ArK	talvez ArK	Não. b)	5	0	0	0	1	0	0	2	1	1	0	0	0	0	ArK, Pd
E8	10 anos	~70	ArK, Res	pc: Res mac: Mod	Isa	Sim.	5	1	1	1	1	3	3	1	1	3	2	2	1	1	ArK, Mod, Neu, Qua, Gep, Max
E9	5 anos	~250	LiV, Res	ArK	Res	Não. b)	2	4	1	5	1	1	1	2	1	5	2	1	2	1	LiV, VRs, ArK, Isa, Max, Pd
E10	7 anos	~80	Key, Fin, Mod, Res, Del, Vee	Res	Isa, Pd ou Key	Não. c) ver novos sws.	2	3	3	2	4	5	2	5	2	5	2	2	0	3	Res, Mod, ArK, Qua, Pd e SOFs em geral
E11	3 meses	8	ArK	VRs	Res	Não. b)	4	0	1	0	0	0	1	0	1	2	1	1	5	0	VRs, ArK, Res
E12	5 anos	-	ArK, Vjo	ArK, Res	-	Falta doc.	4	2	1	1	3	1	2	3	1	2	4	1	1	2	Neu, Res, ArK Vvvv, Max, Pd
E13	3 anos	~50	Mod, ArK	ArK	Mod	Sim.	5	0	0	0	1	5	1	0	2	2	1	1	1	0	ArK, Res, Mod, SOFs em geral
E14	26 anos	N	Não usa sw	-	-	-	1	1	0	0	0	0	0	0	0	0	0	0	0	0	-
E15	3 anos	~50	Res	pc: Res mac: ArK	Mod	Não. b)	2	1	1	1	1	1	2	1	0	5	1	2	2	1	VRs, Res, ArK, Isa, vvvv
Soma de pontos de nível de conhecimento p cada software (questão 7)							56	23	22	19	25	33	29	27	20	50	29	24	24	21	
Soma de quantidade de entrevistados que usam o software (questão 3)							7	0	0	1	1	4	1	0	1	7	1	0	0	1	

Tabela 4 Respostas ao questionário sobre softwares para VJs.

Anexo B: Questionário de Avaliação da Interface de Caixa Aberta

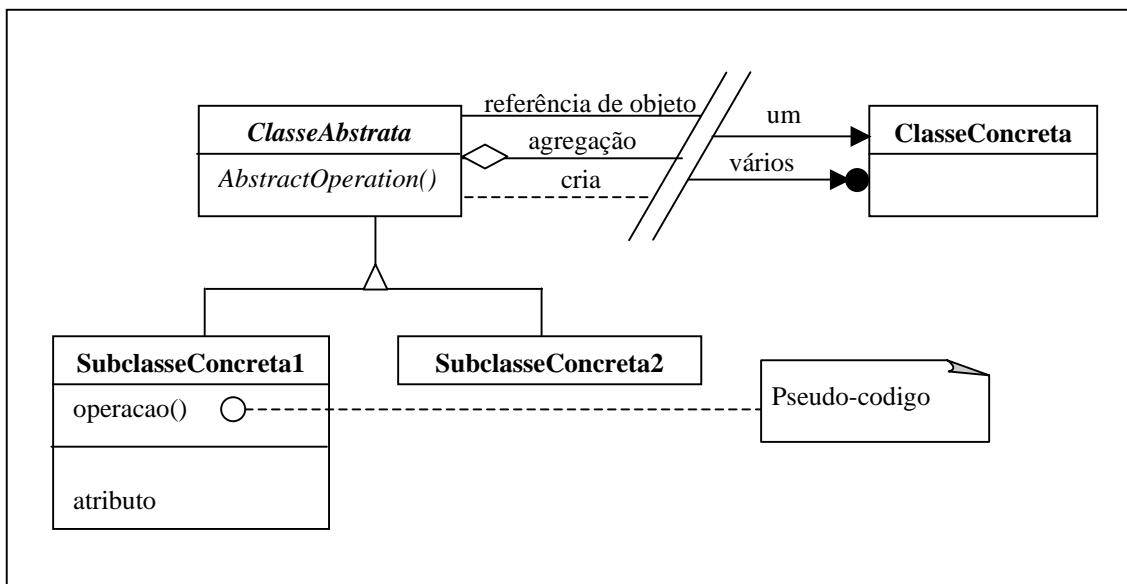
1. Qual o seu conhecimento em sistemas orientados a fluxogramas (*patches*) como Pure Data/GEM, Max/MSP/Jitter, Isadora, Visual Jockey, vvvv, Reaktor, Quartz Composer etc.?
 - a. () Nunca ouvi falar
 - b. () Já ouvi falar
 - c. () Já experimentei um (ou mais) deles
 - d. () Sou usuário iniciante de um (ou mais) deles
 - e. () Sou usuário intermediário de um (ou mais) deles
 - f. () Sou usuário avançado de um (ou mais) deles

Responda as próximas questões comparando o conceito de interface de caixa aberta do ViMus com a interface gráfica dos outros SOFs (baseada em janelas).

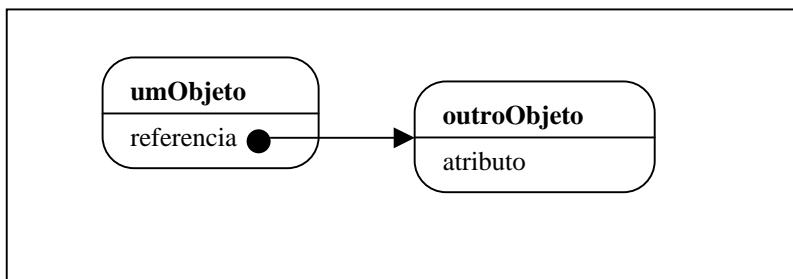
2. Facilidade de aprendizagem: quão intuitiva é a navegação pela interface? Isto é, quão fácil é para um usuário realizar tarefas básicas na primeira vez que encontra esse tipo de interface?
 - a. () Muito mais difícil no ViMus
 - b. () Mais difícil no ViMus
 - c. () Um pouco mais difícil no ViMus
 - d. () Mesma coisa.
 - e. () Um pouco mais fácil no ViMus
 - f. () Mais fácil no ViMus
 - g. () Muito mais fácil no ViMus
3. Eficiência de uso: uma vez que os usuários tenham aprendido a usar este tipo de interface, quão rapidamente eles podem realizar as tarefas?
 - a. () Muito menos eficiente no ViMus
 - b. () Menos eficiente no ViMus
 - c. () Um pouco menos eficiente no ViMus
 - d. () Mesma coisa.
 - e. () Um pouco mais eficiente no ViMus
 - f. () Mais eficiente no ViMus
 - g. () Muito mais eficiente no ViMus
4. Satisfação subjetiva: quão prazeroso é usar a interface?
 - a. () Muito menos prazeroso no ViMus
 - b. () Menos prazeroso no ViMus
 - c. () Um pouco menos prazeroso no ViMus
 - d. () Mesma coisa.
 - e. () Um pouco mais prazeroso no ViMus
 - f. () Mais prazeroso no ViMus
 - g. () Muito mais prazeroso no ViMus

Anexo C: Notações de diagramas

Notação de Diagrama de Classes



Notação de Diagrama de Objetos



Dissertação de Mestrado apresentada por **Jarbas Jácome de Oliveira Junior** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **“Sistemas Interativos de Tempo-real para Processamento Audiovisual Integrado”**, orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Silvio de Barros Melo
Centro de Informática / UFPE

Prof. Etienne Delacroix
Professor Visitante - UnB

Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 29 de agosto de 2007.

Francisco Tenório de Carvalho
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.