UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JOSÉ DIEGO FERREIRA MARTINS

Uma proposta de um motor de animação para o controle de personagens articulados baseado em autômatos finitos

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof^{a}. Dr^{a}. Luciana Porcher Nedel
Orientadora

Prof. Dr. Paulo Blauth
Co-orientador

Porto Alegre, junho de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

José Diego Ferreira Martins,

Uma proposta de um motor de animação para o controle de personagens articulados baseado em autômatos finitos /

José Diego Ferreira Martins. – Porto Alegre: PPGC da UFRGS, 2006.

92 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2006. Orientadora: Luciana Porcher Nedel; Co-orientador: Paulo Blauth.

1. Animação. 2. WWW. 3. Grafo de cena. 4. Autômato finito. I. Porcher Nedel, Luciana. II. Blauth, Paulo. III. Título.

AGRADECIMENTOS

À minha querida Karine, que me acompanha em todos os momentos e me traz muita felicidade e amor.

Aos colegas do grupo, onde me senti acolhido, fiz boas amizades e aprendi bastante com eles.

Aos colegas da EC4-FURG. Se não fosse uma turma tão legal, eu não teria tido paciência nem perseverança para concluir o curso.

Aos colegas do II que me acompanharam nesta jornada, em especial o Mario, o Boffo, o Paulista e o "Bat-Lucc", que me prestaram ajuda em questões de programação, normas e LaTeX.

Por último, é mais do que importante citar meus orientadores, que tiveram bastante paciência comigo e me ajudaram de forma inestimável na realização deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

3D	Três dimensões
AABB	Axis Aligned Bounding Box
AF	Autômato Finito
AFD	Autômato Finito Determinístico
AFN	Autômato Finito Não-Determinístico
AFS	Autômato Finito com Saída
AGA	Automata-based Graphical Animation
AGA-J	AGA for Joints Level
AGA-T	AGA for Tasks Level
API	Application Programming Interface
DAG	Directed Acyclic Graph (grafo acíclico dirigido)
DTD	Document Type Definition
FPS	Frames Per Second (quadros por segundo)
WWW	World Wide Web

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Segundo Thalmann em 1996, animação é a visualização da variação do estado de objetos em relação ao tempo. A meta principal da animação modelada por computador é sintetizar o efeito desejado de movimento que é uma mistura de fenômenos naturais, percepção e imaginação. O sistema de animação deve providenciar ferramentas de controle de movimento para traduzir os desejos do usuário na linguagem do sistema.

O controle dos movimentos de uma animação pode ser dividido em duas classes de algoritmos: a classe intencional, dirigida aos aspectos de controle em alto nível, onde as intenções de uma animação são expressas e a classe concreta, dirigida aos aspectos específicos de animação de objetos. A relação hierárquica destas classes favorece a implementação de um motor de animação usando camadas de abstração.

Uma das abordagens atuais para a construção de um motor de animação para atores humanóides é um sistema implementado em camadas. A camada inferior lida com os movimentos individuais das articulações. As intermediárias são responsáveis por encapsular conjuntos de movimentos que atuam como tarefas simples. O encapsulamento destas tarefas é feito por camadas superiores que lidam com ações sensíveis ao ambiente. Por fim, a camada de mais alto nível atua no nível cognitivo onde um mecanismo dotado de capacidade de raciocínio comanda ações aos atores de acordo com as informações do contexto e também de acordo com as intenções, crenças e desejos dos atores.

Este trabalho visa analisar uma proposta de implementação de um motor de animação para simular o comportamento de atores humanóides, gerando animações em três dimensões via World Wide Web. Dois modelos de animação baseados no formalismo AFS (Autômato Finito com Saída) que lidam com camadas específicas de abstração de um sistema de animação serão apresentados. O modelo AGA-J é uma extensão do modelo AGA para lidar com a classe concreta de movimentos de atores humanóides em 3D e serve como camada de fundação para outros modelos. O modelo AGA gera sequências animadas a partir de um conjunto de AFS que representam os atores da animação e um conjunto de fitas de entrada que determinam os comportamentos dos atores. O alfabeto de saída é representado por imagens e efeitos sonoros e a animação se dá pela sequência das saídas dos autômatos. O modelo AGA-T é um modelo orientado a tarefas, baseado em autômatos probabilísticos e eventos. Este modelo atua como uma camada de nível superior sobre o AGA-J, e foi projetado para lidar com alguns aspectos da classe intencional, sobretudo o encapsulamento de movimentos através de tarefas.

O formalismo de AFS foi escolhido para a construção dos modelos porque propicia muitas características importantes: *saída configurável* através da análise da fita de saída; *reusabilidade* pelo fato de vários atores poderem usar o mesmo AFS; e *recuperação de informação*, observando o estado atual dos autômatos dos atores que compõem a animação.

Palavras-chave: Animação, WWW, grafo de cena, autômato finito.

A proposal of an animation engine for controlling articulated characters based on finite automata

ABSTRACT

According to Thalmann in 1996, animation is the visualization of changings on object states during time. The first goal of computer animation is to synthesize the desired motion effect that is a mix of natural phenomena, perception and imagination. An animation system must provide motion control tools to translate the user intentions into the system language.

The motion control can be divided in two classes of algorithms: the intentional class, directed towards the high-level control aspects where the animation intentions are expressed; and the concrete class, which deals with specific aspects of object animation. The hierarchical relationship between these two classes benefits the implementation of an animation engine based on abstraction layers.

One recent approach for building an animation engine to represent humanoid actors is a system implemented in layers. The lowest layer deals with individual joint movements, while the middle layers are responsible for encapsulating a set of movements acting as simple tasks. The task encapsulation is performed by high-level layers dealing with environment sensitive actions. Finally, the highest-level layer acts in the cognitive level, where a reasoning mechanism commands actions to the actors according to the context information, the beliefs, desires and intentions of the actors.

This work explores an implementation proposal of an animation engine for simulating the behavior of humanoid actors, generating 3D animations to the World Wide Web. Two animation models based on the Finite Automata with Output formalism that deal with specific abstraction layers of an animation system will be presented. The AGA-J model is an extension of the AGA model for dealing with the concrete class of 3D movements of humanoid actors and it suits as a foundation layer for other models. The AGA model generates animated sequences by a set of Finite Automata with Output that implements the animation actors and a set of input tapes which determines the behavior of the actors. The output alphabet is represented by images, and sound effects, and the animation is produced by the sequence of the automata output. The AGA-T model is a task-oriented model, based on probabilistic automata and events. This model acts as a high-level layer over the AGA-J, and it was designed for dealing with some aspects of the intentional class, mainly the movement encapsulation by tasks.

The Finite Automata with Output formalism was chosen for model construction because it provides many important features: *adaptable output* by analysis of output tape; *reusability* by the fact of many actors can use the same automaton; and *information retrieval* by observation of current state of the automata composing the animation.

Keywords: animation,WWW,scenegraph,automata.

1 INTRODUÇÃO

1.1 Motivação

Animação é a visualização da variação do estado de objetos em relação ao tempo. A animação convencional é definida por Thalmann *et al.* (THALMANN; THALMANN, 1985) como a técnica na qual é criada a ilusão de movimento através de uma série de desenhos fotografados individualmente e gravados em sucessivos quadros de um filme. A ilusão de movimento é alcançada quando o filme é projetado a uma certa taxa de quadros por segundo.

A meta principal da animação modelada por computador é sintetizar o movimento desejado, que é uma mescla de fenômenos naturais, percepções e imaginação. Os componentes principais de um sistema de animação são: o modelador de atores, o mecanismo de controle da animação e o renderizador. Há várias formas de descrever, planejar e especificar animações, levando em conta estes componentes. Fatores desejados, como interatividade e inteligência artificial tornam as animações mais realísticas, podendo fazer parte da especificação da animação, dependendo do suporte oferecido pelo modelador de atores e pelo mecanismo de controle. As características do renderizador (modelo de iluminação, posicionamento de câmera, corte de cena, etc.) também podem influenciar bastante na descrição da animação.

O projetista de animação define o comportamento dinâmico dos objetos usando sua representação mental de causas e efeitos. Ele imagina como eles se movem, mudam de forma e suas reações quando são empurrados, puxados, pressionados e torcidos. O sistema de animação deve providenciar recursos que possibilitem ao projetista reproduzir os movimentos imaginados em sua própria linguagem (THALMANN; THALMANN, 1996).

Esta tarefa é mais complicada no caso de animações envolvendo figuras humanóides. (THALMANN; THALMANN, 1985) Como um caso geral, o projetista tem que planejar o comportamento de cada ator no ambiente virtual. Adicionalmente, ele pode atuar como um coreógrafo, tomando conta de todos os detalhes envolvendo os movimentos dos esqueletos dos atores. O fato de um esqueleto humano ter cerca de 200 graus de liberdade e uma sequência de animação ser geralmente composta de mais de um personagem tornam definitivamente complexo produzir este tipo de animação.

Humanos Virtuais são modelos computacionais de pessoas, usados tanto como substitutos para pessoas reais em avaliações ergonômicas de projetos baseados em computador, quanto para embutir representações de participantes reais em ambientes virtuais. Se necessário, humanos virtuais podem apresentar uma aparência bastante realista, baseada em princípios fisiológicos e biomecânicos. Restrições funcionais podem ser aplicadas a eles, a fim de que seus movimentos satisfaçam as limitações humanas (BADLER et al., 1999).

Em aplicações como jogos ou na produção de animação, humanos virtuais podem ser utilizados como atores sintéticos. Um ator sintético, também chamado virtual ou digital, é definido como um ator autônomo, com aparência humana e completamente gerado por computador (THALMANN, 1995).

A modelagem de humanóides é um problema desafiante porque os modelos geométricos e matemáticos da computação gráfica não são adequados para representar a forma do corpo humano. Mesmo usando modelagens simples, o movimento articulado apresenta uma série de complicações. Reposições, torções e as dobras do torso, movimentos reativos do outro braço e o contrabalanço das pernas geralmente são partes de movimentos que aparentemente são relacionados apenas a um braço (PARENT, 2002). A especificação desse tipo de movimento vai além da simples determinação de quais juntas usar em um movimento particular.

Para simular o comportamento de seres humanos, pode-se considerar um sistema implementado em quatro níveis. O nível inferior lida com os movimentos individuais de cada articulação. O nível intermediário é responsável por encapsular movimentos compostos que agrupam um conjunto de movimentos de articulação, representando ações simples (sentar, levantar, pegar algo, dar um passo, etc.). A combinação destas ações podem providenciar um nível maior de abstração, chamados de tarefas (abrir a porta, caminhar da sala até o corredor, etc.). Finalmente, o nível mais alto de abstração considerado é o nível cognitivo. Neste nível, um mecanismo de decisão comanda comportamentos de acordo com informações extraídas de um contexto (posição, orientação e distância de um alvo) e as intenções, crenças e desejos dos personagens envolvidos.

Foi desenvolvido no Instituto de Informática da Universidade Federal do Rio Grande do Sul uma proposta bem sucedida de implementação do nível cognitivo usando lógica BDI para simular animações com agentes autônomos (TORRES; NEDEL; BORDINI, 2003). Entretanto, para um sistema de animação ser convincente, boas soluções para os níveis que lidam com articulações e ações simples também devem ser investigados. Estas soluções devem ser consistentes, providenciar reaproveitamento e simples especificação de movimentos.

Outro trabalho desenvolvido no Instituto de Informática da UFRGS é o modelo AGA (*Automata-based Graphical Animation* - Animação Gráfica baseada na teoria de Autômatos), especificado por Fernando Accorsi (ACCORSI; MENEZES, 2000), que foi utilizado para a construção de um protótipo operacional para reprodução de animações na World Wide Web, visando às necessidades do sistema Hyper-Automaton (MACHADO, 2000), desenvolvido por Júlio Henrique A. P. Machado durante o seu mestrado, realizado também no Instituto de Informática da UFRGS. Trata-se de um ambiente semi-automatizado para o suporte ao gerenciamento de hipertextos destinados ao ensino à distância, cuja arquitetura está baseada em um modelo de organização de hiperdocumentos definido sobre o formalismo de Autômatos Finitos com Saída. O modelo AGA será descrito com mais detalhes na Seção 3.

A reutilização de componentes da animação é um ponto importante no que concerne à redução de espaço de armazenamento e na concentração de informação de especificação, principalmente no caso de parametrização de meta-objetos. A recuperação de informação em animações também é uma questão importante a ser considerada. À medida que o volume de informações de uma animação cresce, o desenvolvimento de estratégias eficientes de recuperação de conteúdo visual se torna ainda mais indispensável.

Outra questão relevante diz respeito à criação e manutenção das animações para exposição via WWW. O conteúdo visual na WWW é alterado e expandido constantemente,

sendo portanto, necessário o desenvolvimento de sistemas de animação que ampliem as possibilidades de reutilização e manutenção de animações, contribuindo assim, com a produtividade para criação desse tipo de conteúdo.

As características particulares da WWW relacionadas às animações têm evidenciado diversas questões em torno deste tema, fomentando pesquisas com o propósito de melhorar os processos de criação, manutenção, apresentação e consulta às animações na WWW. A grande quantidade de espaço de armazenamento necessário para representar as animações é uma dessas questões, pois, mesmo com o aumento da capacidade de armazenamento dos equipamentos e a melhora das taxas de transferência em redes, ainda há restrições que limitam, muitas vezes, a qualidade e a utilização de animações.

Hoje em dia, é comum encontrar programas de animação para a WWW que trabalham com movimentos individuais em vez de usar um nível maior de abstração. Um dos exemplos mais conhecidos de animadores em 2D é o Macromedia Flash (MACROMEDIA, 2002), que trabalha com representações vetoriais dos objetos e utiliza a linguagem *ActionScript*, que é uma linguagem procedural com código similar à linguagem C e orientada a eventos.

A recuperação de informação em animações está fortemente relacionada com uma característica importante da WWW que é sua capacidade de adaptar o fluxo de saída ao dispositivo utilizado. O conteúdo das páginas é disposto conforme a resolução do dispositivo gráfico, ou apresentado apenas em texto, na falta de dispositivo gráfico, com imagens substituídas por descrições textuais.

Em 2002, Fernando Accorsi (ACCORSI, 2002) apontou o fato de que as animações 3D são bastante escassas nas páginas WWW, enquanto são raros os sites que não utilizam alguma animação 2D para enriquecer seu conteúdo. Atualmente, esta escassez ainda se mantém.

O formalismo Autômato Finito Determinístico com Saída é robusto, não sofre o problema da parada oriundo dos formalismos baseados na Máquina de Turing e, no caso da especificação de animações, permite associar cada transformação gráfica de um ator a um símbolo do alfabeto de saída, providenciando reusabilidade e livre configuração da saída. Outra vantagem é a recuperação de informação por meio da observação do estado dos autômatos.

O modelo AGA (ACCORSI; MENEZES, 2000) especifica as alterações sofridas pelos atores na animação a partir de uma estrutura baseada em autômatos finitos. O modelo é definido de tal forma que os mesmos autômatos utilizados na especificação, ao serem simulados, realizem o controle da animação durante a apresentação. A animação é especificada a partir de um conjunto de atores e suas respectivas variações ao decorrer da cena, através de um modelo formal baseado em autômatos finitos com saída, englobando as funcionalidades de reprodução encontradas atualmente nas abordagens quadro-a-quadro (THALMANN; THALMANN, 1985). (CAMPANI, 2005) propôs um método para avaliar a compressão de dados e da qualidade de imagens e animações usando complexidade de Kolmogorov. Como estudo de caso, o método de comparação e avaliação de modelos de animação gráfica foi aplicado na comparação do modelo AGA contra o modelo GIF, provando formalmente que AGA comprime mais as animações que GIF.

1.2 Objetivos

Este trabalho tem como objetivo aplicar a Teoria de Autômatos na proposição de uma alternativa para prover animações 3D com humanos virtuais a serem reproduzidas local-

mente e na WWW, verificando as contribuições alcançadas para as questões relacionadas à transmissão de dados, reusabilidade, manutenibilidade e suporte à recuperação de informação.

A Figura 1.1 mostra um exemplo de camadas de um sistema de animação e o alcance desta proposta nas mesmas. Todas as camadas, exceto a inferior, são de controle intencional.

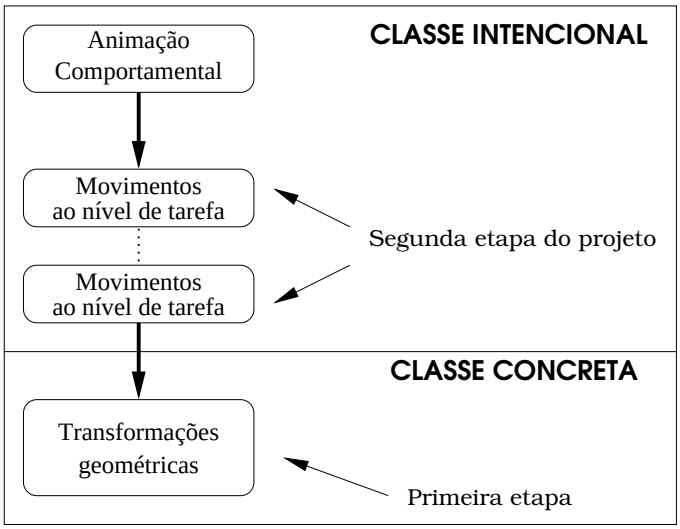


Figura 1.1: Camadas de controle de animação

As metas a seguir foram ordenadas cronologicamente para possibilitar a aplicação da Teoria de Autômatos em animações 3D:

- Projetar uma extensão do modelo AGA (ACCORSI; MENEZES, 2000) para modelar e controlar animações em 3D com humanos virtuais. A intenção deste item é estudar a aplicabilidade do AGA num ambiente 3D com modelos hierárquicos de atores para provar a generalização do modelo AGA;
- Implantar um protótipo operacional baseado na extensão projetada acima, que suporte a especificação de animações na WWW com saída configurável. Como seu antecessor, o protótipo reproduzirá as animações que serão descritas numa extensão da linguagem XML chamada de AGA2ML;
- Providenciar uma avaliação das potencialidades do modelo estendido e um estudo de caso, com o objetivo de provar o reaproveitamento do modelo baseado em autômatos finitos como camada de fundação;
- A partir deste estudo de caso, projetar uma especificação de animação em nível de tarefa. Projetar também uma interface entre esta especificação e a do modelo estendido para torná-la uma camada superior, reaproveitando as capacidades do modelo estendido.

1.3 Organização do texto

O Capítulo 2 apresenta uma visão geral sobre ferramentas para a criação de animação por computador. São apresentados, em ordem cronológica, modelos e sistemas de ani-

mação para demonstrar a evolução dos mesmos e linguagens script usadas na descrição de animações, demonstrando sistemas com aplicações práticas de especificação de objetos animados para controle concreto de animação e para descrever o controle intencional dos atores. É dada uma atenção especial a linguagens de especificação de modelos de comportamento para a interação entre atores.

O Capítulo 3 contém a especificação do modelo AGA-J, que é praticamente o modelo AGA sob nova ótica. O modelo AGA-J é um modelo de animação baseado em autômatos finitos com saída. O modelo AGA-J gera sequências animadas a partir de um conjunto de AFS que representam os atores da animação e um conjunto de fitas de entrada que determinam os comportamentos dos atores. Cada ator é modelado graficamente por um corpo articulado e o alfabeto de saída é representado por conjuntos de transformações nas articulações dos atores. O modelo AGA também é brevemente descrito neste capítulo.

O Capítulo 4 apresenta a proposta do modelo AGA-T, que é um modelo orientado a tarefas, baseado em autômatos probabilísticos que usa o modelo AGA-J como camada inferior. A necessidade de um modelo de mais alto nível surge devido ao fato de que animações mais complexas vão requerer algum tipo de *frontend* porque tornam-se difíceis de serem especificadas, devido ao grau de dependência entre os atores em relação ao tempo.

O Capítulo 5 mostra exemplos de animação gerados por protótipos que implementam os modelos AGA-J e AGA-T e um estudo de caso que ilustra o potencial de recuperação de informação que o modelo oferece, bem como a utilização deste potencial para gerar um modelo de saída adaptável a outros meios de reprodução.

Finalmente, o Capítulo 6 reúne as principais conclusões e contribuições deste trabalho bem como várias potencialidades para serem exploradas através de trabalhos futuros.

2 VISÃO GERAL SOBRE MOTORES DE ANIMAÇÃO

2.1 Introdução

Este capítulo apresenta alguns sistemas utilizados para a construção de animações e modelos usados para o controle de animações em vários níveis de abstração. Como animações são descritas geralmente por alguma espécie de Linguagem Script, este capítulo também apresenta algumas linguagens adequadas ou criadas especialmente para este propósito.

A Seção 2.2 apresenta o Macromedia Flash, um dos mais populares sistemas de animação para a World Wide Web. A Seção 2.3 mostra um modelo para o controle de animações com personagens articulados usando uma hierarquia de máquinas de estado. A Seção 2.4 apresenta três modelos para o controle de animações comportamentais que utilizam redes de máquinas de estado. A Seção 2.5 apresenta um modelo baseados em Autômatos Finitos Sincronizados. As linguagens que descrevem animações de objetos são apresentadas na Seção 2.6 e as linguagens que descrevem animação comportamental e agentes na Seção 2.7.

2.2 SWF - Macromedia Flash

O Flash (MACROMEDIA, 2002) é um sistema de animação destinado à criação de animações bidimensionais para Web. Este sistema tem sido utilizado comumente na criação de interfaces gráficas diferenciadas para *websites*, produção de seqüências animadas e no desenvolvimento de jogos *online*. Suas animações podem conter atores gráficos baseados em curvas ou mapa de bits, textos, vídeos e objetos de interação. O sistema permite a criação de seqüências animadas interativas, assim como a sincronização destas com efeitos sonoros.

O ambiente de criação conta com um modelador geométrico de atores com diversas ferramentas de desenho e colorização, um mecanismo de animação baseado na interpolação de quadros-chave, um editor para a linguagem de script, geradores para diversos formatos de arquivo de intercâmbio e um sistema integrado de visualização das animações.

O modelador de atores provê ferramentas de modelagem baseadas em primitivas geométricas e desenho livre. Os desenhos modelados são convertidos em representações vetoriais e colorizados a partir de esquemas de cores sólidas e gradientes. Os textos são editados através de uma ferramenta especial e podem ser manipulados como figuras vetoriais. Os vídeos e imagens na forma de mapas de bits podem ser inseridos no ambiente a partir de mecanismos de importação de arquivos. Os objetos gráficos podem ser manipulados na área de trabalho a partir de transformações geométricas básicas. O ambiente

ordena a sobreposição dos objetos através da organização da área de trabalho em camadas.

O controle de animação utilizado pelo Flash é explícito. O animador define as alterações através de transformações geométricas dos atores e mudanças de atributos de colorização. O mecanismo de animação também provê processos de interpolação de quadros-chave, no qual o animador define interativamente os parâmetros dos atores nos quadros-chave origem e destino, deixando para o sistema a criação dos quadros intermediários. Os movimentos também podem ser definidos através de scripts escritos em sua linguagem *ActionScript*, que é uma linguagem procedural com código similar à linguagem C e orientada a eventos, que são pertencentes aos objetos da animação.

A Figura 2.1 mostra exemplos de eventos básicos relacionados à cena inteira. **this** refere-se ao objeto dono do evento. No exemplo, quando a animação é carregada, a escala da dimensão X do quadro vai para 50%. A cada quadro mostrado, a escala aumenta em 5% até chegar a 500%. A Figura 2.2 mostra um código que faz um objeto chamado **bob** mover-se até o local onde o usuário clicou com o mouse. O objeto **_root** refere-se à cena inteira e contém todos os objetos criados como seus atributos. O evento de *mouse click* é capturado pelo trecho *onClipEvent(mouseDown)*, que vai salvar as coordenadas atuais do objeto **bob** e as coordenadas da janela onde ocorreu o click. O programa principal efetua uma animação com base nas coordenadas salvas, incrementando de 5 em 5 pixels a cada quadro exibido.

```
// ocorre no carregamento da animação
onClipEvent(load)
{
   this._xscale = 50;
}

// evento que ocorre a cada quadro mostrado
onClipEvent(enterFrame)
{
   if( this._xscale < 500 ) {
       this._xscale += 5;
   }
}
```

Figura 2.1: ActionScript - eventos de cena

Grande parte da popularidade do Flash está atribuída à eficiência do SWF (*Shockwave Flash*) como formato de intercâmbio de animações. O SWF foi desenvolvido especificamente para a distribuição de gráficos vetoriais e animação na Internet. Entre as principais metas do projeto do SWF estão: a alta velocidade de rendering, a distribuição em larguras de banda limitadas e a exploração de várias técnicas de compressão para produção de arquivos pequenos.

Um arquivo SWF pode ser visto como um conjunto de blocos etiquetados distribuídos seqüencialmente. Este conjunto é sempre precedido por um cabeçalho de identificação e concluído por um bloco especial de finalização. Os demais blocos são categorizados entre blocos de definição ou controle. Os blocos de definição especificam as figuras, textos, imagens e sons utilizados na animação, enquanto os blocos de controle determinam as variações dos objetos e o controle de fluxo da animação. Com o objetivo de apresentar a animação como mídia contínua, o conteúdo de cada bloco só depende de blocos anteriores a ele.

A Figura 2.3 mostra um exemplo de seqüência de blocos do SWF entregues ao programa de visualização. À medida que os blocos de definição vão sendo processados, o

```
// ocorre quando o usuário aperta o botão do mouse
onClipEvent(mouseDown)
{
   xtarget = _root._xmouse;
   ytarget = _root._ymouse;
   xstart = this._x;
   ystart = this._y;
}

// parte principal do programa
x = Math.abs(xstart - xtarget);
y = Math.abs(ystart - ytarget);

if( x > y ) {
   xspeed = 5;
   yspeed = 5*(y/x);
 } else {
   yspeed = 5;
   xspeed = 5*(x/y);
}

with(_root.bob) {
   if( _x > (xtarget+2.5) || _x < (xtarget-2.5) ||
       _y > (ytarget+2.5) || _y <(ytarget-2.5) ) {
     if( _x > xtarget ) {
        _x = _x - xspeed;
      } else {
        _x = _x + xspeed;
     }
     if( _y > ytarget ) {
        _y = _y - yspeed;
      } else {
        _y = _y + yspeed;
     }
   }
}
```

Figura 2.2: ActionScript - animação básica com o mouse

programa de visualização forma um dicionário inserindo os objetos definidos em um repositório. O reuso dos elementos do dicionário é uma das técnicas exploradas pelo SWF a fim de reduzir o tamanho do arquivo, pois uma vez inserido o objeto no dicionário, o mesmo não precisa ser retransmitido, apenas recuperado do repositório. Os blocos de controle manipulam instâncias dos objetos do dicionário em uma lista de apresentação organizada em camadas de exibição. Quando a geração de um quadro é solicitada (ShowFrame), a lista de apresentação é processada para obter a imagem referente ao quadro corrente.

A definição do SWF não provê nenhuma estrutura específica que favoreça a recuperação de informação de seu conteúdo. Esta característica dificulta a tarefa de indexação tanto do conteúdo textual quanto visual destes arquivos pelos mecanismos automatizados.

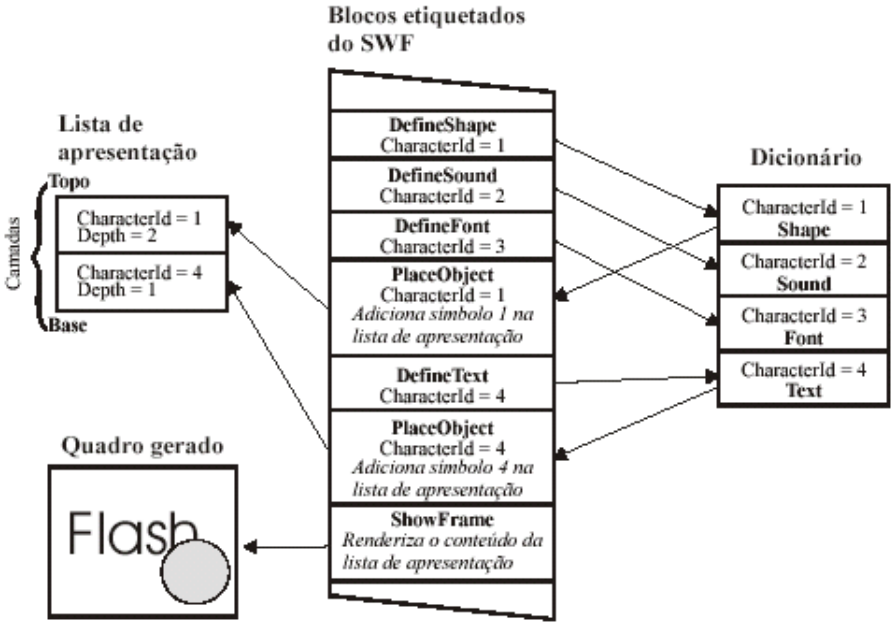


Figura 2.3: SWF - Processamento do arquivo (ACCORSI, 2002)

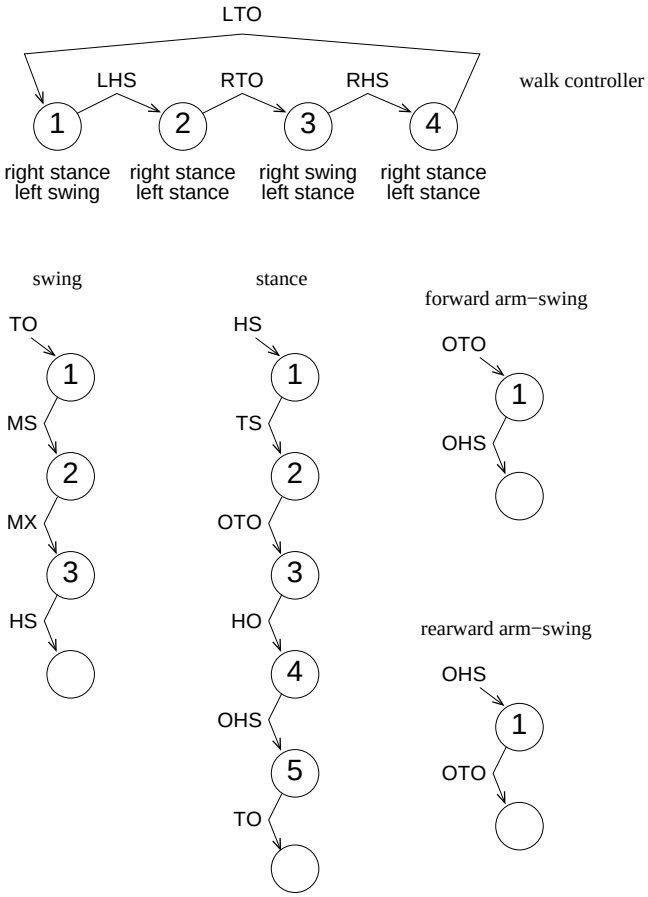
2.3 Técnicas de controle motor para animação de figuras articuladas

Zeltzer (ZELTZER, 1982) criou um modelo onde uma hierarquia de máquinas de estado é usada para o controle cinemático de um esqueleto humano. Os movimentos de maior grau de abstração são controlados por máquinas de estado parametrizáveis chamadas de *Motor Control Programs*. Estas máquinas agregam outras máquinas de estado chamadas de *Local Motor Programs* (LMPs) as quais controlam os movimentos individuais das articulações. A geometria dos objetos é especificada explicitamente por uma descrição cinemática (posição, orientação e deformação, p.ex.), sem levar em conta as causas do movimento.

Um exemplo de hierarquia das máquinas de estado está apresentado na Figura 2.4. Este exemplo implementa o controle dos passos de uma caminhada. A máquina de estado do *Motor Control Program* da caminhada (*walk controller* no exemplo da figura) é responsável pela ativação explícita de duas máquinas de estado (*Local Motor Programs*) chamadas *stance* e *swing*. Estas máquinas recebem os parâmetros LEFT e RIGHT, de acordo

com o movimento desejado. As transições de todas as máquinas de estado são ativadas mediante eventos de articulação e/ou de tempo, apresentados na legenda da Figura 2.4. A Tabela 2.3 descreve as ações realizadas pelos estados das (*Local Motor Programs*).

O modelo é adequado para organizar de forma hierárquica as ações de personagens articulados com movimentos simples e bem definidos. Uma desvantagem desta abordagem é a perda do controle artístico em favor da síntese automática de movimento visto que o modelo não especifica as causas do movimento. Para modelar movimentos mais realísticos e complexos, as máquinas de estado sofreriam explosões de estado para poder abrigar todas as etapas do movimento. A única forma de recuperação de informação é por simples observação do estado das máquinas de estado envolvidas, porque o modelo não provê meios explícitos para recuperação de informação.



Legenda	
O	- Opposite
L	- Left
R	- Right
HS	- Heel-Strike
TO	- Toe-Off
MS	- MidSwing
MX	- Maximum hip fleXion
TS	- Toe-Strike
HO	- Heel-Off

Figura 2.4: Máquinas de estado que controlam uma caminhada (ZELTZER, 1982)

2.4 Animação controlada através de redes de transições paralelas

Redes de Transições Paralelas são um conjunto de máquinas de estado que executam em paralelo. Em animações, elas são usadas para o controle comportamental de um ator/agente, podendo colocar ações em sequência, baseadas no estado atual do ambiente

Tabela 2.1: Comandos dos estados da caminhada (ZELTZER, 1982)

swing	1	HIP FLEXION
		KNEE FLEXION
		ANKLE FLEXION
	2	HIP FLEXION
		KNEE EXTENSION
		ANKLE FLEXION
	3	HIP EXTENSION
		KNEE EXTENSION
		ANKLE EXTENSION
stance	1	DOWNWARD PELVIC CORONAL ROTATION
		ABOUT SUPPORT HIP
		HIP EXTENSION
		KNEE FLEXION
		ANKLE EXTENSION
		SUPPORT ROTATION ABOUT HEEL
	2	UPWARD PELVIC CORONAL ROTATION
		ABOUT SUPPORT HIP
		FORWARD TRANSVERSE ROTATION
		ABOUT SUPPORT HIP
		HIP EXTENSION
		KNEE FLEXION
		ANKLE FLEXION
	3	UPWARD PELVIC CORONAL ROTATION
		ABOUT SUPPORT HIP
		FORWARD PELVIC TRANSVERSE ROTATION
		ABOUT SUPPORT HIP
		HIP EXTENSION
		KNEE EXTENSION
		ANKLE FLEXION
	4	FORWARD PELVIC TRANSVERSE ROTATION
		ABOUT SUPPORT HIP
		HIP EXTENSION
		KNEE FLEXION
		ANKLE EXTENSION
		SUPPORT ROTATION ABOUT BALL OF FOOT
	5	HIP FLEXION
		KNEE FLEXION
		ANKLE EXTENSION
forward arm-swing	1	ELBOW FLEXION
		SHOULDER FLEXION
rearward arm-swing	1	ELBOW EXTENSION
		SHOULDER EXTENSION

ou de alguma meta do agente e representar as tarefas em progresso, condições a serem monitoradas, recursos utilizados e propiciar sincronização temporal.

2.4.1 Modelagem de animações com redes de Petri

Bicho, Raposo e Magalhães (LIMA BICHO; RAPOSO; MAGALHAES, 2001) apresentam o uso de redes de Petri como uma ferramenta para controlar os movimentos de figuras articuladas inspirados no trabalho de Raposo, Magalhães e Ricarte (MAGALHAES; RAPOSO, 1988) que demonstra, de uma forma mais genérica, o uso de redes de Petri para análise e modelagem de animações.

É apresentada uma modelagem orientada a eventos que trabalha com o controle global (intencional) do ator, levando em conta aspectos como concorrência, sincronização e conflitos de evento. São usadas extensões ao modelo básico de redes de Petri: arcos inibidores, prioridades de disparo e redes temporizadas. (MURATA, 1989)

A Figura 2.5 mostra uma rede de Petri que controla o caminhar de uma figura bípede qualquer. Os locais P1, P3, P5 e P9 atuam como condições de controle e estão descritos na Tabela 2.4.1. Os locais P2, P4, P6, P7 e P8 correspondem a ações efetuadas pelo ator que estão apresentadas na Tabela 2.4.1. Note que o número desejado de passos do ator é controlado apenas pela transição de P3 para P5, dando um controle centralizado ao modelo. No exemplo apresentado na Figura 2.5, o número de passos é 3.

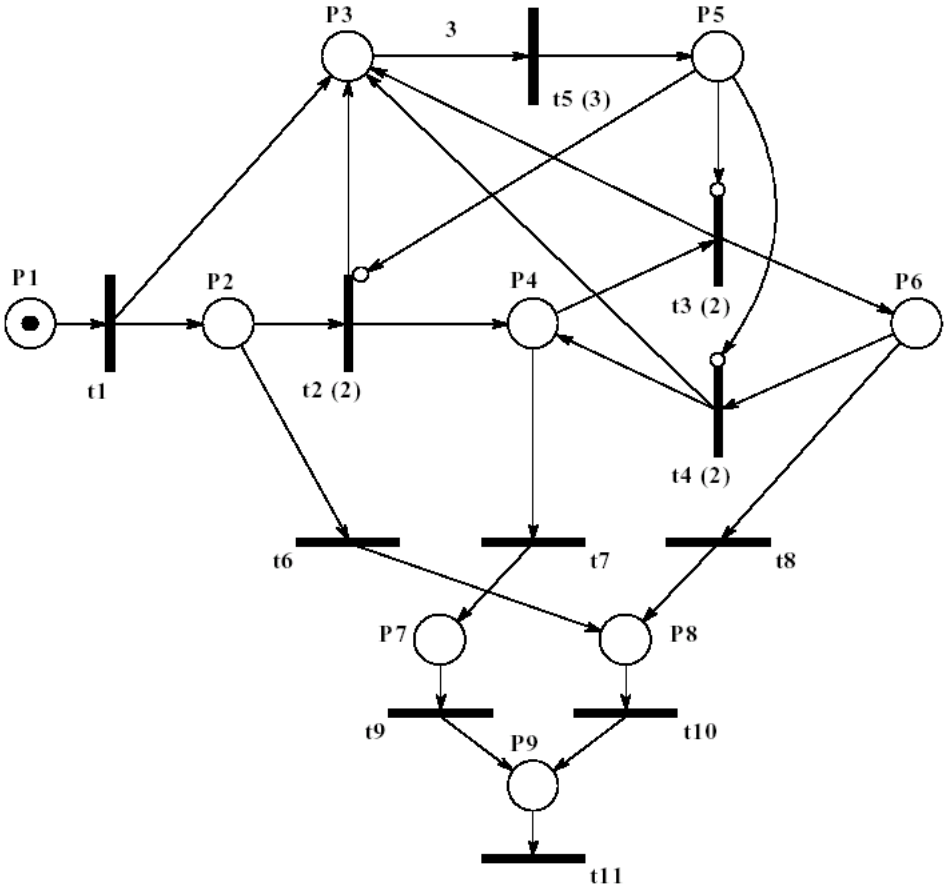


Figura 2.5: O caminhar de uma figura bípede modelado numa rede de Petri. (LIMA BICHO; RAPOSO; MAGALHAES, 2001)

Tabela 2.2: Lugares da rede de Petri da Figura 2.5 que atuam como condições da caminhada

P1	inicia o caminhar
P3	conta os passos dados
P5	número de passos é igual ao desejado
P9	termina o caminhar

Tabela 2.3: Lugares da rede de Petri da Figura 2.5 que atuam como ações

P2,P7	perna esquerda suporta o corpo; perna direita avança; braço esquerdo balança à frente; braço direito balança para trás	meia duração
P8	perna esquerda avança; perna direita suporta o corpo; braço esquerdo balança para trás; braço direito balança à frente	meia duração
P4	perna esquerda avança; perna direita suporta o corpo; braço esquerdo balança para trás; braço direito balança à frente	duração total
P6	perna esquerda suporta o corpo; perna direita avança; braço esquerdo balança à frente; braço direito balança para trás	duração total

Os autores também demonstram que redes de Petri são adequadas para modelagem em um nível de abstração mais alto, como o de animação comportamental. A Figura 2.6 apresenta um modelo de controle comportamental onde dois atores estão jogando bola. O ator com bola executa seu caminhar e continua após arremessar a bola para o outro ator que tem que estar pronto para receber a bola; o ator sem a bola, após executar seu caminhar, só prossegue após receber a bola. A Tabela 2.4.1 contém a descrição dos lugares da rede apresentada.

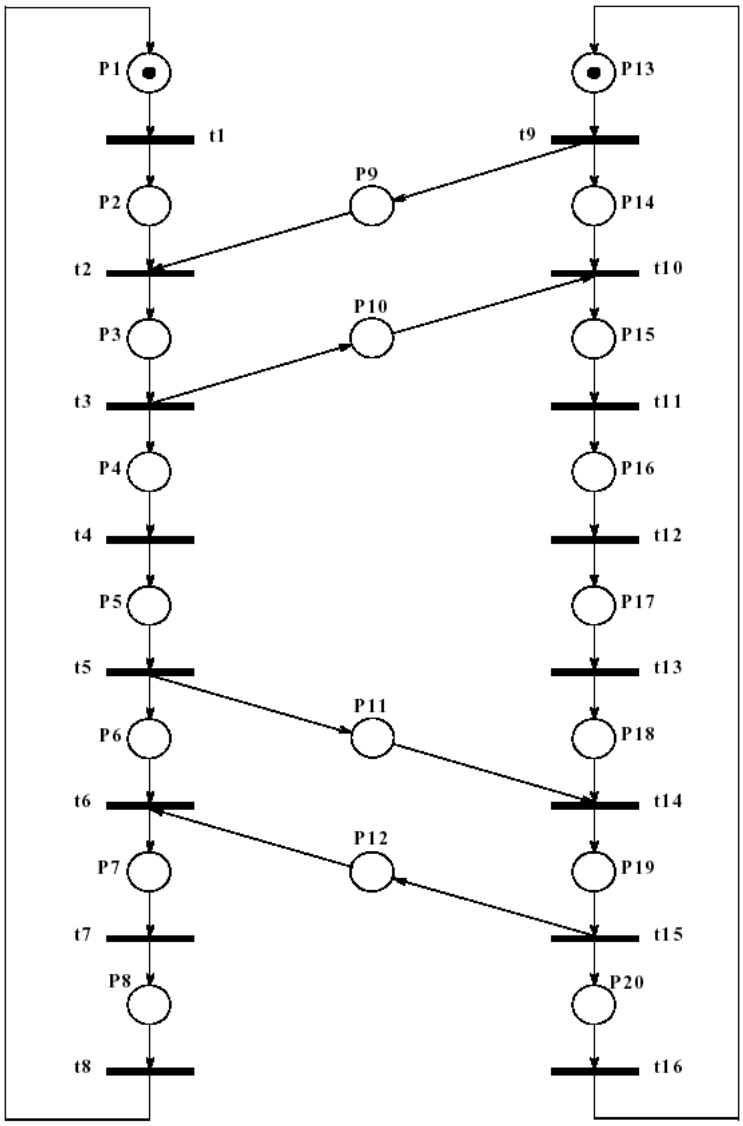


Figura 2.6: Uma rede de Petri modelando dois atores jogando bola (LIMA BICHO; RAPOSO; MAGALHAES, 2001)

Uma vantagem deste modelo é que ele possiblita a centralização do controle da animação e a implementação de um modelo hierárquico de animação pelo fato de suportar diversos níveis de abstração. Uma das principais desvantagens do controle por Redes de Petri é que este modelo sofre do problema da parada.

Tabela 2.4: Lugares da rede de Petri da Figura 2.6 das ações dos atores

Movimentos	Ator A	Ator B
Caminhar segurando a bola	P1	P17
Posicionar para arremesar a bola	P2	P18
Arremessar a bola	P3	P19
Posicionar para caminhar sem a bola	P4	P20
Caminhar sem a bola	P5	P13
Posicionar para pegar a bola	P6	P14
Pegar a bola	P7	P15
Posicionar para caminhar segurando a bola	P8	P16
Ator está pronto para pegar a bola	P11*	P9*
Trajetória da bola (Ator A → Ator B)	P12	
Trajetória da bola (Ator B → Ator A)	P10	

*lugares temporizados, tendo o mesmo tempo de P14 e P6, respectivamente

2.4.2 PaT-Nets

Badler introduziu o conceito de PaT-Nets (*Parallel Transition Networks*) onde há uma memória global para representar o estado interno da rede inteira e cada máquina possui sua memória local. As máquinas trocam mensagens que trabalham com o estado interno da rede e com suas memórias locais. Por possuirem memória, PaT-Nets podem receber parâmetros durante sua instanciação.

Os nós são associados a processos, que podem invocar comportamentos, instanciar outras PaT-Nets e criar planejadores especializados. As transições podem verificar condições locais ou globais, podendo inclusive ser priorizadas e/ou probabilísticas.

Badler, Webber e Reich (BADLER; REICH; WEBBER, 1997, p.9) mostram um exemplo de um controlador que simula a brincadeira de esconde-esconde entre dois agentes: *Hider* (quem se esconde) e *Seeker* (quem procura). O controlador está desenhado na Figura 2.7. O nó *Sync* sincroniza o agente *Hider* forçando-o a esperar até que *Seeker* comece a contar, momento em que *Hider* pode se esconder (nós *Count* e *Hide*). *Hider* espera escondido até que ele seja pego ou perceba que foi visto, momento então que ele tenta fugir de *Seeker* para voltar para a casa. *Seeker* procura por *Hider* até avistá-lo e o persegue até pegá-lo ou até ele voltar para a casa. As setas cheias da Figura 2.7 correspondem a eventos e as pontilhadas corresponde a troca de papel.

Uma das principais vantagens desde modelo é que ele lida com o problema da sincronização entre os atores. Não há proposta de especificação de alocação de recursos nem de meios para controle concreto dos atores. Na prática, o implementador tem que delegar essas tarefas para uma camada de nível inferior.

2.4.3 Sistemas de Transições Paralelas e Hierárquicas - HPTS

HPTS (*Hierarchical Parallel Transition Systems*), proposto por Donikian (DONIKIAN; RUTTEN, 1995) (DONIKIAN, 2001), é um sistema reativo que pode ser visto como um sistema multiagente onde os agentes estão organizados como uma hierarquia de máquinas de estado. Cada agente do sistema é uma caixa preta que suporta entrada e saída e parâmetros de controle. Sub-agentes são organizados em sub-máquinas de estado para o gerenciamento de comportamentos em paralelo do agente.

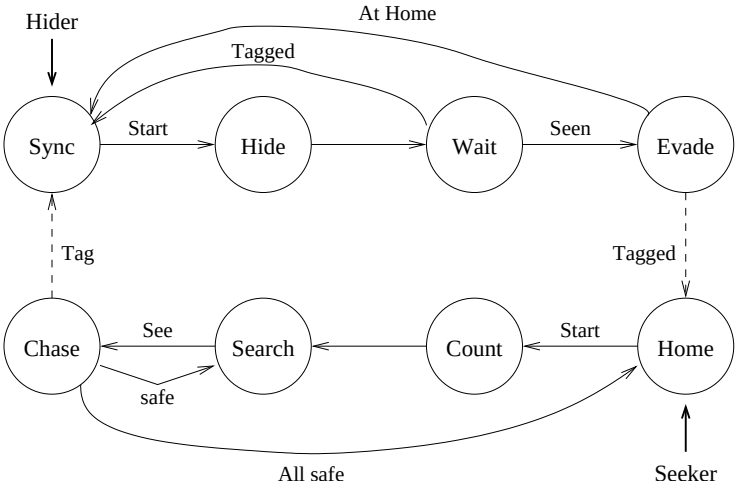


Figura 2.7: PaT-Net que controla esconde-esconde (BADLER; REICH; WEBBER, 1997, p.10)

A tupla $< S, \Gamma, IS, OS, CP, LV, IF, MB >$ define cada máquina de estado do sistema. S é um conjunto de sub-máquinas de estado; Γ é uma função de controle de atividade; IS é um conjunto de sinais de entrada; OS é um conjunto de sinais de saída; CP é um conjunto de parâmetros de controle; LV é um conjunto de variáveis locais; IF é uma função de integração para o processamento dos sinais de entrada e saída e MB é uma caixa de correio para a comunicação entre as máquinas de estado (DONIKIAN, 2001).

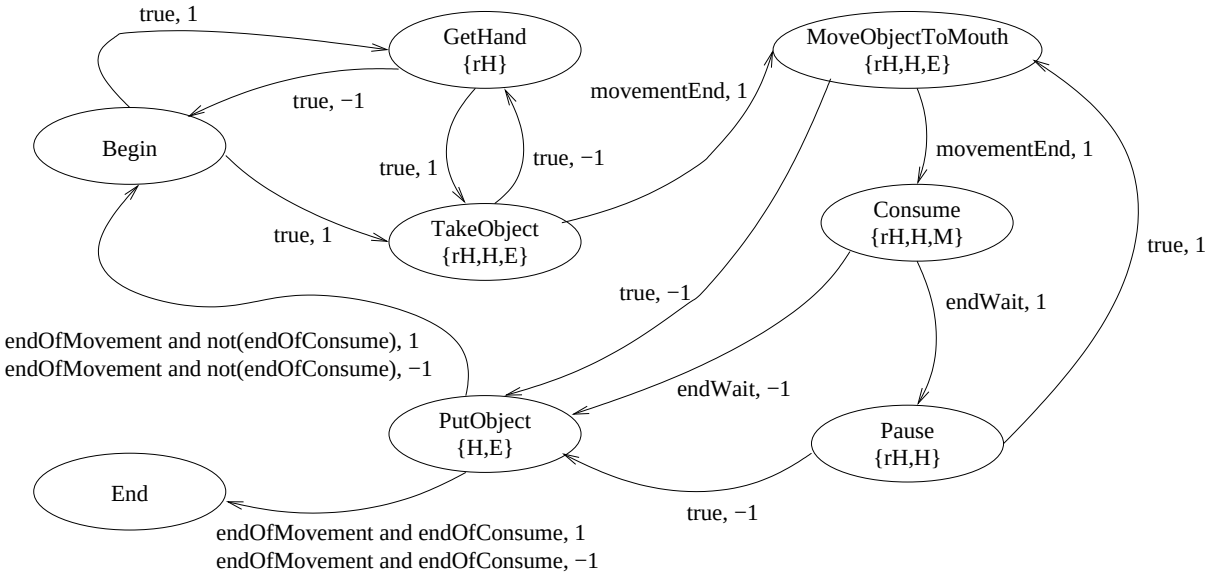
O exemplo apresentado na Figura 2.8 contém três máquinas de estado que descrevem um comportamento de um ator capaz de beber, fumar e ler um jornal simultaneamente. Os símbolos entre chaves dentro dos estados indicam os recursos reservados pela máquina ao entrar em cada estado a seguir: M (*mouth*) para a boca do ator, E (*eyes*) seus olhos, Hl (*left hand*) para a mão esquerda, Hr (*right hand*) para a mão direita e H para qualquer mão disponível. rHl/rHr/rH (*reserve left/right hand*) são usados para a liberação das mãos ao largar um objeto (LAMARCHE; DONIKIAN, 2001). As transições possuem uma condição a ser satisfeita e um número real no intervalo [-1,1] que consiste num grau de preferência para determinar a ordem de escolha das transições para apenas uma transição ser disparada a cada instante, visto que as máquinas são determinísticas.

As características deste modelo flexibilizam o projeto de animações genéricas onde o número de regiões críticas é significativo, visto que o controle de alocação de recursos conflitantes é explícito ao entrar em cada estado. Isto garante que as ações só sejam efetuadas mediante a disponibilização dos recursos. O modelo baseado em PaT-Nets, da seção anterior, somente se preocupa com a sincronização dos atores, sem lidar explicitamente com a reserva e alocação de recursos.

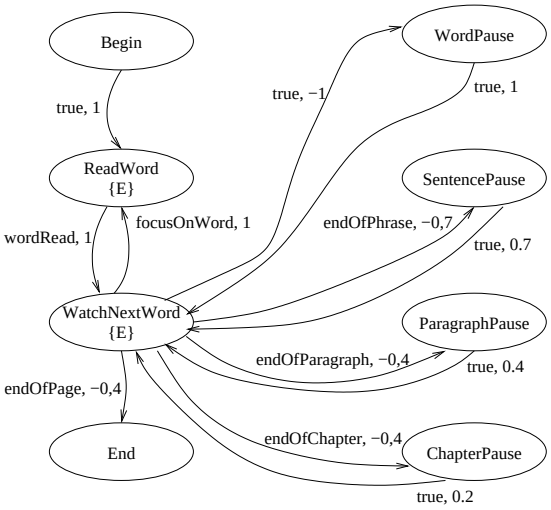
2.5 Sistemas de Transições Rotuladas

Magee *et al.* apresentam um modelo para animação comportamental que combina Autômatos Temporizados e o conceito de Sistemas de Transições Rotuladas (LTS - *Labeled Transition Systems*) para modelar os comportamentos dos atores (MAGEE et al., 2000).

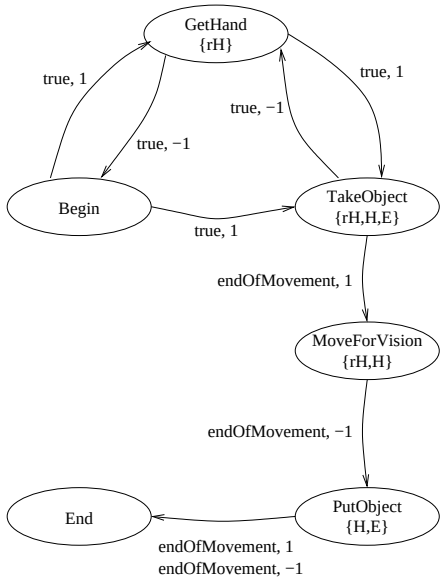
LTS são compostos de autômatos não-determinísticos que possuem um conjunto de rótulos em cada transição. O modelo é abstrato e cada rótulo deve ser mapeado em coman-



(a) Máquina de estado que controla a coleta, uso e troca dos objetos



(b) Máquina de estado que controla a leitura de um objeto



(c) Máquina de estado que controla a coleta de um objeto

Figura 2.8: HPTS - comportamentos para fumar, beber e ler ao mesmo tempo (LAMARCHE; DONIKIAN, 2001)

dos e em condições que devem ser satisfeitas para que a transição correspondente esteja habilitada. A combinação com Autômatos Temporizados ocorre na hora de associar os rótulos a comandos de controle e em condições de teste de relógios, como mostrado na Figura 2.9. Uma das vantagens desta abordagem é que a modelagem é independente da especificação da animação, permitindo que uma animação se aplique a diferentes modelos e que em um modelo sejam aplicadas diferentes animações. Entretanto, a implementação deste modelo pode demandar muitos recursos porque autômatos não-determinísticos complexos exigem muita memória para armazenar as possíveis trocas de estado, visto que uma máquina não-determinística possui a capacidade de se replicar. O Anexo A.2 apresenta o modelo de autômatos não-determinísticos.

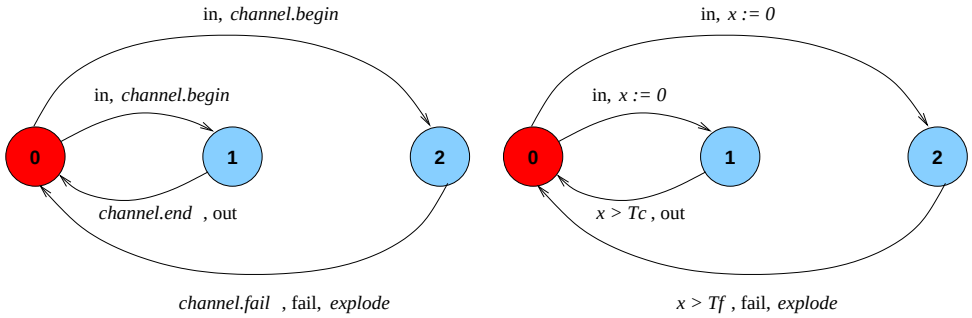


Figura 2.9: Mapeamento de rótulos de um LTS para comandos de temporização em um modelo que descreve um canal de comunicação. Se Tf for maior do que Tc, o canal vai emitir falha. (MAGEE et al., 2000)

2.6 Animação de objetos usando scripts

Esta seção apresenta linguagens de sistemas que tem como preocupação principal a modelagem de objetos animados, sem levar em conta a interação entre esses. O foco é apenas o objeto e seus movimentos. Seção 2.6.1 mostra uma linguagem de baixo nível, que define e anima objetos agregados, suportando interação do usuário; Seção 2.6.2 descreve a linguagem GRAMPS, uma linguagem poderosa de especificação de objetos hierárquicos, suportando a definição de macros e a interação do usuário por meio de dispositivos, inspirando até hoje as linguagens atuais; e a Seção 2.6.3 apresenta uma linguagem de alto nível, baseada em orientação a objeto que trata movimento e objetos animados da mesma maneira, com foco principal na sincronização entre objetos, através de operadores temporais.

2.6.1 Twixt

Gómez (GÓMEZ, 1984) apresenta um sistema que trabalha com hierarquia de objetos e tem a animação controlada por listas de eventos armazenadas em trilhas (*tracks*), as quais podem sofrer interpolação por funções do usuário. A intenção do autor é evitar o custo da criação de quadros-chave. É especificado nas trilhas o que cada objeto está fazendo em vários instantes de tempo, permitindo ao animador gerar os quadros intermediários. A unidade de tempo é o quadro.

Cada objeto tem parâmetros de visualização e uma lista de eventos (*track*) é gerada para lidar com as mudanças dos valores destes parâmetros. O conjunto das *tracks* é nomeado de *script*. Os objetos a serem animados são instâncias de dados previamente definidos pelo animador junto com parâmetros de visualização (exs.: cor, posição e orientação). Existe um tipo de objeto especial (*definer*) que serve apenas para agregar definições de um modelo. Outro objeto especial é nomeado *blender*, que agrega vários *definers* para exercer uma animação. Os objetos podem ser anexados em outros, formando uma hierarquia, como dito no parágrafo anterior. Uma cena completa é representada por uma floresta de objetos hierárquicos.

A Figura 2.10 mostra a implementação de uma animação de uma bola que salta girando. Percebe-se que a linguagem suporta comandos interativos (*dial* e *pencil*) que permitem ao usuário ajustar parâmetros de visualização de um objeto e obter uma prévia da animação, respectivamente. Pode se notar a maneira não-convencional de acessar os campos do objeto *ball*, referindo-se aos subcampos por concatenação. Como pode ser visto no exemplo da figura, o eixo de rotação do objeto *ball* está em *ball.r* e o componente y

deste vetor está em *ball.ry*.

```
call /pic/data/ball     # obtém dados da instância
                        # ** primeiro, vamos trabalhar
                        # na rotação
rot ball 0 y            # seta rotação inicial
event 1 ball.ry         # registra evento
rot ball 720 y          # seta rotação final
event 96 ball.ry        # registra evento
pencil 1 96             # desenha a animação para conferir
                        # como ela está
slide ball.ry.96 72     # reduz a rotação em um segundo
pencil 1 72             # confere de novo a animação
                        # ** vamos trabalhar no salto
event 30 ball.p         # captura posição inicial
event 78 ball.p         # que é também a posição final
place ball at 0 2 0     # posiciona no topo do salto
dial ball               # ajusta
event 54 ball.p         # registra a posição do topo
pencil 1 78             # confere mais uma vez a animação
twerp ball.p para       # usa interpolação parabólica
pencil                  # confere mais uma vez a animação
script jumping          # salva o script
```

Figura 2.10: Twixt - Exemplo de script de animação de uma bola saltitante (GÓMEZ, 1984)

A linguagem é adequada para descrever animações de objetos individuais e agrupados com facilidade e precisão. Apesar da linguagem suportar hierarquias de objetos, ela não possui primitivas para a interação nem sincronização entre os mesmos.

2.6.2 GRAMPS

A linguagem script interpretada pelo sistema GRAMPS (*GRAphics for the Multi-Picture System*) (O'DONNELL; OLSON, 1981) trabalha com construção e animação de cenas com objetos hierárquicos. Ela suporta a criação de novos comandos gráficos por meio de um sistema de macros. Os comandos seguem a seguinte sintaxe:
VERBO [PALAVRAS-CHAVE] [ARGUMENTOS].

A Figura 2.11 mostra um exemplo de uma animação simples de um cubo. O sistema suporta transformações usando valores oriundos de dispositivos de entrada (na figura citada, o argumento D6 do comando TRANSLATE refere-se ao dispositivo de entrada *DIAL 6*, com suas próprias restrições e especificação) ou por valores absolutos, podendo especificar um número de passos (no formato *escalar:passos*) para fazer animação usando interpolação.

```
GET CUBE
ROTATE CUBE X 16384
SCALE CUBE XYZ .+2000
GET SQUARE
GROUP CUBE SQUARE FIGURES
TRANSLATE FIGURES XY -20000<D6<20000
FREEZE/GROUP FIGURES
```

Figura 2.11: Cubo animado na linguagem GRAMPS (O'DONNELL; OLSON, 1981)

O sistema de macros possibilita ao usuário criar procedimentos ou *templates* de comandos gráficos porque os valores dos argumentos são colocados no código diretamente,

sem avaliação prévia.[1] Percebe-se que há comandos interativos, como RESTART e SET e existem comandos que criam novos tipos de dados. O comando GROUP gria um grupo de objetos, o comando FRAME cria um objeto animado e o comando SYNONYM cria um objeto do tipo *synonym*, para capturar dados de outro objeto.

A Figura 2.12 mostra um trecho de uma macro para auxílio na geração de uma animação de uma partícula de um vírus que possui estrutura simétrica em forma de icosaedro. Esta macro trabalha bastante com o tipo de dados *synonym* para gerar as simetrias do objeto. Qualquer manipulação no objeto W é replicada em todos os objetos derivados.

```
(the subunit motif)
GET W
(pick symmetry axes)
PICKAXIS 1 0, 14562, 23561, 0, 0, 0       (5-fold)
PICKAXIS 2 0,     0, 23561, 0, 0, 0       (2-fold)
PICKAXIS 3 0, 23561,     0, 0, 0, 0       (2-fold)
PICKAXIS 4 23561, 23561, 23561, 0, 0, 0   (3-fold)
SYNONYM W W1
GROUP W1 W1 G1W1
SYNONYM W W2
GROUP W2 W2 G1W2
   ...
SYNONYM W W59
GROUP W59 W59 G1W59
USEAXIS 1 G1W1
ROT G1W1 R 13106    (rotate by 5-fold)
FIX G1W1
  ...
USEAXIS 1 G1W44
ROT G1W44 R -13106    (rotate by 5-fold)
FIX G1W44
USEAXIS 2 G1W44
ROT G1W44 R 32767   (rotate by 1st 2-fold)
FIX G1W44
USEAXIS 3 G1W44
ROT G1W44 R 32767   (rotate by 2nd 2-fold)
FIX G1W44
USEAXIS 4 G1W44
ROT G1W44 R 21844   (rotate by 3-fold)
FIX G1W44
  ...
FIX G1W59
```

Figura 2.12: Trecho de macro do GRAMPS para gerar um icosaedro

Há cinco tipos de dados suportados pela linguagem: *simple object* é o tipo básico de dados gráficos que consiste num conjunto de matrizes de transformação para rotação, escala e translação, e nas coordenadas cartesianas dos vértices da figura; *WORLD* é um tipo de dados que contém somente matrizes de transformação para rotação, escala, translação, perspectiva e translação pós-perspectiva;[2] o tipo de dados *GROUP* contém matrizes de transformação que se aplicam entre os níveis da hierarquia de objetos agregados e *framed object* é um *simple object* com uma lista de coordenadas para realizar uma animação. Outro tipo de dados baseado no *simple object* chama-se *synonym*, que possui um apontador para outro objeto para replicar o seu comportamento.

A Figura 2.13 mostra o diagrama em árvore de objetos agrupados para animar um humano virtual. A Figura 2.14 mostra uma macro que efetua um passo de uma caminhada. O passo é controlado pelo dispositivo *DIAL 4*.

[1] Isto é chamado de macrossubstituição.

[2] O sistema de animação suporta apenas dois objetos do tipo *WORLD*: LWORLD e RWORLD, que são usados para visualização em estéreo.

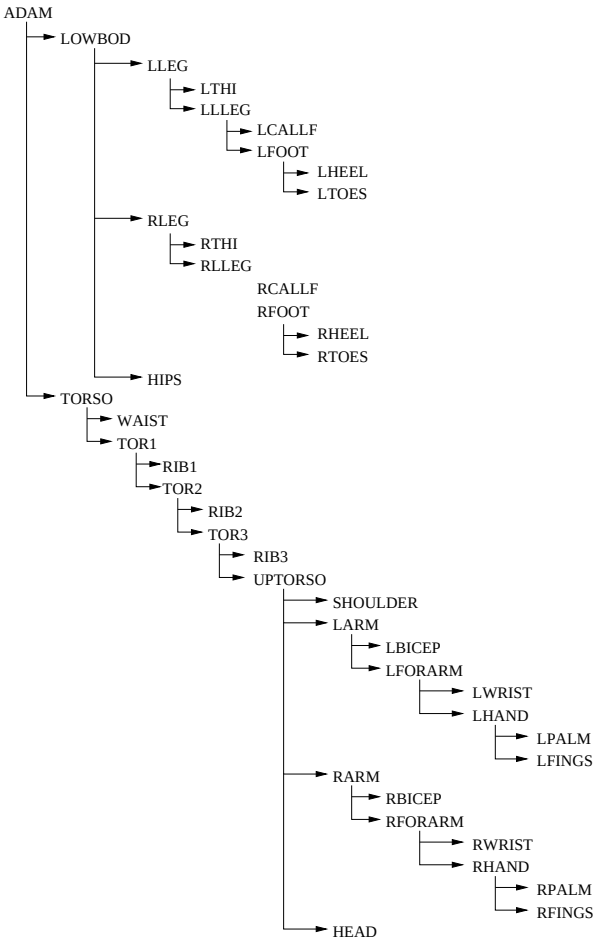


Figura 2.13: Diagrama hierárquico dos objetos de um humano virtual do GRAMPS (O'DONNELL; OLSON, 1981)

```
ROTATE TORSO Z -1500<D4<1500
ROTATE TOR1 Z -1500<D4<1500
ROTATE TOR2 Z -1500<D4<1500
ROTATE TOR3 Z -1500<D4<1500
ROTATE SHOULDER Z -1500<D4<1500
ROTATE LOWBOD Z -1500<D4<1500
ROTATE RLEG Z -1500<D4<1500
ROTATE RLEG X 0<D4<1500
ROTATE RLLEG X -2000<-D4<0
ROTATE LLEG Z -1500<D4<1500
ROTATE LLEG X 0<-D4<1500
ROTATE LLLEG X -2000<D4<0
ROTATE RFOOT X 14384<-D4<16384
ROTATE RTOES X 0<D4<1500
ROTATE LFOOT X 14384<D4<16384
ROTATE LTOES X 0<-D4<1500
```

Figura 2.14: Macro que efetua um movimento de perna do humano virtual do GRAMPS através do dispositivo *DIAL 4* (O'DONNELL; OLSON, 1981)

A linguagem GRAMPS carece de primitivas para maior nível de controle. Um sistema de controle de fluxo e de indexação de objetos poderia tornar o exemplo da Figura 2.12 mais enxuto. O exemplo da Figura 2.14 demonstra que o modelo hierárquico de objetos é apropriado para animações baseadas em cinemática direta.

2.6.3 Linguagem script temporal

Fiume *et al.* (FIUME; TSICHRITZIS; DAMI, 1987) descrevem uma linguagem que trata movimento e objetos animados de modo uniforme. Um objeto animado, no contexto dos autores, é uma entidade ativa que conhece sua duração em unidades abstratas de tempo chamadas *ticks* (pulsos) e conhece seu comportamento a cada pulso. Estes objetos são definidos em um modelo de tempo contínuo e podem trocar mensagens sobre seu comportamento.

Os objetos recebem mensagens que são definidas por um conjunto de comandos gráficos acompanhados de um atributo que determina a duração dos comandos, que pode ser infinita. A Figura 2.15, extraída do artigo, mostra a definição das mensagens. Operadores temporais são usados para a composição de animações entre os objetos animados existentes, incluindo sincronização. O operador de sequenciamento cronológico (*Chronological Sequencing*) define ordem entre duas expressões. O comando $E_1; E_2$ expressa que E_1 é seguido por E_2. O operador de ativação simultânea (*Simultaneous Activation*) no comando $E_1 \& E_2$ indica que E_1 e E_2 devem iniciar ao mesmo tempo. O operador de encerramento simultâneo (*Simultaneous Termination*) no comando $E_1 \wr E_2$ indica que E_1 e E_2 terminam ao mesmo tempo.

$$O.duration \in \mathbf{R} \cup \{\infty\}$$

$$O.t = \begin{cases} \{\text{graphics commands for time } t\} & \text{if } t \in [0, O.duration) \\ \emptyset & \text{otherwise} \end{cases}$$

Figura 2.15: atributos das mensagens da linguagem temporal

O operador de atraso (*Delayed Activation*) cria um objeto que indica uma pausa na animação ou atrasa um objeto em relação a outro. Como exemplos, temos que a expressão $A \& (\textbf{delay}\ 5; B)$ indica que B será ativado em cinco pulsos após A, e a expressão $A \wr (B; \textbf{delay}\ 5)$ indica que B terminará cinco pulsos antes de A.

O operador de sincronização (*General Synchronisation Operator*) é o operador mais importante da linguagem porque, de uma maneira geral, engloba todos os operadores apresentados acima. A expressão $A[t_a] \asymp B[t_b]$ sincroniza a noção de tempo t_a de A com a t_b de B. Como exemplo, a expressão $A[5000] \asymp B[200]$ indica que A iniciará 4800 pulsos antes de B, se $B.duration$ for menor do que 200 pulsos. A Figura 2.16 mostra a definição formal do operador. O símbolo \$ é usado para referenciar a duração da expressão à esquerda dos colchetes. O exemplo $(E_1[\$/2] \asymp E_2[\$/2])[\$/4] \asymp E_3[0]$ faz com que as animações E_1 e E_2 coincidam na metade e que E_3 iniciará ao passar um quarto da animação coincidente.

O operador de repetição (*repeat*) serve para replicar uma animação (**repeat** $3\ E_1 = E_1; E_1; E_1$, por exemplo). O operador para execução assíncrona (*Asynchronous Execution*) indica um tempo absoluto para iniciar uma animação. O comando **at** $500\ E_1$, por exemplo, fará E_1 iniciar após ter passado 500 pulsos desde o início do script.

$$(E_1[t_1] \asymp E_2[t_2]).duration = \max(E_1.duration, E_2.duration, E_1.duration + t_2 - t_1, E_2.duration + t_1 - t_2)$$

$$(E_1[t_1] \asymp E_2[t_2]).t = \begin{cases} E_2.t \cup E_1.(t - \Delta t) & \text{se } t_1 \leq t_2 \\ E_1.t \cup E_2.(t - \Delta t) & \text{senão} \end{cases}$$

operador comutativo

Figura 2.16: operador temporal - sincronização

O operador de escala temporal (*Temporal Scaling*) permite coordenar movimento, redefinindo a duração de uma animação. A Figura 2.17 demonstra o operador e as mudanças em proporção que ele provoca no comportamento do objeto.

$$E_1\langle n\rangle.d = n$$
$$E_1\langle n\rangle.t = E_1 * \left(\frac{tn}{E_1 * d}\right)$$

Figura 2.17: operador temporal - Temporal Scaling

O operador de não determinismo (*Nondeterminism*), mostrado na expressão $E_1|E_2$, faz com que apenas uma das duas animações seja executada. O operador de sobreposição (*Temporal Overlap*) determina que duas animações devem compartilhar um espaço de tempo, que é determinado de forma não-determinística, dando um comportamento aleatório à animação (ex.: $E_1 \star E_2$).

Os objetos animados são instanciados a partir de protótipos oriundos de bibliotecas. A sintaxe de instanciamento é descrita da seguinte forma:
$InstanceName : ObjectPrototype(parameters)\{DynamicsScript\}$.

Existe um tipo especial de objeto animado chamado objeto de movimento (*motion object*) que não possui primitivas gráficas e é usado para a composição de animações. A Figura 2.18 mostra um exemplo que descreve uma animação de três xícaras e um bule. Neste exemplo, pode se ver que o objeto *rot* é um objeto de movimento que efetua uma rotação de 180 graus no sentido horário sobre o eixo Y. *cup1*, *cup2* e *cup3* são instâncias da primitiva **Cup**, tendo posições iniciais e dinâmicas diferentes. O objeto *cup1* efetua duas vezes uma rotação de 90 graus sobre o eixo Z e esta animação está restrita a rodar em 30 pulsos. O objeto *rot* está agindo em paralelo, com uma duração de 30 pulsos também. O objeto *cup2* gira 180 graus em torno do eixo Z em 30 pulsos, em paralelo com uma rotação sobre o eixo Y em 60 pulsos. *cup3* rotaciona 360 graus em torno do eixo Z em 30 pulsos, em paralelo com duas rotações sobre o eixo Y, restritas a 30 pulsos também. O objeto *tea* gira 3 vezes 360 graus em torno do eixo Z, levando 30 pulsos em cada rotação. O script indica que *cup1*, *cup2* e *cup3* iniciam simultaneamente e terminam junto com *tea*. *tea* vai começar antes porque sua animação é maior, então *cup2* vai iniciar no pulso 30 e os cups restantes iniciarão no pulso 60.

Os operadores temporais são a parte mais interessante da linguagem proposta por Fiume *et al.* porque resolvem o problema da sincronização temporal entre as animações dos atores. Entretanto, a linguagem carece de primitivas de alto nível, sendo pouco intuitiva ao projetista de animação e de operadores que providenciassem sincronização

rot : **RotateY**(-180)
$cup1$: **Cup**$(\text{“}origin = 5, 0, 0'')\{$(**repeat** 2 **RotateZ**$(90))\langle 30\rangle \& rot\langle 30\rangle\}$
$cup2$: **Cup**$(\text{“}origin = 3, 0, 0'')\{$**RotateZ**$(180)\langle 30\rangle \& rot\langle 60\rangle\}$
$cup3$: **Cup**$(\text{“}origin = 1, 0, 0'')\{$**RotateZ**$(360)\langle 30\rangle \&($**repeat** 2 $rot)\langle 30\rangle\}$
tea : **Teapot**$\{$**repeat** 3 (**RotateZ**$(360))\langle 30\rangle\}$
$(cup1 \& cup2 \& cup3) \wr tea;$

Figura 2.18: Exemplo de script de animação da linguagem temporal

levando em conta as coordenadas e outros parâmetros dos objetos.

2.7 Animação comportamental e por tarefas

Aqui são apresentadas linguagens em um nível de abstração superior que trabalham com a especificação de tarefas e com modelos de comportamento. Seção 2.7.1 contém um protótipo de linguagem para um sistema de animação orientada a tarefas, abstraindo ao máximo os detalhes para se aproximar da linguagem natural; a linguagem NEM, descrita na Seção 2.7.2 especifica tanto a definição de objetos quanto a relação entre eles, sendo uma espécie de linguagem híbrida; Seção 2.7.3 demonstra uma linguagem derivada da lógica de primeira ordem, para especificar planos de ação seguindo a lógica BDI (*Beliefs, Desires and Intentions* - Crenças, Desejos e Intenções); e a Seção 2.7.4 fala sobre a linguagem STEP, que trabalha com atos comunicativos de agentes encorpados.

2.7.1 Linguagem para animação orientada a tarefas

Esakov *et al.* (ESAKOV; BADLER; JUNG, 1989) apresentam um protótipo de um sistema de animação orientada a tarefas, usando uma linguagem natural como interface para o gerador de movimentos dos atores. A linguagem especifica comandos para os atores visualizarem e alcançarem objetos no cenário.

A Figura 2.19 ilustra um exemplo de um script escrito na linguagem proposta. É bom notar que o nível de abstração da linguagem concentra-se apenas na especificação do sexo e do tamanho médio dos atores (em relação a uma população previamente descrita) e da descrição de tarefas, não especificando posições nem estados dos objetos da cena, que devem estar previamente descritos numa base de conhecimento e numa base geométrica, usadas pelo tradutor. A Figura 2.20 mostra um exemplo da tradução da linguagem para o gerador de movimentos. O tempo das ações poderia ser especificado na linguagem, mas os atores defendem a idéia de que não é comum especificar tempo em linguagens de descrição de tarefas. A idéia proposta é calcular o tempo das ações a partir de dados extraídos das bases de conhecimento e geométrica, como o sexo do ator que efetuará a ação, a distância a ser percorrida até o alvo da ação e o tamanho do alvo.

2.7.2 Linguagem NEM

A linguagem NEM (MARINO; MORASSO; ZACCARIA, 1985) foi projetada para definir modelos de entidades (humanóides e objetos) e para descrever movimentos de grupos ou partes de entidades. A idéia é poder planejar as ações da cena e também efetuar animações em baixo nível. As entidades são representadas em estruturas de dados chamadas GFS (*Geometric Frame Structures*), que providencia um modelo hierárquico para

```
John is a 50 percent man.
Jane is a 50 percent woman.
John, look at switch twf-1.
John, turn twf-1 to state 4.
Jane, look at twf-3.
Jane, look at tglJ-1.
Jane, turn tglJ-1 on.
John, look at tglJ-2.
Jane, look at twf-2.
Jane, turn twf-2 to state 1.
John, look at twf-2.
John, look at S.
Jane, look at J.
```

Figura 2.19: Exemplo de especificação de tarefas em linguagem natural

```
John, look at switch twf-1
    point_at(‘‘ctrlpanel.panel.twf_1’’,’’john.bottom_head.between_eyes’’,(1,0,0));
John, turn twf-1 to state-4
    reach_site(‘‘ctrlpanel.panel.twf_1’’,’’john.right_hand.fingers_distal’’);
Jane, look at tglJ-1
    point_at(‘‘ctrlpanel.panel.twj_1’’,’’jane.bottom_head.between_eyes’’,(1,0,0));
John, turn tglJ-1 on
    reach_site(‘‘ctrlpanel.panel.twj_1’’,’’jane.left_hand.fingers_distal’’);
```

Figura 2.20: Exemplos de interpretação de uma linguagem natural

a montagem de esqueletos.

A descrição dos movimentos é abstrata e pode se estender tanto a uma entidade quanto a uma ou mais partes individuais, a grupos e a generalizações de entidades. Pode se referir aos objetos de forma geométrica como mostrado na Figura 2.21 e é possível se valer de uma base de conhecimento comum suportando conceitos nativos como gravidade, equilíbrio e colisão.

```
estática - ‘‘is the book over the chair’’
dinâmica - ‘‘may john reach the book without walking’’
```

Figura 2.21: Maneiras de se referir aos objetos da linguagem NEM

As entidades são geradas a partir de uma álgebra de referência hierárquica (*Frame Algebra Notation* ou FAN). A Figura 2.22 mostra a definição de uma pirâmide e algumas operações sobre ela. O operador @ serve para definir em qual quadro as definições ocorrem (a palavra-chave ENV refere-se a quadro do ’universo’ ou quadro global). A Figura 2.23 contém uma GFS que define a hierarquia de um humanóide. Os prefixos ’L’ e ’R’ servem para acessar elementos diferentes dentro da mesma estrutura e a palavra-chave LIMB define uma cadeia de quadros.

```
(definição do quadro PYRAMID)
@ ENV FRAME PYRAMID                 (ENV é o quadro do ’universo’)
@ PYRAMID FRAME VERTEX_1 VERTEX_2 VERTEX_3
(inicializa rotação e translação)
[VERTEX_1] = [-,(0,10,0)]           (’-’ é a identidade)
[VERTEX_1] *= [F]                   (aplica uma matriz de transformação)
```

Figura 2.22: Uma pirâmide definida pela FAN da linguagem NEM e exemplos de inicialização

O comando DEFINE serve para definir funções e relações. É possivel criar definições paramétricas de GFS para serem instanciadas depois. Os tipos de dados suportados são listas, escalares, vetores, arrays e eventos. Os eventos serão apresentados junto com as descrições de movimento, na seguinte seção.

Um movimento consiste em variáveis, quadros, primitivas e instâncias de outros movimentos. Há tanto primitivas para trabalhar com cinemática direta quanto com a inversa. As primitivas de cinemática direta são **bend**, **fold** e **rotate** que utilizam eixos de Euler; **turn** para rotacionar em torno de um eixo qualquer e **screw** que executa um movimento semelhante a girar um braço em torno de seu próprio eixo (como um parafuso). As primitivas de cinemática inversa são: **move** que muda a posição de um *end effector*; **fix** e **unfix**, que 'tranca' e 'destranca', respectivamente, a relação do *end effector* a um quadro qualquer; e **emov**, que movimenta uma cadeia de quadros, considerando o *end effector* seu ancestral (atuando como o inverso da primitiva **move**).

```
@ ENV FRAME HUMANOID
@ HUMANOID LIMB PELVIS THORAX NECK
@ PELVIS LIMB 'L' ILIUM HIP KNEE ANKLE FOOT
@ PELVIS LIMB 'R' ILIUM HIP KNEE ANKLE FOOT
@ THORAX LIMB 'L' SCAPULA SHOULDER ELBOW WRIST HAND
@ THORAX LIMB 'R' SCAPULA SHOULDER ELBOW WRIST HAND
@ NECK FRAME HEAD
@ NECK FRAME MOUTH NOSE FOREHEAD
@ HEAD FRAME 'L' EYE EAR
@ HEAD FRAME 'R' EYE EAR
```

Figura 2.23: Humanóide definido pela FAN da linguagem NEM

```
MOTION FUNNY_PYRAMID(TRIPLE ROT_STEP)  /* TRIPLE = vetor */
EXTERN EVENT STOP
@ ENV FRAME PYRAMID                 /* define PYRAMID no quadro global */
{
  PYRAMID( <parameters> );        /* instancia a partir de um GFS paramétrico */
  REPEAT
    PYRAMID *= [(ROT_STEP),-];    /* aplica rotação */
  UNTIL STOP;                     /* gira até o evento disparar */
  THEN {
     T = ACTUALTIME;
     WHILE( ACTUALTIME < (T+10) ) DO {   /* faz por 10 segundos */
        BEND VERTEX_1 & BEND VERTEX_2 & BEND VERTEX_3  /* expande a pirâmide */
     }
  }
}
```

Figura 2.24: Animação de uma pirâmide na linguagem NEM

```
execução sequencial:
   ACTION_1 ; ACTION_2 ; ACTION_3 ; ...
execução em paralelo:
   ACTION_1 & ACTION_2 & ACTION_3 & ...
execução dirigida por um evento:
   (executa enquanto o evento ocorrer)
     WHILE EVENT DO ACTION
   (espera pela ocorrência do evento para executar)
     WAIT EVENT DO ACTION
   (ACTION_1 pode ser interrompida pela ocorrência do evento.
    ACTION_2 é executada caso isso aconteça)
     PERFORM ACTION_1 EXCEPT EVENT THEN ACTION_2
```

Figura 2.25: Operadores de composição de movimento da linguagem NEM

A Figura 2.24 mostra um procedimento que tem a descrição do movimento da pirâmide da Figura 2.22, que vai girar até ser interrompida por um evento externo. Depois disso, ela será ampliada por um tempo. Note que um operador de composição de movimento similar aos do trabalho descrito na Seção 2.7.1 são utilizados. A Figura 2.25 apresenta estes operadores.

2.7.3 AgentSpeak(L)

Rao (RAO, 1996) descreve uma linguagem que trabalha com agentes BDI (*Belief-Desire-Intention* - crença-desejo-intenção), que são sistemas situados num ambiente sujeito a mudanças e que executam ações para afetar este ambiente. Os agentes trabalham enfocando as três atitudes mentais primárias: crenças, desejos e intenções, que correspondem aos seus componentes de informação, de motivação e de decisão.

AgentSpeak(L) é uma linguagem projetada para especificar o comportamento de agentes BDI. É baseada em lógica de primeira-ordem, possuindo eventos e ações. É possível observar as crenças do agente, analisando seu estado atual, seu conhecimento de si mesmo e de outros agentes; observar seus desejos, analisando os estados que o agente quer alcançar; e suas intenções, analisando os programas adotados para alcançar seus desejos.

A linguagem consiste numa base de conhecimento/crenças/fatos e um conjunto de planos, que são uma composição hierárquica de metas junto com a execução de ações. Um agente BDI na linguagem é composto por um conjunto de crenças, planos, intenções, eventos, ações e de funções de seleção. Os símbolos usados em adição aos conetivos de primeira-ordem estão mostrados na Figura 2.26.

!	-	alcance(*achievement*)
?	-	teste
;	-	sequenciamento
&	-	conjunção
$\leftarrow$	-	implicação
$+$	-	adição de crenças ou de metas
$-$	-	deleção de crenças ou de metas

Figura 2.26: Símbolos da linguagem AgentSpeak

Metas, no contexto do autor, são desejos adotados por um agente. $!g(t)$ indica uma meta onde o agente quer alcançar um estado onde $g(t)$ é uma crença verdadeira. $?g(t)$ indica um teste de meta onde o agente quer testar se $g(t)$ é uma crença verdadeira.

Pode-se especificar eventos através dos operadores $+$ e $-$. Eles vão disparar quando uma meta é atingida por um agente ou quando o agente percebe mudanças no ambiente. Um plano é composto por um evento, que indica o motivo do plano, e seu contexto, que corresponde a uma sequência de metas e ações. A Figura 2.27 mostra exemplos de planos para um robô recolher lixo.

2.7.4 Linguagem STEP

Agentes gráficos (*embodied agents*) são agentes autônomos que conhecem seu mundo ou ambiente através de sensores e atuam em seu mundo por meio de atuadores. Huang et. al (HUANG; ELIENS; VISSER, 2002) descrevem a linguagem STEP (*Scripting Technology for Embodied Persona*) que é uma linguagem script para a especificação de gestos e posturas de um agente, tendo semântica formal baseada em lógica dinâmica. A linguagem pode ser estendida para outros atos comunicativos, como expressões faciais, conversação

e a outros tipos de agentes, como robôs cognitivos. Baseado em seu trabalho, os autores desenvolveram a linguagem XSTEP (HUANG; ELIENS; VISSER, 2003), uma versão XML para a troca de dados via World Wide Web.

Os autores não se preocupam com as mudanças internas dos estados mentais dos agentes, defendendo a idéia da simplificação da linguagem, ocultando detalhes de geometria, como especificar rotas ou equações de movimento, a fim de se aproximar da linguagem natural. Baseado em uma posição inicial do humanóide, com a figura de frente para o eixo Z e a origem localizada no chão, logo abaixo dos pés, é definido um sistema para referenciar, de modo natural, as direções das partes do corpo, suportando interpolações entre as direções, inclusive. A Figura 2.28 mostra várias combinações de direções. A especificação do agente contém uma hierarquia de articulações e segmentos, que podem conter sítios para colocar adornos (como roupas ou jóias) e para definir pontos de visão. Como exemplo, a ação "gire vagarosamente o braço esquerdo para a frente" é especificada assim: `turn(Agent,l_shoulder,front,slow)`. Os operadores estão na Figura 2.29.

Como um exemplo derivativo, Ruttkay *et al.* (RUTTKAY; HUANG; ELIENS, 2003) projetaram uma estrutura que providencia definições e controle de gestos em alto nível para avatares em mundos VRML. Os gestos podem ser adaptados a circunstâncias particulares e estilos pessoais. Para alcançarem isso, os autores criaram uma biblioteca na linguagem STEP. A Figura 2.30 mostra um exemplo simples de condução manual em dois tempos que cria uma ação baseada em parâmetros de ritmo que são compassados pelas rotinas `perform_beat` e `wait`.

2.8 Conclusão

Os trabalhos referenciados neste capítulo foram apresentados de forma que o leitor possa acompanhar a evolução e o surgimento de tendências de descrições de animação, demonstrando sistemas com aplicações práticas de especificação de objetos animados para controle concreto de animação e para descrever o controle intencional dos atores.

Foram apresentadas linguagens script para a descrição de objetos animados. Pode-se observar a linguagem Twixt como uma predecessora do modelo AGA, descrito no Capítulo 3, fazendo uma analogia das *tracks* com as fitas de entrada dos autômatos finitos, embora os autômatos finitos providenciem recuperação de informação, através da observação instantânea dos estados atuais dos mesmos. A linguagem GRAMPS serve de base para várias linguagens de descrição de objetos animados, embora ela seja de baixo nível,

```
+location(waste,X):
   location(robot,X) & location(bin,Y) <-
       pick(waste);
       !location(robot,Y);
       drop(waste).

+!location(robot,X): location(robot,X) <- true.

+!location(robot,X):
    location(robot,Y) & (not (X = Y)) &
    adjacent(Y,Z) & (not (location(car, Z))) <-
       move(Y,Z);
       +!location(robot,X).
```

Figura 2.27: Exemplos de planos em AgentSpeak

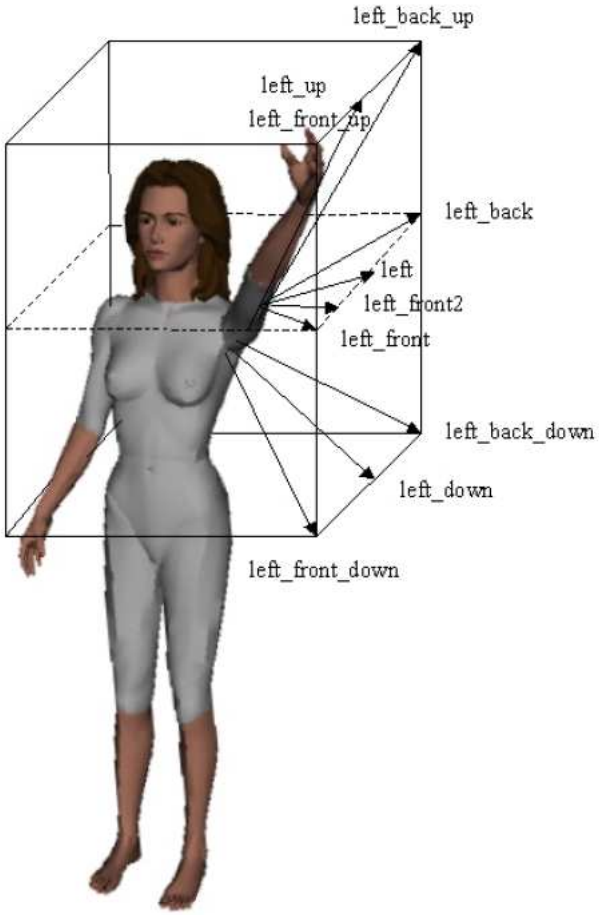


Figura 2.28: STEP - Combinações de direções para o braço esquerdo (HUANG; ELIENS; VISSER, 2002)

`turn(Agent,Parte,Direcao,Duração)`	- especifica uma rotação pelo tempo em uma parte do corpo
`move(Agent,Parte,Direcao,Duração)`	- especifica uma translação
`seq([Ação`$_1$`,...,Ação`$_n$`])`	- gera uma ação que é o resultado das ações em sequência
`par([Ação`$_1$`,...,Ação`$_n$`])`	- gera uma ação que é o resultado das ações em paralelo
`choice([Ação`$_1$`,...,Ação`$_n$`])`	- gera uma ação que é a escolha não determinística de uma das ações
`repeat(Ação,N)`	- gera uma ação que é a `Ação` `N` vezes replicada em sequência
`test(`ϕ`)`	- testa o estado ϕ
`do(`ϕ`)`	- faz estado ϕ ser verdadeiro
`if_then_else(`ϕ`,ação1,ação2)`	- executa `ação1` se ϕ for verdadeiro ou `ação2` em caso contrário
`until(ação,`ϕ`)`	- executa `ação` até que ϕ seja verdadeiro

Figura 2.29: Operadores da linguagem STEP

```
script(conduct_half(Intensity,BeatDuration,DownHold,
       UpHold,DownDynamism,UpDynamism,Hand),Action) :-
   Action = seq([
      get_parameters(beat_down,Intensity,StartHand,EndHand,
         StartWrist,EndWrist,Hand),
      perform_beat(EndHand,EndWand,StrokeDuration,Dynamism,Hand),
      wait(DownHold),
      perform_beat(StartHand,StartWrist,StrokeDuration,Dynamism,Hand),
      wait(UpHold)]),!.
```

Figura 2.30: Exemplo na linguagem STEP de uma condução manual em 2 tempos

com primitivas muito próximas às do hardware gráfico disponível. Depois, é apresentada uma linguagem com operadores temporais que trata movimentos e objetos gráficos da mesma maneira, contendo poderosos operadores para sincronização de animações. Sua especificação em aberto (a descrição dos objetos é apenas derivada de primitivas geométricas) torna ela mais adequada para atuar como uma camada de controle de sincronização de uma animação. A linguagem STEP, apresentada mais adiante, utiliza operadores temporais similares, com o propósito de sincronização de gestos de figuras humanóides, em uma forma próxima à linguagem de primeira ordem.

Quanto às linguagens de especificação de modelos de comportamento para a interação entre atores, o trabalho de Esakov (ESAKOV; BADLER; JUNG, 1989) mostra uma linguagem próxima à linguagem natural para a descrição de tarefas, poupando ao máximo o trabalho do usuário para descrever as animações. O problema é que ela depende muito da base de dados do sistema, que pode se tornar extremamente complexa por ser necessária uma base geométrica com especificação precisa de posicionamento e uma base de conhecimento com descrição de população, pois os atores são criados com base nesta descrição. Uma linguagem mais flexível e híbrida apresentada aqui é a linguagem NEM, que combina definição de modelos de entidades e descrição de movimentos, podendo planejar ações e especificar animações em baixo nível, também tratando movimentos como objetos. Por ser uma linguagem de baixo nível e sendo próxima às linguagens procedurais, como a linguagem GRAMPS, ela apresenta dificuldades na criação de animações complexas, principalmente na sincronização de objetos. AgentSpeak(L) é uma linguagem para qualquer tipo de agente BDI, podendo ser adaptada para o controle em alto nível dos atores de uma animação. A linguagem STEP, por outro lado, trabalha com agentes gráficos, que são agentes que conhecem seu mundo por meio de sensores, se preocupando mais com os gestos e posturas do que com as tarefas.

3 MODELO AGA-J

3.1 Introdução

A modelagem da animação com humanos virtuais é um problema desafiante porque os modelos geométricos e matemáticos da computação gráfica não são adequados para representar a forma do corpo humano. Se necessário, restrições funcionais podem ser aplicadas a eles, a fim de que seus movimentos satisfaçam as limitações humanas. (BADLER et al., 1999)

Este capítulo apresenta o modelo AGA-J, que é o modelo AGA sob uma nova ótica para efetuar animações em 3D de figuras articuladas. O modelo AGA-J gera sequências animadas a partir de um conjunto de AFS que representam os atores da animação e um conjunto de fitas de entrada que determinam os comportamentos dos atores. Cada ator é modelado graficamente por um conjunto de corpos articulados e o alfabeto de saída é representado por conjuntos de transformações nas articulações dos atores. Este modelo foi projetado de uma forma que o mesmo AFS usado na especificação do ator também provê o controle de animação de suas articulações.

A Tabela 3.1 apresenta as mudanças introduzidas pelo modelo AGA-J em relação ao AGA.

Tabela 3.1: AGA e AGA-J

AGA	AGA-J
animações quadro-a-quadro	animações em 3D
imagens estáticas bidimensionais	elementos gráficos em 3D conectados hierarquicamente por articulações, resultando em figuras articuladas em 3D
a saída é produzida unicamente pela interpretação do alfabeto de saída do ator que indica qual imagem exibir	a especificação do ator agrega um modelo de figura articulada; a saída é produzida pela interpretação do alfabeto de saída do ator que indica um conjunto de funções de transformação as quais são aplicadas nas articulações do modelo, gerando uma nova pose
animação por quadros-chave	os quadros são gerados por interpolação entre as poses geradas de acordo com a taxa desejada de quadros por segundo (FPS) na reprodução da animação

A seção seguinte contém uma introdução ao modelo AGA; a Seção 3.3 apresenta a

descrição do modelo AGA-J; a Seção 3.4 contém o modelo formal do AGA-J; a Seção 3.6 demonstra passo a passo o algoritmo do motor de animação; a Seção 3.7 descreve brevemente o conceito de grafo de cena e a Seção 3.8 apresenta detalhes da implementação do modelo.

3.2 Introdução do modelo AGA

É muito comum nos motores de animação os movimentos dos personagens serem controlados através do relógio, onde os momentos em que o comportamento do sistema é alterado são definidos por intervalos discretos de tempo. Accorsi *et al.* (ACCORSI; MENEZES, 2000) apresentaram um modelo para representar animação modelada por computador baseado na teoria dos autômatos finitos (MENEZES, 2000) (HOPCROFT; MOTWANI; ULLMAN, 2001), denominado AGA. Sua primeira versão apresentou um modelo para descrever animações quadro-a-quadro para a Web, onde a saída resultante é uma composição de imagens e efeitos sonoros.

O modelo AGA gera sequências animadas a partir de um conjunto de AFS que representam os atores da animação e um conjunto de fitas de entrada que determinam os comportamentos dos atores. O alfabeto de saída é representado por imagens e efeitos sonoros. Como a animação é quadro-a-quadro, a animação se dá pela sequência das saídas dos autômatos.

Durante a animação, o comportamento dinâmico do ator é produzido através de três elementos pertencentes à cada célula das fitas de entrada. Símbolos (instruções), unidades de tempo (em que a imagem de um ator ficará sendo mostrada para a sincronia temporal dos atores) e um conjunto de operações de transformação de imagem (rotação, escala, translação e visibilidade). Quando um símbolo da fita é lido, uma transição no AFS correspondente é gerada, mudando seu estado atual e a imagem (e/ou efeitos sonoros) é mostrada numa camada, de acordo com o tempo e as operações de transformação especificadas. A animação consiste num conjunto de quadros e cada quadro é composto de camadas. Em outras palavras, cada ator corresponde a um conjunto de imagens estáticas (e/ou efeitos sonoros) que são mostrados e animados de acordo com as transições de um AFS. O encapsulamento em um AFS das propriedades estéticas e comportamentais do ator provê reusabilidade de imagens para diferentes composições de animação. A Figura 3.1 mostra a especificação do ator `BICHO`. As transições contém números, que correspondem ao alfabeto de entrada e imagens, que correspondem ao alfabeto de saída.

Dependendo do instante da animação, a imagem de saída pode sofrer transformações com o objetivo de se adequar ao contexto da cena representada. Por exemplo, a mesma imagem pode aparecer em posições ou escalas diferentes. Além de determinar quais as variações que os atores podem sofrer durante a animação, é necessário especificar em que momentos elas deverão ocorrer. Estes elementos são especificados na fita de entrada do autômato do ator. Cada símbolo da entrada é associado a um número natural que determina o tempo de espera, em milésimos de segundos, após a leitura de um símbolo. Desta maneira, a fita de entrada $ft = (a_1, t_1, \phi_1)(a_2, t_2, \phi_2) \cdots (a_n, t_n, \phi_n)$ possui os triplos ordenados (a, t, ϕ), onde a é um símbolo do alfabeto de entrada, t é um tempo em milésimos de segundo e ϕ uma cadeia de funções de transformação da saída. A Figura 3.2 mostra um exemplo de uma fita de entrada para uma instância do ator `BICHO`. A Tabela 3.2 contém as funções de transformação implementadas no protótipo corrente.[1]

[1]As funções de transformação são independentes do modelo de autômatos. Outras implementações podem agregar novas funções.

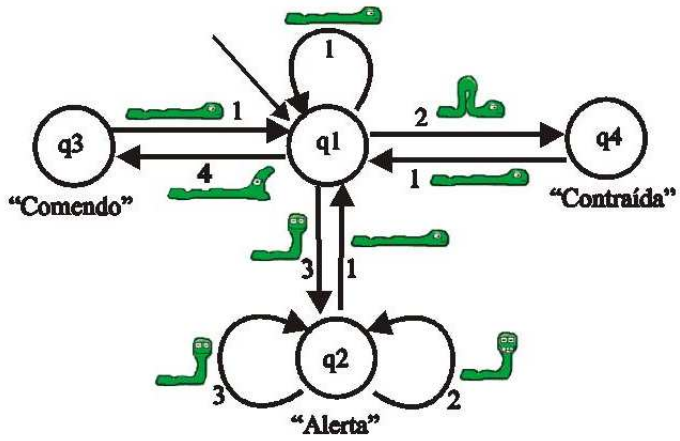


Figura 3.1: AGA - Autômato do ator BICHO

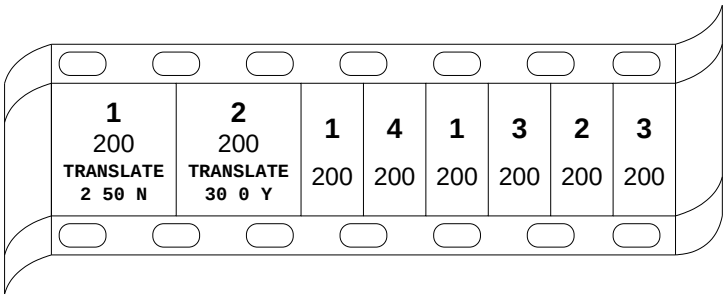


Figura 3.2: Fita de entrada do ator BICHO. Cada célula contém o símbolo a ser consumido, o tempo de duração de uma transição, em milissegundos e a função de transformação, que pode ser opcional

Tabela 3.2: AGA - funções de transformação

ROTATE angulo - transformações geométricas para rotação
TRANSLATE x y - translação em coordenadas cartesianas
SCALE fator - amplia ou reduz a imagem
FLIP V|H - espelhamento vertical ou horizontal
MATRIX matriz - aplica uma matriz de transformação
VISIBLE Y|N - controle de visibilidade
ORDER camada - altera a camada do ator

3.3 Descrição do AGA-J

Um ator AGA-J é descrito através de um modelo formal baseado em AFS junto com um conjunto de corpos articulados para representar a aparência do ator. O modelo de AFS adotado usa a Máquina de Mealy, associando conjuntos de transformações gráficas sofridas pelo ator às transições do autômato, como elementos do alfabeto de saída.

A figura articulada do ator AGA-J é definida através de um modelo hierárquico baseado numa estrutura de dados em árvore, onde um exemplo é mostrado na Figura 3.3. Cada articulação é um nó da árvore e as transformações sofridas por ela são propagadas ao longo de suas sub-árvores (PARENT, 2002). Qualquer objeto gráfico pode ser anexado nas folhas da árvore, desde controles de texto até modelos complexos com textura, por exemplo. É bom notar que este modelo não é restrito a figuras humanóides. Outros tipos de figuras também podem ser projetadas. O exemplo apresentado na Seção 5.3 mostra que um dos atores articulados representa uma cadeira.

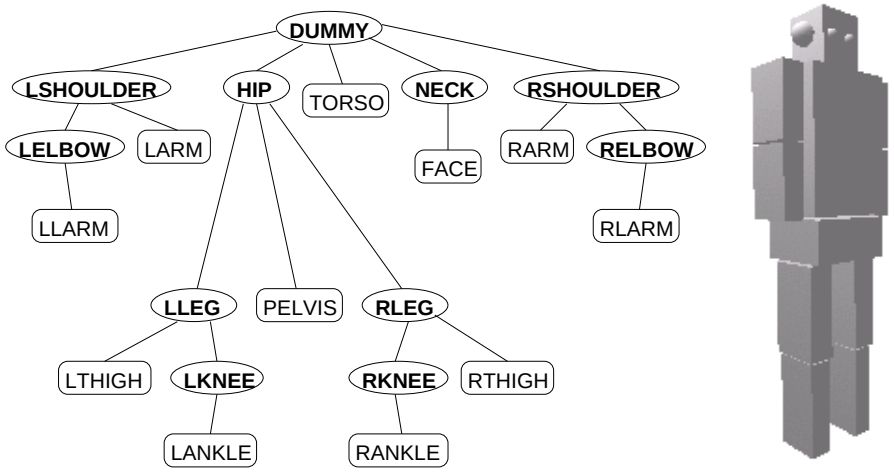


Figura 3.3: Estrutura hierárquica simplificada de um ator humanóide. Os retângulos representam objetos gráficos e as elipses representam articulações.

Uma instância AGA-J é um instanciamento de um ator AGA-J junto com uma fita de entrada. Uma animação no modelo AGA-J é composta por um conjunto de instâncias AGA-J. Mais de uma instância pode ser do mesmo ator e atores e fitas podem ser reusados em várias animações.[2]

3.4 Modelo formal

O modelo de autômatos usado para especificar atores é uma extensão do modelo AGA e, consequentemente, uma extensão da Máquina de Mealy. O modelo AGA-J pode ser formalmente descrito como uma 11-upla

$$ACT = (\Sigma, Q, q_0, D, \sigma, F, \Delta, \Gamma, T_0, \delta')$$

onde

- Σ é um alfabeto de entrada;

[2]Pelo restante do texto, o termo instância será usado para se referir a uma instância de um autômato, que é um ator efetivo em animações baseadas no modelo AGA.

- Q é um conjunto de estados;
- $q_0 \in Q$ é o estado inicial;
- D é um conjunto de descrições;
- $\sigma : Q \rightarrow D$ é uma função de descrição;
- F é um conjunto finito de funções de transformação;
- Δ é um conjunto de corpos articulados;
- Γ é um conjunto finito de estruturas de dados hierárquicas que representam poses de Δ;
- $T_0 \in \Gamma$ é uma estrutura de dados hierárquica que corresponde à pose inicial;
- $\delta' : (Q \times (\Sigma \cup \{\lambda\})) \times (\Gamma \times \mathbb{N} \times F) \rightarrow Q \times \Gamma$ é uma função de transição temporizada[3] .

A função programa, chamada de função de transição temporizada, tem como entrada o estado atual do autômato (Q), a pose atual do ator (Γ), e a célula da fita de entrada, que contém três dados opcionais: um símbolo do alfabeto de entrada (Σ) para selecionar a transição desejada, o tempo ($\mathbb{N}$) que a transição deve durar e uma lista de operações de transformações (F). Durante a transição, o símbolo gerado pela Máquina de Mealy é mapeado para uma lista de transformações de articulação que são aplicadas na estrutura de dados do corpo do ator para produzir uma nova pose. Os dados que representam o estado das articulações são armazenados em nodos especiais da estrutura de dados do corpo do ator. Quadros intermediários dentre a pose atual até a pose gerada são gerados por meio de interpolação de acordo com o tempo da transição e a taxa de quadros por segundo desejada na hora de reproduzir a animação.

Como em toda a Máquina de Mealy, o controle do autômato é obtido apenas pelo estado atual e o símbolo de entrada. O controle da aparência do personagem é obtido pela pose atual, a lista de operações de transformações, o tempo de permanência da transição, a nova pose do ator gerada pela obtenção do símbolo de saída e a taxa de exibição de quadros por segundo, sendo que este último dado é informado pelo usuário, como um parâmetro de reprodução de animação. Isto nos mostra que quadros-chaves não são necessários para providenciar animação e que é possível escolher a taxa de exibição de quadros no momento da reprodução de uma animação AGA-J sem necessidade de alterar sua especificação.

A Figura 3.4 ilustra os elementos da função de transição temporizada.

O espaço de armazenamento de uma animação AGA-J depende do tamanho da estrutura de dados dos corpos dos atores, do tamanho de seus alfabetos de saídas e das células da fitas de entrada utilizadas.

A função de descrição σ mapeia um estado para uma descrição semântica, dando suporte à recuperação de informação.

A Figura 3.5 mostra um autômato AGA-J típico que modela os movimentos de um ator humanóide. A figura mostra diretamente nas transições os símbolos do alfabeto de entrada (`LW`, `PUNCH`, etc.) e ilustra, por meio de imagens, o resultado das transformações

[3] A transição identidade possui sempre o rótulo λ ($\lambda \notin \Sigma$) e é interpretada como operação nula (vazia). Isto torna o autômato reflexivo (MENEZES; HAEUSLER, 2001).

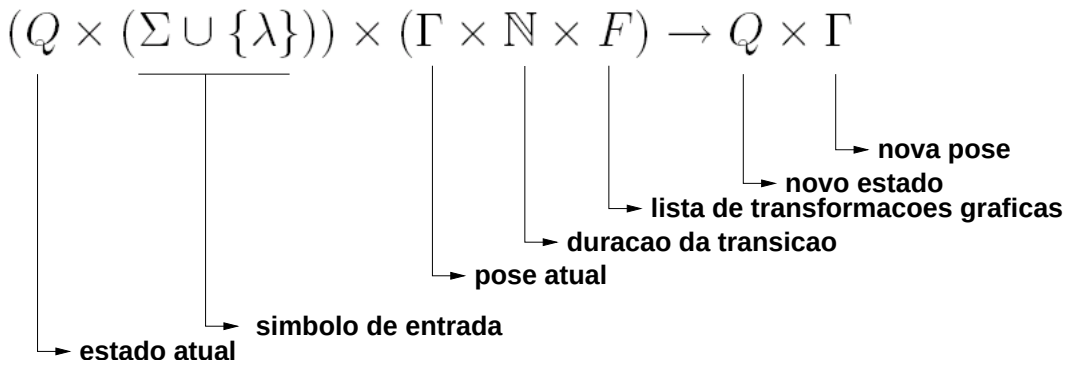


Figura 3.4: Elementos da função de transição temporizada do modelo formal do AGA-J

no modelo do ator produzidas pela interpretação dos símbolos do alfabeto de saída. A pose inicial do ator, bem como as transições identidade, presentes em todos os estados, estão omitidas para simplificar a figura.

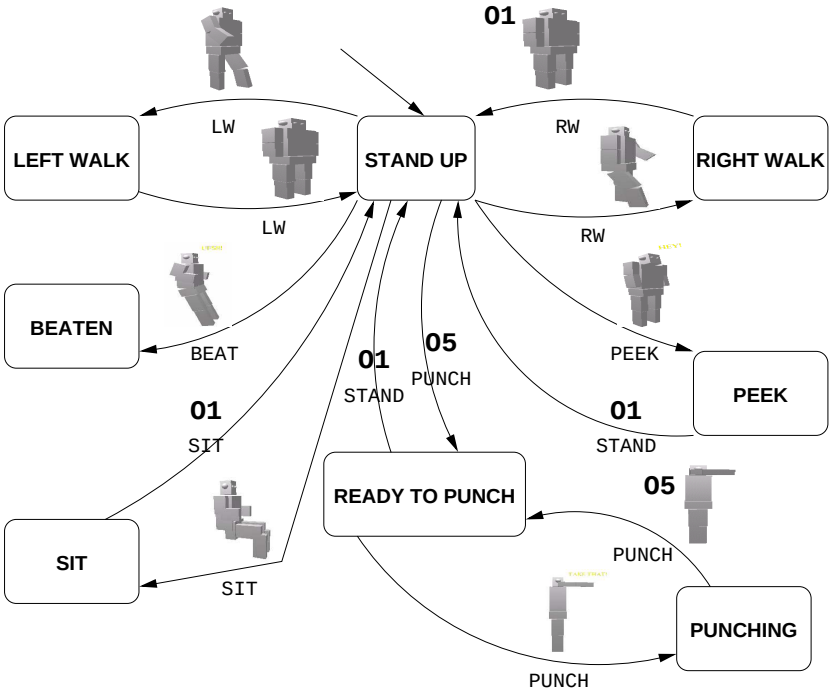


Figura 3.5: Exemplo de autômato AGA-J

3.5 Modelo formal da fita de entrada

A fita de entrada é composta de células onde cada uma é uma tripla

$$CELL = (\alpha, \tau, \Phi)$$

onde

- $\alpha \in (\Sigma \cup \{\lambda\})$ é o símbolo de entrada;

- τ é a duração da transição;
- Φ é uma lista de operações de transformações afim aplicadas na raiz[4] da estrutura de dados do ator.

Cada célula da fita contém as informações necessárias para efetuar um conjunto de transformações nas articulações e no sistema de coordenadas do ator. A Figura 3.6 mostra um trecho de fita de entrada para o ator humanóide da Figura 3.5. Assumindo que o ator esteja no estado `LEFT WALK`, este trecho de fita fará o ator efetuar três movimentos em sequência:

- ficar de pé (`LEFT WALK` $\overset{LW}{\rightarrow}$ `STAND UP`) com duração de 200ms e translação de 0.03 no eixo Z do sistema de coordenadas do ator em relação ao cenário;
- preparar o braço para dar um soco com a mão direita (`STAND UP` $\overset{PUNCH}{\rightarrow}$ `READY TO PUNCH`) com duração de 150ms;
- e esticar o braço esboçando um soco (`READY TO PUNCH` $\overset{PUNCH}{\rightarrow}$ `PUNCH`) com duração de 150ms e translação de 0.05 no eixo Z do sistema de coordenadas do ator em relação ao cenário.

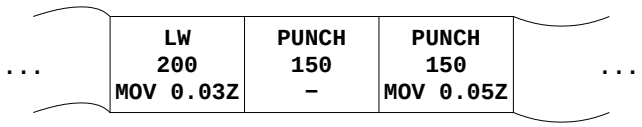


Figura 3.6: Parte de uma fita de entrada para um ator humanóide

A Figura 3.7 demonstra o resultado de uma transição mediante o consumo de uma célula da fita para a construção das poses de um personagem que faz parte de uma animação a ser reproduzida na taxa de 15 quadros por segundo. A transição é feita do estado `STAND UP` para o estado `RIGHT WALK`. Os estados fazem parte do autômato da Figura 3.5.

3.6 Motor do AGA-J

Para gerar animações, o usuário supre um parâmetro que indica a taxa de quadros exibidos por segundo (FPS - *Frames Per Second*). Cada instância possui uma fita de entrada e sua posição inicial em relação ao cenário. As instâncias são renderizadas no cenário segundo sua posição e valores iniciais das articulações do conjunto de corpos articulados associado, obtendo uma pose inicial. Após esta etapa inicial, o seguinte algoritmo é executado para cada instância:

1. leia uma célula da fita de entrada e obtenha α, τ e Φ;
2. aplique as funções de transformação afim contidas em Φ na raiz da estrutura de dados do ator;

[4] As transformações aplicadas na raiz de uma estrutura de dados hierárquica são propagadas a todos os nodos desta estrutura.

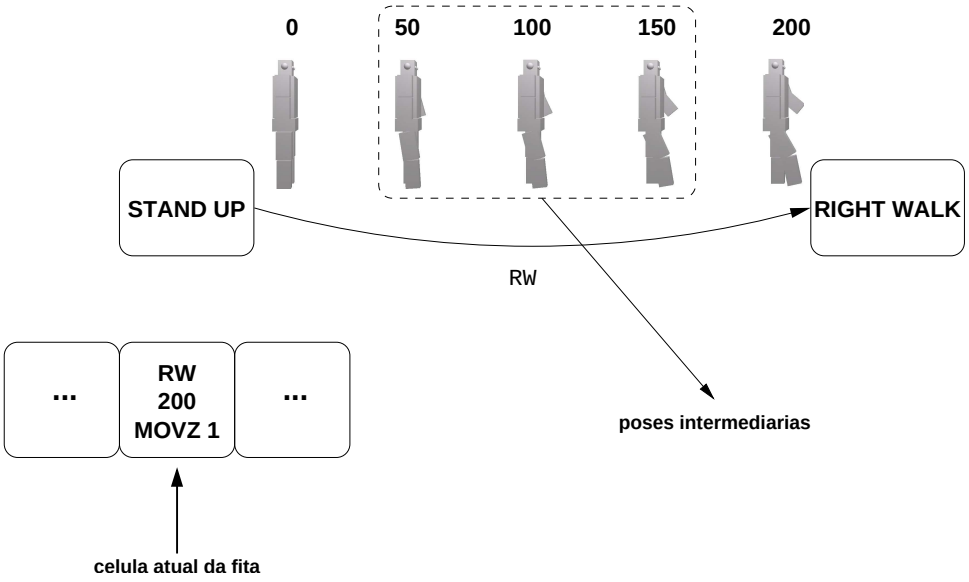


Figura 3.7: Resultado de uma transição de um autômato AGA-J

3. calcule o número de quadros necessário para esta etapa da animação (N) através da função de transição temporizada δ' e do parâmetro FPS usando a seguinte fórmula: $q = (\tau/1000) * FPS$;

4. envie α ao autômato associado para disparar uma transição no autômato e obter a saída A que é uma lista de articulações posicionadas;

5. aplique A em Δ para obter a nova pose da instância;

6. durante os q quadros seguintes, aplique interpolação linear da pose atual até a nova pose obtida, exibindo o quadro gerado;

7. mude o estado atual do autômato de acordo com a transição disparada por α;

8. repita os passos acima até o autômato associado atingir uma condição de parada.

Durante a animação, é possível consultar informações sobre qualquer ator, obtendo o estado atual do autômato associado e usando a função de descrição σ neste estado. A animação é encerrada quando todos os autômatos associados atingirem uma condição de parada.

3.7 Grafo de cena

Um grafo de cena é um grafo acíclico dirigido (DAG - *Directed Acyclic Graph*) com raiz que representa os objetos que compõem a cena mostrada por uma aplicação em 3D. Como exemplos, VRML (ISO/IEC 14772-1, 1997) e X3D (ISO/IEC 19775-200X, 2003) são padrões que adotam esta estrutura de dados.

O grafo indica uma hierarquia entre seus nodos no sentido de que cada nodo pode modificar o estado de todos os nodos apontados por este. Em um DAG com raiz, um nodo pode ser modificado por vários nodos, contrastando com a abordagem tradicional de uma árvore com raiz, onde cada nodo filho só possui um nodo pai ligado diretamente.

Os nodos de um grafo de cena podem conter objetos que representam sistemas de coordenadas que são conectados em uma relação de pai e filho, sendo que nodos folhas podem conter conjuntos de primitivas gráficas. Como exemplo, um carro pode ser representado usando um nodo contendo uma malha de polígonos que representa sua carenagem e quatro nodos que correspondem às suas rodas (representadas por circunferências texturadas, por exemplo), cada uma especificada tendo seu sistema de coordenadas relativo ao eixo correspondente. Além disso, a flexibilidade do grafo permite que haja outros tipos de objetos que representam especificações de iluminação e especificações de eventos oriundos do usuário ou do ambiente de reprodução.

3.8 Detalhes de implementação

Um protótipo escrito na linguagem Java (SUN MICROSYSTEMS, 1994) foi construído para validar o modelo AGA-J. Java foi escolhida devido à sua ampla difusão no mercado e por sua facilidade na construção de applets, contribuindo para a proposta de um animador gráfico em 3D para a WWW. O motor gráfico Java3D (SUN MICROSYSTEMS, 1998) foi escolhido porque trabalha com grafos de cena, visto que a hierarquia de componentes providenciada por este tipo de estrutura de dados facilita a criação de personagens articulados.

Os dados de entrada da animação são compostos pela especificação das instâncias que participam da animação e dos atores instanciados pelas mesmas. A especificação de um ator é composta pela descrição do modelo articulado e do AFS associado (estados, alfabeto de entrada, alfabeto de saída e diagrama de transições). A especificação de uma instância e composta por uma referência ao ator instanciado, sua posição inicial em relação ao cenário da animação e as células da sua fita de entrada. O motor de animação obtém estes dados através de arquivos de entrada com o seu conteúdo em uma linguagem chamada AGA2ML. Esta linguagem é uma extensão particular da linguagem XML que foi desenvolvida para especificar animações AGA-J.

As seguintes seções apresentam uma visão geral do grafo de cena utilizado no Java3D, a linguagem AGA2ML e algumas animações projetadas.

3.8.1 Grafo de cena do Java3D

Um grafo de cena no Java3D é composto por um conjunto de nodos contendo objetos Java3D conectados como mostrado na Figura 3.8. A raiz do grafo é um objeto do tipo *VirtualUniverse* que é um espaço 3D onde os objetos de cena existem e um ou mais objetos do tipo *Locale* que definem sistemas de coordenadas.

Cada subgrafo de cena é composto por nodos conectados a um nodo contendo um objeto do tipo *Locale*, construindo um ambiente virtual. Um nodo contendo um objeto do tipo *TransformGroup* é usado para mudar a posição e orientação dos nodos filhos contendo objetos gráficos. Um objeto do tipo *Shape3D* contém a descrição de um objeto gráfico com informações de cor e iluminação e é sempre encontrado em folhas (nodos terminais).

A modelagem de ator proposta neste trabalho encaixa bem num subgrafo de cena do Java3D porque é bem simples mapear nodos que representam articulações em nodos com objetos do tipo *TransformGroup*. As folhas restantes podem ser facilmente mapeadas à grande variedade de primitivas gráficas da API da Java3D que são objetos de subclasses de *Shape3D*. A Figura 3.8 é um exemplo de um grafo de cena que mapeia diretamente, a partir do nodo *Dummy*, a estrutura de modelo articulado descrita na Figura 3.3.

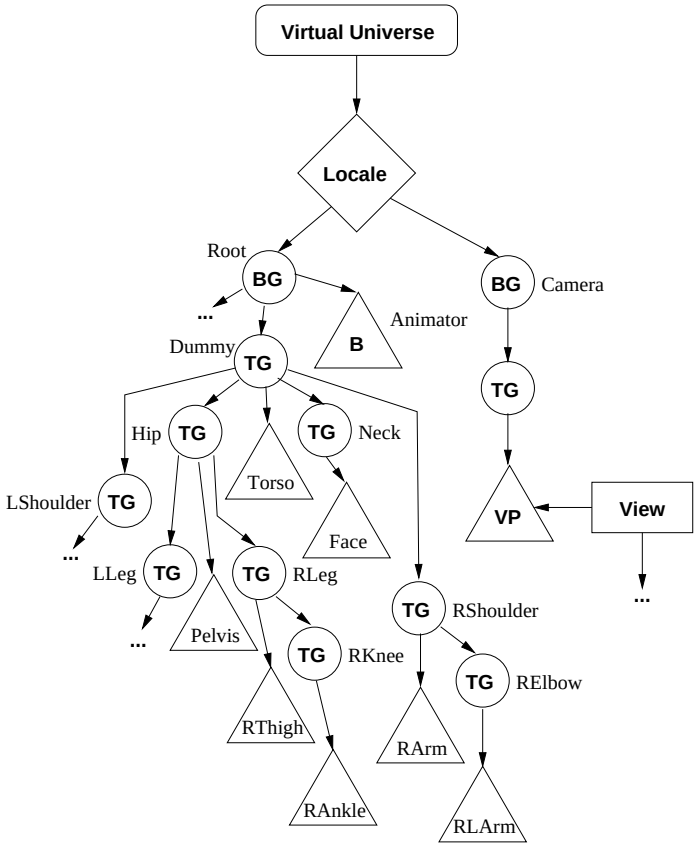


Figura 3.8: Grafo de cena do Java3D usado pelo motor AGA-J. O diagrama apresenta os seguintes nodos: BG (*BranchGroup*); TG (*TransformGroup*); B (*Behaviour*) e VP (*ViewingPlatform*). Triângulos representam nodos *Shape3D*

3.8.2 AGA2ML

Como dito na seção anterior, um vocabulário XML foi projetado para especificar a animação. Um arquivo usando este vocabulário contém grande parte dos dados da animação: modelagem do conjunto de corpos articulados dos atores, autômatos dos atores, refências das instâncias junto com suas fitas e parâmetros de inicialização, e parâmetros para a configuração do cenário. Malhas, texturas, sons e outras informações adjuntas residem em arquivos adicionais, referenciados por atributos de marcas XML deste vocabulário.

As seções seguintes descrevem os elementos definidos na DTD da AGA2ML. Cada novo vocabulário XML possui uma DTD, que é utilizada como padrão para definição de suas regras sintáticas.

3.8.3 Especificação da animação

A Figura 3.9 mostra o trecho da DTD correspondente ao elemento raiz do vocabulário AGA2ML. O elemento AGA3D comporta as especificações dos atores e as definições das instâncias. Além disto, disponibiliza informações adicionais para auxiliar no processo de apresentação da animação.

Os atributos do elemento AGA3D: WIDTH, HEIGHT, FRAMERATE e LOOP, descrevem, respectivamente, a largura e altura da janela onde será exibida a animação, o número de quadros por segundo e o número de vezes que a animação deve ser repetida. A repetição contínua da animação é obtida ao atribuir o valor 0 ao elemento LOOP e o

```
<!ELEMENT AGA3D (HEAD, ACTOR+, INSTANCE*)>
<!ATTLIST AGA3D
        VERSION CDATA #FIXED "1.0"
        WIDTH CDATA #REQUIRED
        HEIGHT CDATA #REQUIRED
        FRAMERATE CDATA #REQUIRED
        LOOP CDATA #IMPLIED>
<!ELEMENT HEAD (TITLE, AUTHOR, SUBJECT)>
<!ELEMENT TITLE (CDATA)>
<!ELEMENT AUTHOR (CDATA)>
<!ELEMENT SUBJECT (CDATA)>
```

Figura 3.9: DTD da raiz do AGA2ML

valor 1 indica que a animação só será reproduzida uma vez. O valor atribuído ao FRAMERATE é usado pelo motor do AGA-J para determinar a suavidade da interpolação dos movimentos dos atores.

Inspirado no trabalho anterior (ACCORSI, 2002), o elemento HEAD tem como objetivo atender as estratégias de indexação orientadas a metadados e taxonomias (ARAÚJO; GUIMARÃES, 2000). Os elementos TITLE, AUTHOR e SUBJECT são utilizados para armazenar informações bibliográficas e descrições adicionais para categorizações subjetivas.

3.8.4 Especificação do ator e de sua modelagem

A Figura 3.10 mostra um trecho da DTD correspondente ao elemento ACTOR, que contém elementos filhos que descrevem a modelagem do ator (JOINT, BOX, SPHERE, CYLINDER, TEXT e MESH). Estes elementos podem ser organizados num formato hierárquico para descrever um modelo com articulações e primitivas gráficas, como mostrado no exemplo da Figura 3.11. O conjunto de elementos do tipo STATE define o autômato do ator por meio de seus estados e alguns elementos do modelo formal do AGA-J. A função descrição σ é especificada pelo atributo DESCRIPTION. As transições de cada estado são definidas pelos elementos do tipo TO, e o alfabeto de saída Δ (OUTPUT) é definido pelo texto contido nestes elementos.

```
<!ELEMENT ACTOR (JOINT*, BOX*, SPHERE*, CYLINDER*, MESH*, TEXT*, STATE* )>
<!ATTLIST ACTOR
        ID CDATA #REQUIRED
        START CDATA #REQUIRED>

<!ELEMENT STATE (TO+)>
<!ATTLIST STATE
        ID CDATA #REQUIRED
        DESCRIPTION CDATA>
<!ELEMENT TO (#PCDATA)>
<!ATTLIST TO
        STATE CDATA #REQUIRED
        SYMBOL CDATA #REQUIRED>
```

Figura 3.10: Trecho da DTD do ator do AGA2ML

A Figura 3.11 apresenta um trecho da especificação do ator `BONECO` na linguagem AGA2ML. O trecho mostra a descrição da articulação `PESCOCO` que contém 4 elementos subordinados: a malha `CABECA`, as esferas `OLHODIREITO` e `OLHOESQUERDO` e `FALA`, um elemento para reprodução de texto posicionado. `OMBRODIREITO` contém a caixa `BRACODIREITO` e outra articulação, nomeada `COTOVELODIREITO` que contém

a caixa ANTEBRACODIREITO, demonstrando que qualquer elemento gráfico pode estar subordinado a uma articulação, inclusive outras articulações. Pode-se perceber que a caixa TRONCO não está subordinada a nenhuma articulação. A marca STATE descreve o estado EMPE, que é um dos estados do autômato do ator. A marca subordinada TO descreve uma transição deste estado para o estado ANDDIR, mediante o processamento do símbolo ANDDIR. O texto contido na marca TO é a saída gerada pela transição, que é um conjunto de transformações sobre as articulações do ator.

```
<ACTOR ID="BONECO" START="EMPE">
   <JOINT ID="PESCOCO" X="0" Y="+0.5">
      <MESH ID="CABECA" FILE="cabeca.mesh" X="0" Y="+0.3" Z="0" SCALE="0.15" />
      <SPHERE ID="OLHODIREITO" X="-0.07" Y="+0.4" Z="0.2" RAIO="0.025" />
      <SPHERE ID="OLHOESQUERDO" X="+0.07" Y="+0.4" Z="0.2" RAIO="0.025" />
      <TEXT ID="FALA" X="0" Y="0.24" Z="+0.2" TEXT="" SIZE="30" />
   </JOINT>

   <BOX ID="TRONCO" X="0" Y="+0.15" SX="0.2" SY="0.3" SZ="0.15" />

   <JOINT ID="OMBRODIREITO" X="-0.2" Y="+0.45">
      <BOX ID="BRACODIREITO" X="-0.06" Y="-0.15" SX="0.05" SY="0.15" SZ="0.1" />
      <JOINT ID="COTOVELODIREITO" X="-0.06" Y="-0.3">
         <BOX ID="ANTEBRACODIREITO" X="0" Y="-0.16" SX="0.05" SY="0.15" SZ="0.1" />
      </JOINT>
   </JOINT>
   ...
   <STATE ID="EMPE" DESCRIPTION="Parado">
      <TO STATE="ANDDIR" SYMBOL="ANDDIR">
         PERNADIREITA ROTX -30;
         JOELHODIREITO ROTX 18;
         OMBROESQUERDO ROTX -20;
         COTOVELOESQUERDO ROTX -30;
      </TO>
      ...
   </STATE>
   ...
</ACTOR>
```

Figura 3.11: Trecho da especificação de um ator em AGA2ML

3.8.5 Especificação das instâncias e de suas fitas de entrada

O elemento INSTANCE cria efetivamente um personagem no cenário da animação. Ele faz a associação de uma fita de entrada com o AFS de um modelo de ator previamente especificado por uma ocorrência do elemento ACTOR. Através deste elemento é possível criar várias instâncias do mesmo ator. A Figura 3.12 mostra o trecho da DTD correspondente a especificação das instâncias. A fita de entrada é definida pelo texto contido nestes elementos. As células são definidas por strings separadas pelo delimitador ponto e vírgula (;).

A Figura 3.13 apresenta um exemplo de especificação de instância que corresponde ao personagem que sofre um golpe da animação descrita na Seção 5.2. No exemplo, a instância sofre uma rotação de 10 graus sobre o eixo Y e é posicionada nas coordenadas $(0.7, 0, 0)$ do sistema de coordenadas do cenário da animação. Sua fita de entrada contém seis símbolos que coordenam a animação do personagem. O processamento da primeira célula faz o personagem girar o corpo e esticar o pescoço; a segunda reverte o pescoço para a posição original; a terceira e a quarta provocam um atraso de 300ms para depois reproduzir um arquivo de som que corresponde a um som de impacto; a quinta translada o ator no eixo X e aplica uma rotação na cintura e nos ombros para simular o golpe recebido.

```
<!ELEMENT INSTANCE (#PCDATA)>
<!ATTLIST INSTANCE
        ID CDTATA #REQUIRED
        ACTOR CDATA #REQUIRED
        X CDATA
        Y CDATA
        Z CDATA
        ROTX CDATA
        ROTY CDATA
        ROTZ CDATA>
```

Figura 3.12: DTD da instância AGA-J do AGA2ML

```
<INSTANCE ID="JOAQUIM" ACTOR="BONECO" X="+0.7" Y="0" Z="0" ROTY="10">
   ESPIA,1200,ROTYMOV -100  -0.15 0 0;
   ESPIA,200;
   _,300,DELAY;
   _,0,PLAY crunch.wav;
   APANHA,150,MOV -0.05 0 0;
   _,1000,MOV 0.55 0 0;
</INSTANCE>
```

Figura 3.13: Exemplo de especificação de instância AGA-J em AGA2ML

Cada célula da fita possui três elementos separados pelo delimitador vírgula (,) e finalizados por ";"no formato $SYMBOL, TIME, OPERATIONS$;.

- $SYMBOL(\alpha)$ é o símbolo de entrada;
- $TIME(\tau)$ é a duração da transição correspondente;
- $OPERATIONS(\Phi)$ corresponde a operações particulares ao evento atual da animação que normalmente se aplicam ao personagem em relação ao cenário (rotações e translações, por exemplo).

A cada leitura da fita, o símbolo $SYMBOL$ é enviado para o autômato, a transição correspondente tem início e a saída desta transição é coletada. Durante o tempo $TIME$, as operações contidas em $OPERATIONS$ são executadas em paralelo com as mudanças nas articulações que foram obtidas pela saída da transição. Por fim, o estado atual do autômato é atualizado.

O caractere especial _ indica uma transição reflexiva implícita no estado atual do autômato associado. Sendo assim, nenhuma mudança nas articulações é efetuada, levando em conta apenas $OPERATIONS$ e $TIME$ neste caso particular. A utilização de _ é conveniente para evitar a criação de arestas na forma de laço nos estados do autômato do ator só com o objetivo de aplicar operações, o que provocaria um aumento na representação da função de transição temporizada.

A Tabela 3.3 mostra as operações implementadas até então. Todos os ângulos especificados são absolutos levando em conta o sistema de coordenadas do cenário e as coordenadas são absolutas em relação à posição inicial da instância. As operações que especificam translações e rotações são implementadas interpolando a posição e rotações atuais do ator até as especificadas pela operação. Isso torna as transformações não-cumulativas, garantindo sempre o resultado da última operação. É bom salientar que operações adicionais podem ser implementadas sem comprometer a sintaxe da AGA2ML.

O Capítulo 5 apresenta algumas animações descritas pelo modelo AGA-J e uma investigação das características de reuso e recuperação de informação que o modelo propicia.

Tabela 3.3: Operações da fita de entrada dos atores

Operação	Argumentos	Descrição
MOV	X Y Z	Posiciona o ator nas coordenadas (X,Y,Z)
ROT	AX AY AZ	Rotaciona o ator em relação aos ângulos especificados.
ROTX, ROTY e ROTZ	A	Rotaciona o ator no ângulo A em relação ao eixo X, Y ou Z, respectivamente.
ROTMOV	AX AY AZ X Y Z	Combina o comando ROT com o MOV
ROTXMOV, ROTYMOV e ROTZMOV	A X Y Z	Combina ROT X (ou Y ou Z) com MOV
DELAY		Nenhuma operação. Útil para sincronismo temporal.
TEXT	ID TEXTO	Altera o texto do componente textual do ator identificado por ID.
PLAY	SOUNDFILE	Reproduz o arquivo de som.

4 AGA-T: IDÉIAS PARA UM MODELO DE ANIMAÇÃO ORIENTADO A TAREFAS

4.1 Introdução

Pode-se dizer que o modelo AGA-J provê ferramentas para trabalhar no nível concreto de controle, sendo possível especificar vários tipos de animações com os controles implementados.[1] Entretanto, uma invariante no modelo AGA-J é a dependência entre os personagens da animação em relação ao tempo. Por causa disso, animações complexas tornam-se difíceis de ser especificadas e, na prática, algum tipo de *front-end* mostra-se necessário para auxiliar na especificação e manutenção destas animações.

Este capítulo apresenta idéias para um modelo de animação que usa o modelo AGA-J como camada inferior. A proposta do modelo AGA-T é um modelo orientado a tarefas (ZELTZER, 1982), baseado em autômatos probabilísticos (HOPCROFT; MOTWANI; ULLMAN, 2001) e eventos gerados pelo ambiente.[2] O modelo proposto usa o modelo AGA-J como camada inferior para demonstrar que o modelo AGA-J pode ser usado como camada de fundação para a implementação de outros modelos mais especializados. Os autômatos AGA-T geram a fita de entrada para os autômatos AGA-J, aproveitando na íntegra todas as características do modelo AGA-J.

A seção seguinte descreve a proposta do modelo AGA-T. A Seção 4.3 contém o modelo formal do AGA-T; a Seção 4.4 demonstra passo a passo o algoritmo do motor de animação; a Seção 4.5 apresenta a especificação do autômato AGA-T numa possível extensão da linguagem AGA2ML para a implementação do modelo AGA-T e a Seção 4.6 apresenta a especificação de uma instância AGA-T.

4.2 Descrição do AGA-T

Uma animação no modelo AGA-T é composta por um conjunto de instâncias de atores descritos através de um modelo formal baseado em autômatos probabilísticos (HOPCROFT; MOTWANI; ULLMAN, 2001) (ALUR et al., 1994) (BEAUQUIER, 2003) e um ator AGA-J associado. Análogo ao modelo AGA-J, o modelo AGA-T também usa a Máquina de Mealy, associando as tarefas efetuadas pelo ator às transições do autômato.

No contexto do modelo AGA-T, uma tarefa é uma sequência ordenada de movimentos. Quando uma transição de um autômato AGA-T é disparada, os movimentos da tarefa associada ao símbolo de saída gerado pela transição são repassados ao ator AGA-J asso-

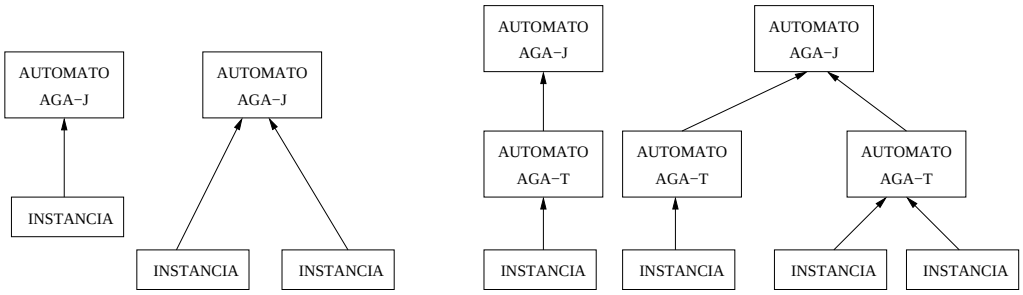
[1] Vários elementos básicos de descrição de cena ainda não estão implementados no protótipo atual, como especificação detalhada de iluminação e controle de câmera.

[2] Os eventos suportados são dependentes da implementação do motor de animação.

ciado. Uma transição AGA-T gera uma ou mais transições AGA-J através da geração de várias células da fita de entrada do autômato do ator AGA-J associado. A aparência do personagem é obtida através do ator AGA-J associado (elementos Δ, Θ e Γ apresentados na Seção 3.4).

Um ator AGA-T é descrito através de um modelo formal baseado na Máquina de Mealy e em autômatos probabilísticos. No AGA-T, uma transição é disparada mediante uma condição de probabilidade satisfeita e uma condição de evento opcional. Visto que o modelo é baseado em autômatos probabilísticos, cada transição de um autômato AGA-T possui um valor real correspondente a um peso para verificar uma condição de probabilidade. Uma condição de evento é um predicado que pode estar presente ou não numa transição de um autômato AGA-T. Um ator AGA-T possui um ator AGA-J associado e a saída gerada por um autômato AGA-T é repassada ao autômato do ator AGA-J associado para gerar as saídas na pose do personagem.

Uma instância AGA-T é um instanciamento de um ator AGA-T junto com sua posição inicial e outros atributos de inicialização.[3] Mais de uma instância AGA-T pode usar um mesmo ator AGA-T e mais de um ator AGA-T pode usar um mesmo ator AGA-J. A Figura 4.1 mostra o instanciamento de atores dos modelos AGA-J e AGA-T e a relação entre os modelos. A figura sugere que, assim como no AGA-J em que é possível reaproveitar o mesmo autômato em várias instâncias AGA-J, vários autômatos AGA-T podem reaproveitar o mesmo autômato AGA-J e várias instâncias AGA-T podem reaproveitar o mesmo autômato AGA-T.



(a) Modelo AGA-J: uma instância AGA-J usando um autômato AGA-J e várias instâncias AGA-J usando o mesmo autômato AGA-J

(b) Modelo AGA-T: uma instância AGA-T usando um autômato AGA-T que usa um autômato AGA-J e vários autômatos AGA-T usando o mesmo autômato AGA-J. Várias instâncias AGA-T podem usar o mesmo autômato AGA-T

Figura 4.1: AGA-J e AGA-T

O modelo AGA-T pode ser estendido, sem maiores dificuldades, para ser recursivo que nem o AGA-J. Isto não foi abrangido nesta proposta porque não é o objetivo deste capítulo.

4.3 Modelo formal

O modelo de autômatos usado para especificar atores AGA-T é baseado no formalismo autômato probabilístico e é também uma extensão da Máquina de Mealy que delega

[3]Os atributos adicionais de inicialização são dependentes da implementação do motor de animação, visto que o modelo AGA-T não especifica nem restringe os tipos de atributos que uma instância AGA-T pode suportar.

sua saída para um autômato AGA-J associado. O modelo AGA-T pode ser formalmente descrito como uma 9-upla

$$ACT2 = (Q, q_0, \Upsilon, \Gamma, \delta', J, \lambda, \sigma, D)$$

onde

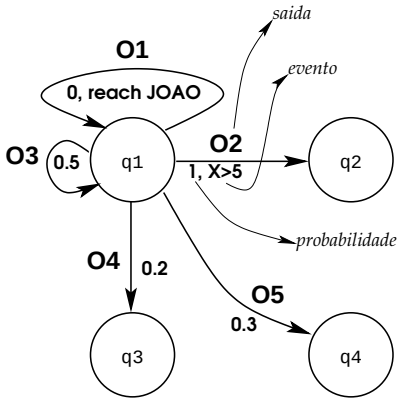
- Q é um conjunto de estados;
- $q_0 \in Q$ é o estado inicial;
- Υ é um conjunto de condições de evento;
- Γ é um conjunto de tarefas;
- $\delta' : Q \times \mathbb{R} \times \Upsilon \rightarrow Q \times \Gamma^*$ é uma função de transição probabilística;
- J é o ator AGA-J associado, possuindo (α, τ, Φ) como célula de entrada do seu autômato;
- $\lambda : \Gamma \rightarrow (\alpha, \tau, \Phi)^*$ é uma função de tradução de tarefas;
- $\sigma : Q \rightarrow D$ é uma função de descrição;
- D é um conjunto de descrições.

A função de transição probabilística dispara uma transição mediante um teste de probabilidade e uma condição de evento opcional. Eventos são gerados internamente ou pelo usuário. Na escolha de uma transição, as transições dependentes de evento são verificadas primeiro. Das transições verificadas, as que tiverem eventos observados são selecionadas mediante a função de seleção por probabilidade, descrita a seguir. Se nenhuma transição tiver eventos observados, as transições que não possuirem condições de evento (transições restantes) são selecionadas mediante a função de seleção por probabilidade.

A função de seleção por probabilidade seleciona uma transição dentre um conjunto de transições. Cada transição possui um peso que corresponde a uma condição de probabilidade. O simulador gera um número aleatório que varia de 0 até a soma dos pesos das transições do conjunto e uma transição é escolhida de acordo com este número. A Figura 4.2 mostra um exemplo do processo de escolha de uma transição do estado $q1$ com a função de seleção por probabilidade sendo aplicada nas transições $q1 \rightarrow q1(O3)$, $q1 \rightarrow q3$ e $q3 \rightarrow q4$, no caso de nenhum evento acontecer. Se ambos os eventos acontecerem, a transição $q1 \rightarrow q2$ é selecionada porque a transição $q1 \rightarrow q1(O1)$ possui peso zero.

A saída das transições do autômato AGA-T é uma lista de tarefas (Γ), que são mapeadas pela função de tradução de tarefas (λ) para uma palavra de entrada (ω) para o autômato do ator AGA-J associado. A função gera ω baseada nos parâmetros da instância e nas informações associadas ao símbolo do alfabeto de saída. As instâncias suportam parâmetros de inicialização que acarretam o fato de que as saídas do mesmo autômato AGA-T podem ser usadas para gerar palavras diferentes de acordo com estes parâmetros. A Seção 4.4 apresenta uma explicação dos parâmetros de uma instância AGA-T.

Durante uma transição do autômato AGA-T, o autômato do ator AGA-J associado é alimentado com a palavra gerada antes do novo estado ser alcançado e o autômato do ator AGA-J associado realiza a animação mediante o processamento da palavra. O espaço de



transições	$q1 \rightarrow q1$	$q1 \rightarrow q3$	$q1 \rightarrow q4$
pesos	0.5	0.2	0.3
selecionadas			
rnd <= 0.5	X		
0.5 < rnd <= 0.7		X	
0.7 < rnd <= 1			X

Figura 4.2: Função de seleção por probabilidade

armazenamento da animação com atores AGA-T depende somente do tamanho do ator AGA-J associado, do tamanho do alfabeto de saída (Γ) e do tamanho do conjunto das condições de evento (Υ). Análogo ao AGA-J, quadros-chaves não são necessários para providenciar animação.

4.4 Motor AGA-T

O motor de animação do AGA-T é implementado como uma camada sobre o modelo AGA-J. Análogo ao AGA-J, a animação é composta de instâncias parametrizadas de autômatos AGA-T. Os parâmetros de uma instância AGA-T correspondem à sua posição inicial do cenário, valores iniciais das articulações do conjunto de corpos articulados do ator AGA-J associado e parâmetros personalizados que funcionam como variáveis a serem lidas nos predicados dos eventos e na geração das tarefas.

As instâncias são renderizadas no cenário segundo sua posição e valores iniciais das articulações. Após esta etapa inicial, o seguinte algoritmo é executado para cada instância AGA-T:

1. o conjunto E corresponde às transições que partem do estado atual que satisfazem a condição de evento através da função de verificação de condição de evento;

2. se E for vazio, insira no conjunto E todas as transições sem eventos que partem do estado atual;

3. selecione uma transição T do conjunto E através da função de seleção por probabilidade e pegue a string de saída gerada pela transição (Γ^*);

4. para cada elemento da string de saída, aplique a função de tradução de tarefas (λ) para obter a palavra ω;

5. execute o motor AGA-J (Seção 3.6) do ator AGA-J associado (J) usando a palavra ω;

6. mude o estado do autômato AGA-T através da transição T e

7. repita os passos acima até o autômato associado AGA-T e/ou o autômato associado AGA-J atingir uma condição de parada.

O protótipo atual suporta, na mesma animação, a especificação de instâncias AGA-J e AGA-T, que são animadas mediante os respectivos motores. O motor AGA-T permite que as transições dependentes de eventos que reconhecem instâncias possam reconhecer instâncias AGA-T e AGA-J. Como no modelo anterior, também é possível consultar informações sobre qualquer ator, obtendo o estado atual do autômato associado e usando a função de descrição (σ) neste estado. A animação é encerrada quando todos os autômatos associados atingirem uma condição de parada.[4]

4.5 Especificação do autômato AGA-T

A DTD da especificação do ator AGA-T na linguagem AGA2ML é mostrada na Figura 4.3. O atributo ACTOR indica o autômato AGA-J associado para o qual a fita será gerada. START indica o estado inicial do autômato AGA-T.

```
<!ELEMENT ACTOR2 (OUTPUT*, STATE* )>
<!ATTLIST ACTOR2
        ID CDATA #REQUIRED
        START CDATA #REQUIRED
        ACTOR CDATA #REQUIRED>

<!ELEMENT OUTPUT (#PCDATA)>
<!ATTLIST OUTPUT
        ID CDATA #REQUIRED>

<!ELEMENT STATE (TO+)>
<!ATTLIST STATE
        ID CDATA #REQUIRED
        DESCRIPTION CDATA>
<!ELEMENT TO (#PCDATA)>
<!ATTLIST TO
        STATE CDATA #REQUIRED
        PROB CDATA
        EVENT CDATA>
```

Figura 4.3: DTD do ator AGA-T

O conjunto de elementos do tipo OUTPUT determina o alfabeto de saída dos autômatos AGA-T. Cada elemento OUTPUT possui uma identificação que corresponde a um símbolo do alfabeto de saída e o texto contido no elemento corresponde a descrição textual de células para o autômato AGA-J associado.

Cada elemento do tipo STATE é um estado do autômato AGA-T. As transições de cada estado são descritas pelos elementos do tipo TO contidos. O autômato AGA-T também é uma Máquina de Mealy e o conjunto de símbolos do alfabeto de saída gerado pela transição corresponde ao texto contido no elemento TO. Como no AGA-J, os estados do autômato AGA-T também possuem uma função descrição, especificada pelo atributo DESCRIPTION.

Para uma transição ser disparada, ela tem que atender uma condição de probabilidade, especificada pelo atributo PROB e uma condição de evento, especificada pelo atributo EVENT. É feita uma consulta ao ambiente para verificar a condição de evento e uma

[4]O processamento do automato do ator AGA-J associado não valida a entrada, simplesmente ignorando eventuais símbolos inválidos gerados pela função de tradução de tarefas.

consulta a uma função de probabilidade que é fornecida pelo motor de animação para verificar a condição de probabilidade. A Seção 4.4 descreve isso com mais detalhes.

As figuras 4.4 e 4.5 mostram um exemplo da especificação de um autômato AGA-T usando parâmetros, que são descritos com mais detalhes na Seção 4.6. O evento SEE dispara quando a instância que está usando este autômato enxerga a instância com o nome contido no parâmetro :OPONENT. Uma implementação possível deste evento é fazer com que um raio seja lançado a partir de uma posição e ângulos previamente estabelecidos. O evento REACH dispara quando a instância aproxima-se da instância :OPONENT. Uma aproximação entre atores pode ser implementada de várias maneiras: teste de distância radial, colisão de AABB, colisão de polígonos, etc.

```
<ACTOR2 ID="BONECO2" START="INICIO" ACTOR="BONECO">
  <OUTPUT ID="WALK">
      ANDDIR,400,MOV 0.1 0 0;
      ANDDIR,400,MOV 0.2 0 0;
      ANDESQ,400,MOV 0.3 0 0;
      ANDESQ,400,MOV 0.4 0 0;
  </OUTPUT>
  <OUTPUT ID="TURN">
      _,200,ROTY 90;
  </OUTPUT>
  <OUTPUT ID="FIGHT">
      SOCO,150;
      SOCO,150,MOV 0.6 0 0;
      SOCO,100;
      VOLTA,100;
  </OUTPUT>
  <OUTPUT ID="ESPERA">
      _,1000,DELAY;
  </OUTPUT>
  ...
```

Figura 4.4: Exemplo da especificação do alfabeto de saída de um ator AGA-T em AGA2ML

4.6 Especificação da instância

No modelo AGA-T, os atores são criados pelo elemento INSTANCE2. Como visto na Figura 4.1, um autômato AGA-T pode ser usado em várias instâncias. O texto contido no elemento INSTANCE2 determina um conjunto de parâmetros que são enviados ao autômato para particularizar eventos e/ou saídas. Há também os parâmetros de inicialização que são os mesmos utilizados no elemento INSTANCE (ver Figura 3.12).

A Figura 4.6 apresenta um exemplo de especificação de uma instância do AGA-T que usa o autômato BONECO2, mostrado nas figuras 4.4 e 4.5. Além dos parâmetros de inicialização, há três parâmetros personalizados dentro do elemento INSTANCE2. Os parâmetros são descritos na forma $NOME$=$VALOR$ e são separados pelo delimitador ponto e vírgula (;).

```
  <STATE ID="INICIO" DESCRIPTION="Inicio">
     <TO STATE="INICIO" PROB="0.5">
         ESPERA;
     </TO>
     <TO STATE="CAMINHA" PROB="0.5" EVENT="SEE :OPONENT">
         WALK;
     </TO>
  </STATE>
  <STATE ID="CAMINHA" DESCRIPTION="Caminhando">
     <TO STATE="LUTA" PROB="0.1" EVENT="REACH :OPONENT">
         FIGHT;
     </TO>
     <TO STATE="CAMINHA" PROB="0.5">
         TURN;
         TURN2;
         TURNFRONT;
     </TO>
     <TO STATE="CAMINHA" PROB="0.1">
         TURNBACK;
         WALK2;
         TURNFRONT;
         WALK;
     </TO>
  </STATE>
  <STATE ID="LUTA" DESCRIPTION="Lutando">
     <TO STATE="INICIO" PROB="1">
         TURNBACK;
         WALK2;
         TURNFRONT;
     </TO>
  </STATE>
</ACTOR2>
```

Figura 4.5: Exemplo da especificação dos estados de um ator AGA-T em AGA2ML

```
<INSTANCE2 ID="JOAO2" ACTOR="BONECO2" X="-0.8" Y="0" Z="0" ROTY="90">
OPONENT=JOAO;
SPEED=1.2;
JUMP=3;
</INSTANCE2>
```

Figura 4.6: Exemplo de instância do AGA-T

5 RESULTADOS OBTIDOS

5.1 Introdução

Para validar o modelo AGA-J, um protótipo em Java foi implementado usando o motor Java3D. As seções seguintes apresentam os resultados de duas animações produzidas por este protótipo. O site `http://www.inf.ufrgs.br/cg/aga/` contém as applets correspondentes para visualização em tempo real.

O protótipo de animação foi ampliado para validar parte do modelo AGA-T, descrito na Seção 4. Desta vez, foi usado o motor gl4java (JAUSOFT, 2001) que é mais leve que o Java3D e permite acesso direto às primitivas do OpenGL. As animações AGA-J foram validadas também neste motor que contém mais recursos, como texturas, por exemplo.

As seções 5.2, 5.3 e 5.4 apresentam animações especificadas em AGA2ML, junto com algumas telas geradas pelo protótipo de animação, a título de exemplo; a Seção 5.5 fala sobre a usabilidade do AGA-J na WWW; a Seção 5.7 demonstra uma forma de extrair informação das instâncias e a Seção 5.8 apresenta um método que se vale da recuperação de informação para adaptar a saída da animação a outros meios de reprodução.

5.2 Simulação de briga de rua

Esta animação descreve uma cena onde quatro instâncias do mesmo ator interagem entre si. Este exemplo demonstra a capacidade de reuso do modelo AGA-J. A animação consiste em duas instâncias que começam a brigar após uma rápida discussão enquanto as outras assistem de longe. Algumas telas da animação estão na Figura 5.1.

O grafo de cena do ator utilizado foi apresentado anteriormente na Figura 3.8 com o autômato correspondente na Figura 3.5. O trecho XML que especifica as instâncias da animação está na Figura 5.2. A instância que dá um soco durante a animação se chama JOHN e o ator se chama DUMMY. Como consequência, todas as instâncias desta animação são do ator DUMMY. As fitas de entrada das instâncias estão em forma textual, dentro das marcas `INSTANCE`. Cada linha corresponde a uma célula no formato:

SÍMBOLO DE ENTRADA,TEMPO,TRANSFORMAÇÃO;

5.3 Sala de estar

Esta animação descreve uma cena onde uma instância do ator DUMMY caminha por uma sala, chuta uma bola, sem querer, e resolve sentar numa cadeira, que é uma instância do ator CHAIR. No topo da sala, há um lustre que balança. Toda a animação sofre uma

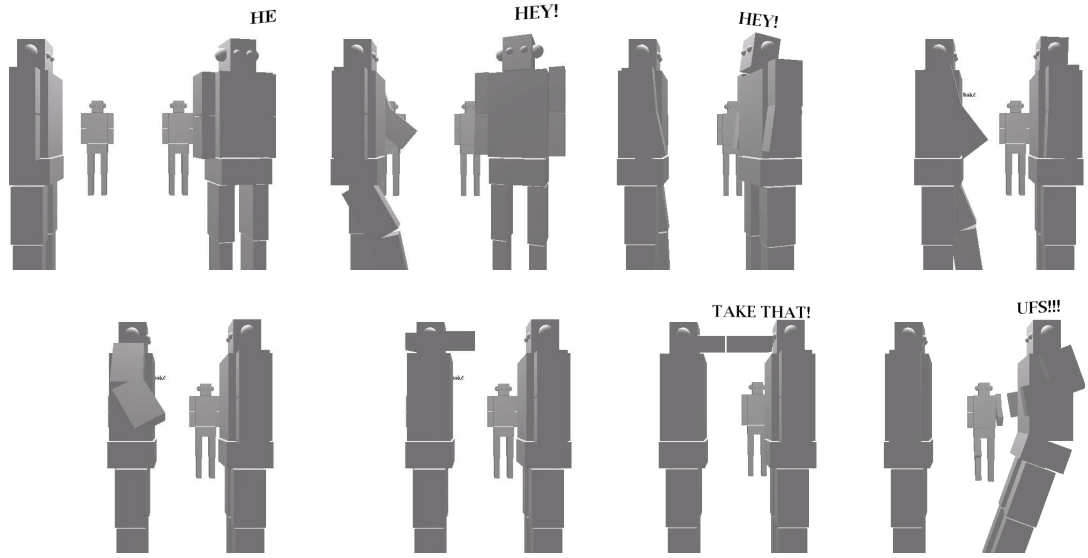


Figura 5.1: Simulação de luta

```
<INSTANCE ID="JOHN" ACTOR="DUMMY" X="-0.8" Y="0" Z="0" ROT="90 0 1 0">
   RW,400,MOV 0 0 0.3;
   RW,400,MOV 0 0 0.15;
   LW,400,MOV 0 0 0.05;
   LW,200,MOV 0 0 0.03;
   PUNCH,150;
   PUNCH,150,MOV 0 0 0.05;
   PUNCH,100;
   STAND,100;
</INSTANCE>

<INSTANCE ID="DUNKY" ACTOR="DUMMY" X="+0.7" Y="0" Z="0" ROT="10 0 1 0">
   PEEK,1200,ROTMOV -100 0 1 0  0 0 +0.15;
   PEEK,200;
   _,400,DELAY;
   _,0,PLAY crunch.wav;
   BEAT,150,MOV 0 0 -0.1;
   _,1000,MOV 0 0 -0.5;
</INSTANCE>

<INSTANCE ID="SPECTATOR1" ACTOR="DUMMY" X="-0.9" Y="0" Z="-4">
   _,1000,DELAY;
   _,850,TEXT FALA IH! vai dar bolo!;
   _,0,TEXT FALA;
</INSTANCE>

<INSTANCE ID="SPECTATOR2" ACTOR="DUMMY" X="+0.7" Y="0" Z="-4">
   _,1850,DELAY;
   _,200,MOV 0 0.2 0;
   _,100,MOV 0 -0.2 0;
   RW,100,MOV 0 0 0.3;
   RW,100,MOV 0 0 0.3;
   LW,100,MOV 0 0 0.3;
   LW,100,MOV 0 0 0.3;
   RW,100,MOV 0 0 0.3;
   RW,100,MOV 0 0 0.3;
</INSTANCE>
```

Figura 5.2: Especificação AGA2ML das instâncias da animação Briga de Rua.

narração por meio de uma instância de um ator-legenda. Este exemplo demonstra a ca-

pacidade de reutilizar os mesmos modelos de atores em animações diferentes[1] e também que o modelo é propenso para representar alguns objetos deformáveis.

A cena mostra o personagem humanóide sofrendo um acidente ao sentar na cadeira. Alguns instantâneos são mostrados na Figura 5.5. O modelo da cadeira está na Figura 5.3. O diagrama do autômato da cadeira está na Figura 5.4 e sua definição XML está na Figura 5.6. A Figura 5.7 mostra a especificação XML das instâncias da animação. Esta animação está usando recursos novos do protótipo, como malhas e texturas.

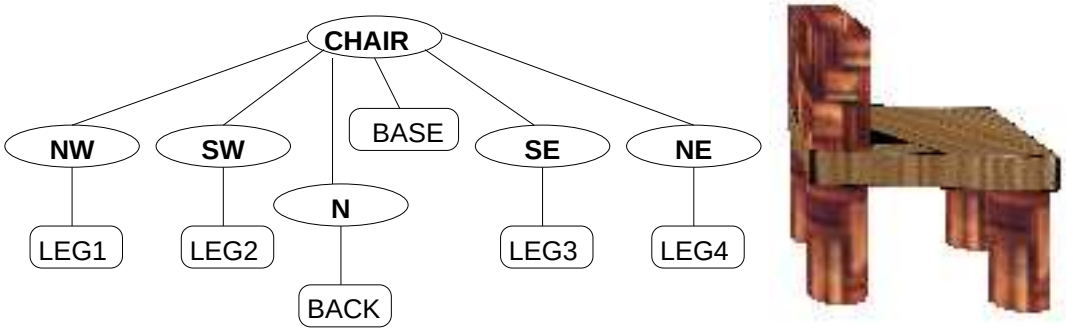


Figura 5.3: Modelo hierárquico do ator `CHAIR`. Retângulos são objetos gráficos. Elipses representam articulações

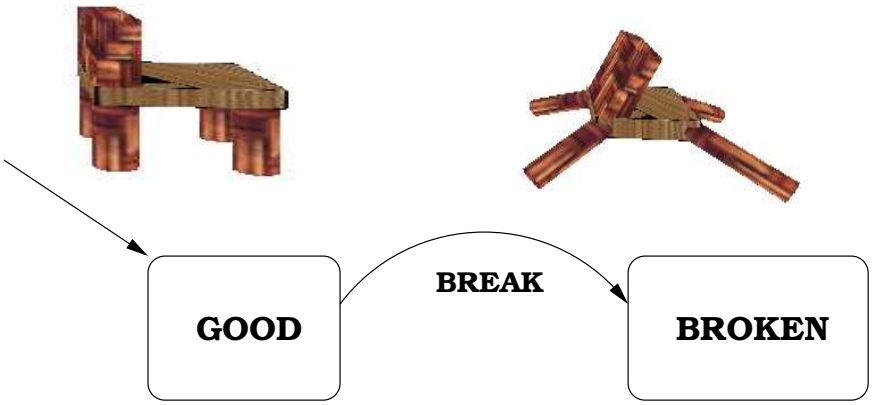


Figura 5.4: Autômato do ator `CHAIR`

5.4 Acidentes Recorrentes

A animação apresentada nesta seção é a utilizada como estudo de caso na Seção 5.7. Ela tem a duração de 16 segundos e sua especificação levou cerca de oito horas para ser construída. Foram usados 4 atores, reaproveitados da animação Sala de Estar, que é apresentada na Seção 5.3 e 10 instâncias. O ator BONECO é instanciado quatro vezes, o ator BOLA três vezes, o ator CHAIR duas vezes e o ator LUSTRE uma vez. O ator BONECO e o ator CHAIR possuem texturas, como parte do experimento.

A cena apresenta a atuação de vários atores humanóides. No primeiro plano, um deles está caminhando pelo cenário e sofre acidentes sucessivos. Primeiro ele tropeça

[1] É uma boa prática colocar cada definição de ator em um arquivo XML separado para poder usá-las como entidades.

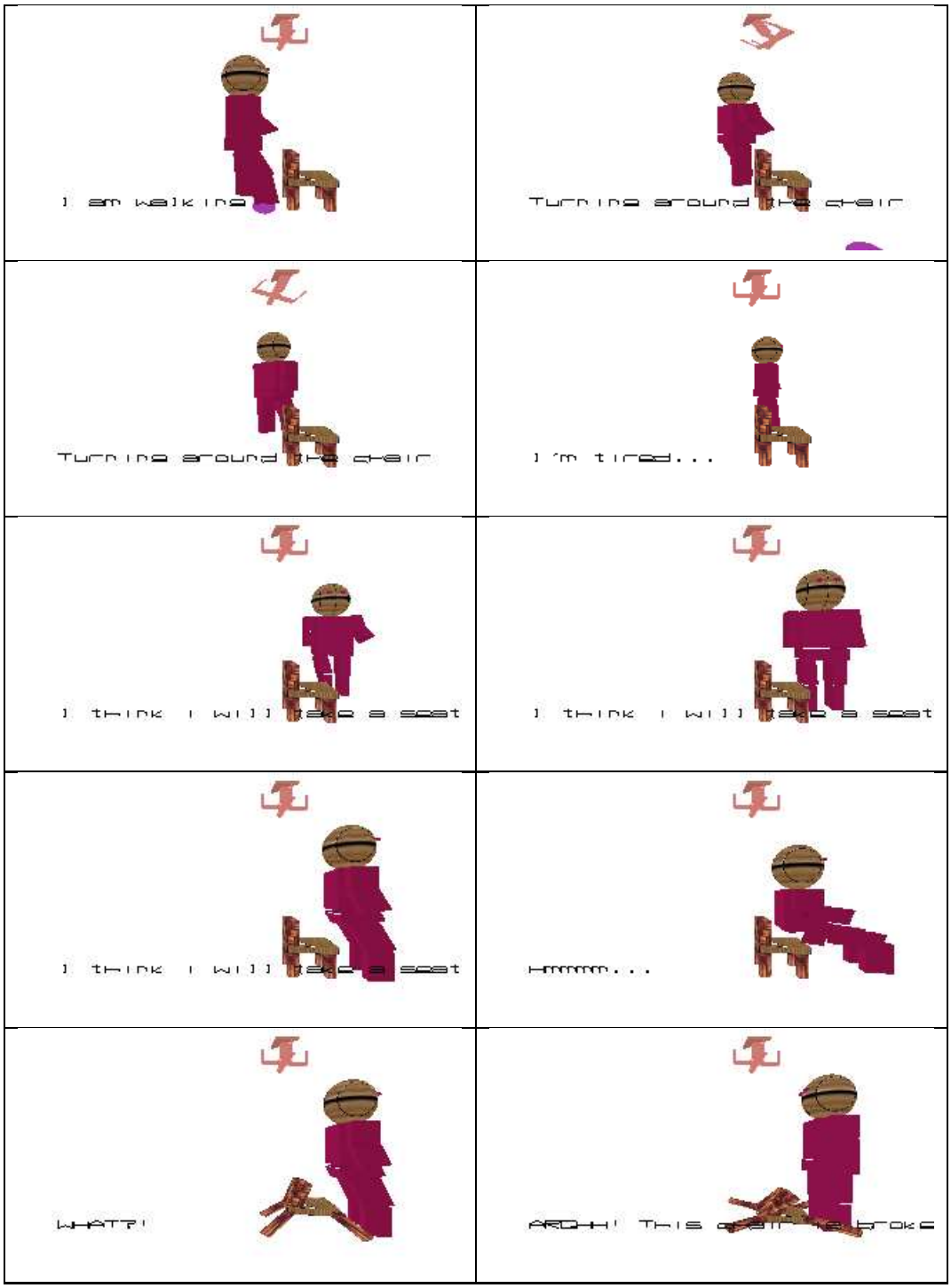


Figura 5.5: JOHN tendo problemas na sala de estar

numa bola e cai no chão. Depois, ele se levanta e tenta seguir em frente, mas uma bola cai do céu, balançando um lustre e acertando a cabeça do ator, que afunda no cenário com o golpe recebido. Logo, ele se recupera e continua caminhando. Quando tudo parece bem, uma bola vem pela direita do cenário e atinge o rosto do ator, que vai para trás sentindo o baque do golpe e depois cai no chão, se mantendo lá até o final da animação. Durante a sina do ator, outros dois humanóides estão conversando de pé no plano de fundo. Depois sentam-se em cadeiras e começam a conversar sobre os acidentes que o outro ator está sofrendo. Perto do final da animação, outro ator humanóide entra correndo na cena pela esquerda, efetua uma estrelinha e sai de cena pela direita.

A Figura 5.8 mostra alguns instantâneos da animação descrita e as figuras 5.9 e 5.10 mostram a especificação XML das instâncias da animação. A complexidade da criação de animações no modelo AGA-J aumenta em função da capacidade de interação entre as instâncias da animação, visto que a sincronização entre suas ações é tratada apenas pelo tempo.

5.5 Uso do AGA-J na WWW

Atualmente, uma série de *plug-ins* e programas auxiliares são exigidos para o usuário poder lidar com os formatos atuais de WEB3D como VRML (ISO/IEC 14772-1, 1997) e X3D (ISO/IEC 19775-200X, 2003), em navegadores populares. Em virtude disso, ainda é comum usar o formato GIF ou AVI para uso e distribuição de animações estáticas pela WWW, visto que os navegadores modernos reproduzem estes formatos nativamente.

O protótipo animador para o AGA-J foi construído na linguagem Java devido ao fato dos navegadores modernos suportarem a exibição de *Java Applets*. Apesar disto,

```
<ACTOR ID="CHAIR" START="GOOD">
   <BOX ID="BASE" X="0" Y="0" SX="0.2" SY="0.05" SZ="0.2" />
   <JOINT ID="NW" X="-0.15" Y="0" Z="-0.15">
      <CYLINDER ID="LEG1" X="0" Y="-0.2" RADIUS="0.05" HEIGHT="0.4" />
   </JOINT>
   <JOINT ID="SW" X="-0.15" Y="0" Z="+0.15">
      <CYLINDER ID="LEG2" X="0" Y="-0.2" RADIUS="0.05" HEIGHT="0.4" />
   </JOINT>
   <JOINT ID="SE" X="+0.15" Y="0" Z="+0.15">
      <CYLINDER ID="LEG3" X="0" Y="-0.2" RADIUS="0.05" HEIGHT="0.4" />
   </JOINT>
   <JOINT ID="NE" X="+0.15" Y="0" Z="-0.15">
      <CYLINDER ID="LEG4" X="0" Y="-0.2" RADIUS="0.05" HEIGHT="0.4" />
   </JOINT>
   <JOINT ID="N" X="0" Y="0" Z="-0.15">
      <BOX ID="BACK" X="0" Y="+0.2" SX="0.2" SY="0.2" SZ="0.05" />
   </JOINT>

   <STATE ID="GOOD" DESCRIPTION="The chair is intact">
      <TO STATE="BROKEN" SYMBOL="BREAK">
         NW ROT 90 1 0 -1;
         SW ROT 90 -1 0 -1;
         SE ROT 90 -1 0 1;
         NE ROT 90 1 0 1;
         N ROTX 35;
      </TO>
   </STATE>
   <STATE ID="BROKEN" DESCRIPTION="The chair is destroyed">
   </STATE>
</ACTOR>
```

Figura 5.6: Especificação AGA2ML do ator AGA-J `CHAIR`

```
<INSTANCE ID="JOHN" ACTOR="DUMMY" X="-0.2" Y="0" Z="0" ROTY="90">
   RW,500,MOV 0.07 0 0;
   RW,500,MOV 0.14 0 0;
   LW,500,ROTYMOV 20  0.2 0 -0.6;
   LW,500,ROTYMOV 40  0.26 0 -0.7;
   RW,500,MOV 0.33 0 -0.8;
   RW,500,MOV 0.36 0 -0.9;
   LW,500,MOV 0.39 0 -1.0;
   LW,500,MOV 0.42 0 -1.1;
   LW,500,ROTYMOV 20  0.5 0 -1.15;
   LW,500,ROTYMOV 0  0.6 0 -1.2;
   LW,500,ROTYMOV -45  0.8 0 -1.05;
   LW,500,ROTYMOV -90  1.0 0 -0.9;
   RW,500,MOV 1.0 0 -0.7;
   RW,500,MOV 1.0 0 0;
   _,300,ROTY 0;
   SIT,2000,MOV 0.7 -0.45 0;
   SIT,200,MOV 1.0 0 0;
   _,200,ROTY 180;
</INSTANCE>

<INSTANCE ID="CHAIR" ACTOR="CHAIR" X="0.5" Y="-0.8" Z="-1.5" ROT="90 0 1 0">
   _,9300,DELAY;
   BREAK,400,MOV 0 -0.2 0;
</INSTANCE>

<INSTANCE ID="LEG" ACTOR="LEGENDA" X="-1" Y="-0.7" Z="1">
   _,1000,TEXT TEXTO I am walking;
   _,3000,TEXT TEXTO Turning around the chair;
   _,2000,TEXT TEXTO I’m tired...;
   _,2000,TEXT TEXTO I think i will take a seat;
   _,1300,TEXT TEXTO Hmmmm...;
   _,200,TEXT TEXTO WHAT?!;
   _,600,TEXT TEXTO ARGHH! This chair is broken!!!;
</INSTANCE>

<INSTANCE ID="LUS" ACTOR="LUSTRE" X="0.2" Y="1.7" Z="0" ROTY="10">
   _,1000,DELAY;
   BDIR,800,DELAY;
   BDIR,500,DELAY;
   BESQ,800,DELAY;
   BESQ,500,DELAY;
</INSTANCE>

<INSTANCE ID="BOL" ACTOR="BOLA" X="0" Y="-1.2" Z="0.1">
   _,300,DELAY;
   _,2000,MOV 1 0 1;
</INSTANCE>
```

Figura 5.7: Especificação AGA2ML das instâncias da animação Sala de Estar.

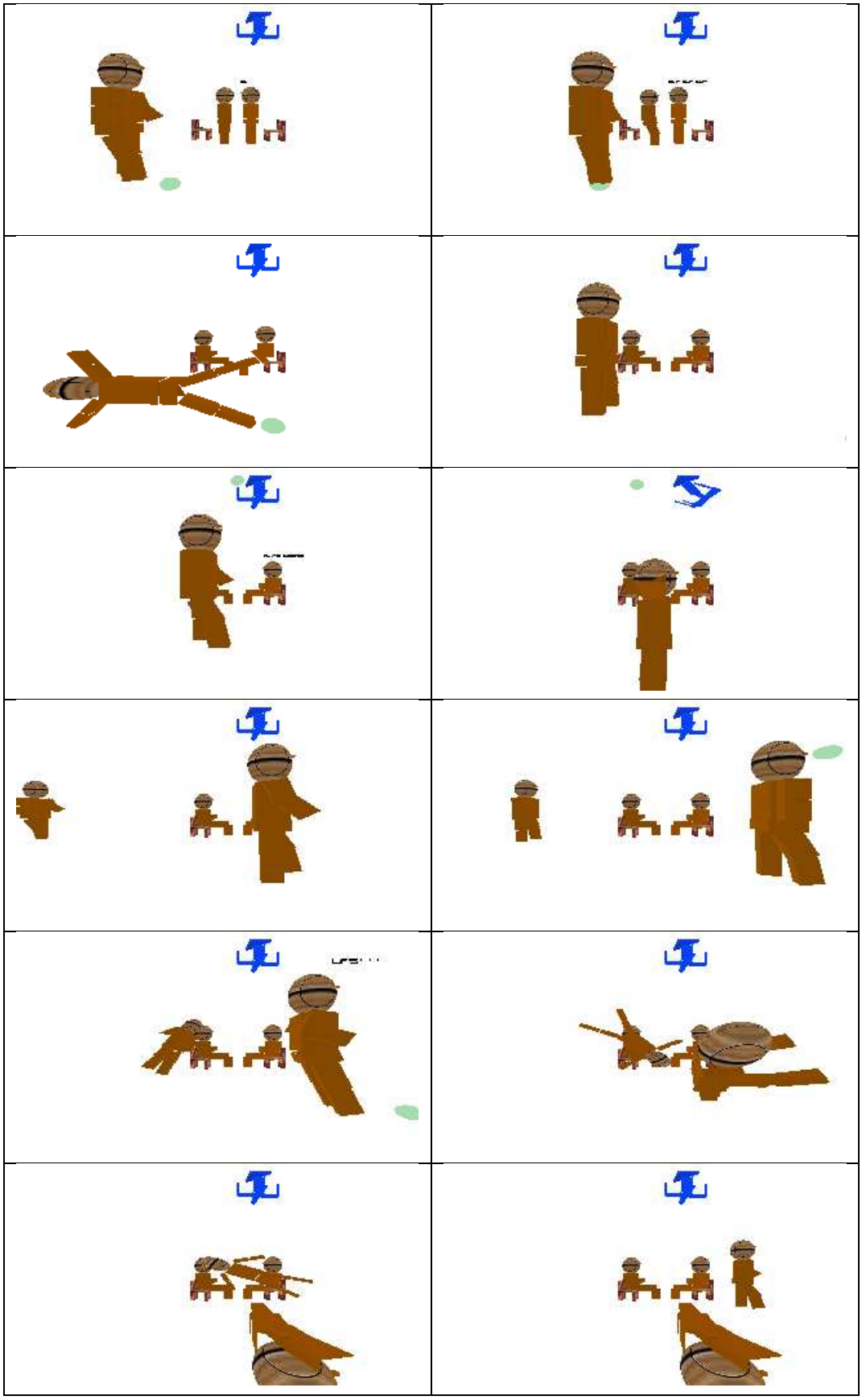


Figura 5.8: Imagens obtidas da animação Acidentes Recorrentes

```
<INSTANCE ID="CAD" ACTOR="CADEIRA" X="-0.8" Y="-0.6" Z="-2.5" ROTY="90">
_,2900,DELAY;
_,150,MOV -0.03 -0.06 0;
</INSTANCE>

<INSTANCE ID="CAD2" ACTOR="CADEIRA" X="+0.8" Y="-0.6" Z="-2.5" ROTY="-90">
</INSTANCE>

<INSTANCE ID="JOAO" ACTOR="BONECO" X="-0.3" Y="0" Z="-2.7" ROTY="90">
_,500,DELAY;
_,500,TEXT FALA OI;
SENTA,2000,MOV -0.5 -0.45 0;
_,300,TEXT FALA BURP!;
_,3000,DELAY;
</INSTANCE>

<INSTANCE ID="PAULO" ACTOR="BONECO" X="+0.3" Y="0" Z="-2.7" ROTY="-90">
_,500,TEXT FALA OI;
_,500,DELAY;
_,1000,TEXT FALA BLA BLA BLA;
SENTA,1500,MOV 0.5 -0.45 0;
_,1500,DELAY;
_,1000,TEXT FALA MUITO PORCAO;
</INSTANCE>

<INSTANCE ID="WALKER" ACTOR="BONECO" X="-1.4" Y="0" Z="0" ROTY="90">
   ANDDIR,300,MOV 0.15 0 0;
   ANDDIR,300,MOV 0.3 0 0;
   ANDESQ,300,MOV 0.45 0 0;
   ANDESQ,300,MOV 0.6 0 0;
   ANDDIR,300,MOV 0.75 0 0;
   QUEDA,600,ROTMOV 0 -90 90 0.5 -0.8 0;
   _,2000,DELAY;
   EMPE,200,ROTMOV 0 0 0 0.5 0 0;
   ANDDIR,300,MOV 0.6 0 0;
   ANDDIR,300,MOV 0.7 0 0;
   ANDESQ,300,MOV 0.8 0 0;
   ANDESQ,300,MOV 0.9 0 0;
   ANDDIR,300,MOV 1.0 0 0;
   ANDDIR,300,MOV 1.1 0 0;
   ANDESQ,150,MOV 1.2 0 0;
   ANDESQ,150,MOV 1.3 0 0;
   QUEDA,200,MOV 1.3 -0.5 0;
   _,2800,MOV 1.3 -7.5 0;
   EMPE,500,MOV 1.3 0 0;
   ANDDIR,400,MOV 1.4 0 0;
   ANDDIR,400,MOV 1.55 0 0;
   ANDESQ,400,MOV 1.75 0 0;
   ANDESQ,400,MOV 2.1 0 0;
   ANDDIR,400,MOV 2.4 0 0;
   ANDDIR,100,MOV 2.5 0 0;
   APANHA,1000,MOV 2 0 0;
   QUEDA,1200,ROTMOV 0 0 90 1.7 -1 0;
</INSTANCE>

<INSTANCE ID="BOL" ACTOR="BOLA" X="-0.65" Y="-1.2" Z="0.1">
   _,1300,DELAY;
   _,4000,MOV 2.2 0 0.4;
</INSTANCE>

<INSTANCE ID="LUS" ACTOR="LUSTRE" X="0.2" Y="1.7" Z="0" ROTY="10">
   _,5950,DELAY;
   BDIR,800,DELAY;
   BDIR,500,DELAY;
   BESQ,500,DELAY;
   BESQ,300,DELAY;
</INSTANCE>
```

Figura 5.9: Especificação AGA2ML das instâncias CAD, CAD2, JOAO, PAULO, WALKER, BOL e LUS da animação Acidentes Recorrentes.

```
<INSTANCE ID="BOLA2" ACTOR="BOLA" X="0" Y="4.5" Z="-2.1">
   _,5350,DELAY;
   _,1000,MOV 0 -3.5 2.1;
   _,1000,MOV -1 0 0;
</INSTANCE>

<INSTANCE ID="BOLA3" ACTOR="BOLA" X="1.7" Y="0.9" Z="0.1">
   _,11500,DELAY;
   _,100,MOV -0.1 0 0;
   _,400,MOV -0.35 0 0;
   _,600,MOV -0.27 -2 0;
   _,1000,MOV 0 -2 0;
</INSTANCE>

<INSTANCE ID="BAILARINO" ACTOR="BONECO" X="-5.4" Y="0" Z="-2" ROTY="90">
_,10000,DELAY;
ANDDIR,1000,MOV 1 0 0;
ANDDIR,700,MOV 2 0 0;
ANDESQ,500,MOV 3 0 0;
ANDESQ,300,MOV 4 0 0;
QUEDA,1500,ROTMOV 0 -90 -180 5 0 0;
EMPE,800,ROTMOV 0 -90 -360 6 0 0;
ANDDIR,300,ROTMOV 0 0 0 7 0 0;
ANDDIR,300,MOV 8 0 0;
</INSTANCE>
```

Figura 5.10: Especificação AGA2ML das instâncias BOLA2, BOLA3 e BAILARINO da animação Acidentes Recorrentes.

as duas versões desenvolvidas do protótipo de animação necessitam que o usuário instale o Java3D *plug-in* (SUN MICROSYSTEMS, 1998) e o GL4Java *plug-in* (JAUSOFT, 2001) para poder reproduzir as animações no navegador. Isto ocorre porque, atualmente, nenhum navegador possui suporte nativo para motores 3D.

Para efeito de ilustração, o Anexo B apresenta um comparativo do espaço de armazenamento ocupado por animações especificadas no modelo AGA-J com animações equivalentes no formato GIF e no formato AVI.

5.6 Recuperação de informação do ator

Como herança do modelo AGA, o modelo AGA-J possui, engendrado na definição do ator, a função descrição, que efetua a associação de descrições semânticas aos estados do ator AGA. Ou seja, é possível explorar o modelo de autômatos finitos no qual o AGA-J foi construído para propiciar um nível de recuperação de informação.

Uma análise de grafo do autômato permite determinar quais são os estados alcançáveis a partir de um estado desejado. (HOPCROFT; MOTWANI; ULLMAN, 2001) Esta análise em combinação com a função descrição provém obtenção de informação sobre as capacidades do ator e a sequência de ações que um ator tem que proceder para que a ação desejada se realize, a partir de um estado atual.

A seguir, é feita uma breve análise do ator apresentado na Figura 3.5, a título de exemplo. A partir de *STAND UP*, todas as ações podem ser realizadas imediatamente com exceção de *PUNCHING*, que requer obrigatoriamente a ação *READY TO PUNCH*. Quando o autômato chega em *BEATEN*, o ator não é capaz de realizar nenhuma ação além do próprio *BEATEN* (a Seção 3.4 diz que todos os estados dos autômatos AGA-J possuem transições reflexivas). Pode-se concluir que *BEATEN* é um forte candidato a estado final. Por simplicidade, é considerado o rótulo do estado como resultado da função descrição.

5.7 Recuperação de informação da instância

A seção anterior mostrou que é possível obter informação sobre as capacidades do ator. Também é possível obter informação sobre as ações das instâncias em uma determinada animação. A animação Acidentes Recorrentes com duração de 16 segundos, usando 4 atores e 10 instâncias em AGA-J foi construída a título de estudo de caso para demonstrar essa característica observada. A Seção 5.4 mostra os dados da animação.

Para possibilitar isso, o animador protótipo suporta a opção do usuário escolher a geração de um arquivo de log que abrange todas as mudanças de estado dos autômatos dos atores que instanciaram os personagens da animação e o tempo de permanência em cada estado, ao decorrer da animação. A Seção 5.7.1 apresenta o log completo da animação. Os dados estão delimitados pelo caractere espaço, com exceção da função descrição de estado, que pode conter qualquer valor. O significado dos dados é apresentado na seguinte ordem: quadro-inicial, quadro-final, tempo inicial em segundos, duração em segundos, identificador da instância e o resultado da função descrição de estado. O identificador do estado é usado para os estados que não definirem a função descrição.

O interessante desta ferramenta é que um simples *parser* de texto pode revelar um histórico independente da animação de uma instância e também um histórico da animação baseado na função descrição de estado. A Tabela 5.1 mostra todo o histórico da animação da instância WALKER extraído do log da animação. Neste caso, foram extraídas e interpretadas todas as linhas do log que referenciavam a instância WALKER. Neste exemplo, não foi feita nenhuma operação de agrupamento, que é mostrada no exemplo a seguir.

As tabelas 5.2 e 5.3 mostram os passos realizados para que seja possível saber quais instâncias sentaram nas cadeiras e o instante em que isso aconteceu. Primeiro, foi feita uma extração de todas as linhas do log que tinham `Sentado` como resultado da função descrição de estado. JOAO e PAULO só estão sentados uma vez, apesar da Tabela 5.2 revelar três entradas para JOAO e três para PAULO. Isto acontece porque as transições reflexivas do autômato também aparecem no arquivo de log para expor o máximo possível de informação sobre a animação. No caso da animação Acidentes Recorrentes, um símbolo da fita levou o autômato de JOAO para o estado SENTADO e dois outros símbolos mantiveram-no no mesmo estado. O mesmo aconteceu com a instância PAULO.

Para deixar mais claro o resultado, foi feito um agrupamento por contiguidade temporal das linhas que contêm a mesma instância, a fim de descobrir quando uma instância sentou e por quanto tempo ela se manteve no estado SENTADO. Isto é mostrado na Tabela 5.3.

5.7.1 Log da animação Acidentes Recorrentes

```
0 72 0.0 2.88 CAD Cadeira inteira
0 12 0.0 0.48 JOAO Parado
0 12 0.0 0.48 PAULO Parado
0 7 0.0 0.28 WALKER Andando com o pe direito
0 32 0.0 1.28 BOL PARADA
0 148 0.0 5.92 LUS Still
0 133 0.0 5.32 BOLA2 PARADA
0 287 0.0 11.48 BOLA3 PARADA
0 250 0.0 10.0 BAILARINO Parado
7 15 0.28 0.32 WALKER Parado
12 25 0.48 0.52 JOAO Parado
12 25 0.48 0.52 PAULO Parado
15 22 0.6 0.28 WALKER Andando com o pe esquerdo
22 30 0.88 0.32 WALKER Parado
25 75 1.0 2.0 JOAO Sentado
25 50 1.0 1.0 PAULO Parado
30 37 1.2 0.28 WALKER Andando com o pe direito
```

```
32 132 1.28 4.0 BOL PARADA
37 52 1.48 0.6 WALKER Caiu no chao
50 87 2.0 1.48 PAULO Sentado
52 102 2.08 2.0 WALKER Caiu no chao
72 76 2.88 0.16 CAD Cadeira inteira
75 82 3.0 0.28 JOAO Sentado
82 157 3.28 3.0 JOAO Sentado
87 125 3.48 1.52 PAULO Sentado
102 107 4.08 0.2 WALKER Parado
107 115 4.28 0.32 WALKER Andando com o pe direito
115 122 4.6 0.28 WALKER Parado
122 130 4.88 0.32 WALKER Andando com o pe esquerdo
125 150 5.0 1.0 PAULO Sentado
130 137 5.2 0.28 WALKER Parado
133 158 5.32 1.0 BOLA2 PARADA
137 145 5.48 0.32 WALKER Andando com o pe direito
145 152 5.8 0.28 WALKER Parado
148 168 5.92 0.8 LUS Right swing
152 156 6.08 0.16 WALKER Andando com o pe esquerdo
156 160 6.24 0.16 WALKER Parado
158 183 6.32 1.0 BOLA2 PARADA
160 165 6.4 0.2 WALKER Caiu no chao
165 235 6.6 2.8 WALKER Caiu no chao
168 181 6.72 0.52 LUS Still
181 193 7.24 0.48 LUS Left swing
193 201 7.72 0.32 LUS Still
235 247 9.4 0.48 WALKER Parado
247 257 9.88 0.4 WALKER Andando com o pe direito
250 275 10.0 1.0 BAILARINO Andando com o pe direito
257 267 10.28 0.4 WALKER Parado
267 277 10.68 0.4 WALKER Andando com o pe esquerdo
275 292 11.0 0.68 BAILARINO Parado
277 287 11.08 0.4 WALKER Parado
287 297 11.48 0.4 WALKER Andando com o pe direito
287 290 11.48 0.12 BOLA3 PARADA
290 300 11.6 0.4 BOLA3 PARADA
292 305 11.68 0.52 BAILARINO Andando com o pe esquerdo
297 300 11.88 0.12 WALKER Parado
300 325 12.0 1.0 WALKER Tomou pau
300 315 12.0 0.6 BOLA3 PARADA
305 312 12.2 0.28 BAILARINO Parado
312 350 12.48 1.52 BAILARINO Caiu no chao
315 340 12.6 1.0 BOLA3 PARADA
325 355 13.0 1.2 WALKER Caiu no chao
350 370 14.0 0.8 BAILARINO Parado
370 377 14.8 0.28 BAILARINO Andando com o pe direito
377 385 15.08 0.32 BAILARINO Parado
```

5.8 Saída adaptável ao meio de reprodução

Uma das características que enriquece qualquer meio de distribuição de informação é a capacidade de adaptação à mídia que vai reproduzir a informação. A WWW possui essa característica, através da especificação HTML.

No AGA-J, bem como no AGA-T é possível usar a função de descrição de estado para agregar informação coletável para ser usada por algum analisador ou *frontend*. O log da animação, comentado na seção anterior, abre a possibilidade de adaptar a animação para outros meios de reprodução porque as entradas contém informação de estado e duração do mesmo em que a instância se encontra. Os dados fornecidos pelo log permitem que cada *frontend* aplique o critério que lhe aprouver para gerar a nova saída.

A Tabela 5.4 apresenta um exemplo hipotético que pôde ser elaborado analisando o conteúdo da Tabela 5.1. Este exemplo mostra a saída da instância WALKER adaptada para uma saída sonora. O trecho do log foi analisado e convertido em comandos para um hipotético reprodutor de voz. O critério utilizado foi a agregação de entradas com

Tabela 5.1: Resultado descritivo da animação da instância WALKER

início (s)	duração (s)	descrição
0.0	0.28	Andando com o pe direito
0.28	0.32	Parado
0.6	0.28	Andando com o pe esquerdo
0.88	0.32	Parado
1.2	0.28	Andando com o pe direito
1.48	0.6	Caiu no chao
2.08	2.0	Caiu no chao
4.08	0.2	Parado
4.28	0.32	Andando com o pe direito
4.6	0.28	Parado
4.88	0.32	Andando com o pe esquerdo
5.2	0.28	Parado
5.48	0.32	Andando com o pe direito
5.8	0.28	Parado
6.08	0.16	Andando com o pe esquerdo
6.24	0.16	Parado
6.4	0.2	Caiu no chao
6.6	2.8	Caiu no chao
9.4	0.48	Parado
9.88	0.4	Andando com o pe direito
10.28	0.4	Parado
10.68	0.4	Andando com o pe esquerdo
11.08	0.4	Parado
11.48	0.4	Andando com o pe direito
11.88	0.12	Parado
12.0	1.0	Tomou pau
13.0	1.2	Caiu no chao

Tabela 5.2: Análise da animação que revela quando as instâncias sentam nas cadeiras

início (s)	duração (s)	instância
1.0	2.0	JOAO
2.0	1.48	PAULO
3.0	0.28	JOAO
3.28	3.0	JOAO
3.48	1.52	PAULO
5.0	1.0	PAULO

Tabela 5.3: Operação de agrupamento por contiguidade temporal na análise da animação que revela quando as instâncias sentam nas cadeiras

início (s)	duração (s)	instância
1.0	5.28	JOAO
2.0	4.0	PAULO

saídas similares da função descrição de estado. Foram agregadas as entradas contíguas

com a mesma descrição de estado e o conjunto de descrições `Andando com o pé direito`, `Andando com o pé esquerdo` e `Parado`.

Tabela 5.4: Animação da instância WALKER adaptada para uma saída sonora hipotética

comando	duração (s)
FALE CAMINHANDO	RAPIDO
FALE CAIU NO CHAO	NORMAL
FALE CAMINHANDO	NORMAL
FALE CAIU NO CHAO	LENTO
FALE CAMINHANDO	NORMAL
FALE TOMOU PAU	RAPIDO
FALE CAIU NO CHAO	RAPIDO

O fato de usar um log para gerar outros tipos de saída torna trivial a implementação de uma saída textual, como mostra a Figura 5.11. Outras informações podem ser agregadas às entradas do log para aumentar as possibilidades do meio alternativo de saída, como por exemplo, o símbolo do alfabeto de saída usado para disparar a transição que gerou o novo estado e dados relativos à posição e orientação da instância em relação ao cenário.

WALKER caminha rapidamente, caiu no chão, se levantou e caminha normalmente.
De novo, caiu no chão lentamente, se levantou novamente e caminha normalmente.
WALKER foi golpeado rapidamente e caiu no chão.

Figura 5.11: Animação da instância WALKER adaptada para uma espécie de saída textual

6 CONCLUSÕES

O modelo AGA-J e a proposta do modelo AGA-T são novas alternativas para prover animações 3D de figuras humanóides, sendo que o modelo AGA-J é o modelo AGA (ACCORSI; MENEZES, 2000) sob uma nova ótica. Como no modelo AGA, a definição do alfabeto de saída como um conjunto de transformações na estrutura do ator contribui para a transmissão de dados, pois um mesmo objeto pode ser apresentado em diversos momentos da animação sem a necessidade de replicação. A forma de codificação de cada objeto é independente da estrutura dos modelos. Apesar de existirem alternativas para prover animações na WWW, algumas questões relacionadas à escassez de animações em 3D, ao espaço de armazenamento necessário para representá-las, manutenção de seu conteúdo e suporte à recuperação de informação, têm motivado esforços para melhorar os processos de criação, manutenção, apresentação e consulta de animações.

Seguindo as bases do modelo AGA, foi mantida a escolha da Máquina de Mealy em vez da Máquina de Moore como modelo de AFS para reduzir a explosão de estados, evitando a criação de estados irrelevantes, com o intuito de não degradar a função descrição de estado. A Figura 6.1 demonstra a flexibilidade da Máquina de Mealy em relação à Máquina de Moore usando como exemplo uma variação do autômato AGA-J do ator `DUMMY`.

A proposta do modelo AGA-T é um modelo orientado a tarefas para gerar animações não-determinísticas utilizando um formalismo de autômatos probabilísticos para o controle das tarefas executadas pelos personagens. Ele se vale do modelo AGA-J para controlar a representação e os movimentos dos personagens. Isto pode ser considerado parte de um sistema de animação implementado em níveis de abstração bem definidos.

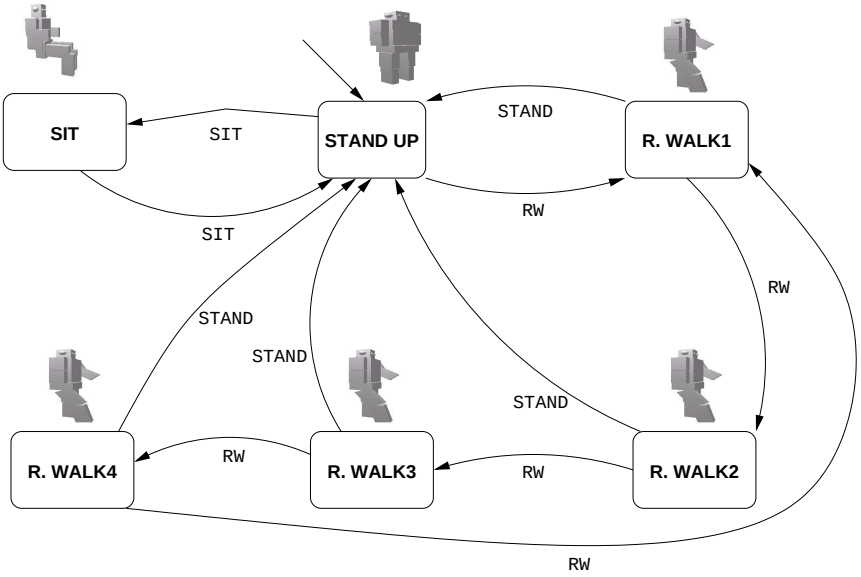
6.1 Contribuições

As metas propostas apresentadas na Seção 1.2 foram atingidas em sua totalidade, gerando as contribuições apresentadas nesta seçao. No final desta seção é apresentado um resultado que transcende os objetivos propostos.

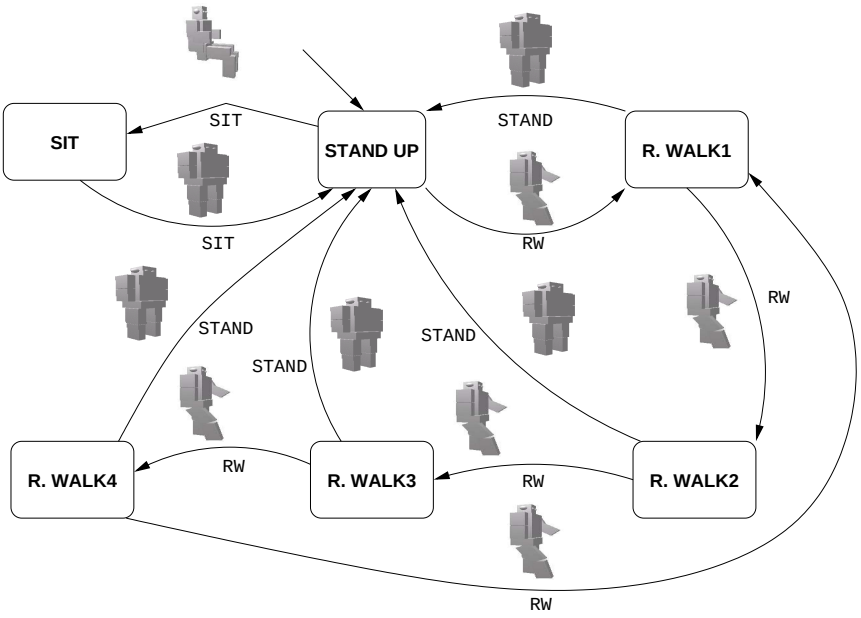
Redução no espaço de armazenamento

As características da linguagem AGA2ML contribuem com a redução do espaço de armazenamento e da transmissão de dados no servidor HTTP quanto ao desenvolvimento de várias animações, pois as estruturas definidas para especificação de atores, fitas e instâncias podem ser compartilhadas por diversas animações.

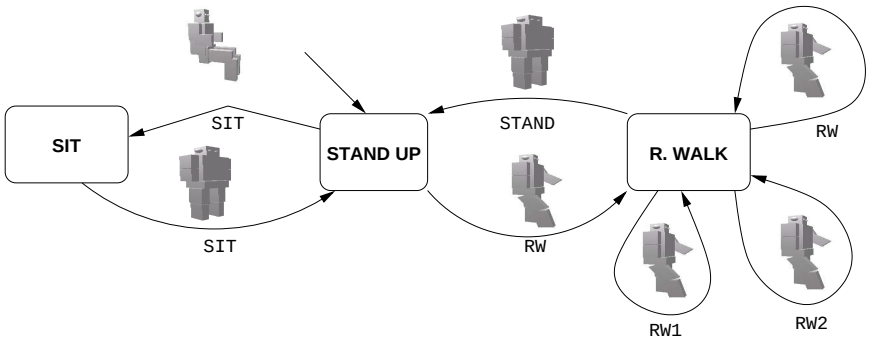
O AGA-J armazena apenas o conjunto de corpos articulados e as poses dos atores. Desta maneira, o mesmo ator pode repetir seus movimentos em diversos momentos da



(a) Moore com 3 estados a mais



(b) Mealy equivalente



(c) Mealy com redução de estados

Figura 6.1: Máquina de Mealy: evitando estados irrelevantes

animação sem a necessidade de codificá-los novamente. Todas as instâncias de um mesmo ator compartilham o mesmo alfabeto de saída, reduzindo o espaço de armazenamento necessário. O tamanho da especificação em AGA2ML varia de acordo com a complexidade dos atores e o comprimento de suas fitas de entrada. Portanto, o tamanho da animação especificada em AGA2ML não cresce em função do número de quadros-chave como por exemplo, o GIF e o AVI, mas sim, de acordo com a complexidade de cada tipo de ator, suas texturas e as transformações nas suas articulações associadas ao alfabeto de saída.

Recuperação de informação

A estrutura básica dos autômatos finitos utilizados como base nos modelos propostos suporta a modelagem de estados nos atores que correspondem a condições semanticamente relevantes durante a animação. Desta forma, a descrição semântica, mapeada a partir da função descrição, possibilita a recuperação de informação relativa ao conteúdo da animação baseada na verificação dos estados dos atores. Como o mapeamento semântico está associado aos estados dos atores independentemente da fita de entrada, qualquer animação em que eles participem está automaticamente mapeada para consulta.

É bom notar que é possível construir um autômato finito com saída contendo apenas um estado, gerando todas as saídas desejadas através de transições reflexivas. Durante a construção deste trabalho, foi observado em alunos a tendência em colocar estados nos autômatos. A colocação de estados no autômato é o que, de fato, propicia a recuperação de informação.

Reuso de componentes

As definições dos atores no modelo AGA-J encapsulam as propriedades estéticas e comportamentais dos objetos por eles modelados. Essa característica contribui para que os componentes da animação sejam facilmente reutilizados por outras, pois uma vez especificado o ator, as animações podem reutilizá-lo atribuindo fitas de entradas diferentes para adaptá-lo aos seus contextos. A fita de entrada é outro componente dos modelos que pode ser reutilizada sempre que o mesmo comportamento for desejado para o ator. A fita de entrada pode ser reusada em outros atores com alfabeto de entrada compatível.

O encapsulamento encontrado nos modelos propostos cria a possibilidade de formação de bibliotecas de atores e fitas de entrada que podem ser reutilizadas em várias animações. A estrutura da linguagem AGA2ML suporta essa possibilidade devido ao mecanismo desenvolvido para a criação de instâncias e os recursos herdados do XML para a criação de entidades externas.

Manutenção da animação

A manutenção das animações é facilitada de várias maneiras pelos modelos propostos. A alteração dos atores tanto para a função programa do autômato quanto para os objetos do alfabeto de saída podem ser realizadas de forma independente, conservando inalterados os demais componentes da animação, como exemplos: inclusão ou exclusão de estados e modificações nos objetos do alfabeto de saída. Essa característica subsidia a construção de processos automatizados para a manutenção das animações.

O AGA-J é um modelo recursivo e pode ser usado como camada para uma implementação dele mesmo. Como demonstrado no Capítulo 4, a proposta do modelo AGA-T toma vantagem disso, agindo como camada superior sobre o AGA-J, aproveitando na íntegra todas as características deste.

Saída adaptável a outros meios de reprodução

Um resultado que transcende aos objetivos esperados é a possibilidade de gerar saída adaptável a outros meios de reprodução. O formalismo de autômatos finitos com saída usado para a especificação de animações provê ferramentas que possibilitam sua auditoria. Estas mesmas ferramentas podem ser usadas por *frontends* para adaptar a saída gerada a outros meios de reprodução, de acordo com o nível de detalhe desejado, como mostra a Seção 5.8. A estrutura da linguagem AGA2ML suporta essa possibilidade devido aos recursos herdados do XML para a criação de entidades externas.

6.2 Limitações

O modelo AGA-J herdou algumas limitações do modelo AGA, visto que, como dito antes, o modelo AGA-J constitui-se de uma nova ótica do modelo AGA, aproveitando as características do mesmo.

Uma das características que enriquece uma animação é a capacidade de interação entre seus atores. Apesar de todas as vantagens oferecidas pelo modelo AGA-J, experimentos com alunos de disciplinas relacionadas constataram que o modelo é mais adequado como fundação para modelos de mais alto nível do que a especificação manual de animações que apresenta o problema da sincronização entre os atores que é tratada apenas pelo tempo, tornando difícil e trabalhosa a manutenção de animações complexas em AGA2ML, principalmente onde sejam comuns as interações entre os atores.

No AGA-J, pode-se notar que parte da especificação da animação fica no autômato e parte fica na fita de entrada do ator, através das funções de transformação (ex.: mover, girar, sumir). Em parte, isso facilita o reaproveitamento do ator em animações diferentes porque cada animação pode ter uma instância personalizada do mesmo ator, mas isso revela uma limitação quando há a necessidade de reaproveitar sequências de movimento junto com o ator. Um bom exemplo é a geração de multidões, onde vários atores compartilham os mesmos movimentos em instantes diferentes. Isto foi uma grande motivação para a proposta do modelo AGA-T.

6.3 Produção Científica

Até o momento do fechamento deste texto, esta dissertação de mestrado resultou na publicação de um artigo que descreve o modelo AGA-J (MARTINS et al., 2004).

6.4 Trabalhos Futuros

O experimento representativo correspondente ao estudo de caso apresentado na Seção 5.4 levou cerca de oito horas para ser criado, apesar do tempo total de exibição da animação resultante ocupar apenas 16 segundos. Uma ferramenta de edição pode solucionar boa parte dos problemas que o projetista de animação passa por trabalhar diretamente com um modelo de controle concreto.

O reaproveitamento de atores é comprometido quando animações curtas reusam modelos de atores com estados em excesso, caracterizando um possível desperdício de recursos demandados durante a simulação dos autômatos correspondentes. Este problema pode ser solucionado por uma arquitetura que permita suporte a Orientação a Objetos na criação de atores, podendo reaproveitar apenas as partes desejadas do autômato e da fita

de entrada.

Investigar possíveis algoritmos de minimização de autômatos finitos com saída que sejam aplicáveis ao modelo AGA-J para desenvolver uma ferramenta a fim de eliminar dados em excesso na especificação da animação.

Investigar o desenvolvimento de *frontends* para o uso da saída adaptável a outros meios de reprodução que o modelo AGA-J oferece. Descobrir a potencialidade das informações que podem ser agregadas às entradas do log gerado pelo protótipo para aumentar as possibilidades dos meios de saída usados. Exemplos possíveis são o símbolo do alfabeto de saída usado para disparar a transição que gerou o novo estado e dados relativos à posição e orientação do ator em relação ao cenário.

Aplicar mais investigações e experimentos no modelo AGA-T a fim de atingir um grau maior de maturação. Investigar quais características de alto nível devem pertencer a uma ferramenta de auxílio ao projetista de animação e quais que devem fazer parte do AGA-T. Investigar a possibilidade de estender o modelo AGA-T para torná-lo recursivo, suportando múltiplas camadas, assim como no modelo AGA-J. Investigar meios para um ator AGA-T controlar vários atores AGA-J em paralelo, em vez de apenas um.

Pesquisar modelos que tratem o problema da sincronização entre atores. Investigar quais questões relacionadas cabem à uma ferramenta de edição, quais cabem à proposta do modelo AGA-T e quais podem caber a um modelo de mais alto nível.

Estender adequadamente os modelos AGA-J e AGA-T para conceitos não determinísticos.

Investigar um projeto de extensões para atuar como camadas superiores ao modelo AGA-T para abstrair a especificação da animação. Um exemplo a ser considerado é uma possível extensão chamada AGA-B (*AGA for Behaviour oriented animation*), que será uma camada para controle de animação comportamental.

REFERÊNCIAS

ACCORSI, F. **Animação Bidimensional para World Wide Web Baseada em Autômatos Finitos**. 2002. 112 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

ACCORSI, F.; MENEZES, P. F. B. AGA - Animação Gráfica Baseada na Teoria de Autômatos. In: WORKSHOP ON FORMAL METHODS, 2000, Paraíba, Brazil. **Proceedings...** [S.l.: s.n.], 2000. v.3, p.122–127.

ALUR; RAJEEV; DILL; L., D. A theory of timed automata. **Theoretical Computer Science**, [S.l.], v.126, p.183–235, 1994.

ARAÚJO, A. d. A.; GUIMARÃES, S. J. F. Recuperação de informação visual com base no conteúdo em imagens e vídeos. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v.7, n.2, p.46–71, 2000.

BADLER, N.; BINDIGANAVALE, R.; BOURNE, J.; ALLBECK, J.; SHI, J.; PALMER, M. Real time virtual humans. In: INTERNATIONAL CONFERENCE ON DIGITAL MEDIA FUTURES, 1999, Bradford, UK. **Proceedings...** [S.l.: s.n.], 1999.

BADLER, N. I.; REICH, B. D.; WEBBER, B. L. Towards Personalities for Animated Agents with Reactive and Planning Behaviors. In: CREATING PERSONALITIES FOR SYNTHETIC ACTORS, 1997, London, UK. **Proceedings...** Berlin: Springer-Verlag, 1997. p.43–57.

BEAUQUIER, D. On probabilistic timed automata. **Theoretical Computer Science**, [S.l.], v.292, n.1, p.65–84, 2003.

CAMPANI, C. A. P. **Avaliação da Compressão de Dados e da Qualidade de Imagem em Modelos de Animação Gráfica para Web**: uma nova abordagem baseada em complexidade de kolmogorov. 2005. 111 f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

DONIKIAN, S. HPTS: a behaviour modelling language for autonomous agents. In: INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2001, Montreal, Canada. **Proceedings...** New York: ACM Press, 2001. p.401–408.

DONIKIAN, S.; RUTTEN, E. Reactivity, Concurrency, Data-flow and Hierarchical Preemption for Behavioural Animation. In: WORKSHOP ON PROGRAMMING PARADIGMS IN GRAPHICS, 1995. **Proceedings...** [S.l.: s.n.], 1995. p.137–153.

ESAKOV, J.; BADLER, N. I.; JUNG, M. An investigation of language input and performance timing for task animation. In: GRAPHICS INTERFACE, 1989, London, Canada. **Proceedings...** New York: ACM Press, 1989. p.19–23.

FIUME, E.; TSICHRITZIS, D.; DAMI, L. A Temporal Scripting Language for Object-Oriented Animation. In: EUROGRAPHICS, 1987, Amsterdam, The Netherlands. **Proceedings...** Amsterdam: Elsevier Science, 1987. p.283–294.

GÓMEZ, J. E. Twixt: a 3d animation system. In: EUROGRAPHICS, 1984, Amsterdam, The Netherlands. **Proceedings...** Amsterdam: Elsevier Science Publishers, 1984. p.121–134.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. **Introduction to Automata Theory, Languages and Computation**. 2.ed. Boston: Addison-Wesley, 2001. 521p.

HUANG, Z.; ELIENS, A.; VISSER, C. STEP: a scripting language for embodied agents. In: WORKSHOP ON LIFELIKE ANIMATED AGENTS, TOOLS, AFFECTIVE FUNCTIONS AND APPLICATIONS, 2002, Tokyo, Japan. **Proceedings...** [S.l.: s.n.], 2002. p.46–51.

HUANG, Z.; ELIENS, A.; VISSER, C. XSTEP: a markup language for embodied agents. In: INTERNATIONAL CONFERENCE ON COMPUTER ANIMATION AND SOCIAL AGENTS, 16., 2003. **Proceedings...** Los Alamitos: IEEE Computer Society, 2003.

ISO/IEC 14772-1. **The Virtual Reality Modeling Language (VRML)**. Disponível em: <http://www.web3d.org/Specifications/VRML97>. Acesso em: março 2005.

ISO/IEC 19775-200X. **Extensible 3D (X3D)**. Disponível em: <http://www.web3d.org/technicalinfo/specifications/ISO_IEC_19775/>. Acesso em: março 2005.

JAUSOFT. **GL4Java**. Disponível em: <http://www.jausoft.com/gl4java.html>. Acesso em: março 2005.

LAMARCHE, F.; DONIKIAN, S. The orchestration of behaviours using resources and priority levels. In: COMPUTER ANIMATION AND SIMULATION, 2001, Manchester, UK. **Proceedings...** London: Springer-Verlag, 2001. p.171–182.

LIMA BICHO, A. de; RAPOSO, A. B.; MAGALHAES, L. P. Control of Articulated Figures Animations Using Petri Nets. In: BRAZILIAN SYMPOSIUM ON COMPUTER GRAPHICS AND IMAGE PROCESSING, 14, 2001, Florianópolis, Brazil. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.200–207.

MACHADO, J. H. A. P. **Hyper-Automaton**: hipertextos e cursos na web usando autômatos finitos com saída. 2000. 148p. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

MACROMEDIA. **Macromedia Flash Application Development Center**. Disponível em: <http://www.macromedia.com/ desdev/ mx/ flash/>. Acesso em: maio 2002.

MAGALHAES, L. P.; RAPOSO, A. B. Animation Modeling with Petri Nets. **Computer & Graphics**, London, v.22, n.6, p.735–743, 1988.

MAGEE, J.; PRYCE, N.; GIANNAKOPOULOU, D.; KRAMER, J. Graphical Animation of Behavior Models. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2000, Limerick, Ireland. **Proceedings...** [S.l.: s.n.], 2000.

MARINO, G.; MORASSO, P.; ZACCARIA, R. NEM: a language for animation of actors and objects. In: EUROGRAPHICS, 1985. **Proceedings...** Amsterdam: Elsevier Science, 1985. p.129–142.

MARTINS, J. D. F.; NEDEL, L. P.; MENEZES, P. F. B.; ACCORSI, F. An Automata-based Animation Engine to Control Articulated Characters. In: SYMPOSIUM ON VIRTUAL REALITY, 2004, São Paulo, Brazil. **Proceedings...** São Paulo: SBC, 2004. p.315–326.

MENEZES, P. F. B. **Linguagens Formais e Autômatos**. Porto Alegre: Sagra Luzatto, 2000. (Livros Didáticos, v.4).

MENEZES, P. F. B.; HAEUSLER, E. H. **Teoria das Categorias para Ciência da Computação**. Porto Alegre: Sagra Luzatto, 2001. (Livros Didáticos, v.12).

MURATA, T. Petri Nets: properties, analysis and applications. **Proceedings of the IEEE**, Piscataway, v.77, n.4, p.541–580, April 1989.

O'DONNELL, T. J.; OLSON, A. J. GRAMPS – A graphics language interpreter for real-time, interactive, three-dimensional picture editing and animation. **SIGGRAPH Computer Graphics**, New York, v.15, n.3, p.133–142, 1981.

PARENT, R. **Computer Animation**: algorithms and techniques. San Francisco: Morgan Kaufmann, 2002.

RAO, A. S. AgentSpeak(L): BDI agents speak out in a logical computable language. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD, 1996, Eindhoven, The Netherlands. **Proceedings...** Berlin: Springer-Verlag, 1996. v.7.

RUTTKAY, Z.; HUANG, Z.; ELIENS, A. Reusable Gestures for Interactive Web Agents. In: INTERNATIONAL WORKSHOP INTELLIGENT VIRTUAL AGENT, 4., 2003, Kloster Irsee, Germany. **Proceedings...** Berlin: Springer-Verlag, 2003. p.80–87.

SUN MICROSYSTEMS. **Java Technology**. Disponível em: <http://java.sun.com/>. Acesso em: março 2005.

SUN MICROSYSTEMS. **Java3D API**. Disponível em: <http://java.sun.com/products/java-media/3D/>. Acesso em: março 2005.

THALMANN, D. Autonomy and Task-Level Control for Virtual Actors. **Programming and Computer Software**, New York, v.4, 1995.

THALMANN, N. M.; THALMANN, D. **Computer Animation Theory and Practice**. Tokyo: Springer-Verlag, 1985.

THALMANN, N. M.; THALMANN, D. Computer animation. **ACM Computing Surveys**, New York, v.28, n.1, p.161–163, 1996.

TORRES, J. A. R.; NEDEL, L. P.; BORDINI, R. H. Using the BDI Architecture to produce autonomous characters in virtual worlds. In: INTERNATIONAL WORKSHOP ON INTELLIGENT VIRTUAL AGENTS, 2003, Kloster Irsee, Germany. **Proceedings...** Heidelberg: Springer-Verlag, 2003. v.4, p.197–201.

VERHOEVEN, J. **Movies 13**. Disponível em: <http://jansfreeware.com/>. Acesso em: março 2005.

VIRTUALDUB.ORG. **VirtualDub**. Disponível em: <http://www.virtualdub.org/>. Acesso em: março 2005.

ZELTZER, D. Motor Control Techniques for Figure Animation. **IEEE Computer Graphics and Applications**, Los Alamitos, v.2, n.9, p.53–59, 1982.

APÊNDICE A REVISÃO SOBRE AUTÔMATOS FINITOS

Autômato Finito é um sistema de estados finitos (portanto possui um número finito e predefinido de estados) o qual constitui um modelo computacional do tipo seqüencial muito comum em diversos estudos teórico-formais da Computação e Informática, com destaque para Linguagens Formais, Compiladores, Semântica Formal e Modelos para Concorrência.

A.1 Autômato Finito Determinístico

Um Autômato Finito Determinístico ou simplesmente Autômato Finito pode ser visto como uma máquina constituída, basicamente, de três partes:

1. Fita. Dispositivo de entrada que contém a informação a ser processada;

2. Unidade de Controle. Reflete o estado corrente da máquina. Possui uma unidade de leitura (cabeça da fita) a qual acessa uma célula da fita de cada vez e movimenta-se exclusivamente para a direita;

3. Programa, Função Programa ou Função de Transição. Função que comanda as leituras e define o estado da máquina.

A fita é finita, sendo dividida em células, cada uma das quais armazena um símbolo. Os símbolos pertencem a um alfabeto de entrada. Não é possível gravar sobre a fita (e não existe memória auxiliar). Inicialmente, a palavra a ser processada (ou seja, a informação de entrada para a máquina) ocupa toda a fita.

A unidade de controle possui um número finito e predefinido de estados, originando o termo controle finito. A unidade de controle lê o símbolo de uma célula de cada vez. Após a leitura, a cabeça da fita move-se uma célula para a direita. Inicialmente, a cabeça está posicionada na célula mais à esquerda da fita, como ilustrado na Figura A.1.

O programa é uma função parcial tal que: dependendo do estado corrente e do símbolo lido, determina o novo estado do autômato.

Um Autômato Finito Determinístico M é uma 5-upla ordenada: $M = (\Sigma, Q, \delta, q_0, F)$ na qual:

1. Σ é um alfabeto de símbolos de entrada, ou simplesmente alfabeto de entrada;

2. Q é um conjunto de estados possíveis do autômato o qual é finito;

3. δ é uma função programa ou simplesmente programa, ou ainda função de transição: $\delta = Q \times \Sigma \rightarrow Q$ a qual é uma função parcial. Supondo que a função programa é

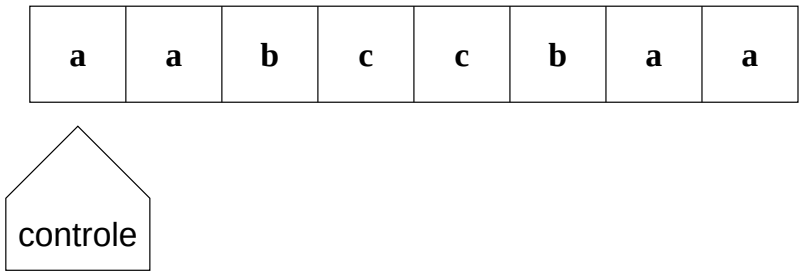


Figura A.1: Autômato finito como uma máquina com controle finito.

definida para um estado p e um símbolo a, resultando no estado q, então: $\delta(p, a) = q$ é uma transição do autômato;

4. q_0 é um elemento distinguido de Q, denominado estado inicial;
5. F é um subconjunto de Q, denominado conjunto de estados finais.

Um autômato finito pode ser representado na forma de um diagrama no qual:

1. estados são nodos, representados por círculos;
2. transições são arestas, ligando os correspondentes nodos. Por exemplo, uma transição do tipo $\delta(q, a) = p$ é como ilustrada na Figura A.2;
3. estados iniciais e finais são representados de forma distinta dos demais, como ilustrado na Figura A.3;
4. transições paralelas (mesmos nodos origem e destino) podem alternativamente ser representadas como ilustrado na Figura A.4 (supondo que $\delta(q, a) = p$ e $\delta(q, b) = p$).

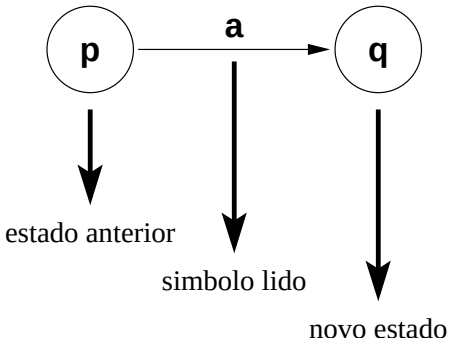


Figura A.2: AFD: representação gráfica de uma transição.

A computação de um autômato finito, para uma palavra de entrada w, consiste na sucessiva aplicação da função programa para cada símbolo de w (da esquerda para a direita) até ocorrer uma condição de parada.

Um autômato finito sempre pára ao processar qualquer entrada pois, como qualquer palavra é finita, e como um novo símbolo da entrada é lido a cada aplicação da função programa, não existe a possibilidade de ciclo (loop) infinito.

A parada do processamento de um autômato finito para uma entrada w pode ser de duas maneiras:

Figura A.3: AFD: representação de estados inicial e final.

Figura A.4: AFD: representações alternativas para transições paralelas.

1. Aceita a entrada w. Após processar o último símbolo da fita, o autômato finito assume um estado final;

2. Rejeita a entrada w. São duas possibilidades:

   (a) após processar o último símbolo da fita, o autômato finito assume um estado não-final;

   (b) em algum momento, ao longo do processamento de w, a função programa é indefinida para o argumento (estado corrente e símbolo lido da fita).

A.2 Autômato Finito Não-Determinístico

O não-determinismo é uma importante generalização dos modelos de máquinas, sendo de fundamental importância no estudo dos Modelos para Concorrência, da Teoria da Computação e das Linguagens Formais, entre outros.

A facilidade de não-determinismo para autômatos finitos é expressa no programa, que é uma função parcial tal que: dependendo do estado corrente e do símbolo lido, determina um conjunto de estados do autômato.

Visto como uma máquina composta por fita, unidade de controle e programa, um Autômato Finito Não-Determinístico assume um conjunto de estados alternativos, como se houvesse uma multiplicação da unidade de controle, uma para cada alternativa, processando independentemente, sem compartilhar recursos com as demais. Assim, o processamento de um caminho não influi no estado, símbolo lido e posição da cabeça dos demais caminhos alternativos.

Um Autômato Finito Não-Determinístico M é uma 5-upla ordenada:
$M = (\Sigma, Q, \delta, q_0, F)$ na qual:

1. Σ é um alfabeto de símbolos de entrada, ou simplesmente alfabeto de entrada;

2. Q é um conjunto de estados possíveis do autômato o qual é finito;

3. δ é uma função programa ou simplesmente programa, ou ainda função de transição: $\delta = Q \times \Sigma \rightarrow 2^Q$ a qual é uma função parcial. Supondo que a função programa é definida para um estado p e um símbolo a, resultando no conjunto de estados $\{q_1, q_2, \ldots, q_n\}$, então: $\delta(p, a) = \{q_1, q_2, \ldots, q_n\}$ é uma transição do autômato;

4. q_0 é um elemento distinguido de Q, denominado estado inicial;

5. F é um subconjunto de Q, denominado conjunto de estados finais.

Portanto, excetuando-se pela função programa δ, as componentes Σ, Q, q_0 e F são como na definição do autômato finito determinístico.

A representação da função programa como um diagrama é análoga à do autômato finito determinístico. Assim, a representação diagramática de uma transição do tipo: $\delta(p, a) = \{q_1, q_2, \ldots, q_n\}$ resulta em diversas arestas etiquetadas por a, com origem em p, e com destino em cada estado $q_1, q_2, \ldots, q_n$, como ilustrado na Figura A.5.

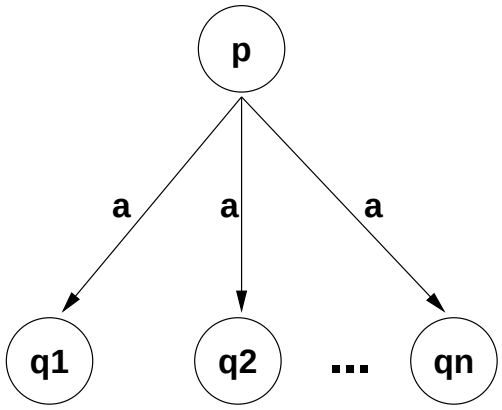


Figura A.5: AFN: uma transição não-determinística para o mesmo símbolo.

Analogamente aos autômatos finitos determinísticos, a computação de um autômato finito não-determinístico, para uma palavra de entrada w, consiste na sucessiva aplicação da função programa para cada símbolo de w (da esquerda para a direita) até ocorrer uma condição de parada. Como cada transição do autômato não-determinístico resulta em um conjunto de estados, é necessário estender a definição da função programa, usando como argumento um conjunto finito de estados e uma palavra.

A parada do processamento de um autômato finito não-determinístico para uma entrada w pode ser de duas maneiras:

1. Aceita a entrada w. Após processar o último símbolo da fita, existe pelo menos um estado final pertencente ao conjunto de estados alternativos atingidos;

2. Rejeita a entrada w. São duas possibilidades:

   (a) após processar o último símbolo da fita, todos os estados alternativos atingidos são não-finais;

   (b) em algum momento, ao longo do processamento de w, a função programa é indefinida para o argumento (conjunto de estados alternativos e símbolo lido da fita).

Embora a facilidade de não-determinismo seja, aparentemente, um significativo acréscimo ao autômato finito, na realidade não aumenta seu poder computacional. Assim, para cada AFN, é possível construir um AFD equivalente que realiza as mesmas computações. O contrário também é verdadeiro.

A.3 Autômatos Finitos com Saída

O conceito básico de autômato finito limita a saída a uma informação binária: aceita ou rejeita. Porém, há duas abordagens possíveis: Máquina de Mealy e Máquina de Moore (HOPCROFT; MOTWANI; ULLMAN, 2001), que estendem esse conceito básico para modelos que possibilitam a geração de palavras como saída. A Máquina de Mealy associa as saídas às transições, enquanto a Máquina de Moore aos estados.

Algumas características quanto ao processo de geração da saída são comuns nas duas abordagens. As seguintes são relacionadas:

1. a saída é armazenada em uma fita independente, a qual não pode ser utilizada para leitura;
2. a cabeça da fita de saída move uma célula para direita a cada símbolo gravado, o qual pertence a um alfabeto especial, denominado alfabeto de saída;
3. o resultado do processamento do autômato é a informação contida na fita de saída.

Os dois modelos de autômato finito com saída são capazes de produzir o mesmo mapeamento da entrada para a saída, ou seja, há equivalência entre eles. Contudo, esta equivalência não é válida para a entrada vazia. Essa exceção não é relevante neste texto, posto que o modelo AGA e suas extensões desconsideram tal situação como entrada. A Figura A.6 tem um exemplo de equivalência entre Máquina de Moore e Máquina de Mealy.

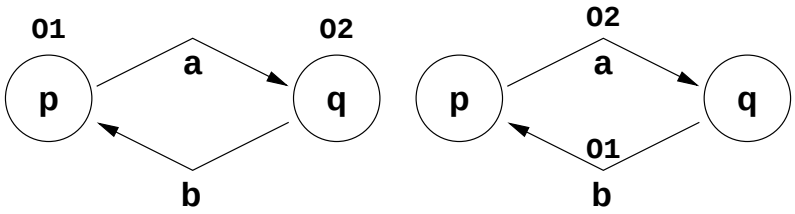


Figura A.6: Equivalência entre Máquina de Moore e Máquina de Mealy

A.3.1 Máquina de Moore

A Máquina de Moore é uma extensão do AFD com saída associada a cada estado. Uma Máquina de Moore M é uma 6-upla ordenada: $M = (\Sigma, Q, \delta, q_0, F, \Delta)$, onde:

1. Σ é um alfabeto de entrada;
2. Q é um conjunto de estados possíveis do autômato o qual é finito;
3. $\delta : Q \times \Sigma \rightarrow Q \times \Delta^*$ é uma função de transição;
4. $q_0 \in Q$ é o estado inicial;
5. $F \subseteq Q$ é um denominado conjunto de estados finais;
6. Δ é um alfabeto de saída.

O processamento de uma Máquina de Moore é similar ao de um AFD mais a gravação de símbolos na fita de saída do autômato a cada vez que um estado é alcançado. Uma saída vazia (ε) indica que não há gravação na fita de saída e, consequentemente, a cabeça da fita de saída não se move. Conclui-se que se todas as transições gerarem saídas vazias, a Máquina de Moore processa exatamente como um AFD.

A.3.2 Máquina de Mealy

A Máquina de Mealy é uma extensão do AFD com saída associada a cada transição. Uma Máquina de Mealy M é uma 6-upla ordenada: $M = (\Sigma, Q, \delta, q_0, F, \Delta)$, onde:

1. Σ é um alfabeto de entrada;
2. Q é um conjunto de estados possíveis do autômato o qual é finito;
3. $\delta : Q \times \Sigma \rightarrow Q \times \Delta^*$ é uma função de transição;[1]
4. $q_0 \in Q$ é o estado inicial;
5. $F \subseteq Q$ é um denominado conjunto de estados finais;
6. Δ é um alfabeto de saída.

O processamento de uma Máquina de Mealy é similar ao de um AFD mais a gravação de símbolos na fita de saída do autômato para cada transição efetuada. Uma saída vazia (ε) indica que não há gravação na fita de saída e, consequentemente, a cabeça da fita de saída não se move. Conclui-se que se todas as transições gerarem saídas vazias, a Máquina de Mealy processa exatamente como um AFD.

[1] No protótipo de animação, optou se por utilizar um único símbolo como saída, em vez de uma string, por motivos de simplificação.

APÊNDICE B COMPARATIVO COM AGA-J, GIF E AVI

Este anexo mostra um comparativo do espaço de armazenamento das animações apresentadas com animações equivalentes no formato GIF e no formato AVI. Para o AGA-J, levou-se em conta os arquivos de malhas, das texturas e das especificações em AGA2ML, e para o GIF e AVI, os arquivos que contêm a seqüência de quadros.

Para evitar diferenças entre as versões produzidas, foram usados os softwares VirtualDub (VIRTUALDUB.ORG, 2005) e Movies 13 (VERHOEVEN, 2004). VirtualDub é um programa de captura de quadros com o código fonte aberto sob a licença GNU GPL. Este programa capturou os quadros-chave necessários para compor a animação em AVI a partir da animação gerada pelo protótipo que reproduz as animações especificadas pelo AGA2ML. Desta forma, cada quadro-chave gerado é uma cópia fiel do apresentado no AGA-J. É bom deixar claro que foram apenas coletados os quadros-chave, ou seja, apenas quadros onde ocorreram mudanças na imagem. O software Movies 13 é um utilitário *freeware* que possui várias ferramentas para geração de imagens GIF animadas. Com o Movies 13, foi possível fazer uma conversão de AVI para GIF.

A Tabela B.1 resume os valores encontrados na análise do espaço de armazenamento das animações. Todas as animações comparadas nesta figura possuem os quadros com tamanho fixo de 792x572 pixels. A coluna do AGA se refere ao gasto em bytes com o armazenamento das especificações em AGA2ML e com os arquivos de malhas. Para demonstrar a reusabilidade provida pelo AGA-J, a última linha da figura corresponde ao total de armazenamento necessário para as 3 animações. Veja bem que, no caso do AGA-J, este total não corresponde à soma exata do espaço de armazenamento das 3 animações, já que as especificações de alguns atores são compartilhadas pelas mesmas.

Tabela B.1: Comparativo do espaço de armazenamento entre animações AGA-J, GIF e AVI com quadros de dimensão fixa de 792x572 pixels

Animação	Tempo	Tamanho (AGA-J)	Tam (GIF)	Tam (AVI)
Briga de Rua	2,5s	10.458B (0,009MB)	1,01MB	5,85MB
Sala de Estar	11s	18.015B (0,017MB)	3,41MB	28,30MB
Acidentes Recorrentes	16s	19.016B (0,018MB)	5,01MB	37,60MB
Total das animações	30s	22.427B (0,021MB)	9,43MB	71,75MB

Nota-se que o espaço de armazenamento gasto com as animações produzidas em AGA-J são extremamente inferiores às animações GIF equivalentes no estudo de caso. Na primeira animação, o AGA-J utiliza menos de 0,1% do espaço necessário para representar a mesma animação em GIF, que ocupa cinco vezes menos do que o AVI equivalente. Seguindo a ordem da lista, as outras animações utilizam 0,49% do espaço necessário pelo

GIF e 0,36%, mostrando que esta porcentagem vai diminuindo nas animações maiores devido ao reuso de poses ao longo do tempo que o AGA-J propicia.

No caso das 3 animações juntas, o AGA-J necessita apenas de 0,22% do espaço requerido pelas animações em GIF, devido ao reaproveitamento dos atores e das texturas utilizadas entre as animações.

A Tabela B.2 resume os valores encontrados na análise do espaço de armazenamento das mesmas animações com o tamanho de quadro reduzido para 312x172 pixels a título de comparação. Pode-se notar que, diferente do GIF e do AVI, o espaço de armazenamento gasto com as animações produzidas em AGA-J não varia em função do tamanho do quadro. Mesmo assim, o ganho de armazenamento em relação ao GIF e o AVI ainda é muito grande. Na primeira animação, o AGA-J utiliza 6,37% do espaço necessário para representar a mesma animação em GIF, que ocupa cerca de dez vezes menos do que o AVI equivalente. Seguindo a ordem da lista, as outras animações utilizam 3,65% do espaço necessário pelo GIF e 3,13%, também apresentando um ganho nas animações complexas mesmo para animações com quadros menores.

Tabela B.2: Comparativo do espaço de armazenamento entre animações AGA-J, GIF e AVI com quadros de dimensão fixa de 312x172 pixels

Animação	Tempo	Tamanho (AGA-J)	Tam (GIF)	Tam (AVI)
Briga de Rua	2,5s	10.458B (0,009MB)	160KB	1,52MB
Sala de Estar	11s	18.015B (0,017MB)	482KB	7,47MB
Acidentes Recorrentes	16s	19.016B (0,018MB)	594KB	10,00MB
Total das animações	30s	22.427B (0,021MB)	1,20MB	18,99MB

O GIF armazena a animação a partir da codificação dos sucessivos quadros-chave. Cada quadro-chave é codificado utilizando o algoritmo de compressão sem perdas LZW [GIB 98] e anexado na forma de bloco gráfico ao conjunto de blocos que formam o arquivo. Desta maneira, o tamanho da animação em GIF cresce em função do número de quadros-chave. A proporção de crescimento do arquivo pode variar devido à taxa de compressão alcançada pelo LZW para cada quadro.

Já o AGA-J armazena apenas o conjunto de corpos articulados e as poses dos atores. Desta maneira, o mesmo ator pode repetir seus movimentos em diversos momentos da animação sem a necessidade de codificá-los novamente. Todas as instâncias de um mesmo ator compartilham o mesmo alfabeto de saída, reduzindo o espaço de armazenamento necessário. O tamanho da especificação em AGA2ML varia de acordo com a complexidade dos atores e o comprimento de suas fitas de entrada. Portanto, o tamanho da animação especificada em AGA2ML não cresce em função do número de quadros-chave como o GIF e o AVI, mas sim, de acordo com a complexidade de cada tipo de ator, suas texturas e as transformações nas suas articulações associadas ao alfabeto de saída.