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

Animação Bidimensional para World Wide Web Baseada em Autômatos Finitos

por

FERNANDO ACCORSI

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Ciência da Computação

Prof. Paulo Fernando Blauth Menezes
Orientador

Prof.ª Luciana Porcher Nedel
Co-Orientadora

Porto Alegre, setembro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Accorsi, Fernando

Animação Bidimensional para World Wide Webe baseada em Autômatos Finitos / por Fernando Accorsi. – Porto Alegre: PPGC da UFRGS, 2002.

112 f. : il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2002. Orientador: Menezes, Paulo Fernando Blauth; Co-orientadora: Nedel, Luciana Porcher.

1. WWW. 2. Animação. 3. Teoria dos Autômatos. 4. Autômatos temporizados. 5. Recuperação de informação. I. Menezes, Paulo Fernando Blauth. II. Nedel, Luciana Porcher.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitora: Prof.ª Wrana Panizzi
Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann
Pró-Reitor Adjunto de Pós-Graduação: Prof. Jaime Evaldo Fensterseifer
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador do PPGC: Prof. Carlos Alberto Heuser
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Primeiramente, gostaria de agradecer aos meus pais Harry e Geni pelo exemplo de trabalho e perseverança, o qual muitas vezes foi minha inspiração para enfrentar os momentos difíceis até aqui.

A minha querida Dinha e aos meus irmãos Jane e Edson pelas constantes palavras de incentivo e otimismo.

Aos professores Blauth e Luciana pela orientação no desenvolvimento deste trabalho e pela disposição de me aceitar como orientando, mesmo sabendo das dificuldades que envolvem a realização de um trabalho à distância.

A todos os colegas do grupo de trabalho, em especial ao Guilherme e ao Campani, pelas grandes contribuições dadas no desenvolvimento do AGA.

Por último gostaria de agradecer aos membros do Instituto de Informática da UFRGS pela iniciativa de promover o Mestrado Remoto, sem o qual, eu não teria a oportunidade de desenvolver este trabalho.

À minha querida companheira Alifrancy,
doce fortaleza presente em todos
os momentos difíceis.

Sumário

Lista de Abreviaturas

API	Application Program Interface
AVI	Audio Video Interleave
DOM	Document Object Model
DTD	Document Type Definition
ECMA	European Computer Manufacturers Association
GIF	Graphics Interchange Format
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JPEG	Joint Photographic Experts Group
MIME	Multipurpose Internet Mail Extensions
MPEG	Moving Picture Expert Group
ms	milésimos de segundo
NCSA	National Center for Supercomputing Applications
RDF	Resource Description Framework
SAX	Simple API for XML
SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics
SWF	Shockwave Flash
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Lista de Símbolos

$\preceq$	precede
$\prec$	precede estritamente
2^A	conjunto das partes do conjunto A

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho aplica a Teoria de Autômatos na proposição de uma nova alternativa para prover animações 2D na *World Wide Web*, verificando as contribuições alcançadas para as questões relacionadas ao espaço de armazenamento, reutilização e manutenção do conteúdo e suporte à recuperação de informação. Para este objetivo, é proposto o modelo AGA (Animação Gráfica baseada em Autômatos Finitos), o qual especifica a animação a partir de uma estrutura baseada em autômatos finitos com saída. Esse modelo é definido de tal forma que os mesmos autômatos utilizados na especificação, ao serem simulados, realizam o controle da animação durante a apresentação. O modelo AGA apresenta características que favorecem a redução do espaço de armazenamento da animação, provêem suporte à recuperação de informação, colaboram com a reutilização e manutenção do conteúdo das animações. Uma implementação multiplataforma foi desenvolvida para apresentar animações especificadas nesse modelo na *Web*. Essa implementação proporciona a elaboração de consultas ao conteúdo da animação, além dos recursos tradicionais de reprodução. A partir dessa implementação, o AGA foi submetido a um estudo de caso prático, onde os resultados obtidos são comparados com o produzidos pelo GIF (*Graphic Interchange Format*). Esse comparativo demonstra que o AGA possui várias vantagens em relação à estrutura adotada pelo GIF. O modelo AGA é estendido utilizando autômatos temporizados para prover restrições temporais às especificações e também ampliar as funcionalidades de interação com o observador da animação. Essa extensão, chamada de modelo AGA-S (Animação Gráfica baseada em Autômatos Temporizados Sincronizados), é definida a partir do autômato temporizado proposto por Alur e Dill. Para esse modelo, é definida uma operação formal para sincronização dos componentes da animação e adicionada uma estrutura baseada em autômatos finitos para controlar a interação do observador com a animação.

Palavras-chave: WWW, animação, teoria dos autômatos, autômatos temporizados, recuperação de informação.

TITLE: "TWO-DIMENSIONAL ANIMATION FOR WORLD WIDE WEB BASED ON FINITE AUTOMATA"

Abstract

This work propose a new alternative to provide two-dimensional animation in the World Wide Web applying the Theory of Automata, verifying the contributions reached for the questions related to the amount of storage space, content reuse and maintenance, and information retrieval. For this purpose, AGA model (Graphical Animation based on Finite Automata) is considered, which specifies the animation from a structure based on finite automata with output. This model is defined of such form that the same automata used in the specification, when being simulated, do the control the animation during the presentation. AGA has characteristics that favor the reduction of the storage space of the animation, to provide has supported to the information retrieval, collaborates with the content reuse and maintenance. A multiplatform implementation was developed for playing AGA animations in the Web. This implementation provides the elaboration of queries for the animation content, beyond the traditional reproduction resources. From this implementation, the AGA was submitted to a study of practical case, where the gotten results are compared with the GIF's (Graphic Interchange Format). This comparative exposes some advantages of AGA in relation to the structure adopted for the GIF. AGA is extended using timed automata to provide time restrictions to the specifications and also to extend the functionalities of interaction between observer and animation. This extended model, called AGA-S (Graphical Animation based on Synchronized Timed Automata), is defined from the timed automata proposed by Alur and Dill. For this model, a formal operation for synchronization animation components is defined and added a structure based on finite automata to control the interaction between observer and animation.

Keywords: WWW, animation, theory of automata, timed automata, information retrieval.

1 Introdução

1.1 Motivação

Atualmente, grande parte do conteúdo da *World Wide Web* consiste em informações visuais, tais como: imagens, animações, vídeos e gráficos [LEW2000]. Em especial, as animações tem sido amplamente utilizadas na elaboração de interfaces mais amigáveis, apresentação de propagandas *on-line*, criação de jogos e ambientes virtuais, ilustração de conteúdos didáticos e de entretenimento, entre outras.

Vinculadas às páginas *Web*, são encontradas animações providas por sistemas de animação 2D e 3D. As animações 3D ainda são bastante escassas, enquanto são raros os *sites* que não utilizam alguma animação 2D para enriquecer seu conteúdo.

Os diferentes sistemas de animação utilizados para a criação e apresentação de animações 2D na *Web* podem ser categorizados, de acordo com o modo de produção, em sistemas de tempo real ou quadro-a-quadro [THA 85]. Os sistemas de animação em tempo real computam a imagem final para visualização no momento da apresentação, enquanto os sistemas quadro-a-quadro (também denominados *playback*) apresentam a seqüência animada a partir de sucessivos quadros previamente computados e armazenados.

Entre as abordagens quadro-a-quadro, são encontrados na *Web*: o *AVI* (*Audio Video Interleave*) [MIC 2002], *MPEG* (*Moving Picture Expert Group*) [INT 96], QuickTime [APP 2000] e, especialmente, o *GIF* (*Graphics Interchange Format*) [COM 90], atualmente o mais utilizado para a criação de animações 2D.

Embora a abordagem quadro-a-quadro seja responsável pela maior parte das animações na *Web*, o volume de animações baseadas em sistemas de tempo real, como o SVG (Scalabre Vector Graphics) [W3C 2002a] e o Macromedia Flash [MAC 2002] tem crescido e ocupado um lugar de destaque em animações que possuam interação com o observador.

Mesmo existindo essas várias alternativas, as características particulares da *Web* 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 *Web*.

A grande quantidade de espaço de armazenamento necessária 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 [CHE 01, REJ 99, ACH 98].

A recuperação de informação em animações também é uma questão importante a ser considerada, principalmente para a *Web*, que reúne um volume bastante grande de informações visuais [ARA 2000]. À medida que esse volume de informações cresce, o desenvolvimento de estratégias eficientes de recuperação de conteúdo visual se torna ainda mais indispensável [SMI 97, GEV 99, SCL 97, LEW 2000].

Outra questão relevante diz respeito à criação e manutenção das animações. O conteúdo visual na *Web* 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.

Várias dificuldades em torno dessas questões foram experimentadas pela equipe de trabalho do Projeto TEIA (Técnicas de Ensino Interativas Assistidas por

Computador), desenvolvido no Instituto de Informática da Universidade Federal do Rio Grande do Sul, durante a elaboração das animações ilustrativas para o conteúdo didático disponibilizado na *Web*. Em especial, na elaboração das animações dos autômatos contidas na versão eletrônica do livro "Linguagens Formais e Autômatos" do autor Paulo Fernando Blauth Menezes [MEN 2000].

Atualmente, a versão eletrônica desse livro é suportada pelo sistema Hyper-Automaton [MAC 2000], desenvolvido por Júlio Henrique A. P. Machado durante o seu mestrado também realizado no Instituto de Informática da Universidade Federal do Rio Grande do Sul. 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.

As questões evidenciadas em relação às animações motivaram o desenvolvimento deste trabalho com o propósito de apresentar uma nova alternativa para prover animações 2D na *Web*, a qual pudesse ser facilmente integrada às demais ferramentas já desenvolvidas no Instituto de Informática da UFRGS, como, por exemplo, o Hyper-Automaton.

1.2 Objetivo

Este trabalho tem como objetivo aplicar a Teoria de Autômatos na proposição de uma nova alternativa para prover animações 2D na *World Wide Web*, verificando as contribuições alcançadas para as questões relacionadas ao espaço de armazenamento, reusabilidade, manutenibilidade e suporte à recuperação de informação.

A fim de atingir esse objetivo foram traçadas as seguintes metas específicas ordenadas cronologicamente:

a) desenvolver um modelo baseado em autômatos finitos para suportar animações 2D que englobe as funcionalidades de reprodução encontradas atualmente nas abordagens quadro-a-quadro;

b) implementar um protótipo operacional para reprodução na *World Wide Web* de animações elaboradas seguindo o modelo proposto;

c) verificar as contribuições alcançadas através da utilização do modelo quanto ao espaço de armazenamento, reusabilidade, manutenibilidade e suporte à recuperação de informação;

d) estender o modelo proposto inicialmente para suportar outras funcionalidades encontradas em sistemas de animação em tempo real, como a interação com o observador e a reprodução de som durante a apresentação.

Como é possível constatar pela leitura dos demais capítulos dessa dissertação, as metas específicas traçadas foram atingidas em sua totalidade. Porém, é importante ressaltar, que na implementação do modelo (item b acima), o resultado obtido ultrapassa a qualidade de protótipo, pois consiste em um produto final com a totalidade dos recursos implementados.

1.3 Descrição e Apresentação da Dissertação

Os capítulos dessa dissertação foram organizados na mesma ordem das metas estabelecidas. Desta forma, o leitor pode acompanhar gradativamente a evolução do trabalho e constatar os principais resultados alcançados a cada etapa.

No capítulo 2 são descritos os conceitos fundamentais relacionados à criação de animações por computador. Apoiado nesses conceitos, foi delimitado o escopo do trabalho para as animações 2D aplicadas a *World Wide Web*, cujas características

relacionadas às animações são apontadas evidenciando as questões principais quanto ao espaço de armazenamento necessário para representá-las e o suporte à recuperação de informação. Nesse capítulo, também são apresentadas as estratégias mais utilizadas na *Web* para prover animações 2D classificadas de acordo com o modo de produção. Entre elas são enfocados: o GIF [COM 90], o SVG [W3C 2002a] e o Macromedia Flash [MAC 2002].

No capítulo 3 é proposto pelo autor o modelo AGA (Animação Gráfica baseada em Autômatos Finitos), o qual especifica a animação a partir de uma estrutura baseada em autômatos finitos com saída. Esse modelo é definido de tal forma que os mesmos autômatos utilizados na especificação, ao serem simulados, realizam o controle da animação durante sua apresentação. O modelo AGA apresenta características que: favorecem a redução do espaço de armazenamento da animação; provêem suporte à recuperação de informação; colaboram com a reutilização e manutenção da animação.

O modelo proposto foi publicado através do artigo "Animação Gráfica Baseada na Teoria de Autômatos" [ACC 2000] no III Workshop de Métodos Formais (WMF 2000), realizado em João Pessoa em outubro de 2000. Esse artigo, presente no anexo 1, define o AGA a partir do modelo de autômato finito com saídas associadas aos estados (Máquina de Moore). É importante ressaltar que no capítulo 3, o AGA é definido a partir do modelo de autômato finito com as saídas associadas às transições (Máquina de Mealy), modelo equivalente ao apresentado no artigo, porém com algumas conveniências observadas durante a evolução do trabalho para a modelagem das animações.

No capítulo 4, a implementação do AGA é descrita enfocando as tecnologias escolhidas para o desenvolvimento do programa de visualização na *Web*, batizado de AGA Player, e a linguagem criada especialmente para ele com o objetivo de especificar as animações a serem apresentadas. O AGA Player é uma solução multiplataforma que possui recursos que proporcionam a elaboração de consultas ao conteúdo da animação, além dos recursos tradicionais de reprodução encontrados na maioria dos programas de visualização. A linguagem criada para o AGA Player, denominada de AgaML (AGA *Markup Language*), é um vocabulário XML *(Extensible Markup Language)* que possibilita a especificação dos componentes do modelo proposto com significativas facilidades para reutilização e manutenção das animações. A definição da sintaxe da AgaML é apresentada no anexo 4 na forma de DTD (*Document Type Definition*).

No capítulo 5, a aplicação do modelo AGA é demonstrada e avaliada através da criação de animações ilustrativas para o livro eletrônico "Linguagens Formais e Autômatos" do autor Paulo Fernando Blauth Menezes [MEN 2000]. As seqüências de imagens produzidas por essas animações estão ilustradas no anexo 5. Essas animações são utilizadas para traçar um comparativo entre o modelo AGA e o GIF, utilizado anteriormente na produção dessas animações. Esse comparativo demonstra que o AGA possui várias vantagens em relação à estrutura adotada pelo GIF quanto ao espaço de armazenamento necessário para representação, reusabilidade e manutenibilidade das animações, assim como acrescenta estruturas que suportam a recuperação de informação.

Os experimentos quantitativos, apresentados no capítulo 5, referentes ao espaço de armazenamento confirmam os resultados obtidos no artigo *"Comparing Data Compression in Web-based Animation Models using Kolmogorov Complexity"* submetido à revista *IEEE Transactions on Information Theory*, para a qual aguarda resposta. Nesse artigo, encontrado no anexo 3, o primeiro autor Carlos A. P. Campani prova que o AGA é mais eficiente para a compressão de dados do que o GIF utilizando a Complexidade de Kolmogorov. O autor desta dissertação contribuiu no artigo com as

descrições referentes às características do modelo AGA e as informações quantitativas reais utilizadas para ilustrar os resultados teóricos.

No capítulo 6, é proposta a extensão do modelo AGA utilizando autômatos temporizados para prover restrições temporais às especificações do modelo e ampliar as funcionalidades de interação com o observador (usuário) da animação. Essa extensão, chamada de modelo AGA-S (Animação Gráfica baseada em Autômatos Temporizados Sincronizados), é definida a partir do autômato temporizado proposto por Alur-Dill [ALU 94], o qual é descrito nas primeiras seções deste capítulo. O modelo AGA-S conserva os benefícios do modelo original AGA quanto ao espaço de armazenamento, reusabilidade, manutenibilidade e suporte à consulta. Para esse modelo foi definida uma operação formal para sincronização dos componentes da animação e adicionada uma estrutura, também baseada em autômatos finitos, para controlar a interação do observador com a animação.

O modelo AGA-S foi publicado através do artigo "Animação Gráfica Baseada em Autômatos Temporizados Sincronizados" [ACC 2001a] no IV Workshop de Métodos Formais (WMF 2001), realizado no Rio de Janeiro em outubro de 2001. Esse artigo, presente no anexo 2, descreve o modelo AGA-S exceto quanto às características de interação, as quais foram definidas posteriormente à sua publicação.

No capítulo 7, são reunidas as principais contribuições deste trabalho obtidas a cada etapa de seu desenvolvimento, enfocando principalmente as questões delimitadas no capítulo 2 quanto à utilização de animações 2D na *World Wide Web*.

Durante a definição e utilização dos modelos propostos, foram observadas várias potencialidades que podem ser exploradas ainda mais através de trabalhos de futuros. Diversos desses trabalhos são apontados no capítulo 7.

2 Animações na *World Wide Web*

2.1 Introdução

A animação convencional[1] é definida em [THA 85] como a técnica na qual cria-se a ilusão de movimento através de uma série de desenhos fotografados individualmente e gravados em sucessivos quadros em um filme. A ilusão de movimento é alcançada quando o filme é projetado a uma certa taxa de quadros por segundo.

A animação por computador teve sua origem na assistência dos processos convencionais e atingiu sua independência nas animações modeladas. A animação assistida por computador consiste, basicamente, em automatizar processos empregados na criação de animações convencionais, enquanto a animação modelada engloba desde a modelagem geométrica dos objetos até a computação das diferentes ações que podem realizar na animação.

Em [PUE 88], o processo de animação por computador é organizado nas seguintes etapas:

a) **Pré-processamento** – A animação é planejada através de *storyboards* onde as seqüências animadas são esboçadas a partir das cenas principais. Essa fase é amplamente usada em filmes de animação devido à complexidade do projeto, já em animações mais simples geralmente é descartada;

b) **Edição das cenas e modelagem dos objetos** – Os objetos que participam da animação são modelados e combinados para comporem as cenas. A modelagem dos objetos pode ser realizada através de várias abordagens dependendo da natureza estética. Em geral, as modelagens mais comuns são as geométricas em domínios bidimensionais e tridimensionais através de modelos representados por listas de triângulos [THA 85];

c) **Animação** – As variações dos objetos na animação são determinadas e calculadas para gerar a seqüência animada. Os movimentos dos objetos na animação podem ser descritos através de diversos métodos de controle movimento e em níveis de abstração diferentes. Esses métodos vão desde o controle explícito, no qual o animador é responsável pela descrição dos atributos posicionais dos objetos, até controles altamente automatizados através de sistemas baseados em conhecimento [FOL 90];

d) ***Rendering* das imagens –** As imagens finais de cada quadro são obtidas através da computação das propriedades visuais dos objetos na cena. Nesta fase são levados em conta, por exemplo, os parâmetros de iluminação e posicionamento de câmeras para a computação de sombras, transparências, eliminação de faces, texturas, etc. [THA 85] salienta que o processo de *rendering* pode ser realizado em tempo real durante a apresentação da animação ou ser computado antes;

e) **Pós-processamento** – A seqüência de imagens geradas é freqüentemente sincronizada com som e gravada em outro tipo de mídia caso necessário. Nas animações destinadas a filmes, a seqüência é combinada com outras cenas e gravada em película;

f) **Análise dos resultados** – As características visuais reveladas pela animação são exploradas com o objetivo de extrair informações sobre os modelos utilizados. Essa fase é amplamente utilizada em animações destinadas à simulação de processos naturais e industriais.

---

[1] O termo "convencional" é utilizado para as animações tradicionalmente criadas em sistemas não computacionais.

Os sistemas responsáveis por essas tarefas, chamados sistemas de animação, possuem três componentes básicos: modelador de atores[1], mecanismo de controle da animação e mecanismo de *rendering* e visualização.

Devido à variedade de técnicas utilizadas no processo de animação, vários enfoques para a classificação dos sistemas de animação são propostos [ZEL 85, THA 85, FOL 90, PUE 88]. Em [PUE 88] são reunidos os seguintes critérios para classificação: histórico, tipo de aplicação, nível de controle do movimento no sistema, dimensão do sistema, modelo de animação, complexidade de *rendering*, complexidade de pós-processamento e modo de produção. Para a organização deste trabalho são evidenciados os critérios de classificação quanto à dimensão do sistema e ao modo de produção.

O critério baseado na dimensão do sistema, apresentado em [PUE 88], estabelece a classificação dos sistemas de animação em 2D, 2.5D e 3D. A dimensão do sistema está diretamente ligada à natureza geométrica da modelagem dos objetos e da construção da cena. Em sistemas 2D, as modelagens dos objetos e das cenas são realizadas sobre um espaço bidimensional. Em animações 2.5D, os objetos bidimensionais são distribuídos em um ambiente tridimensional, onde transformações geométricas tridimensionais são permitidas e a cena é construída a partir da computação de um campo de visão sobre este ambiente. Já em sistemas 3D, tanto os objetos quanto à construção das cenas são modelados em um espaço tridimensional.

Independente da dimensão, os sistemas de animação por computador também podem ser classificados, segundo o critério de modo de produção, em sistemas de tempo real ou quadro-a-quadro [THA 85]. Os sistemas de animação em tempo real geram a imagem final para visualização no momento de sua apresentação. Esta abordagem favorece principalmente as animações interativas, onde a imagem visualizada deve corresponder às ações instantâneas tomadas pelo usuário. Por outro lado, os sistemas de animação quadro-a-quadro, geram a seqüência animada a partir de seus modelos de objetos e cenas, armazenando as imagens geradas em sucessivos quadros. Posteriormente, esses quadros são apresentados a uma taxa adequada para visualizar a animação. Em geral, esta abordagem é utilizada quando a complexidade de *rendering* é alta devido ao realismo das cenas.

O escopo deste trabalho está direcionado às animações 2D com a aplicação direcionada a WWW. Esse ambiente possui características bastante particulares que influenciam diretamente o processo de criação das animações. Essas características juntamente com as principais dificuldades que envolvem a produção de animações para *Web* são discutidas na seção 2.2. Na seção 2.3, são apresentadas as principais estratégias para animação na *Web* organizadas segundo o modo de produção. O Graphics Interchange Format (GIF), Scalable Vector Graphics (SVG) e o Macromedia Flash (SWF) são descritos em mais detalhes por serem estratégias predominantes atualmente na WWW para animações 2D.

2.2 Características da *Web* relacionadas às animações

Grande parte do conteúdo da *Web* consiste atualmente em informações visuais, tais como imagens, animações, vídeos e gráficos [LEW 2000]. As animações, em especial, são empregadas para diversos fins: as propagandas *on-line*, jogos, ícones animados, ambientes virtuais, interfaces gráficas, desenhos animados, simulações, entre outros.

[1] Os objetos que participam da animação são comumente chamados de atores.

Os sistemas de animação para *Web* materializam a seqüência animada em um arquivo para o intercâmbio da animação na rede. Este arquivo é utilizado para a apresentação da animação pelos módulos de visualização localizados, em geral, nos navegadores. O arquivo de animação e o módulo de visualização são os componentes que interagem diretamente com a *Web*, as demais partes do sistema de animação são, na maioria das vezes, independentes.

Há diversos formatos de arquivos[1] que permitem o intercâmbio de animações na *Web*, sendo que vários já eram utilizados anteriormente em outros ambientes e foram adaptados às necessidades da WWW. Outros formatos, porém, foram desenvolvidos especialmente para este ambiente, considerando as características de rede, a heterogeneidade das plataformas, a dinâmica do conteúdo, entre outras.

Em [LEE 98], os formatos de arquivos para animação são classificados em formatos baseados em quadros ou formatos baseados em conteúdo. A primeira categoria armazena as animações como uma seqüência de imagens estáticas (quadros). Já a segunda, armazena os modelos que descrevem os objetos e suas ações ao invés das imagens. O GIF [COM 90], MPEG-1 [INT 96], QuickTime [APP 2000] e AVI [MIC 2002] são exemplos de formatos baseados em quadros herdados pela *Web*, enquanto o SVG [W3C 2002a] e o SWF [MAC 2002] são propostas baseadas no conteúdo e desenvolvidas especificamente para este ambiente.

A apresentação das animações veiculadas nas páginas *Web* é mediada em geral pelos navegadores. Estes programas para exploração da *Web* utilizam basicamente três abordagens para prover esta tarefa: uso de um decodificador do formato da animação embutido no navegador; execução de um programa externo para visualização; execução de programas em linguagens específicas para Internet.

Para alguns formatos, o decodificador foi incluído no desenvolvimento do navegador. Desta forma, a animação referenciada pela página, logo que carregada, é decodificada e apresentada pelo próprio navegador, dispensando o uso de programas externos para visualização. Um representante histórico desta abordagem é o *Graphics Interchange Format* versão 89a, popularmente chamado de GIF animado.

Por outro lado, os formatos não decodificados internamente necessitam de programas externos para visualização. Esta abordagem é utilizada desde o primeiro navegador para a *Web*, o Mosaic, desenvolvido pelo NCSA [VET 94]. Esses programas que complementam os navegadores, chamados atualmente de *plug-ins*, são invocados quando um formato associado é encontrado. O navegador verifica qual *plug-in* é compatível com o tipo MIME (*Multipurpose Internet Mail Extensions*), o carrega para a memória e cria uma nova instância para execução. A possibilidade de ligação de programas externos complementares aos navegadores vem estimulando a criação de novas estratégias tanto para especificação quanto para apresentação de animações. Programas como o QuickTime, Macromedia Flash Player, Windows Media Player e Adobe SVG Viewer são exemplos de *plug-ins* destinados à visualização de animações e vídeos.

Além das abordagens já descritas, as linguagens de programação direcionadas a *Web* também podem ser utilizadas para promover animações. A linguagem Java, por exemplo, contém extensões com o objetivo de agregar capacidades multimídia à linguagem. Desta maneira, através da construção de *applets*[2] Java, é possível explorar esses recursos tanto para apresentação de animações quadro-a-quadro quanto para

[1] Formato neste contexto refere-se à especificação para armazenamento da animação em arquivos de dados.

[2] O termo *applet* é utilizado para representar um miniaplicativo em Java dedicado a *Web* desenvolvido para executar sobre um navegador.

criação de animações em tempo real. As linguagens para script, como o JavaScrip e o VBScript, podem ser utilizadas para o mesmo fim. Esta abordagem é bastante empregada em animações destinadas aos jogos *on-line*.

Embora existam diversos formatos de animação e abordagens alternativas de apresentação na *Web*, algumas dificuldades tem sido evidenciadas neste ambiente com o aumento do conteúdo visual vinculado às páginas. Estas dificuldades estão principalmente ligadas ao grande espaço de armazenamento necessário às animações, às baixas taxas de transmissão alcançadas na rede, e à complexidade das estratégias de recuperação de informação visual de alta volatilidade.

O espaço de armazenamento necessário para uma animação depende do formato de arquivo utilizado para a codificação. Nos formatos baseados em seqüência de quadros, o número de quadros por segundo, número de cores da paleta, tempo de duração e dimensão da imagem, são fatores que implicam diretamente no tamanho do arquivo. Já nos formatos baseados em conteúdo, outros fatores são mais determinantes, como por exemplo, o número de objetos animados e a complexidade geométrica dos modelos e seus movimentos.

Em geral, animações de boa qualidade[1] implicam em arquivos mais longos, os quais exigem, por exemplo, mais espaço nos servidores de páginas, estratégias de *caching* específicas [CHE 2001, REJ 99], tempos de transmissão maiores, recursos de armazenamento e processamento adicionais para indexação [LEW 2000, SMI 97].

Nota-se que, muitas vezes, a qualidade da animação é reduzida para adequá-la às restrições da WWW. O experimento apresentado em [ACH 98] envolvendo os formatos MPEG-1, AVI e QuickTime, mostra que o tamanho médio destes arquivos na *Web* era de 1.1 MB, onde 90% destes continham apresentações com apenas 45 segundos de duração ou menos, em dimensões de imagens pequenas ou médias[2]. Ainda assim, somente 1% dos arquivos pesquisados poderiam ser apresentados como mídia contínua a uma taxa de 56 Kbits por segundo.

A recuperação de informação visual na *Web* é outro ponto importante a ser considerado. Muitos sistemas de indexação de documentos estão disponíveis na WWW [GUD 97, LAW 98], porém, a maior parte destes é baseada somente no conteúdo textual dos documentos. Desta maneira, as animações não são recuperadas atualmente pelas informações que representam, mas sim pelo conteúdo textual da página onde são vinculadas.

A carência da recuperação de informação visual tem estimulado vários esforços para a construção de sistemas de indexação específicos para imagens, animações e vídeos na *Web* [SMI 97, GEV 99, SCL 97, LEW 2000]. Esses sistemas têm explorado paradigmas de consulta baseados em palavras-chave, similaridade de imagens, casamento de curvas e ícones, obtendo resultados interessantes. Porém, a falta de precisão na elaboração da consulta e a presença de imagens inoportunas no resultado são problemas encontrados por estes sistemas [LEW 2000].

O WebseeK [SMI 97], PicToSeek [GEV 99] e ImageRover [SCL 97], por exemplo, possuem mecanismos de consultas baseados na similaridade de imagens. Estes sistemas restringem a especificação da consulta à seleção de uma imagem de referência. Como estes sistemas se baseiam na similaridade de vetores de características baseados em esquemas globais (cor, texturas e formas), as imagens que possuem as mesmas características globais, mas têm conteúdo visual diferente, são também recuperadas.

---

[1] A qualidade neste contexto se refere à riqueza de detalhes estéticos e a maior proximidade da ilusão do movimento com o fenômeno real representado.

[2] As dimensões de 160x120 e 320x240 são padrões encontrados e classificados, respectivamente, como pequenos e médios em [ACH 98].

Em contra partida à construção de mecanismos de pesquisa, novas estratégias tem sido desenvolvidas para adicionar esquemas de descrição do conteúdo visual à mídia com o objetivo de favorecer a recuperação de informação. O MPEG-7 (padrão proposto pelo MPEG para descrição de conteúdo multimídia) [INT 2001] ilustra os esforços neste sentido, permitindo acesso orientado ao conteúdo em vários níveis de abstração.

2.3 Animações bidimensionais na *Web*

2.3.1 Animação quadro-a-quadro

2.3.1.1 Graphics Interchange Format (GIF)

O GIF, desenvolvido pela Compuserve Incorporate, é um protocolo para intercâmbio de imagens. A versão 89a desse protocolo, chamada popularmente de GIF animado, tem sido amplamente utilizada na *Web* para criação de pequenas animações. Entre as principais aplicações, está a criação de animações para propagandas on-line e ícones animados.

Ao contrário das abordagens dos sistemas de animação em tempo real, o GIF armazena as imagens resultantes do processo de animação. Embora seja mais aplicado em animações 2D na *Web*, o GIF também pode armazenar imagens geradas a partir de sistemas de animação tridimensionais ou até mesmo vídeo sem som.

Há diversos sistemas de animação dedicados a construção de GIF animados. Em geral, esses sistemas proporcionam a produção da animação a partir da criação de cada quadro da seqüência animada. Por exemplo, o Corel PHOTO-PAINT e o Adobe ImageReady oferecem um ambiente de trabalho com ferramentas de desenho e efeito baseadas em processamento de imagem, e disponibilizam uma área de trabalho organizada para a criação de cada quadro. Esse processo de criação, embora proporcione uma grande liberdade de desenho ao autor, uma vez concluído, não contribui com o reuso de partes da animação nem com a manutenção da seqüência animada.

A figura 2.1 ilustra a estrutura do GIF, a qual é formada a partir de blocos e sub-blocos dedicados a armazenar informações de controle, gráficas e de propósito geral. Todo arquivo GIF possui um cabeçalho de identificação, um descritor de tela com os parâmetros necessários para definir os recursos de visualização, e é finalizado com um bloco terminador. As tabelas de cor utilizadas como referência pelas imagens, tanto global quanto local, são opcionais. Em geral, para manter o arquivo menor, as imagens contidas na animação fazem referência apenas à tabela de cor global, eliminando assim, as tabelas locais. Entre a tabela de cor global e o bloco finalizador podem existir vários blocos gráficos ou de propósito geral.

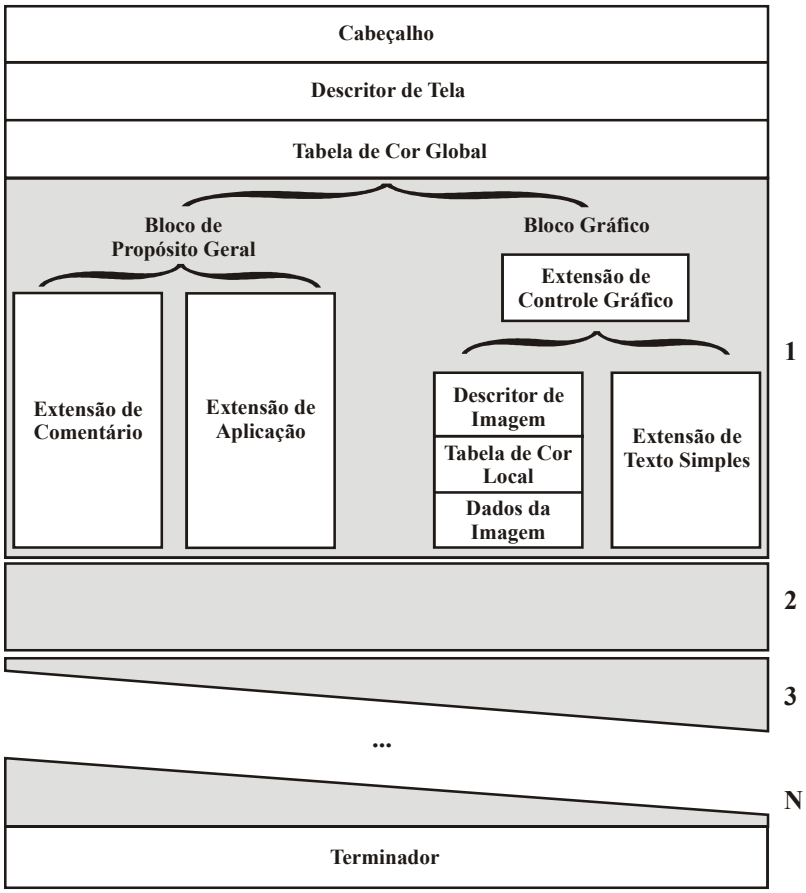


FIGURA 2.1 – Estrutura do GIF versão 89a

Os blocos gráficos são utilizados para armazenar as informações relativas aos quadros da animação ou textos[1]. A extensão de controle gráfico possui informações quanto ao tempo de permanência do próximo elemento gráfico e também indica a maneira como este deve ser tratado[2] após ter sido mostrado.

Com o objetivo de produzir arquivos pequenos, o GIF codifica as imagens utilizando o algoritmo de compressão sem perdas LZW [GIB 98]. Desta forma, os pontos de cada imagem são armazenados nos blocos de dados seguindo essa codificação e decodificados pelo programa de visualização no momento da apresentação. Em geral, a imagem de cada quadro não é armazenada integralmente. Os programas que geram para o formato GIF procuram codificar somente a área da imagem que foi alterada de um quadro para outro, utilizando o descritor de imagem para armazenar as dimensões e coordenadas da área selecionada.

Quanto à recuperação de informação em arquivos GIF, os blocos de propósito geral podem ser explorados para o armazenamento de informações adicionais destinadas a este fim. O bloco de extensão de comentário, por exemplo, é dedicado a armazenar texto sem propósito de *rendering*. Esse texto pode ser utilizado para adicionar informações quanto à autoria ou conteúdo da animação.

[1] A utilização da extensão de texto simples é um recurso pouco utilizado para unir textos às imagens nas animações.

[2] Os gráficos podem ser deixados no lugar ou removidos restaurando a cor de fundo ou a imagem anterior.

2.3.1.2 Moving Picture Expert Group (MPEG-1), QuickTime e Audio Video Interleave (AVI)

Os formatos MPEG-1, QuickTime e AVI foram desenvolvidos com o objetivo de codificar vídeo e áudio associados para armazenamento em mídia digital. Esses formatos têm sido utilizados na *Web* para o intercâmbio de vídeos.

A codificação desses formatos está estruturada no armazenamento dos sucessivos quadros do filme destinados à apresentação em tempo real. Com esse objetivo, a codificação das imagens envolve algoritmos de compressão com perdas para a redução das redundâncias tanto espaciais quanto temporais, favorecendo a transferência em ambientes como a *Web*. A codificação com perdas baseia-se no conceito de comprometimento da precisão da imagem reconstruída em troca de uma maior compressão.

Embora esses formatos também possam ser aplicados à codificação de animações bidimensionais, a distorção nas imagens causadas pelos algoritmos de compressão com perda não tem estimulado esta prática. Por exemplo, o MPEG-1 e QuickTime utilizam algoritmos para compressão espacial semelhantes ao JPEG, o qual distorce imagens que contenham bordas bem definidas [MAR 99]. Essa característica estética é bastante comum nas imagens de síntese geradas por sistemas de animação 2D.

2.3.2 Animação em tempo real

2.3.2.1 Macromedia Flash (SWF)

O Flash [MAC 2002], desenvolvido pela Macromedia®, 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 *sites*, produção de seqüências animadas com baixo nível de realismo e no desenvolvimento de jogos *on-line*.

As animações em Flash 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 mapa 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 como escala, translação e rotação. 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 na linguagem específica do sistema, ActionScript. Essa última alternativa de controle de movimento é bastante utilizada na criação de animações interativas.

A seqüência animada pode ser codificada em diversos formatos de arquivos, como: GIF, AVI e QuickTime, porém, o SWF, padrão do sistema Flash, é o formato de arquivo que comporta a especificação de todos recursos previstos pelo ambiente. O SWF é lido como mídia contínua pelo Macromedia Flash Player, *plug-in* de visualização disponibilizado pela mesma empresa.

Grande parte da popularidade do Flash está atribuída à eficiência do SWF 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 2.2 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 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.

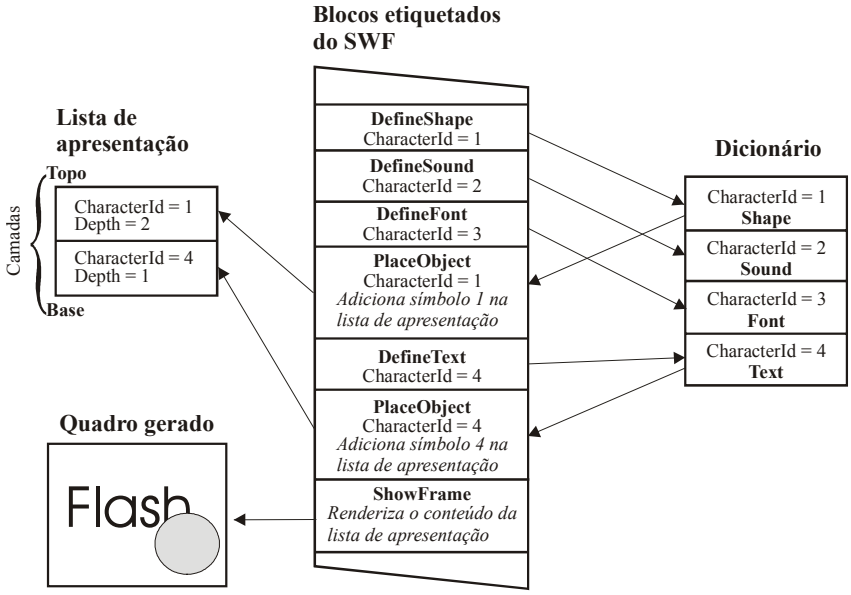


FIGURA 2.2 – Processamento do SWF no programa de visualização

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.

O SWF também é utilizado pelos sistemas de animação Corel R.A.V.E. e Adobe Live Motion como formato padrão para o intercâmbio de animações. Estes sistemas são bastante similares ao Flash, porém, até o momento não atingiram a mesma popularidade.

Além do Flash, a Macromedia® disponibiliza o sistema Director Shockwave Studio, direcionado à produção de aplicações multimídia interativas na *Web*. Este sistema complementa a área de atuação do Flash, pois implementa ferramentas para a criação de animações tridimensionais. O sistema disponibiliza a linguagem orientada a objetos Lingo para a criação de scripts. As aplicações produzidas pelo Director necessitam do programa de visualização específico Macromedia ShockWave Player para serem veiculadas na *Web*.

2.3.2.2 Scalable Vector Graphics (SVG)

O SVG é um vocabulário[1] proposto pelo W3C (*World Wide Web Consortium*) para representar gráficos bidimensionais, estáticos ou animados, em XML (*Extensible Markup Language*). O alto nível das estruturas sintáticas e semânticas da linguagem vem possibilitando não só sua utilização como linguagem alvo em sistemas de animação, mas também como linguagem fonte destinada à edição humana. O SVG, somado a tecnologias como o ECMAScript[2] e SMIL (*Synchronized Multimedia Integration Language*) [W3C 2002b], tem se mostrado um próspero concorrente do Flash (SWF).

Ao contrário do SWF, um arquivo SVG não é binário, consiste em um documento no formato texto estruturado segundo as marcações definidas pela sua DTD (*Document Type Definition*). Esse arquivo texto é utilizado como arquivo de intercâmbio na Internet e processado pelos programas de visualização, em geral, *plug-ins* instalados nos navegadores. O programa de visualização é responsável por validar o arquivo segundo a DTD do SVG, e gerar as imagens correspondentes às especificações contidas no documento. O Adobe SVG Viewer é um exemplo de *plug-in* de visualização para o SVG, o qual tem ganhado destaque por sua alta compatibilidade com as especificações do W3C.

A DTD do SVG define tipos de elementos[3] para manipulação de três categorias de objetos gráficos: figuras vetoriais, imagens e texto. A modelagem das figuras vetoriais, o principal enfoque do SVG, é realizada através da composição de elementos que descrevem estruturas geométricas primitivas, como por exemplo: retângulos, círculos, elipses, linhas e polígonos. Elementos especiais, denominados contêineres, agrupam essas composições, possibilitando transformações geométricas ou estéticas conjuntas.

---

[1] Um vocabulário XML é uma descrição de dados XML usados como meio de troca de informação, freqüentemente dentro de um domínio específico de atividade humana [AND 01]. Os termos: aplicativo XML e linguagem baseada em XML, também são empregados com esta mesma semântica [KIR 00, HOL 01].

[2] O ECMAScript é a padronização do JavaScript pela ECMA (*European Computer Manufactures Association*).

[3] O termo elemento, utilizado nesta seção, faz referência ao conceito de elemento empregado na nomenclatura do XML.

O SVG, de forma semelhante ao SWF, promove o reuso dos objetos gráficos através de estruturas de definição e instanciamento. Essas estruturas, além de contribuírem para a criação de padrões gráficos, proporcionam a produção de arquivos menores.

Os objetos gráficos do SVG podem ser animados através de três abordagens: usando os elementos de animação do próprio vocabulário, usando o SVG DOM (*Document Object Model*) ou integrando o conteúdo com o SMIL. O SMIL[1], também desenvolvido pelo W3C, é uma linguagem para integração de multimídia sincronizada que tem como objetivo a criação de apresentações multimídias na *Web*. A especificação da integração do SVG e SMIL não se encontra totalmente definida até o momento, portanto não será abordada neste trabalho.

Os elementos de animação do SVG foram desenvolvidos em colaboração com o W3C Synchronized Multimedia Working Group, criadores da especificação para animações no SMIL. Com isso, exceto por algumas regras específicas do SVG, a definição normativa para seus elementos e atributos de animação seguem as especificações das animações em SMIL.

O SVG possui os seguintes elementos de animação: *animate*, *set*, *animateMotion*, *animateColor* e *animateTransform*. Esses elementos descrevem as variações dos valores de atributos e propriedades dos elementos gráficos através do tempo. O controle do tempo é determinado por atributos específicos nos elementos de animação. Entre os atributos de controle de tempo, estão atributos que determinam o momento de início (*begin*), término (*end*) e duração (*dur*) da animação. Exceto pelo elemento *set,* que apenas utiliza um valor em sua definição, os demais elementos especificam as variações dos valores através de atributos que determinam valores chave de origem (*from*) e destino (*to*), ou ainda, listas de valores (*values*) que devem ocupar ao longo da animação.

A figura 2.3 ilustra um documento em SVG que representa a animação de um retângulo azul aumentando de tamanho. Note que *rect* é um elemento gráfico que contém uma animação descrita pelo elemento de animação *animate*. O atributo *attributeName* desse elemento especifica que a variação de valores será aplicada ao atributo *width* do elemento gráfico. A animação inicia no instante zero e dura nove segundos, provocando uma variação dos valores para a largura do objeto, de 50 até 400 *pixels*, neste intervalo de tempo. Os valores intermediários são interpolados segundo o modo especificado, neste caso, a ausência da especificação indica a interpolação linear.

```
<svg width="8cm" height="3cm"  viewBox="0 0 800 300"
     xmlns="http://www.w3.org/2000/svg">

  <rect x="10" y="100" width="0" height="50"
        fill="rgb(0,0,255)">
    <animate attributeName="width" attributeType="XML"
             begin="0s" dur="9s" fill="freeze" from="50" to="400" />
  </rect>
</svg>
```

FIGURA 2.3 – Animação de um retângulo em SVG

[1] A Microsoft, Macromedia e Compaq possuem uma alternativa ao SMIL, chamada Timed Interactive Multimedia Extension (HTML+TIME).

Além dos elementos de animação, os objetos gráficos também podem ser animados utilizando linguagens de script. O SVG oferece um conjunto adicional de interfaces DOM para suportar animação via linguagens de script, o que possibilita que animações sejam iniciadas e controladas através do uso de linguagens como o ECMAScript. Esta abordagem é utilizada principalmente na construção de animações interativas.

Quanto à recuperação de informação em documentos SVG, o vocabulário provê o elemento *Metadata*, o qual permite a inserção de metadados em qualquer elemento gráfico ou contêiner. A estrutura contida no *Metadata* pode ser definida pelo autor, porém, uma tendência incentivada pelo W3C, é a utilização de modelos de conteúdos baseados em RDF[1] (*Resourse Description Framework*). Um modelo bastante utilizado para descrever recursos na *Web* é o Dublin Core [DUB 2002], atualmente empregado por museus, bibliotecas, órgãos do governo e grupos comerciais.

2.4 Considerações Finais

As características da *Web*, descritas na seção 2.2, evidenciam algumas dificuldades a serem consideradas na formulação de novas estratégias para prover animações bidimensionais nesse ambiente.

A grande quantidade de espaço de armazenamento necessária para representar as animações é uma dessas dificuldades. Animações representadas em arquivos longos exigem mais espaço nos servidores de páginas, estratégias de *caching* específicas, tempos de transmissão maiores, recursos de armazenamento e processamento adicionais para indexação.

A recuperação de informação em animações é também uma questão relevante. À medida que mais informações são representadas na forma de animação, são necessárias estratégias de recuperação eficientes para o resgate deste conteúdo, assim como representações que colaborem com isso.

Outro ponto a ser considerado, diz respeito à criação e manutenção de animações. O conteúdo visual na *Web* é alterado constantemente. A facilidade de reutilização e alteração de animações contribui diretamente com o aumento da produtividade na criação e manutenção de páginas. A possibilidade de composição de novas animações a partir de outras proporciona a elaboração de conteúdos mais complexos em menos tempo.

[1] O RDF é um vocabulário XML utilizado como base para o processamento de metadados. Seu principal objetivo é o de facilitar o intercâmbio de informações entre aplicativos via *Web*.

3 Animação Baseada em Autômatos Finitos – Modelo AGA

3.1 Introdução

Durante uma animação, os atores podem sofrer diversos tipos de alteração, podendo ser alterados quanto à forma, posição, cor, textura, etc. Diversas dessas alterações são percebidas pelo observador de maneira contínua, mas na verdade são computadas para um número discreto de instantes.

Nas animações quadro-a-quadro, o arquivo de intercâmbio possui as imagens de cada um dos instantes armazenadas (quadros). Desta forma, as variações dos atores estão implicitamente representadas nas diferenças de um quadro para outro. Nas animações em tempo real, o arquivo de intercâmbio possui a especificação destas variações explicitamente, pois esta é necessária ao programa de visualização para o controle da animação dos atores durante a apresentação.

O modelo proposto AGA (Animação Gráfica baseada em Autômatos Finitos), apresentado neste capítulo, é uma alternativa para a especificação e controle de animações. Este modelo 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 seção 3.2 descreve em linhas gerais o AGA, evidenciando a estruturação da animação usando autômatos. Na seção 3.3, o modelo é definido utilizando como base o ator AGA, unidade básica estendida a partir de autômatos com saídas. A aplicação do modelo em um exemplo é demonstrada na seção 3.4. A seção 3.5 reúne as principais características do modelo para animações em tempo real.

3.2 Visão Geral do Modelo AGA

De maneira semelhante às animações em tempo real, descritas no capítulo 2, o AGA especifica a animação a partir de um conjunto de atores (objetos) e suas respectivas variações durante a animação. As especificações em AGA, entretanto, são suportadas por um modelo formal baseado em autômatos com saída [HOP 69, MEN 2000].

Nesse modelo, cada um dos atores na animação é especificado a partir de uma extensão proposta para o autômato com saída, a qual vincula as variações estéticas do ator à saída do autômato. Deste modo, quando os autômatos são simulados, mediante a leitura da fita de entrada, as transições entre seus estados controlam a animação dos atores.

A figura 3.1 ilustra a estrutura básica de especificação do modelo AGA aplicada a uma animação com dois atores. Os atores *maçã* e *bicho* são especificados a partir de autômatos com saída onde representações gráficas estão associadas às transições. Essas representações correspondem às variações gráficas que o ator pode sofrer durante a animação. Desta forma, quando o ator *maçã* realiza uma transição do estado *q1* para o estado *q2*, a representação gráfica do ator apresentada na animação é alterada para uma maçã mordida.

O AGA não restringe a forma de codificação das representações gráficas de cada ator, sendo assim podem ser empregadas tanto representações vetoriais quanto matriciais. No texto, o termo imagem é utilizado de forma genérica e pode ser entendido

como a representação pictórica do ator modelado independente de seu tipo de codificação.

A fita de entrada utilizada pelo autômato contém, além dos símbolos de entrada, especificações complementares sobre os instantes em que devem ocorrer as transições, assim como funções de controle para a saída do autômato. Essas funções de controle são responsáveis por diferentes tarefas, como por exemplo, posicionar o ator na área de animação.

A área de atuação dos atores pode ser dividida em camadas com o objetivo de estabelecer a ordem de sobreposição das figuras de saída. Neste exemplo, a representação gráfica do ator *maçã* (camada 1) deve ser sobreposta pela representação gráfica do ator *bicho* (camada 2) quando ambas ocuparem o mesmo lugar no plano. Assim, as cenas correspondentes a cada instante da animação são formadas pela unificação das camadas.

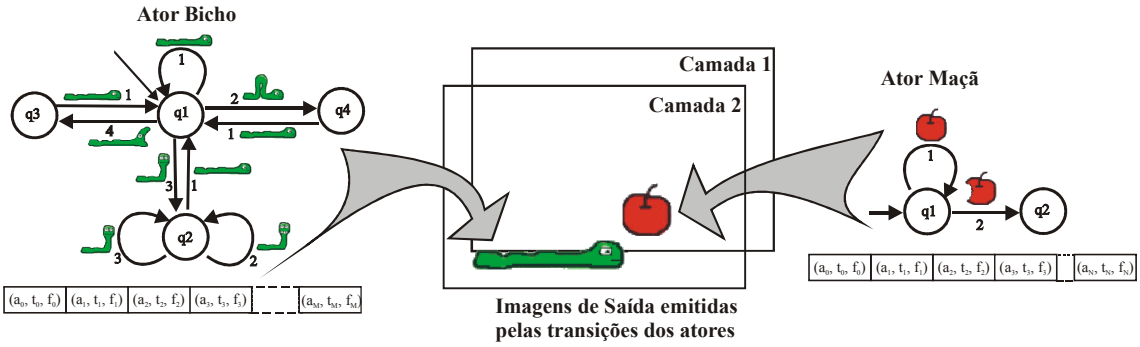


FIGURA 3.1 – Atores especificados no modelo AGA

O modelo AGA também provê a vinculação de descrições semânticas aos estados dos atores com o objetivo de favorecer mecanismos de consulta. Por exemplo, na figura 3.1, o estado *q2* do ator *maçã* pode ser vinculado à descrição "mordida". Desta maneira, uma verificação na especificação da animação possibilita a recuperação dos intervalos de tempo que o ator *maçã* ocupa o estado correspondente à situação "mordida".

3.3 Modelo AGA

A estrutura do modelo AGA é baseada em uma unidade básica, chamada de ator AGA, a qual é definida a partir de extensões propostas para os autômatos com saída. Essas extensões complementam a definição de autômato com saída com o objetivo de: suportar o controle temporal das transições; prover imagens como saída com adaptações contextuais; associar descrições semânticas aos estados.

3.3.1 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 [HOP 79, MEN 2000], 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:

- a saída é armazenada em uma fita independente, a qual não pode ser utilizada para leitura;

- 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;
- 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 desconsidera tal situação como entrada.

Devido à equivalência entre as duas abordagens de autômatos com saída, o ator AGA pode ser estendido tanto vinculando a saída ás transições quanto aos estados. Nos artigos publicados pelo autor, as duas abordagens são exploradas. Em [ACC 2000] (Anexo 1), o modelo AGA é apresentado utilizando como base a Máquina de Moore e, em [ACC 2001a] (Anexo 2), utilizando a abordagem de saída proposta pela Máquina de Mealy combinada ao modelo de autômato temporizado.

Contudo, a extensão do ator AGA a partir da Máquina de Mealy foi preferida para a apresentação e implementação do modelo, pois como associa as imagens de saída às transições, os estados podem ser incluídos de acordo com a modelagem desejada para o ator, independente do número de imagens no alfabeto de saída. As descrições semânticas associadas aos estados também não são vinculadas à mesma imagem, pois várias transições com imagens de saída diferentes podem ser mapeadas para o mesmo estado. Essas características proporcionam uma flexibilidade maior para a modelagem dos atores.

Assim, para as extensões propostas à frente, a Máquina de Mealy [HOP 79, MEN 2000], utilizada como base, é um Autômato Finito Determinístico com saídas associadas às transições e representada pela 6-upla, $ME = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, onde:

Q conjunto finito de estados possíveis do autômato;
Σ alfabeto de símbolos de entrada;
Δ alfabeto de símbolos de saída;
δ função programa ou função de transição (função parcial $\delta: Q \times \Sigma \rightarrow Q$);
λ função de saída[1] (função parcial $\lambda: Q \times \Sigma \rightarrow \Delta^*$);
q_0 estado inicial do autômato ($q_0 \in Q$).

O processamento de uma Máquina de Mealy, para uma palavra de entrada w, consiste na sucessiva aplicação da função programa para cada símbolo de w até ocorrer uma condição de parada[2]. Á medida que cada símbolo de w é lido, a palavra, vinculada como saída à transição efetuada, é inserida na fita de saída. A palavra vazia como saída da função λ indica que nenhuma gravação é realizada na fita de saída.

3.3.2 Função de Saída Contextual

A primeira extensão para a Máquina de Mealy é realizada quanto à saída do autômato. O objetivo dessa extensão é adequar a saída à emissão de imagens, de tal forma que estas possam ser contextualizadas a diversas situações durante a animação.

Com esse objetivo, o alfabeto de saída Δ, para o ator AGA, é definido como um conjunto finito de imagens estáticas, onde cada elemento é uma imagem distinta que o

[1] Em [HOP 79], o modelo definido para a Máquina de Mealy grava apenas um símbolo a cada transição, já em [MEN 00], a definição promove a associação de uma palavra de saída a cada transição. Para este trabalho, a segunda definição foi utilizada por ser mais oportuna para a extensão do modelo.
[2] As condições de parada são as mesmas estabelecidas para o Autômato Finito Determinístico.

ator pode utilizar durante a apresentação. O conjunto Δ^*, portanto, é composto por todas as imagens que podem ser produzidas pela composição de zero ou mais imagens de Δ. A operação de composição pode ser definida da maneira mais conveniente à classe de animações representadas. Para a elaboração dos exemplos e a implementação do modelo, a palavra $i_1 i_2 .. i_n \in \Delta^*$ representa a composição $i_1+i_2+...+i_n=i_r$ que produz a imagem i_r a partir da sobreposição da imagem i_1 pela i_2 e assim sucessivamente até i_n.

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. Para esse propósito, é introduzido o conjunto F, cujos elementos são funções do tipo $f(\vec{v},i)=i^t$, onde i e i^t são elementos de Δ^F e $\vec{v}=v_1 v_2 ... v_n$ é um vetor de argumentos necessários para o mapeamento de i para i^t. O conjunto Δ^F é composto por todas as imagens de Δ^* transformadas por zero ou mais funções de F.

Assim, a função de saída estendida, chamada de função de saída contextual, é definida como a função parcial $\lambda_c: Q \times \Sigma \times F^* \rightarrow \Delta^F$, onde F^* é o conjunto de cadeias $\varphi=f_1 f_2 ... f_m$ formadas por zero ou mais funções de F. Logo $\lambda_c(q,a,\varphi) = f_1(v1\vec{}, f_2(v2\vec{},... f_m(vm\vec{},i)))$, onde a imagem $i \in \Delta^*$ está associada a transição a que diverge do estado q. Uma cadeia φ é associada a cada um dos símbolos da fita de entrada para determinar quais transformações devem ser aplicadas a cada transição do autômato.

Por exemplo, para o ator *bicho* apresentado na figura 3.1, o conjunto F possui a função *Mirror*, a qual produz o espelhamento da imagem no seu eixo horizontal (argumento *H*) ou vertical (argumento *V*). Então, para a animação representada na figura 3.2, a leitura da terceira célula da fita de entrada *(1,... , Mirror(V))* provoca a transição do estado *q4* para o estado *q1* e permuta a imagem 2 corrente pela imagem 1 transformada pela função *Mirror(V)*.

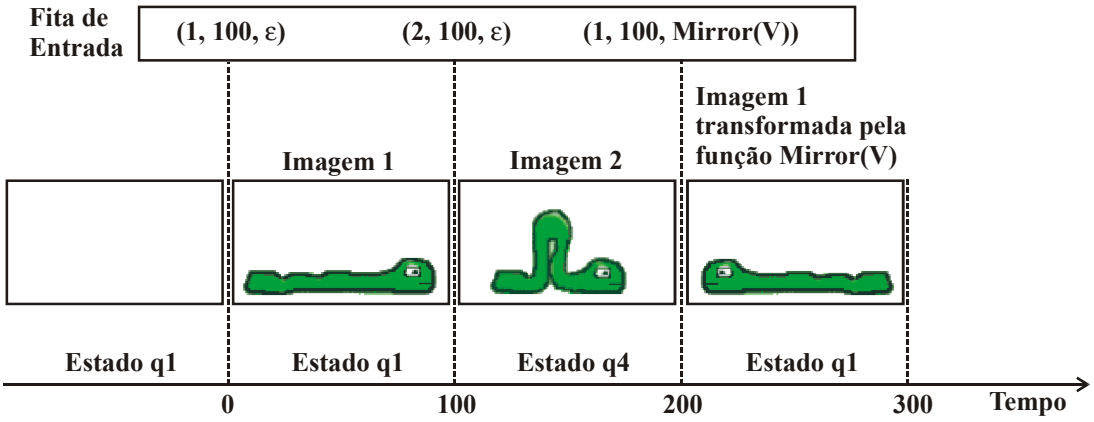


FIGURA 3.2 – Troca de imagens do ator bicho provocadas pela leitura da fita

Note que a animação do ator *bicho* é decorrente da leitura da fita de entrada, a qual determina o comportamento do ator durante a apresentação, pelas trocas de imagens provocadas pelos símbolos de entrada. Assim, para alterar o comportamento do ator na animação, basta modificar a fita de entrada

3.3.3 Função de Transição Temporal

Além de determinar quais as variações que os atores podem sofrer durante a animação, é necessário especificar em que momentos deverão ocorrer. Como as variações estão associadas às transições do autômato, é preciso determinar em que momentos essas variações deverão ocorrer durante o processo de avaliação da fita de entrada.

Com este propósito, 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 $ft=(a_1,t_1,\varphi_1)(a_2,t_2,\varphi_2)...(a_n,t_n,\varphi_n)$ possui os triplos ordenados $(a,t,\varphi) \in \Sigma \times N \times F^*$, 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.

Para o comportamento temporal do autômato, as transições são consideradas instantâneas e o tempo passa durante a ocupação de um estado. Assim, para a entrada $(a_1,t_1,\varphi_1)(a_2,t_2,\varphi_2)...(a_n,t_n,\varphi_n)$ que provoca a seqüência de ocupação dos estados $q_0q_1...q_n$, o tempo t_k, para algum $1 \leq k \leq n$, determina o tempo de ocupação do estado q_k.

Deste modo, a função de transição é estendida para comportar um argumento adicional que determina o tempo que deve ser esperado após ser efetuada a transição. Esse argumento é obtido a cada célula lida a partir da fita de entrada. Portanto, a função de transição temporal é definida como a função parcial $\delta_t: Q \times \Sigma \times N \rightarrow Q$, para a qual, a especificação $\delta_t(q, a, t)=q'$ determina que deve ocorrer uma transição de q para q' quando o símbolo a é lido, e depois, aguardado o tempo t no estado q' antes da leitura do próximo símbolo.

A figura 3.2 ilustra uma seqüência animada com 300 milésimos de segundo de duração. Inicialmente, o ator AGA se encontra no estado *q1* e no instante 0, com a leitura da primeira célula $(1, 100, \varepsilon)$, ocorre uma transição instantânea para o estado *q1*, projetando a imagem vinculada a este arco como saída. Após essa transição é esperado um tempo de 100 milésimos de segundo para a leitura da próxima célula. Note que a imagem emitida como saída permanece visível até a próxima transição.

3.3.4 Função Descrição

A última extensão proposta para a Máquina de Mealy é destinada à vinculação de descrições semânticas aos estados. A estrutura da especificação do modelo AGA baseada em um conjunto de atores colabora para que o comportamento de cada ator possa ser verificado a partir de seu autômato correspondente. Desta forma, os estados dos autômatos podem ser utilizados para indicar diferentes situações durante a animação.

Com esse propósito, foi criada a função descrição que mapeia os estados de cada ator AGA para um conjunto de descrições. A forma de descrição semântica do estado não é definida neste texto, pois pode ser explorada de diferentes maneiras dependendo do mecanismo de consulta. Contudo, para os exemplos foram empregadas descrições baseadas em palavras-chave.

A função descrição é definida como a função parcial $\sigma: Q \rightarrow D$, onde D é o conjunto de descrições utilizadas pelo ator AGA especificado. Nem todo estado precisa necessariamente estar mapeado para alguma descrição em D. Por exemplo, para o ator A na figura 3.3, os seguintes mapeamentos são definidos para o conjunto $D=\{$*"Alerta", "Boca Aberta", "Contraído"*$\}$: $\sigma(q2)=$*"Alerta"* , $\sigma(q3)=$*"Boca Aberta"* e $\sigma(q4)=$*"Contraído"*. Desta maneira, toda vez que o ator A ocupar o estado 2 durante a animação é sabido que o ator está na situação de *"Alerta"*.

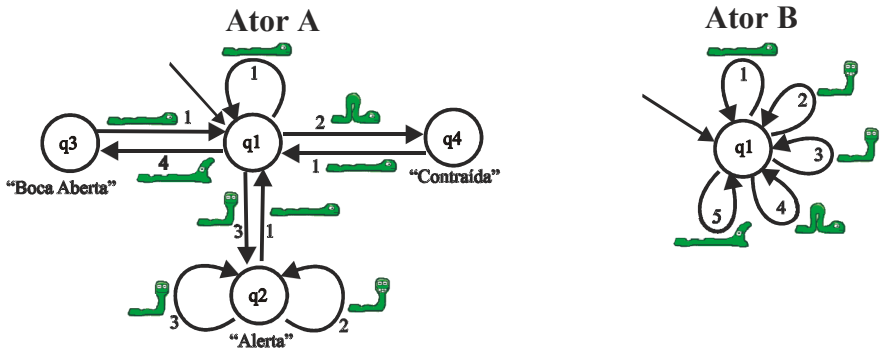


FIGURA 3.3 – Alternativas de construção dos atores AGA

Note que a criação de estados na modelagem do ator AGA pode ser explorada com dois objetivos principais: definir o padrão de comportamento das trocas de imagens e documentar semanticamente os estados. Por exemplo, a figura 3.3 ilustra dois atores que possuem a capacidade de emitir as mesmas imagens como saída. O ator A é modelado com vários estados, enquanto o ator B possui um único estado. O ator B permite que qualquer troca de imagem seja efetuada e não possui nenhuma atribuição de descrição que indique a situação do ator na animação. Já, no ator A, é possível saber quando o ator está *"Alerta"*, *"Contraído"* ou *"Boca Aberta"*, como também verificar que este ator nunca poderá emitir uma imagem com a boca aberta e logo em seguida contrair-se. Essas características são propriedades individuais de cada ator e são respeitadas em qualquer animação.

3.3.5 Animação como um Conjunto de Atores AGA

Reunindo as extensões propostas, o ator AGA é definido então como uma 9-upla: $At=(Q, \Sigma, \Delta, \delta_t, \lambda_c, q_0, \sigma, F, D)$, onde:

- Q conjunto finito de estados possíveis do autômato;
- Σ alfabeto de símbolos de entrada;
- Δ alfabeto de símbolos de saída;
- δ_t função de transição temporal (função parcial $\delta_t: Q \times \Sigma \times N \rightarrow Q$);
- λ_c função de saída contextual (função parcial $\lambda_c: Q \times \Sigma \times F^* \rightarrow \Delta^F$);
- q_0 estado inicial do autômato ($q_0 \in Q$);
- σ função descrição (função parcial $\sigma: Q \rightarrow D$);
- F conjunto finito de funções de transformação de Δ^F ;
- D conjunto finito de descrições.

A fita de entrada utilizada pelo ator AGA é limitada ao tamanho da entrada e definida como $ft=(a_1,t_1,\varphi_1)(a_2,t_2,\varphi_2)...(a_n,t_n,\varphi_n)$, na qual cada célula possui o triplo ordenado $(a,t,\varphi) \in \Sigma \times N \times F^*$, 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 de saída.

Portanto, seja A um conjunto de atores AGA e T um conjunto de fitas de entrada para os atores AGA, uma animação no modelo AGA, ou simplesmente AGA, é um conjunto totalmente ordenado de pares ordenados $(At, ft) \in A \times T$, cuja relação de ordem total, denotada por $\preceq$, é definida como: $(At_1, ft_1) \preceq (At_2, ft_2)$ se e somente se At_1 atua em uma camada menor ou igual à camada de At_2 na área de animação. Assim, $AGA= (\{(At, ft) \mid At \in A \text{ e } ft \in T\}, \preceq)$, onde todos os atores iniciam o processamento de suas fitas de entrada no mesmo instante.

Desta maneira, a apresentação de uma animação modelada em AGA pode ser obtida a partir da simulação dos atores AGA, na qual estes processam em paralelo suas

fitas de entrada correspondentes, e emitem as imagens de saída em uma mesma cena animação respeitando as camadas determinadas pela relação de ordem.

3.4 Exemplo

O exemplo proposto ilustra a modelagem em AGA da animação representada na figura 3.4. A animação tem duração de 1900 ms e cada uma das imagens representa a situação dos atores a partir do instante indicado no canto superior esquerdo.

A modelagem da animação é criada a partir de dois atores: *Bicho* e *Maçã*, os quais são especificados como segue:

- *Bicho = ({q1, q2, q3, q4}, {1, 2, 3, 4}, Δ_1, δ_{t1}, λ_{c1}, q1, σ_1, {Trans(x,y)}, {"Boca Aberta", "Contraído", "Alerta"})*, onde Δ_1, δ_{t1} e λ_{c1} estão descritos na figura 3.1 (Ator *Bicho*). A imagem mais próxima à transição indica a associação da saída. A função σ_1 é mapeada como: σ_1 *(q2)="Alerta"*, σ_1 *(q3)="Boca Aberta"* e σ_1 *(q4)="Contraída"*.
- *Maçã = ({q1, q2}, {1, 2}, Δ_2, δ_{t2}, λ_{c2}, q1, σ_2, {Trans(x,y)}, {"Mordida"})*, onde Δ_2, δ_{t2} e λ_{c3} estão descritos na figura 3.1 (Ator *Maçã*) e σ_2*(q2)="mordida"*.

A função *Trans (x,y)*, pertencente ao conjunto de funções *F* de cada ator, efetua a translação da imagem de saída nos eixos *X* e *Y*. A origem dos eixos é o canto superior esquerdo da área de animação.

Para esta animação são especificadas duas fitas de entrada, uma para o ator *Bicho* e outra para o ator *Maçã*, respectivamente, como segue:

- ft_{Bicho}*=(1, 200, Trans(2,50)) (2, 200, Trans(32,0)) (1, 200, Trans(32,0)) (4, 200, Trans(32,0)) (1, 200, Trans(32,0)) (3 ,200, Trans(32,0)) (2, 500, Trans(32,0)) (3, 200, Trans(32,0))*.
- $ft_{Maçã}$*=(1, 800, Trans(142,70)) (2, 1100, Trans(142, 70))*.

Portanto, a animação é especificada como *AGA= ({(Bicho,* ft_{Bicho}*), (Maçã,* $ft_{Maçã}$*)}, $\preceq$)*, onde *Maçã*$\prec$*Bicho*. Note que o modelo descreve todo o comportamento dos atores na animação de tal forma que a simulação dos autômatos pode ser utilizada para compor as imagens de saída em qualquer um dos instantes do intervalo 0 até 1900 ms.

Na figura 3.4, abaixo de cada imagem, são representados os estados correntes de cada ator e suas transições durante a simulação. Note que, primeiramente, ambos os atores ocupam o estado inicial, e no instante 0 com a leitura da primeira célula de cada fita, as transições dos atores são provocadas emitindo as respectivas imagens de saída. Os atores realizam suas transições, indicadas pelas setas na figura 3.4, de forma independente mediante a leitura do conteúdo de cada fita. Na animação representada, o ator *Maçã* realiza apenas duas transições durante a animação, enquanto o outro ator realiza oito. As fitas foram construídas para que o fechamento da boca do bicho coincida com a troca de imagem da maçã.

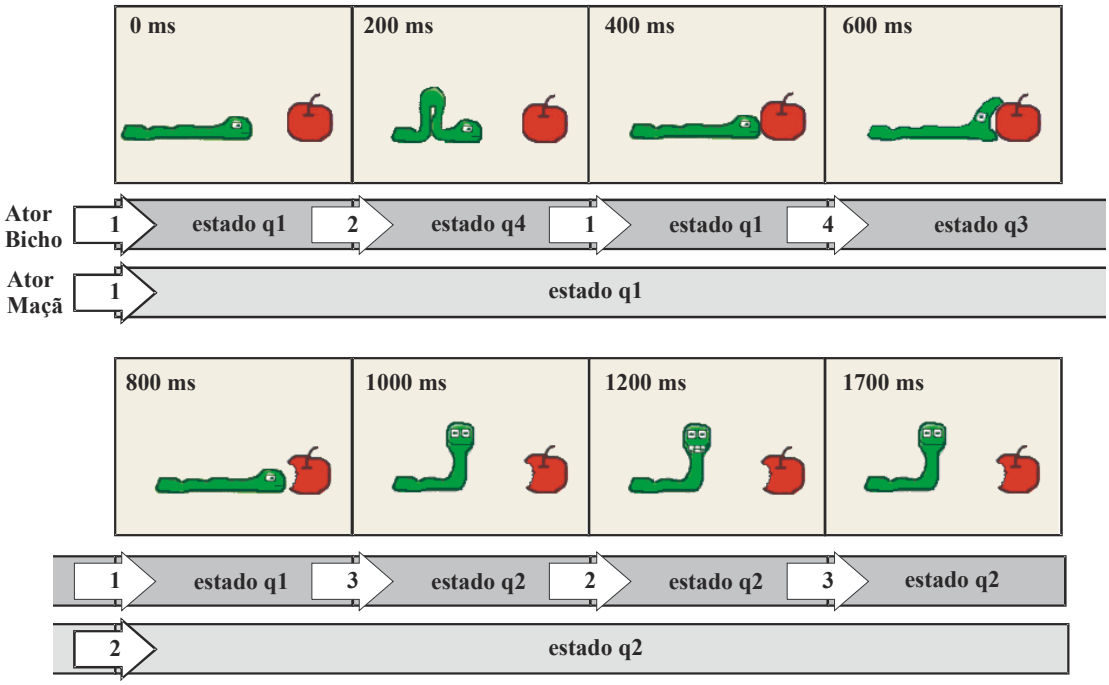


FIGURA 3.4 – Animação envolvendo dois atores AGA

É importante ressaltar, que pela análise dos estados correntes mediante a leitura da fita de entrada, é possível verificar que o ator *Bicho* está em "alerta" e *Maçã* está "mordida" no intervalo de 1000 ms até 1900 ms. Da mesma forma, várias outras situações podem ser verificadas pelas associações estabelecidas pela função descrição.

3.5 Considerações Finais

O modelo AGA possui características que favorecem sua aplicação na especificação e controle de animações para a *Web*. Essas características estão ligadas às questões evidenciadas no capítulo 2, quanto ao espaço de armazenamento, suporte a recuperação de informação e manutenção do conteúdo das animações.

A definição do alfabeto de saída como um conjunto de imagens básicas utilizadas na animação está diretamente ligada ao conceito de dicionário proposto em abordagens como o SWF e SVG. Este tipo de organização contribui para a redução do espaço de armazenamento da animação, pois uma mesma imagem é reutilizada em diversos momentos sem a necessidade de representá-la novamente. O AGA não restringe a forma de codificação de cada imagem do alfabeto, sendo assim podem ser empregadas tanto representações vetoriais quanto matriciais.

A descrição semântica, mapeada a partir da função descrição, colabora para a recuperação de informação baseada na verificação dos estados dos atores AGA. Assim, é possível resgatar intervalos na animação onde os atores AGA estejam em situações requeridas por alguma consulta. Como as descrições estão associadas ao ator AGA, qualquer animação que ele participe está inerentemente mapeada para consulta.

O encapsulamento das propriedades estéticas e comportamentais dos atores em uma unidade básica, o ator AGA, favorece que estes sejam reutilizados em diferentes animações. As variações comportamentais desses atores em cada animação podem ser exploradas simplesmente pela troca da fita de entrada. Essas características também colaboram para que animações novas sejam formadas a partir da combinação de animações existentes.

4 Implementação do Modelo AGA

4.1 Introdução

A implementação foi desenvolvida com o objetivo de prover a reprodução de animações em AGA na *Web*. Desta forma, as decisões tomadas quanto às tecnologias empregadas no desenvolvimento tiveram como objetivo manter a portabilidade dos produtos desenvolvidos para várias plataformas.

O desenvolvimento foi realizado em duas etapas: a definição de uma linguagem para a especificação destas animações, chamada de AgaML; e a implementação de um programa para visualização destas na *Web*, tratado no texto como *AGA* Player. Tanto a linguagem AgaML como o AGA Player foram desenvolvidos especificamente para o AGA a fim de demonstrar as potencialidades do modelo.

A seção 4.2 descreve o funcionamento geral da proposta implementada, enfocando a interação com a *Web*. A sintaxe e a semântica da linguagem AgaML são descritas na seção 4.3. As tecnologias utilizadas na implementação do AGA Player e suas funcionalidades de reprodução e consulta são descritas na seção 4.4. A seção 4.5 evidencia a potencialidade da implementação atual para expansões futuras.

4.2 Visão Geral da Implementação

A abordagem utilizada na implementação do AGA é baseada na produção de animações que executem em tempo real. A estratégia adotada é semelhante às empregadas pelo Flash e o SVG, nas quais os quadros são gerados no momento da apresentação a partir do processamento de um arquivo de intercâmbio contendo as especificações da animação.

Nas animações desenvolvidas em AGA, o arquivo de intercâmbio contém as especificações dos atores AGA e suas respectivas fitas de entrada descritas em AgaML. Essa linguagem é um vocabulário XML desenvolvido especificamente para esse propósito. Os arquivos em AgaML são disponibilizados pelo servidor HTTP juntamente com as imagens e sons[1] utilizados como saída pelos atores AGA[2].

A carga do arquivo de intercâmbio e a geração dos quadros são realizadas pelo programa de visualização, AGA Player, executado no navegador do cliente. O AGA Player, desenvolvido em linguagem Java na forma de *applet,* é carregado pelo cliente quando alguma página *Web* que o vincula é processada pelo navegador. Como o *applet* desenvolvido utiliza recursos da plataforma Java que os navegadores ainda não suportam totalmente (processamento de documentos em XML), é necessário que o cliente tenha instalado adicionalmente o Java *plug-in,* fornecido pela Sun Microsystems, para processá-lo. Este *plug-in* basicamente redireciona a interpretação do *applet* para JRE (*Java Runtime Enviroment*) instalada no cliente que provê esses recursos adicionais.

A animação a ser reproduzida é associada à página *Web* através de um dos parâmetros fornecidos ao *applet*. Assim, quando o AGA Player é executado, o arquivo

---

[1] Atualmente as imagens podem ser gravadas como mapa de bits nos formatos: *Graphics Interchange Format* (GIF) e *Joint Experts Group* (JPEG), e os sons nos formatos: *Sun Audio* (extensão *.au*), *Windows Wave* (extensão *.wav*), *Macintosh AIFF* (extensões *.aif* ou *.aiff*) e *Musical Instrument Digital Interface* – MIDI (extensões *.mid* ou *.rmi*).

[2] A vinculação de sons como saída nos atores AGA é um recurso proposto no modelo AGA-S (que será descrito no capítulo 6). A implementação atual para este recurso consiste em uma etapa intermediária para a extensão da implementação para modelo AGA-S.

em AgaML atribuído ao parâmetro é carregado juntamente com as mídias externas e processado produzindo a animação.

O *applet* possui outro parâmetro que determina se a animação deve ser reproduzida com ou sem o painel de controle. Este recurso favorece tanto a criação de animações simples de aplicação direta nas páginas, como o caso dos *banners* de propaganda, quanto animações mais complexas, as quais o observador precisa explorara o conteúdo da seqüência animada.

Note que o desenvolvimento do AGA Player na forma de *applet* proporciona uma solução multiplataforma, pois este é interpretado pela máquina virtual Java instalada no próprio navegador da máquina cliente. Os formatos de arquivos utilizados para o AgaML e a codificação das mídias externas são padrões amplamente utilizados na Internet e podem ser manipulados independentemente de plataforma. A figura 4.1 ilustra a arquitetura multiplataforma do sistema envolvida na implementação do modelo AGA.

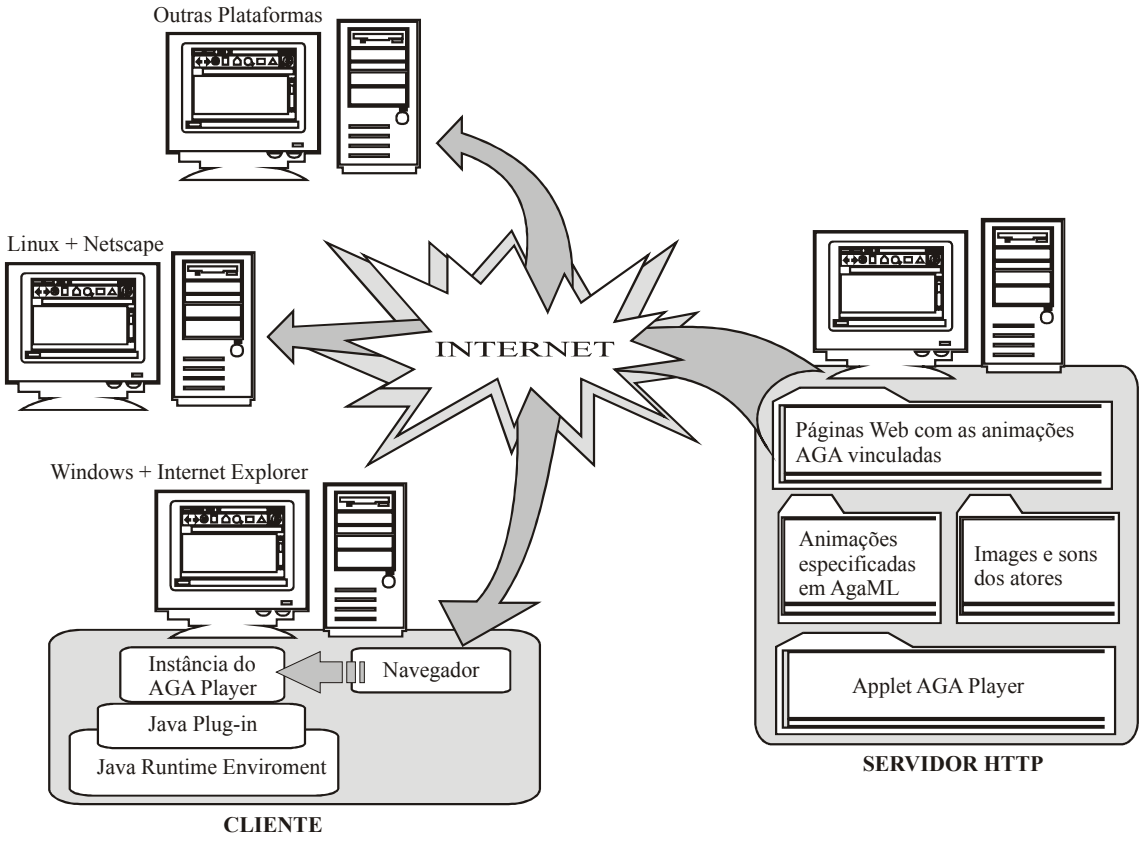


FIGURA 4.1 – Arquitetura do Sistema

4.3 Linguagem AgaML

A AgaML (AGA *Markup Language*) é um vocabulário XML desenvolvido especialmente para especificar as animações no modelo AGA. Duas características do XML motivaram sua escolha para o desenvolvimento da AgaML: sua ampla aplicabilidade como padrão de intercâmbio de documentos na *Web*; e a facilidade de manipulação tanto manual quanto por processos automatizados. A segunda característica colabora para que a AgaML seja utilizada tanto como linguagem fonte

para edição direta pelo usuário quanto em ambientes automatizados de animação na qualidade de linguagem intermediária.

O desenvolvimento da linguagem AgaML teve como foco os seguintes requisitos:

- prover uma estrutura sintática tanto para a especificação dos atores como para as fitas de entrada seguindo a definição formal proposta pelo modelo AGA;
- possibilitar a utilização da mesma especificação de ator ou fita para a criação de múltiplas instâncias na animação;
- agrupar as especificações dos atores em unidades sintáticas coesas de tal modo a possibilitar sua fácil reutilização e compartilhamento pelas facilidades herdadas do XML;
- integrar símbolos do alfabeto de saída definidos externamente, com o objetivo de utilizar imagens e sons em diferentes formatos e localizações;
- fornecer informações adicionais para catalogação da animação com o objetivo de favorecer mecanismos automáticos de consulta.

AgaML organiza a especificação da animação a partir de três componentes básicos: a especificação dos atores AGA, a especificação das fitas de entrada e a criação das instâncias dos atores. Uma instância pode ser entendida como a associação da especificação de um ator AGA com uma fita de entrada. A figura 4.2 ilustra a especificação de um ator AGA sendo associado a duas fitas de entrada diferentes, criando assim, duas instâncias independentes.

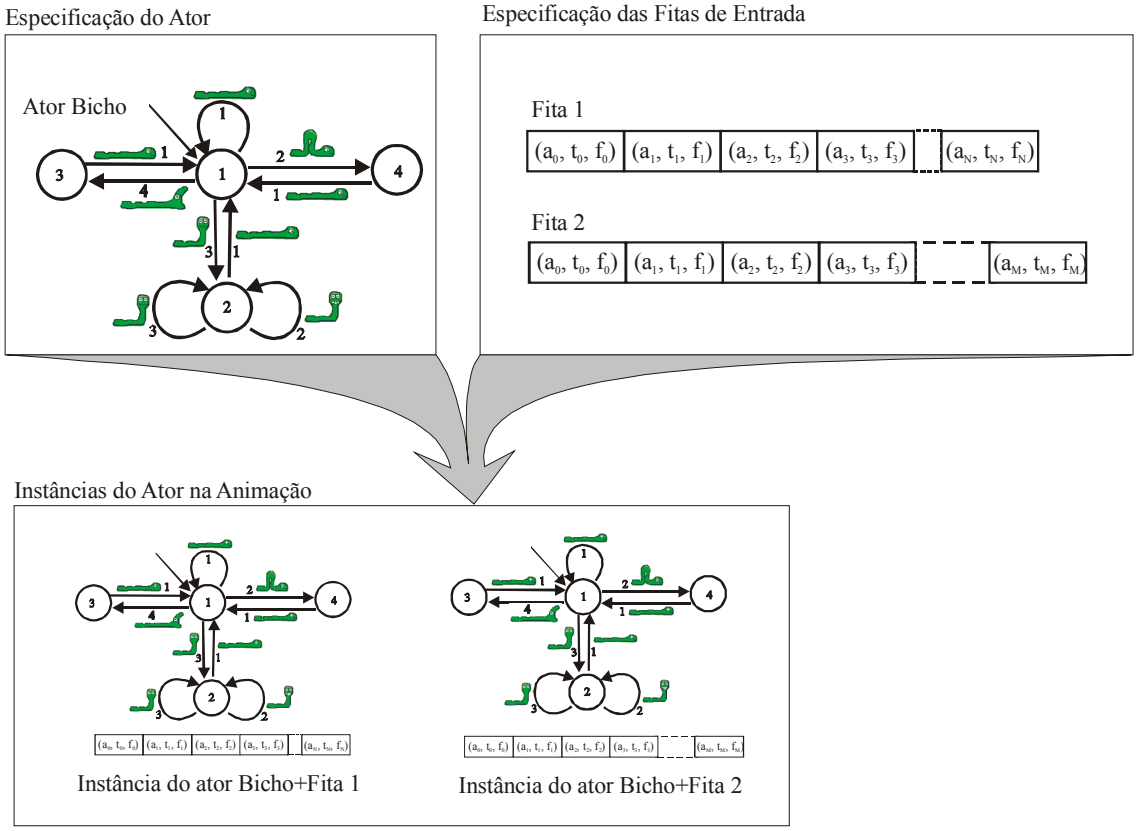


FIGURA 4.2 – Estrutura de instanciação dos atores

As seções seguintes descrevem os elementos definidos na DTD da AgaML acompanhados de exemplos aplicados. A DTD é utilizada como padrão para definição

das regras sintáticas de novos vocabulários XML. A DTD completa da AgaML versão 2.0 se encontra no anexo 4.

4.3.1 Especificação da Animação

O elemento raiz *AGA* da DTD (figura 4.3) comporta as especificações dos atores, das fitas e as definições das instâncias dos atores. Além disto, disponibiliza informações adicionais para auxiliar no processo de apresentação da animação.

```
<!ELEMENT AGA (HEAD, ACTOR+, TAPE+, INSTANCE+)>
<!ATTLIST AGA
        VERSION CDATA #FIXED "2.0"
        WIDTH CDATA #REQUIRED
        HEIGHT CDATA #REQUIRED
        BACKGROUND CDATA #REQUIRED
        FRAMERATE CDATA #REQUIRED
        REPEAT CDATA #IMPLIED>
<!ELEMENT HEAD (TITLE, AUTHOR, SUBJECT)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
<!ELEMENT SUBJECT (#PCDATA)>
```

FIGURA 4.3 – Trecho da DTD com os elementos *AGA*, *HEAD*, *TITLE, AUTHOR* e *SUBJECT*

Os atributos do elemento *AGA*: *WIDTH*, *HEIGHT*, *BACKGROUND*, *FRAMERATE* e *REPEAT,* descrevem, respectivamente, a largura e altura da janela onde será exibida a animação, a cor de fundo, o número de quadros por segundo e o número de vezes que a animação deve ser repetida. A palavra *LOOP* pode ser associada ao atributo *REPEAT* para provocar a repetição contínua da animação.

O valor atribuído ao *FRAMERATE* determina a freqüência com que os autômatos serão verificados pelo AGA Player para a formação dos quadros de saída. É recomendado que o valor do *FRAMERATE* seja compatível com as temporizações especificadas na fita de entrada, pois se o intervalo entre a composição de um quadro e outro for superior ao intervalo entre as transições, nem todas mudanças das imagens de saída serão observadas.

A figura 4.4 mostra a definição de uma animação de dimensões 200 por 100 (*pixels*) com a cor de fundo branca[1] que deve ser repetida continuamente com uma freqüência de renovação de 24 quadros por segundo.

```
<AGA VERSION="2.0" WIDTH="200" HEIGHT="100" BACKGROUND="255 255 255"
FRAMERATE="24" REPEAT="LOOP">
        <HEAD>
                <TITLE>Exemplo de AgaML </TITLE>
                <AUTHOR>Fernando Accorsi </AUTHOR>
                <SUBJECT> Acadêmico; Linguagem AgaML</SUBJECT>
        </HEAD>
        <!- -Especificação dos Atores e Fitas- ->
        <!- - Criação das Instâncias- ->
</AGA>
```

FIGURA 4.4 – Exemplo com o cabeçalho *HEAD*

O elemento *HEAD* tem como objetivo atender as estratégias de indexação orientadas a metadados e taxonomias [ARA 2000]. Composto por *TITLE*, *AUTHOR* e

[1] O atributo BACKGROUND é especificado utilizando o padrão de cores RGB.

SUBJECT é utilizado para armazenar informações bibliográficas e descrições adicionais para categorizações subjetivas, como por exemplo, classes semânticas e classes visuais.

4.3.2 Especificação do Ator AGA

O ator na AgaML é definido a partir da extensão da Máquina de Mealy, assim como descrito no capítulo 3. O *ACTOR*, elemento criado para este propósito, contém elementos filhos que descrevem o alfabeto de saída Δ (*OUTPUT*), a função descrição σ (*DESCF*) e a função transição temporal δ_t (*TRANSF*) (figura 4.5). Os atributos *ID*, *TYPE*, *STATES* e *SYMBOLS* dessa estrutura comportam, respectivamente, as informações sobre a identificação do ator, o seu tipo, as quantidades de estados e símbolos do seu alfabeto de entrada. A identificação é utilizada na criação da instância para a associação da especificação do ator com a fita de entrada. O tipo determina se o ator possui imagens (GRAPHIC) ou sons (SOUND) como alfabeto de saída associado.

```
<!ELEMENT ACTOR (OUTPUT*, DESCF?, TRANSF)>
<!ATTLIST ACTOR
        ID CDATA #REQUIRED
        TYPE CDATA #REQUIRED
        STATES CDATA #REQUIRED
        SYMBOLS CDATA #REQUIRED>
<!ELEMENT OUTPUT EMPTY>
<!ATTLIST OUTPUT
        ID CDATA #REQUIRED>
        SOURCE CDATA #REQUIRED>
        X CDATA #IMPLIED>
        Y CDATA #IMPLIED>
<!ELEMENT DESCF (DESCRIPTION)+>
<!ELEMENT DESCRIPTION>
<!ATTLIST DESCRIPTION
        STATE CDATA #REQUIRED>
<!ELEMENT TRANSF (FROM+)>
<!ELEMENT FROM (TO+)>
<!ATTLIST FROM
        STATE CDATA #REQUIRED>
<!ELEMENT TO EMPTY>
<!ATTLIST TO
        STATE CDATA #REQUIRED
        SYMBOL CDATA #REQUIRED
        OUTPUT CDATA #IMPLIED>
```

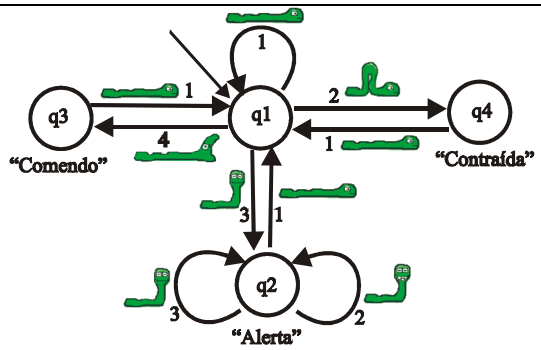
FIGURA 4.5 – Trecho da DTD com os elementos *ACTOR*, *OUTPUT*, *DESCF*, *DESCRIPTION*, *TRANSF*, *FROM* e *TO*

A integração das imagens e sons do alfabeto de saída Δ é realizada através do elemento *OUTPUT*, o qual define uma identificação interna para o símbolo (*ID*) e a associa com o arquivo que contém a imagem ou som (SOURCE). Os atributos *X* e *Y* representam os deslocamentos que devem ser aplicados à imagem para o enquadramento com as demais. Por exemplo, na figura 4.6, o ator *bicho*, possui imagens com alturas diferentes. Deste modo, quando uma imagem é trocada pela outra na saída, é preciso efetuar um enquadramento para que não haja uma variação na posição do ator. Alternativamente, esta correção pode ser feita através das transformações contidas na célula de entrada, porém, isto implica na replicação desta informação em cada célula.

É importante notar que o elemento *OUTPUT* possibilita independência entre a linguagem AgaML e o formato de codificação de cada mídia. Novos formatos de mídia

podem ser utilizados a partir da implementação dos respectivos decodificadores no AGA Player.

A função descrição, definida no modelo AGA, é representada por *DESCF*. Os elementos filhos *DESCRIPTION* associam o estado do ator AGA (atributo *STATE*) com um texto. Neste texto podem ser incluídas várias palavras que descrevam semanticamente a situação do ator naquele estado. Estas palavras são utilizadas como palavras-chave para consulta no AGA Player. Na figura 4.6, por exemplo, o estado *q2* do ator é associado à descrição "Alerta", pois, durante a ocupação deste estado, a imagem mostrada na animação será do bicho com a cabeça levantada.




```
<ACTOR ID="bicho" TYPE="GRAPHICS" STATES="4" SYMBOLS="4">
	<OUTPUT ID="1" SOURCE="contraida.gif" X="0" Y="13"/>
	<OUTPUT ID="2" SOURCE="estendida.gif" X="0" Y="42"/>
	<OUTPUT ID="3" SOURCE="aberta.gif" X="0" Y="22"/>
	<OUTPUT ID="4" SOURCE="alerta.gif" X="0" Y="0"/>
	<OUTPUT ID="5" SOURCE="rindo.gif" X="0" Y="0"/>
	<DESCF>
		<DESCRIPTION STATE="q3">Comendo</DESCRIPTION>
		<DESCRIPTION STATE="q2">Alerta</DESCRIPTION>
		<DESCRIPTION STATE="q4">Contraída</DESCRIPTION>
	</DESCF>
	<TRANSF>
		<FROM STATE="q1">
			<TO STATE="q1" SYMBOL="1" OUTPUT="2"/>
			<TO STATE="q4" SYMBOL="2" OUTPUT="1"/>
			<TO STATE="q2" SYMBOL="3" OUTPUT="4"/>
			<TO STATE="q3" SYMBOL="4" OUTPUT="3"/>
		</FROM>
		<FROM STATE="q2">
			<TO STATE="q1" SYMBOL="1" OUTPUT="2"/>
			<TO STATE="q2" SYMBOL="2" OUTPUT="5"/>
			<TO STATE="q2" SYMBOL="3" OUTPUT="4"/>
		</FROM>
		<FROM STATE="q3">
			<TO STATE="q1" SYMBOL="1" OUTPUT="2"/>
		</FROM>
		<FROM STATE="q4">
			<TO STATE="q1" SYMBOL="1" OUTPUT="2"/>
		</FROM>
	</TRANSF>
</ACTOR>
```

FIGURA 4.6 – Exemplo do elemento *ACTOR* para a especificação do ator *bicho*

As transições do ator e a vinculação com a saída são obtidas através do elemento *TRANSF*. Cada elemento filho *FROM* contém os arcos que divergem do estado (*STATE*) representados pelos elementos vazios *TO*. Os atributos de *TO* indicam o estado destino

(*STATE*), o símbolo do alfabeto de entrada (*SYMBOL*) que provoca a transição, e a saída associada (*OUTPUT*). O atributo *OUTPUT* utiliza os identificadores definidos através do elemento *OUTPUT*. É possível definir a composição de imagens para saída a partir da enumeração das componentes em *OUTPUT*. Por exemplo, *OUTPUT="1;2"* provoca a sobreposição da imagem 1 com a 2 para produzir a imagem resultante de saída.

4.3.3 Especificação da Fita de Entrada

A fita de entrada reúne as informações que descrevem o comportamento do ator durante a animação. O elemento *TAPE* (figura 4.7) contém uma coleção de elementos vazios *CEL* que descrevem cada célula da fita. Como descrito no capítulo 3, a célula possui um símbolo de entrada (*SYMBOL*), um tempo de espera para a leitura do próximo símbolo (*TIME*) e a coleção de transformações que devem ser aplicadas à imagem de saída (*FN*). O valor de *TIME* é dado em milésimos de segundo.

Alguns comportamentos em animações são repetitivos e conseqüentemente implicam na criação de células com o mesmo conteúdo. Devido a esta característica, o elemento *GROUP* foi definido com o propósito de agrupar células que irão se repetir, diminuindo assim, o tamanho da representação da fita. O número de repetições é determinado pelo atributo *ITERATION*. A definição da DTD permite que existam grupos dentro de grupos, o que favorece a representação de comportamentos repetitivos mais complexos. A figura 4.8, por exemplo, ilustra uma fita de entrada criada para provocar o comportamento de andar no ator *bicho*. Nesta fita, duas células serão repetidas 3 vezes, totalizando assim, oito células a serem lidas pelo ator AGA.

```
<!ELEMENT TAPE (CEL|GROUP)+>
<!ATTLIST TAPE
        ID CDATA #REQUIRED>
<!ELEMENT CEL EMPTY>
<!ATTLIST CEL
        SYMBOL CDATA #REQUIRED
        TIME CDATA #REQUIRED
        FN CDATA #IMPLIED>
<!ELEMENT GROUP (CEL|GROUP)+>
<!ATTLIST GROUP
        ITERATION CDATA #REQUIRED>
```

FIGURA 4.7 - Trecho da DTD com os elementos *TAPE*, *CEL* e *GROUP*

```
<TAPE ID="andar" >
        <CEL SYMBOL="1" TIME="200" FN="TRANSLATE 20 30 N"/>
        <GROUP ITERATION="3">
                <CEL SYMBOL="2" TIME="100" FN="TRANSLATE 20 0 Y"/>
                <CEL SYMBOL="1" TIME="100" />
        </GROUP>
        <CEL SYMBOL="3" TIME="100" />
</TAPE>
```

FIGURA 4.8 – Exemplo do elemento *TAPE* para a especificação da fita de entrada

Inicialmente foram definidas para a linguagem AgaML sete transformações que podem ser utilizadas no atributo *FN*. Transformações geométricas para rotação (*ROTATE*), translação (*TRANSLATE*), escala (*SCALE*), espelhamento (*FLIP*), além de oferecer a possibilidade de aplicar-se diretamente uma matriz homogênia de transformação (MATRIX). Como também transformações para o controle da visualização do ator, como a visibilidade (*VISIBLE*) e alteração de camada (*ORDER*).

Há dois símbolos especiais definidos para o atributo *SYMBOL*, o *WAIT* e o *EMPTY*. A presença do *WAIT* na célula indica que animação deve parar até que um evento externo ocorra. Por exemplo, até que o botão do mouse seja pressionado. Este símbolo especial proporciona a interação com o usuário do tipo pró-ativa[1] [HEL 2001].

O símbolo *EMPTY* indica que o ator deve permanecer no estado atual e as transformações descritas em *FN* devem ser aplicadas à imagem de saída mais recente. A utilização do *EMPTY* é conveniente para evitar a criação de arestas na forma de laço nos estados do ator só com o objetivo de aplicar transformações, o que provoca um aumento na representação da função de transição.

4.3.4 Especificação das Instâncias do Ator AGA

A associação entre a especificação do ator AGA e uma fita de entrada é realizada pelo elemento INSTANCE (figura 4.9). Através deste elemento é possível criar várias instâncias do mesmo ator, assim como ter a mesma fita de entrada compartilhada por vários atores.

```
<!ELEMENT INSTANCE (USE+)>
<!ATTLIST INSTANCE
        ID CDATA #IMPLIED
        ACTOR CDATA #REQUIRED
        ORDER CDATA #IMPLIED>
<!ELEMENT USE EMPTY>
<!ATTLIST USE
        TAPE CDATA #REQUIRED>
```

FIGURA 4.9 - Trecho da DTD com os elementos *INSTANCE* e *USE*

O atributo *ACTOR* determina qual a especificação de ator utilizada, enquanto os elementos filhos *USE* determinam quais são as fitas de entrada (*TAPE*) que devem ser lidas. As fitas de entrada são utilizadas uma após a outra durante a apresentação da animação. O atributo *ORDER* indica qual a camada de atuação da instância (relação de ordem definida no modelo AGA).

O atributo *ID* de *INSTANCE* determina uma identificação para cada instância. Esta identificação é importante para suportar a construção de predicados destinados a consultas baseadas nos estados dos atores pelo AGA Player. Por exemplo, através da análise da fita *andar* (figura 4.8) e o autômato do ator *bicho* (figura 4.6) é possível recuperar o momento exato em que a instância *bicho faminto* (figura 4.10) ocupa um estado cuja descrição é “alerta”.

A figura 4.10 ilustra a criação de duas instâncias, *bicho faminto* e *bicho lento*, a partir da mesma especificação de ator *bicho*. A primeira é associada à fita de entrada *andar* enquanto a segunda instância associa a outra fita *andar lento.*

```
<INSTANCE ID=”bicho faminto” ACTOR=”bicho“ ORDER=”1”>
        <USE TAPE=”andar”/>
</INSTANCE>
<INSTANCE ID=”bicho lento” ACTOR=”bicho” ORDER=”2”>
        <USE TAPE=”andar lento”/>
</INSTANTE>
```

FIGURA 4.10 – Exemplo do elemento *INSTANCE* para a criação de instâncias

[1] A taxonomia aplicada em [HEL 01] divide o atributo interatividade em quatro categorias: passiva, reativa, pró-ativa e diretiva. A possibilidade de o usuário controlar funções como *start*, *stop*, *pause*, *forward* e *reverse* é qualificada como pró-ativa.

As especificações dos atores e fitas para a criação das instâncias não precisam ser necessariamente escritas no mesmo documento AgaML. Através do conceito de entidade externa do XML (*ENTITY*) é possível vincular as especificações localizadas em arquivos independentes. Este recurso colabora para que a mesma especificação seja compartilhada por várias animações no servidor HTTP.

A figura 4.11 ilustra um exemplo onde as especificações dos atores e fitas estão armazenadas separadamente nos arquivos *bicho.xml*, *andar.xml* e *andarlento.xml*. Estes arquivos são vinculados através do *ENTITY* e referenciados internamente pelos símbolos atribuídos na definição (*defbicho*, *defandar* e *defandarlento*).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE AGA SYSTEM "aga.dtd" [
        <!ENTITY defbicho SYSTEM "actors/bicho.xml">
        <!ENTITY defandar SYSTEM "tapes/andar.xml">
        <!ENTITY defandarLento SYSTEM "tapes/andarlento.xml"> ]>
<AGA VERSION=”2.0” WIDTH=”200” HEIGHT=”100” BACKGROUND=”255 255 255”
FRAMERATE=”24” REPEAT=”LOOP”>
        <HEAD>
                <TITLE>Exemplo de AgaML </TITLE>
                <AUTHOR>Fernando Accorsi </AUTHOR>
                <SUBJECT> Acadêmico; Linguagem AgaML</SUBJECT>
        </HEAD>
        &defbicho;
        &defandar;
        &defandarLento;
        <INSTANCE ID=”bicho faminto” ACTOR=”bicho“ ORDER=”1”>
                <USE TAPE=”andar”/>
        </INSTANCE>
        <INSTANCE ID=”bicho lento” ACTOR=”bicho” ORDER=”2”>
                <USE TAPE=”andar lento”/>
        </INSTANCE>
</AGA>
```

FIGURA 4.11 – Exemplo completo com as especificações vinculadas por entidades externas

4.4 AGA Player

O programa de visualização, AGA Player, foi desenvolvido em linguagem Java na forma de *applet*. A escolha pelo Java foi motivada principalmente pela necessidade de portabilidade do visualizador e o amplo repositório de objetos existentes para manipulação gráfica e processamento do XML encontrado na linguagem.

Para prover os recursos gráficos necessários para o processamento das imagens e sons foi utilizada a plataforma Java 2 [JAV 2002], em especial, os componentes de interface gráfica com o usuário (GUI) do *Swing*[1]. A manipulação da linguagem AgaML no *applet* é realizada a partir de uma API destinada ao processamento de documentos XML específica para Java, o JDOM [JDO 2002]. O JDOM pode ser visto como uma alternativa para o DOM (*Document Object Model*) e o SAX (*Simple API for XML*) direcionada para programadores Java.

Como descrito na seção 4.1, o AGA Player é executada no cliente para realizar a reprodução da animação com base nas especificações em AgaML. A execução do AGA Player pode ser dividida esquematicamente em três fases:

[1] Pacote que contém componentes escritos, manipulados e exibidos completamente em JAVA, denominados comumente de componentes Java puros.

a) **Carga do AgaML e mídias externas** – O *applet*, ao ser inicializado, carrega o arquivo em AgaML cujo nome foi passado como parâmetro. Uma vez carregado, o arquivo AgaML é processado, e as informações das especificações dos atores e fitas são extraídas. Entre as informações estão as referências para os arquivos de imagens e sons utilizados, os quais são carregados na seqüência, e armazenados em estruturas de dados do *applet*. Como as mídias se encontram em arquivos independentes, o processo de carga aguarda a confirmação de leitura de todas antes de iniciar as outras fases;

b) **Construção das instâncias** – A partir das especificações das instâncias extraídas do AgaML, os atores AGA são efetivamente instanciados em estruturas de dados internas e associados às suas respectivas fitas de entrada. Após esta fase, as instâncias se encontram no estado inicial e prontas para serem simuladas;

c) **Simulação dos autômatos e geração dos quadros** – De acordo com o modelo descrito no capítulo 3, os autômatos devem ser simulados em paralelo para obter a formação dos quadros de saída. Porém, a implementação do modelo utilizando processos concorrentes para cada autômato (*multithreading*) implica em uma dificuldade adicional para a sincronização precisa das trocas de imagens dos autômatos. Assim, uma alternativa de implementação foi escolhida para garantir que as imagens geradas representem fielmente as especificações determinadas pelas fitas de entrada. Um processo de temporização (*timer*) foi criado para gerar eventos de análise dos autômatos com a freqüência determinada pelo *FRAMERATE*. A cada evento do *timer*, as transições pertinentes são realizadas em cada um dos autômatos e o quadro de saída é formado, reproduzindo assim a imagem respectiva para aquele instante na animação. Durante o processo de simulação, o AGA Player está disponível para intervenções de reprodução e consulta solicitadas pelo observador através dos botões de controle.

A figura 4.12 ilustra o painel de controle do AGA Player visualizado pelo observador durante visita à página *Web*. O painel conta com 3 botões de controle de reprodução para as funções corriqueiras de reproduzir, pausar e parar. Uma barra de busca é disponibilizada para o usuário com o objetivo de indicar o tempo decorrido de animação e proporcionar a busca temporal da seqüência animada.

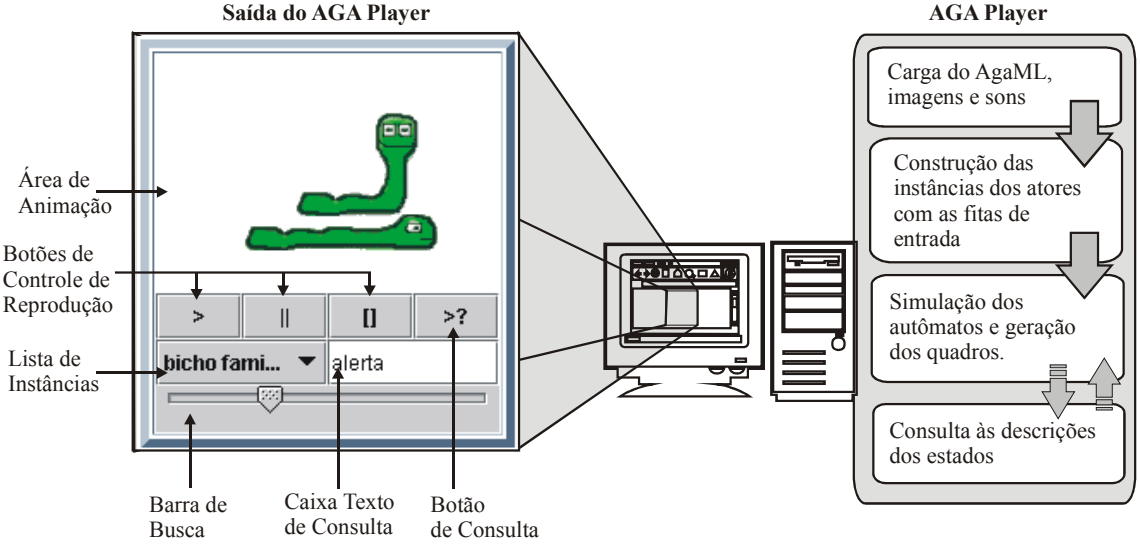


FIGURA 4.12 – Painel de controle do AGA Player visualizado pelo observador

O AGA Player conta com um recurso de busca incomum nos programas de visualização disponíveis na Internet. Este recurso é particular do modelo AGA e

proporciona um mecanismo refinado de pesquisa baseado em palavras-chave. A lista de instâncias, a caixa de texto e o botão de consulta dão suporte operacional para a execução deste mecanismo.

O procedimento de pesquisa é operacionalmente simples. O observador escolhe um ator na lista de instâncias e digita alguma palavra-chave que represente a situação semântica que deseja verificar. Assim, quando o botão de consulta é pressionado, é realizada uma análise na instância, investigando se em algum instante da animação ela ocupa um estado cuja descrição vinculada coincide com a palavra-chave requerida. Caso afirmativo, a animação é posicionada neste instante, pronta para iniciar a reprodução. A análise é realizada a partir do instante corrente. Desta maneira, o botão pode ser pressionado sucessivas vezes a fim de pesquisar novas situações ao longo da animação. A potencialidade de casamento das palavras-chave cresce à medida que o projetista da animação associa um amplo vocabulário aos estados através da função descrição.

4.5 Considerações Finais

O mecanismo de consulta implementado no AGA Player demonstra como o modelo AGA pode favorecer a recuperação de informação em animações. Esta implementação, porém, não esgota a potencialidade do modelo para este propósito. É possível implementar mecanismos de consulta baseados na construção de predicados mais complexos que levem em conta várias instâncias dos atores AGA.

A estrutura do AgaML promove a reutilização das especificações em vários níveis. As especificações dos atores e fitas podem ser utilizadas por várias instâncias diferentes, assim como, podem ser compartilhadas por diversas animações se armazenadas em arquivos independentes. Essa última característica colabora também para que as especificações possam ser criadas dinamicamente por processos vinculados às páginas no momento do acesso pelo usuário.

Embora o AGA Player e a linguagem AgaML tenham sido desenvolvidos para atender as especificações do modelo AGA, estes possuem uma estrutura favorável para expansão a fim de comportar as extensões propostas no modelo AGA-S (descrito no capítulo 6).

5 Estudo de Caso

5.1 Introdução

O livro "Linguagens Formais e Autômatos" escrito por Paulo Fernando Blauth Menezes [MEN 2000] conta com ilustrações esquemáticas que demonstram o comportamento dos formalismos reconhecedores para as classes de linguagens estabelecidas pela Hierarquia de Chomsky. Entre essas ilustrações, estão demonstrados: o Autômato Finito Determinístico, o Autômato com Pilha e a Máquina de Turing.

Atualmente, o livro possui uma versão eletrônica suportada pelo sistema Hyper-Automaton [MAC 2000] e disponibilizada na *Web*. Algumas das ilustrações da versão impressa foram animadas utilizando como recurso o GIF a fim de enriquecer a versão eletrônica.

O estudo de caso descrito neste capítulo utiliza como objeto de estudo a aplicação do modelo AGA para a criação dessas animações. Com o objetivo de comparar as animações em AGA com as produzidas em GIF, foi escolhida uma animação de cada tipo de autômato e realizadas análises quanto ao espaço de armazenamento, a reusabilidade e manutenibilidade das animações e o suporte à recuperação de informação.

A seção 5.2 descreve o projeto das animações dos autômatos utilizando o AGA (enfocando os atores desenvolvidos) e de que maneira foram utilizados para comporem as animações. A seção 5.3 contém as análises comparativas realizadas entre o AGA e o GIF no estudo de caso. A seção 5.4 apresenta algumas dificuldades encontradas na construção das animações para o estudo de caso e trás considerações finais sobre o comparativo entre o AGA e o GIF.

5.2 Projeto das Animações em AGA

O modelo AGA ainda não possui um editor próprio para o desenvolvimento das animações. Desta maneira, foram utilizadas ferramentas de desenho e edição de texto[1] de uso geral para a criação do alfabeto de saída e das especificações em AgaML.

O desenvolvimento das animações em AGA foi efetuado seguindo as etapas relacionadas abaixo:

a) **Esboço das animações** – Para cada uma das animações foi criado um *storyboard* com o objetivo de subsidiar a identificação dos atores e suas ações realizadas durante a animação;

b) **Criação dos atores AGA** – O autômato de cada ator foi projetado, e as especificações em AgaML gravadas em arquivos independentes contendo apenas o elemento *ACTOR*. Para a definição dos autômatos, levou-se em conta as imagens distintas de cada ator e os estados semanticamente interessantes para consulta. As imagens do alfabeto de saída foram desenhadas separadamente em um editor gráfico e gravadas em arquivos no formato GIF (estático) com os mesmos nomes referenciados no elemento *ACTOR*;

c) **Criação do AgaML principal com as instâncias** – Para cada animação, foi criado um arquivo em AgaML contendo o elemento raiz *AGA*. Nesse arquivo, foram inseridas as informações de autoria e apresentação, como também, as especificações das instâncias. Os arquivos com os atores AGA foram associados por meio de entidades

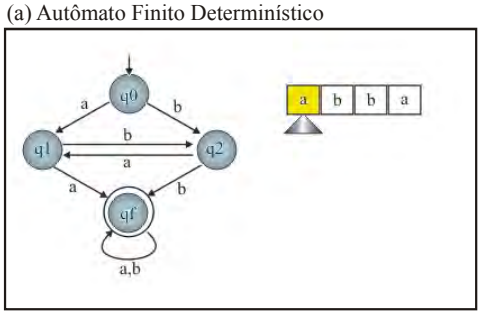
[1] Para a criação dos desenhos do alfabeto de saída foram utilizados os editores gráficos: COREL Photo Paint e COREL Draw. As especificações em AgaML foram editadas no editor Bloco de Notas disponibilizado junto com os sistemas operacionais da Microsoft.

externas providas pelo XML. Nesta etapa, também foram criadas páginas em HTML para apresentar as animações a fim de facilitar os testes de animação da etapa seguinte;

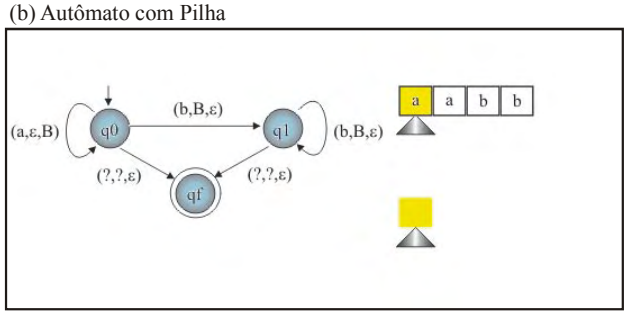
d) **Animação das instâncias** – A animação das instâncias foi realizada pela criação e associação das fitas de entrada. O conteúdo de cada fita foi determinado tendo como referência o *storyboard* desenvolvido na primeira etapa. Durante o processo de animação, o AGA Player foi utilizado várias vezes para auxiliar na determinação dos valores de posicionamento e temporização incluídos na fita de entrada.

De acordo com as etapas descritas, foram realizadas 3 animações que ilustram: o Autômato Finito Determinístico, o Autômato com Pilha e a Máquina de Turing. A figura 5.1 mostra a imagem do primeiro quadro formado em cada uma das animações.

(a) Autômato Finito Determinístico



(b) Autômato com Pilha



(c) Máquina de Turing

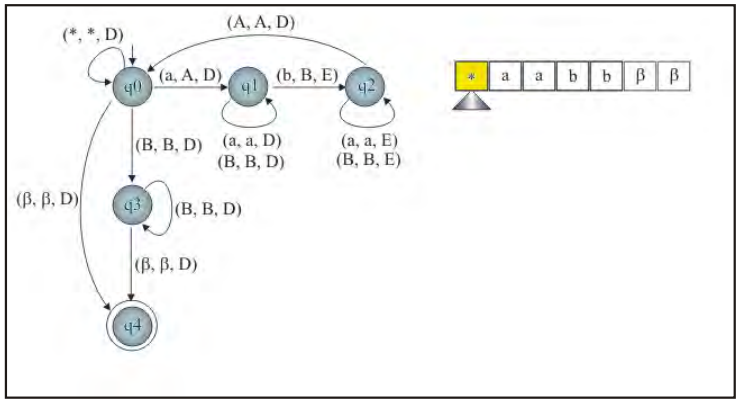


FIGURA 5.1 – Primeiro quadro das três animações utilizadas no estudo de caso

Nota-se que as 3 animações possuem componentes com representações gráficas iguais. Os estados (vértices dos grafos), representados por círculos sombreados, aparecem em múltiplas ocorrências nas 3 animações. Da mesma maneira, as células da fita de entrada (quadrados com as letras) e a unidade de controle (triângulos sombreados) também são comuns. Além da representação gráfica comum, esses componentes possuem comportamento dinâmico semelhante nas animações.

As semelhanças encontradas motivaram a construção de 3 atores AGA: *Estado*, *Célula* e *Unidade*. Esses atores foram projetados para serem compartilhados pelas 3 animações. Sendo assim, possuem todas as variações de imagens necessárias para atendê-las.

A figura 5.2 ilustra a estrutura desses atores com seus respectivos alfabetos de saída (retângulo com as imagens). Nota-se que cada imagem é associada a um número, visando a representação interna (quadro preto escrito em branco). A presença deste número na aresta do grafo de transições indica qual a imagem que será produzida na saída. Por exemplo, no ator *Unidade* a transição do estado *q0* para o estado *q1* mediante

a leitura do símbolo *d*, produz a imagem 2 (imagem da unidade com a seta para direita) como imagem de saída.

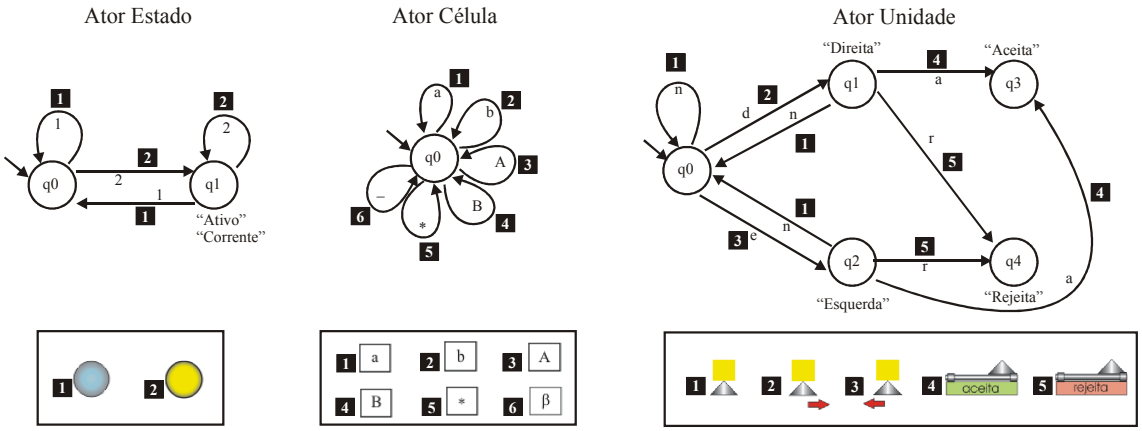


FIGURA 5.2 – Atores AGA compartilhados pelas animações

O ator *Estado* possui 2 estados que indicam as situações de ativo (*q1*) e inativo (*q0*) durante as animações. Quando há uma transição de *q0* para *q1*, a imagem exibida é um círculo amarelo (ativo), e na transição oposta, *q1* para *q0*, um círculo azul (inativo). O estado *q1* foi associado às descrições: "Ativo" e "Corrente", para que o observador possa consultar está condição para cada um dos estados durante a apresentação.

O ator *Célula* possui apenas um estado sem nenhuma descrição associada. Esse ator trabalha apenas com um seletor de imagens de saída para compor os símbolos da fita de entrada.

O ator *Unidade* possui 5 estados, 4 deles são associados a descrições que indicam a situação de movimento da unidade de controle. A unidade de controle pode se mover para a direita (*q1*) ou para esquerda (*q2*), ou ainda, representar que a palavra na fita de entrada foi aceita (*q3*) ou rejeitada (*q4*). Nota-se que as animações dos dois primeiros autômatos (Autômato Finito Determinístico e Autômato com Pilha) não utilizam todas as imagens de movimento, porém, como o ator é compartilhado, permanecem previstas todas as imagens utilizadas pelas 3 animações.

5.2.1 Animação do Autômato Finito Determinístico

A primeira animação foi criada para demonstrar o processo de reconhecimento de uma palavra por um Autômato Finito Determinístico. Durante a animação, são mostradas as transições do autômato e os movimentos da unidade de controle até a palavra ser aceita. A seqüência de imagens produzidas pela animação do Autômato Finito Determinístico pode ser vista no anexo 5 (item a).

Além dos atores já definidos, foi criado um novo ator para coordenar as mudanças de imagens com o objetivo de indicar as transições do autômato durante o processo de reconhecimento. Quando uma transição é realizada na animação, a aresta do grafo muda da cor preta para vermelha, indicando assim que aquela determinada transição foi utilizada. A figura 5.3 ilustra o ator *Autômato* criado para este fim.

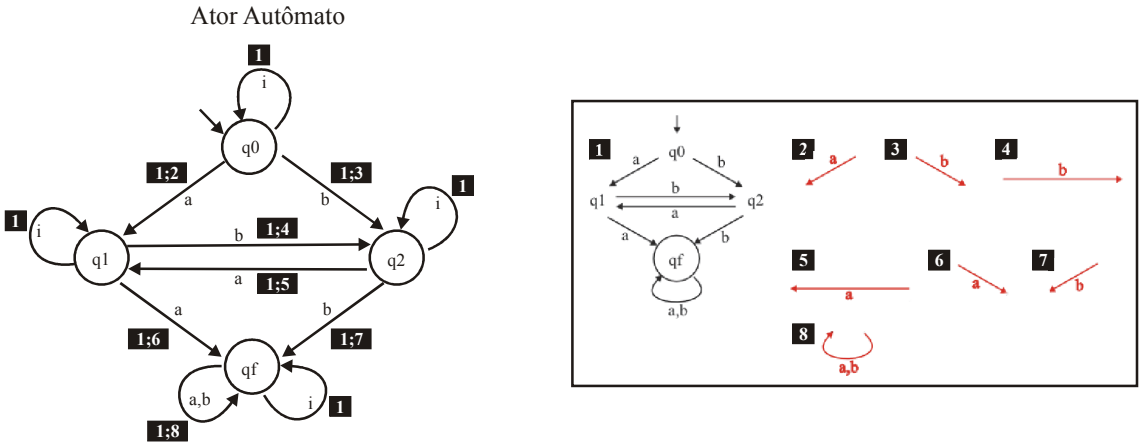


FIGURA 5.3 – Ator Autômato dedicado apenas à primeira animação

Nota-se que a composição de imagens descrita nos capítulos anteriores é utilizada neste ator para compor as imagens de saída. Por exemplo, a transição do estado *q0* para o estado *q1* do ator está associada à palavra *1;2*. Essa palavra indica que a imagem 1 deve ser sobreposta pela imagem 2 para formar a imagem de saída. A imagem 1 contém todas as arestas do grafo na cor preta e a imagem 2 possui apenas uma aresta em vermelho. Assim, quando são sobrepostas as imagens, a resultante contém apenas uma aresta em vermelho, enquanto todas as outras se mantém na cor original. Esse recurso foi utilizado para reduzir o espaço de armazenamento das imagens do alfabeto de saída, pois, caso contrário, seriam necessárias imagens maiores com todas as arestas representadas.

Baseadas nesses 4 tipos de atores AGA, foram criadas 10 instâncias para realizar a animação. A tabela 5.1 relaciona as instâncias criadas para cada tipo de ator. Os nomes das instâncias na tabela são os mesmos nomes disponibilizados no painel de controle do AGA Player para consulta. A figura 5.4 ilustra a posição de cada instância na animação.

TABELA 5.1 – Relação das instâncias na animação do Autômato Finito Determinístico

Tipo do Ator AGA	Instâncias
Ator Estado	(1) Estado q0, (2) Estado q1, (3) Estado q2 e (4) Estado qf.
Ator Unidade	(5) Unidade de Controle.
Ator Célula	(6) 1º Célula da Fita, (7) 2º Célula da Fita, (8) 3º Célula da Fita e (9) 4º Célula da Fita.
Ator Autômato	(10) Transições do Autômato.

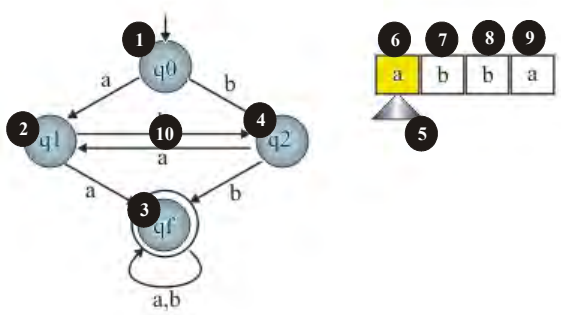


FIGURA 5.4 – Localização das instâncias na animação do Autômato Finito Determinístico

A distribuição espacial das instâncias é determinada pelas funções de transformação contidas na fita de entrada. Cada instância é completamente independente da outra e comporta-se na animação de acordo com o conteúdo da fita de entrada associada.

5.2.2 Animação do Autômato com Pilha

A segunda animação foi criada para demonstrar o processo de reconhecimento de uma palavra por um Autômato com Pilha. Durante a animação, são mostradas as transições do autômato e os movimentos das unidades de controle da fita e da pilha. A seqüência de imagens produzidas pela animação do Autômato com Pilha pode ser vista no anexo 5 (item b).

De forma semelhante à primeira animação, foi criado um ator novo para controlar as mudanças de cor das transições do Autômato com Pilha (figura 5.5). O recurso de composição de imagens também foi utilizado neste ator para reduzir o espaço de armazenamento.

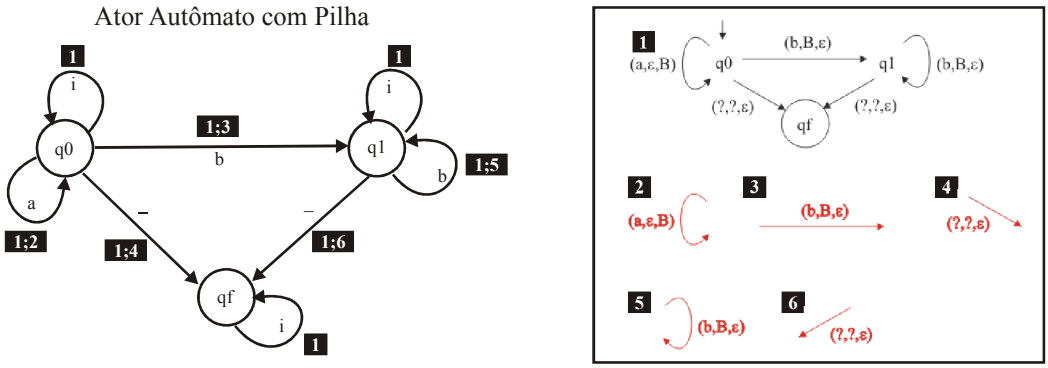


FIGURA 5.5 –Ator Autômato com Pilha dedicado apenas à segunda animação

Para esta animação foram criadas 12 instâncias. A tabela 5.2 relaciona as instâncias criadas para cada tipo de ator. A figura 5.6 ilustra a posição de cada instância na animação.

TABELA 5.2 - Relação das instâncias na animação do Autômato com Pilha

Tipo do Ator AGA	Instâncias
Ator Estado	(1) Estado q0, (2) Estado q1 e (3) Estado qf.
Ator Unidade	(4) Unidade de Controle da Fita e (5) Unidade de Controle da Pilha.
Ator Célula	(6) 1º Célula da Fita, (7) 2º Célula da Fita, (8) 3º Célula da Fita, (9) 4º Célula da Fita, (10) 1º Célula da Pilha e (11) 2º Célula da Pilha.
Ator Autômato com Pilha	(12) Transições do Autômato com Pilha

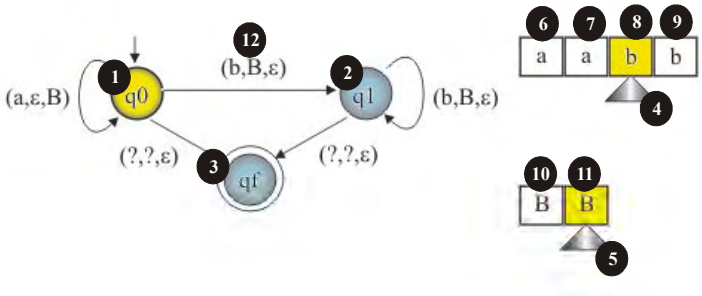


FIGURA 5.6 - Localização das instâncias na animação do Autômato com Pilha

5.2.3 Animação da Máquina de Turing

A terceira animação foi criada para demonstrar o processo de reconhecimento de uma palavra por uma Máquina de Turing. Esta animação representa, além das transições da máquina e os movimentos da unidade de controle, as operações realizadas na fita de trabalho. A seqüência de imagens produzidas pela animação da Máquina de Turing pode ser vista no anexo 5 (item c).

Nesta animação, ao contrário das animações anteriores, os atores compartilhados: *Célula* e *Unidade*, têm praticamente todas as imagens de saída utilizadas. Pois, a animação da Máquina de Turing envolve movimentos da unidade de controle para esquerda e direita, e os símbolos na fita de trabalho são alterados durante o processo.

Para a animação das transições foi adotada a mesma estratégia que nos casos anteriores. Foi criado um autômato (figura 5.7) dedicado à coordenação das mudanças de cor das arestas do grafo que representam as transições.

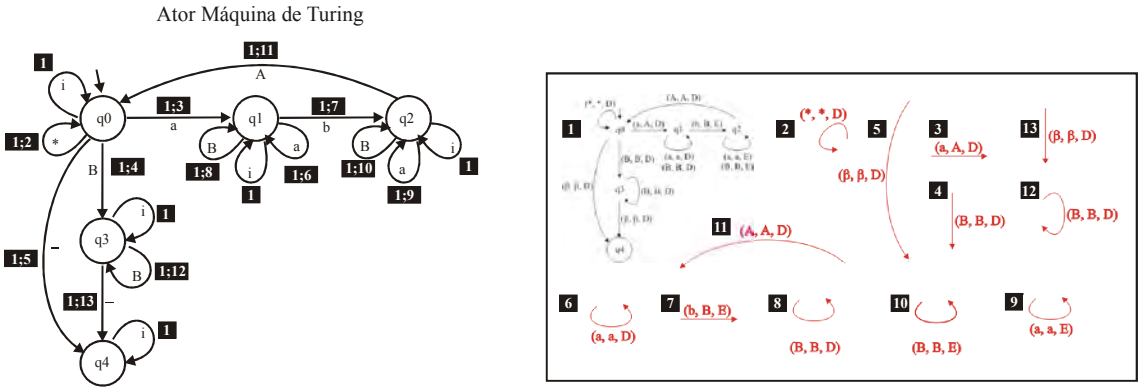


FIGURA 5.7 – Ator Máquina de Turing dedicado apenas à terceira animação

Para esta animação foram criadas 14 instâncias. A tabela 5.3 relaciona as instâncias criadas para cada tipo de ator, enquanto a figura 5.8 ilustra a posição de cada instância na animação.

TABELA 5.3 – Relação das instâncias na animação da Máquina de Turing

Tipo do Ator AGA	Instâncias
Ator Estado	(1) Estado q0, (2) Estado q1, (3) Estado q2, (4) Estado q3 e (5) Estado q4.
Ator Unidade	(6) Unidade de Controle.
Ator Célula	(7) 1º Célula da Fita, (8) 2º Célula da Fita, (9) 3º Célula da Fita, (10) 4º Célula da Fita, (11) 5º Célula da Fita, (12) 6º Célula da Fita e (13) 7º Célula da Fita.
Ator Máquina de Turing	(14) Transições da Máquina de Turing

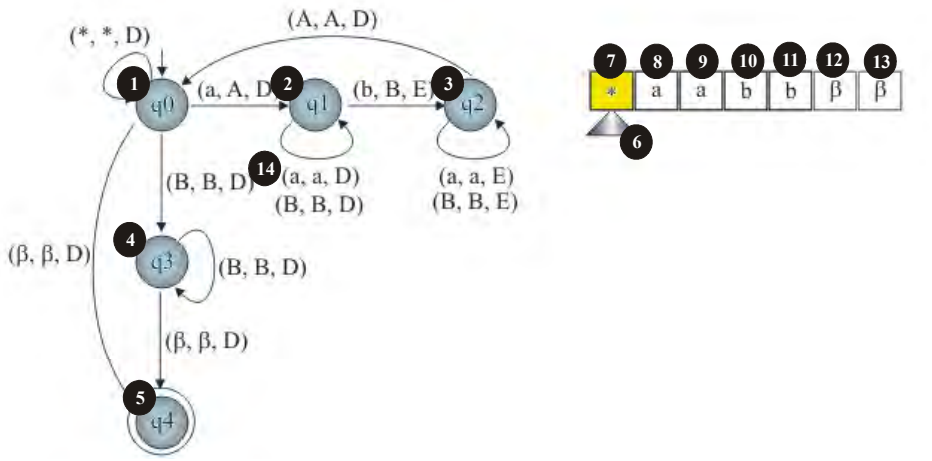


FIGURA 5.8 - Localização das instâncias na animação da Máquina de Turing

5.3 Análise Comparativa com o GIF

A análise comparativa entre o AGA e o GIF foi realizada enfocando as características das duas abordagens quanto ao espaço de armazenamento, à reusabilidade e manutenibilidade das animações e ao suporte à recuperação de informação.

A escolha desses 3 itens baseia-se no estudo realizado no capítulo 2 sobre as características da *Web* relacionadas às animações, onde as contribuições nesses tópicos são caracterizadas como requisitos relevantes para o desenvolvimento de novas abordagens para animações na *Web*.

Embora a análise comparativa esteja baseada principalmente no estudo de caso realizado, várias propriedades do AGA discutidas nesta seção são inerentes ao modelo e se mantém na produção de animações em geral.

5.3.1 Espaço de Armazenamento

A análise de espaço de armazenamento foi realizada a partir da quantificação do tamanho dos arquivos envolvidos na representação de cada animação. Para o AGA, levou-se em conta os arquivos de imagem e das especificações em AgaML, e para o GIF, os arquivos que contêm a seqüência de quadros.

Com o objetivo de evitar diferenças entre as versões produzidas em AGA e GIF, foi desenvolvido um programa para gerar os quadros-chave necessários para compor a animação em GIF a partir das especificações em AGA. Desta forma, cada quadro-chave gerado é uma cópia fiel do apresentado em AGA. Nota-se que foram apenas coletados os quadros-chave, ou seja, apenas quadros onde ocorreu alguma mudança na imagem.

A geração das versões em GIF a partir dos quadros-chave foi realizada por duas ferramentas diferentes, o Corel Photo Paint (versão 10) e o Adobe ImageReady (versão 3.0)[1]. A diferença em bytes entre versões produzidas pelas duas ferramentas foi inferior a 15 bytes, desprezível para o estudo.

As ferramentas gráficas utilizadas trazem opções de otimização quanto ao número de cores e a forma de armazenamento dos quadros. O número de cores foi mantido em 256 em todas as animações a fim de realizar um comparativo justo com as

[1] As ferramentas Corel Photo Paint (versão 10) e o Adobe ImageReady (versão 3.0) são ferramentas recomendadas por seus fabricantes para a produção de GIF animado para Web, entre outras funcionalidades.

imagens utilizadas pelo AGA[1], também geradas em 256 cores. O armazenamento dos quadros pode ser integral ou parcial. No integral, os quadros são armazenados integralmente e não são otimizadas redundâncias entre os sucessivos quadros. Já no parcial, apenas são armazenadas as diferenças entre um quadro e outro, reduzindo assim o tamanho do arquivo. Foram avaliados os GIF animados produzidos com e sem a otimização.

A tabela 5.4 resume os valores encontrados na análise das animações do estudo de caso. Na seção do AGA, o item 3 se refere à soma dos itens 1 e 2, que correspondem, respectivamente, ao gasto em bytes com o armazenamento das imagens do alfabeto de saída e das especificações em AgaML. Por exemplo, para a animação do Autômato Finito Determinístico, foram necessários 32.495 bytes, sendo gastos 25.750 com arquivos de imagens e 6.745 com a especificação em AgaML. O item 4 corresponde ao total de armazenamento necessário para as 3 animações no servidor HTTP. Nota-se que não corresponde a soma exata das 3 animações, já que várias imagens e especificações são compartilhadas pelas animações.

TABELA 5.4 – Espaço de armazenamento utilizado nas duas abordagens

	Animação Autômato Finito Determinístico 420x250 Pixels 10 quadros-chave 256 cores	**Animação Autômato com Pilha** 550x250 Pixels 14 quadros-chave 256 cores	**Animação Máquina de Turing** 650x350 Pixels 30 quadros-chave 256 cores
AGA			
(1) Imagens	25.750	26.190	44.648
(2) AgaML	6.745	7.353	9.820
(3) Total	32.495	33.543	54.468
(4) Total das três animações			89.134
GIF versão 89a			
(5) Quadros inteiros	111.305	176.020	655.151
(6) Quadros parciais	64.558	93.508	278.513
(7) Total das três animações (quadros inteiros)			942.476
(8) Total das três animações (quadros parciais)			436.579

Na seção do GIF, são mostrados os tamanhos dos arquivos para as versões geradas com quadros inteiros (item 5) e com quadros parciais (item 6). Nota-se que a otimização alcançada com o armazenamento dos quadros parcialmente é bastante significativa. Os itens 7 e 8, correspondem, respectivamente, aos gastos das 3 animações no servidor HTTP utilizando quadros inteiros e quadros parciais. No caso do GIF, os valores correspondem à soma das 3 animações, posto que não há nenhum tipo de compartilhamento entre as mesmas.

Nota-se que o espaço de armazenamento gasto com as animações produzidas em AGA são inferiores às animações GIF equivalentes no estudo de caso. Na primeira animação, o AGA utiliza 50,3% do espaço necessário para representar a mesma animação em GIF com quadros parciais. Nas outras animações, os valores para este mesmo cálculo são, respectivamente, de 35,9% e 19,6%. No caso das 3 animações juntas, o AGA necessita apenas de 20,4% do espaço requerido pelas animações em GIF.

[1] Como as imagens utilizadas pelo AGA estão separadas em arquivos independentes, a possibilidade de redução do número de cores é maior que da representação dos quadros nos GIF animados.

É possível demonstrar que os resultados obtidos pelo AGA em relação ao GIF são decorrentes das características particulares do modelo, e portanto, se mantém nas animações em geral além das descritas.

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. O armazenamento parcial dos quadros não altera esta relação, apenas reduz a área a ser codificada em cada quadro.

Ao contrário do GIF, o AGA armazena apenas as imagens distintas do alfabeto de saída de cada tipo de ator. Desta maneira, a mesma imagem pode ser emitida como saída pelo ator em diversos momentos da animação sem a necessidade de codificá-la novamente. Nota-se também, que todas as instâncias de um mesmo ator compartilham o mesmo alfabeto de saída, o que reduz ainda mais o espaço de armazenamento necessário. O tamanho da especificação em AgaML varia de acordo com a complexidade[1] dos atores e o comprimento de suas fitas de entrada. Portanto, o tamanho da animação em AGA não cresce em função do número de quadros-chave como o GIF, mas sim, de acordo com a complexidade de cada tipo de ator e o tamanho de suas imagens associadas ao alfabeto de saída.

A partir das características do modelo AGA, pode-se determinar que o pior caso para este modelo é quando a animação se constitui em uma sucessão de imagens completamente distintas entre si. Por exemplo, uma animação em que cada quadro apresentado possua um retângulo do tamanho da área de animação tingido com uma cor diferente. Desta forma, a animação em AGA necessita de um ator que vincule um símbolo do alfabeto de saída para representar cada uma dos quadros distintos da animação.

A figura 5.9 ilustra o ator construído para modelar uma animação de pior caso. Esse ator tem apenas a função de selecionar o quadro que deverá ser apresentado, o qual será utilizado apenas uma vez. Quanto maior o número de quadros, maior será o número de arcos no ator e o número de imagens vinculadas a eles.

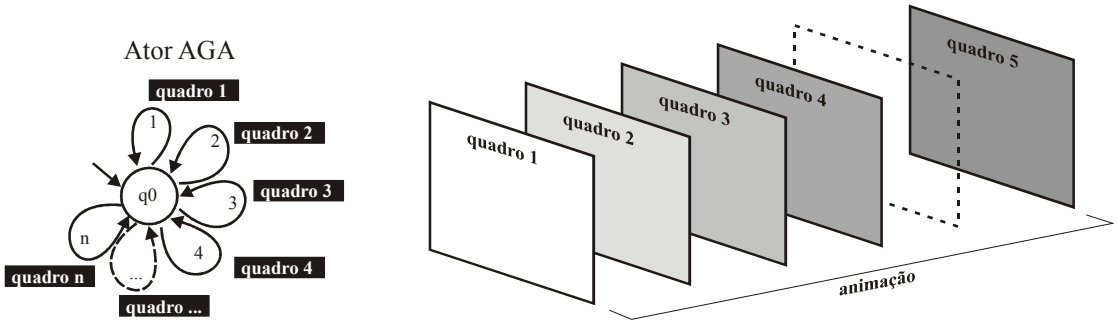


FIGURA 5.9 – Ator AGA construído para modelar o pior caso

Nota-se que no pior caso, o AGA se comporta como o GIF, pois acumula os sucessivos quadros da animação e os apresenta em ordem. O gráfico da figura 5.10

[1] A complexidade do ator AGA é quantificada pela quantidade de estados, transições e símbolos do alfabeto de saída necessários para sua representação.

ilustra esse comportamento a partir de um experimento realizado com animações[1] do pior caso. Tanto o AGA quanto o GIF crescem linearmente em função do número de quadros no pior caso.

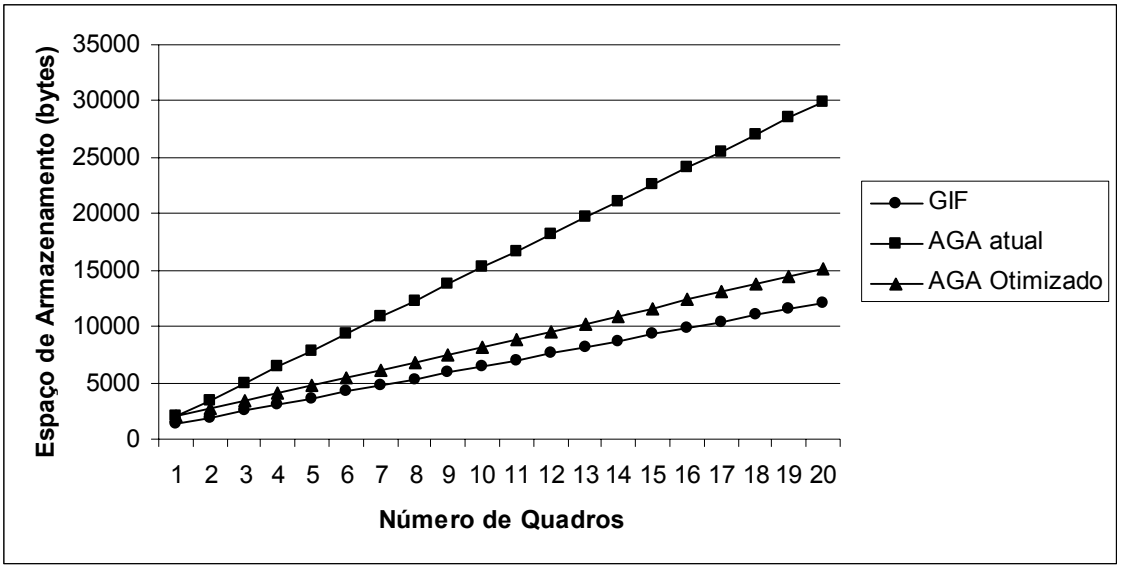


FIGURA 5.10 – Gráfico comparativo entre o AGA e GIF em animações do pior caso

Observa-se que o coeficiente angular da reta que representa o AGA (atual) é maior que a do GIF. Isto acontece porque a cada quadro incluído na animação em AGA, a especificação em AgaML também aumenta para representar mais um arco do ator. Outra característica da implementação atual do AGA que contribui para que esse coeficiente seja maior, é o fato das imagens do alfabeto de saída serem gravadas em arquivos independentes que incluem um preâmbulo com informações redundantes. Como por exemplo, a paleta da imagem com 256 cores. A reta que representa o AGA (otimizado) simula a evolução do espaço de armazenamento caso essas redundâncias fossem excluídas. Nota-se, portanto, que o aumento da especificação em AgaML contribui muito pouco com o crescimento do espaço de armazenamento.

Outro resultado que fica demonstrado, é que para toda animação em GIF é possível construir pelo menos uma equivalente em AGA que terá no máximo uma evolução do espaço de armazenamento assintomaticamente igual ao GIF.

Os experimentos quantitativos discutidos nesta seção ilustram os resultados obtidos no artigo *"Comparing Data Compression in Web-based Animaiton Models using Kolmogorov Complexity"*. Neste artigo, o primeiro autor, Carlos A. P. Campani, prova, utilizando a Complexidade de Kolmogorov, que o AGA é mais eficiente para a compressão de dados do que o GIF. Os conceitos sobre a Complexidade de Kolmogorov e os detalhes da prova podem ser vistos no artigo completo contido no anexo 3.

É importante ressaltar que as características do modelo AGA discutidas nesta seção a respeito da redução do espaço de armazenamento afetam de forma bastante direta os custos de transferência das animações pela Internet. Um dos custos mais evidentes para o usuário é o tempo de espera para a carga da animação.

[1] As animações geradas constituem em seqüências de quadros de 420x250 pixels tingidos com cores distintas.

5.3.2 Reusabilidade e Manutenibilidade

Reusabilidade e manutenibilidade são dois atributos de grande importância na especificação de animações para *Web*, pois este ambiente pressupõe constantes alterações e expansões do seu conteúdo visual.

O reuso das animações em AGA é promovido principalmente pela possibilidade de reutilização de três componentes do modelo: o alfabeto de saída, o ator AGA e a fita de entrada. Várias das características desses três componentes têm um papel importante na manutenção[1] das animações.

Nas especificações em AgaML, os atores AGA têm seus símbolos do alfabeto de saída associados a imagens e sons codificados em arquivos independentes. Desta forma, esses arquivos, disponibilizados no servidor HTTP, podem ser reutilizados por várias especificações. Nas animações do estudo de caso, por exemplo, as imagens dos estados, células da fita e unidade de controle são utilizadas pelas 3 animações apresentadas.
Nota-se também que, no desenvolvimento de novos atores, esses arquivos podem ser facilmente reaproveitados pela simples associação com os símbolos do alfabeto de saída. Esse recurso, por exemplo, proporciona que sejam mantidas diferentes versões de um ator compartilhando o mesmo alfabeto de saída. A figura 5.11 ilustra duas versões do ator *Célula* que compartilham o mesmo alfabeto de saída, sendo que a primeira não contém a documentação dos estados implementada pela segunda versão.

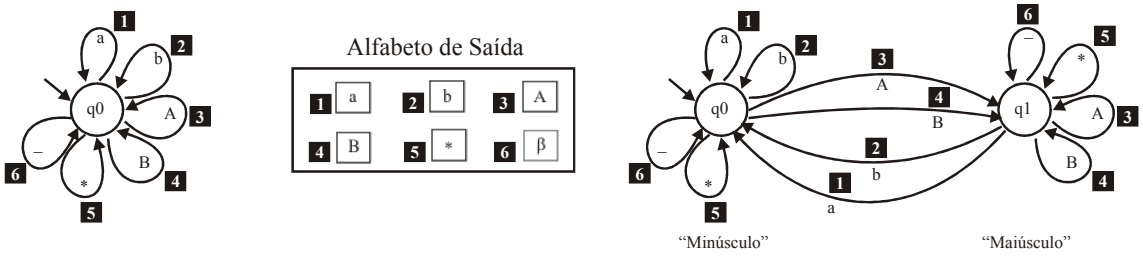


FIGURA 5.11 – Duas versões diferentes para o ator Célula

As imagens e sons associados aos símbolos do alfabeto de saída dos atores AGA podem ser alterados de forma independente sem necessariamente envolver alterações nas especificações das animações. Por exemplo, o círculo em amarelo adotado para indicar o estado ativo nas animações do estudo de caso, poderia ter sua cor permutada para verde apenas pela edição individual da imagem associada ao símbolo do alfabeto de saída. Após essa edição, as 3 animações estariam alteradas e prontas para serem exibidas com os estados ativos em verde.

A especificação do ator AGA é outro componente que pode ser reutilizado e facilmente alterado. O mesmo ator AGA pode ser combinado a diferentes fitas de entrada pela criação das instâncias e ser utilizado não só várias vezes pela mesma animação, como também por diversas animações diferentes. A criação das animações para o estudo de caso explorou estas duas possibilidades. O ator *Estado*, por exemplo, é utilizado para criação de todos os estados dos autômatos nas 3 animações.

A alteração do ator AGA pode ser realizada de maneira bastante simples pela edição da especificação em AgaML. A partir dessa edição, é possível alterar tanto a estrutura do autômato com saída quanto às associações semânticas determinadas pela função descrição. Essa característica do AGA pode ser explorada principalmente para

[1] Entende-se como manutenção, qualquer operação realizada com o objetivo de alterar ou mesmo expandir o conteúdo da animação.

expansão dos atores. A figura 5.11 ilustra um exemplo de expansão proposta para o ator *Célula* onde um novo estado é criado para indicar que o símbolo da célula foi alterado de minúsculo para maiúsculo.

Nota-se que a expansão ilustrada na figura 5.11 conserva o mesmo padrão de saída esperado pelas animações existentes. Como a estrutura básica do ator AGA é um autômato, diversas operações formalmente verificáveis podem ser definidas para alterar os atores garantindo propriedades como esta.

De forma semelhante à especificação do ator AGA, a mesma especificação de fita de entrada pode ser reutilizada na criação de instâncias em várias animações. Como a fita de entrada contém as informações que determinam o comportamento dos atores nas animações, esse tipo de reuso pode ser utilizado para a criação de padrões de comportamento para o desenvolvimento de novas animações.

Assim como o ator AGA, o conteúdo da fita de entrada pode ser alterado pela edição da especificação em AgaML. Essa operação pode alterar o comportamento do ator quanto à ordem de apresentação das imagens de saída, as temporizações entre as trocas de imagem, e também, as funções que transformam o alfabeto de saída.

Durante a criação das animações do estudo de caso, a facilidade de alteração do comportamento dos atores pela fita de entrada proporcionou ao autor a experimentação de vários tipos de comportamentos até a eleição dos preferidos.

O reuso e manutenção das animações em AGA não se limitam a operações efetuadas nesses três componentes isoladamente. Uma animação pode ser criada ou atualizada aproveitando grande parte da estrutura de uma animação existente. Por exemplo, caso fosse necessário criar uma animação que demonstrasse o comportamento do Autômato Finito Determinístico (ilustrado no estudo de caso) rejeitando uma palavra, bastaria reutilizar a animação atual alterando o conteúdo de algumas fitas de entrada.

Ao contrário do AGA, o GIF não possui uma estrutura que colabora com o reuso e manutenção das animações. Sua estrutura baseada em quadros limita a edição da animação a operações gráficas realizadas diretamente nas imagens de cada quadro.

Devido à forma de codificação do GIF não é possível criar seqüências de quadros que sejam compartilhadas ao mesmo tempo por várias animações. Cada animação constitui em um único arquivo independente.

Várias das tarefas de manutenção e reuso descritas nesta seção, facilmente efetuadas em AGA, constituem em operações bastante trabalhosas nas animações em GIF. Por exemplo, para mudança da cor do estado ativo nas animações do estudo de caso, seria necessário editar cada quadro das 3 animações e com uma ferramenta gráfica de pintura preencher cada um dos estados com a cor desejada. De maneira semelhante, a alteração de comportamento de apenas um ator na animação em GIF, envolveria a edição gráfica de cada quadro comprometendo muitas vezes as demais imagens contidas neles.

Nota-se, portanto, que o AGA possui uma estrutura que favorece mais do que o GIF as tarefas de manutenção e reuso freqüentemente necessárias para o desenvolvimento de conteúdo visual para a *Web*.

5.3.3 Recuperação de Informação

O modelo AGA possui, engendrado na definição do ator, a função descrição, que efetua a associação de descrições semânticas aos estados do ator AGA. Essa particularidade do modelo é suportada pelo mecanismo de consulta implementado no AGA Player. Como descrito no capítulo 4, a partir do painel de controle do AGA Player

é possível inserir palavras-chave e resgatar instantes na animação onde alguma instância de ator possua um estado cuja descrição coincida com a palavra consultada.

No estudo de caso realizado, os atores *Estado* e *Unidade* tiveram alguns de seus estados mapeados para palavras-chave. O ator *Estado* possui um estado que indica a condição de estado ativo ou inativo, e o ator *Unidade* possui estados que indicam os movimentos da unidade de controle à esquerda ou à direita, como também os resultados de reconhecimento, aceita ou rejeita.

Como esses mapeamentos fazem parte da estrutura dos atores, todas as instâncias criadas a partir deles herdaram automaticamente essa associação semântica, proporcionando uma variedade de consultas às animações. Por exemplo, em qualquer uma das animações é possível escolher uma instância do ator *Estado* pelo AGA Player, e através da palavra-chave "ativo" ou "corrente", resgatar o momento exato na animação em que aquele determinado estado está ativo.

Outras consultas bastante interessantes podem ser realizadas a partir das instâncias do ator *Unidade*. Por exemplo, é possível resgatar nas 3 animações, exatamente em que condições se encontram as máquinas quando a unidade emite a mensagem de "aceita". Na animação da Máquina de Turing, em especial, é possível recuperar os momentos na animação em que a unidade de controle realiza um movimento à esquerda ou à direita.

Nota-se que, essas possibilidades de consulta pelo observador da animação têm um papel bastante valioso na investigação do conteúdo visual. No caso particular do estudo de caso, proporciona ao visitante do livro eletrônico confrontar visualmente os conceitos descritos nas páginas do *site*.

Além do mapeamento efetuado pela função descrição, a especificação da animação em AgaML provê elementos que objetivam atender as estratégias de indexação orientadas a metadados e taxonomias. Por meio desses elementos é possível armazenar informações bibliográficas e descrições adicionais para categorizações subjetivas, como por exemplo, classes semânticas e classes visuais [ARA 2000].

Quanto ao suporte à recuperação de informação, o GIF não possui nenhuma estrutura projetada especificamente para este propósito. Porém, a especificação do GIF contém a definição de blocos de propósito geral, os quais podem ser explorados para o armazenamento de informações adicionais. O bloco de extensão de comentário, por exemplo, é dedicado a armazenar texto sem propósito de *rendering*. Esse texto pode ser utilizado para disponibilizar informações quanto à autoria ou conteúdo da animação.

Embora haja essa possibilidade, o autor deste texto desconhece algum programa de visualização de animações que aproveite esses blocos para prover funcionalidades de recuperação de informação semelhantes às descritas para o AGA.

5.4 Considerações Finais

Durante a criação das animações do estudo de caso, encontrou-se alguma dificuldade para o posicionamento espacial das instâncias e a sincronização de seus comportamentos. Porém, acredita-se que com o desenvolvimento de um editor de animações especializado para o AGA, as dificuldades encontradas serão minimizadas, pois muitos dos valores contidos na fita de entrada serão adquiridos a partir da interface visual de criação e não determinados diretamente pelo usuário como é atualmente. Quanto à sincronização dos atores, em especial, o modelo AGA-S, apresentado no próximo capítulo, acrescenta ao modelo atual, a operação de sincronização baseada na composição de autômatos temporizados.

O comparativo realizado entre o AGA e o GIF demonstra que o modelo proposto possui várias vantagens em relação à estrutura adotada pelo GIF quanto ao espaço de

armazenamento necessário para representação, reusabilidade e manutenibilidade das animações e ao suporte à recuperação de informação. O estudo de caso desenvolvido evidencia de forma bastante prática cada uma dessas contribuições. Porém, é importante ressaltar que para a criação de pequenas animações, como por exemplo, a criação de ícones animados para páginas *Web*, essas vantagens causam pouco impacto em relação ao GIF e, portanto, o suporte oferecido atualmente pelos navegadores para a decodificação desse formato sem a necessidade de transferência de um *applet* para visualização é um ponto favorável a ser considerado.

6 Extensão do Modelo AGA utilizando Autômatos Temporizados – Modelo AGA-S

6.1 Introdução

O modelo AGA-S (Animação Gráfica baseada em Autômatos Temporizados Sincronizados), apresentado neste capítulo, estende o modelo AGA para prover restrições temporais às especificações dos atores e também ampliar as funcionalidades do modelo quanto à interação com o observador da animação.

As restrições temporais são adicionadas ao modelo com o objetivo de atribuir propriedades de temporização ao padrão de comportamento dos atores, como também subsidiar a apresentação de mídias contínuas na animação, além das imagens estáticas utilizadas como saída no modelo AGA.

Para o desenvolvimento do modelo AGA-S foi realizado um estudo sobre autômatos temporizados que resultou na redação de uma monografia [ACC 2001b]. Nesse trabalho, foram descritas as propostas de Merrit, Modugno e Tuttle [MER 91] para a temporização do autômato de entrada e saída [LYN 87], e a de Alur e Dill [ALU 94] que pode ser vista com uma generalização das máquinas de estados finitos. As propostas estudadas possuem modelos sintáticos e semânticos próprios.

O modelo proposto por Alur e Dill se mostrou mais adequado para a extensão do AGA, pois é definido a partir de um conjunto finito de estados, possibilitando assim a aplicação de métodos de verificação baseados na abordagem de modelos. Este modelo também estabelece as restrições temporais a partir do uso de relógios fictícios, o que não altera a estrutura básica do autômato já utilizada pelo modelo AGA.

O modelo de Alur e Dill é descrito em detalhes na seção 6.2. Em especial, a seção 6.2.4 apresenta a operação de composição de autômatos temporizados utilizada como base para a sincronização dos atores no modelo AGA-S. Na seção 6.3, o modelo AGA-S é apresentado, enfocando principalmente as restrições temporais dos atores, a operação de sincronização e o controle de interação com o usuário. A seção 6.4 reúne as principais características do modelo AGA-S em relação ao modelo AGA anteriormente definido.

6.2 Autômato Temporizado

O autômato temporizado de Alur e Dill [ALU 94] foi proposto com o objetivo de modelar o comportamento de sistemas de tempo real. Esse modelo pode ser visto como um sistema de transição de estados com restrições de tempo [ALU 99]. As restrições são construídas a partir de variáveis de controle de tempo, chamadas de relógios. O autômato temporizado pode representar tanto características qualitativas, como o não determinismo, quanto características quantitativas, como periodicidade. O modelo de tempo utilizado pelo autômato temporizado é denso, já que utiliza valores pertencentes aos reais para os relógios.

Uma abordagem deste modelo enfocando a teoria de linguagens formais pode ser vista em [ALU 94], onde é definido como uma generalização das máquinas de estados finitos de cadeias infinitas do tipo autômatos-ω [BUC 62, CHO 74, MCN 66, THO 90]. Nesse enfoque, o autômato temporizado é definido como um reconhecedor de seqüências infinitas de símbolos, onde cada símbolo é associado a um valor de tempo real, formando palavras temporizadas.

Aplicações para especificação e verificação de sistemas de tempo real vêm sendo desenvolvidas tendo como base o modelo de Alur-Dill. Entre elas, se encontram: o COSPAN temporizado[HAR 96], KRONOS [DAW 96] e UPPAAL [LAR 97]. Em geral, estas ferramentas possibilitam a modelagem dos autômatos, proporcionando verificações automáticas. Por exemplo, analisar se determinados estados são alcançáveis na composição dos autômatos.

6.2.1 Visão Geral do Modelo

O autômato temporizado, proposto em [ALU 99], pode ser visto como um sistema de transição, o qual possui um grafo de transição finito com restrições de temporização associadas a suas arestas e estados.

As restrições são definidas a partir da verificação de um conjunto finito de relógios, os quais armazenam valores reais que determinam o tempo decorrido desde a sua última inicialização. O incremento dos valores dos relógios é realizado de acordo com uma referência global de tempo. As transições entre os estados são consideradas instantâneas e o tempo passa durante a ocupação de um estado, também chamado de locação. Uma transição só pode ser realizada se a avaliação do relógio satisfizer a restrição da aresta. Da mesma maneira, um estado pode ser ocupado apenas pelo tempo tolerado pela restrição associada a ele, também chamada de restrição invariável.

O exemplo da figura 6.1 ilustra um autômato temporizado que possui os estados *q0* e *q1*, sendo o primeiro um estado inicial. Nesse exemplo, existe apenas a presença de um relógio rotulado como *x*, e duas restrições, uma associada à aresta *(q1, q0)*, e uma restrição invariável associada ao estado *q1*. A restrição ligada à aresta permite que ocorra uma transição de *q1* para *q0* apenas quando o valor do relógio *x* for superior ou igual a *1*. Por outro lado, a restrição invariável em *q1*, restringe a permanência do sistema neste local apenas enquanto o relógio *x* possuir valores inferiores a *2*. Na aresta *(q0, q1)* existe uma instrução de inicialização de *x* (*x:=0*), a qual determina a atribuição de *x* com *0*, toda vez que for utilizada. Desta maneira, o modelo expressa que um evento *b* só pode ocorrer dentro do intervalo de tempo *(1, 2)* após a ocorrência de um *a*.

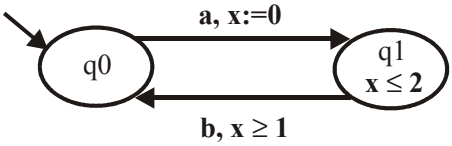


FIGURA 6.1 – Exemplo de autômato temporizado

6.2.2 Restrições e Interpretações de Relógio

Como foi visto na seção anterior, as propriedades temporais são expressas por restrições baseadas nos relógios. Estas restrições são construídas a partir de restrições atômicas que comparam um relógio com uma constante de tempo. A constante pode ser qualquer valor racional não negativo (Q).

Formalmente, para um conjunto de variáveis de relógio X, o conjunto de restrições, denotado por $\Phi(X)$, possui as restrições de relógio φ definidas pela gramática: $\varphi := x \leq c \mid c \leq x \mid x<c \mid c < x \mid \varphi_1 \wedge \varphi_1$, onde x é um relógio em X e c é uma constante em Q. Restrições como *verdadeiro*, $x=c$ e $x \in [2,5)$, podem ser definidas como formas abreviadas.

A interpretação de relógio v para um conjunto de relógios X, associa um valor real para cada relógio, ou seja, realiza o mapeamento de X para o conjunto $\mathbb{R}$ de números

reais não negativos. Uma interpretação v para X satisfaz uma restrição de relógio φ sobre X, se e somente se, φ é verdadeira para os valores dados por v. Para $\delta \in \mathbb{R}$, $v + \delta$ denota a interpretação de relógio que mapeia todo relógio x para o valor $v(x) + \delta$. Para $Y \subseteq X$, $v[Y:=0]$ denota a interpretação de relógio para X que associa 0 para cada $x \in Y$, e mantém a evolução dos outros relógios segundo v.

6.2.3 Modelo

O autômato temporizado é definido como uma 6-upla $A = (L, L^0, \Sigma, X, I, E)$, onde:

L conjunto finito de estados (locações);

L^0 conjunto de estados iniciais ($L^0 \subseteq L$);

Σ alfabeto de símbolos de entrada (rótulos);

X conjunto finito de relógios;

I mapeamento que associa cada estado s em L com alguma restrição de relógio em $\Phi(X)$;

E conjunto de transições ($E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$). Uma transição $\langle s, a, \varphi, \lambda, s' \rangle$ representa uma transição do estado s para s' mediante o símbolo a. A restrição de relógio φ determina quando a transição está habilitada, enquanto o conjunto $\lambda \subseteq X$ determina os relógios a serem inicializados nessa transição.

O comportamento do autômato temporizado pode ser definido a partir dos conceitos tradicionais de sistemas de transição. Seja A um autômato temporizado e S_A um sistema de transição associado a ele, onde um estado de S_A é um par (s, v), tal que s é um estado de A e v é uma interpretação de relógio para X que satisfaz a restrição invariável $I(s)$. Um estado (s, v) é considerado inicial, se s é um estado de A e $v(x)=0$ para todos os relógios. O alfabeto de símbolos de entrada ou rótulos (Σ_A) do sistema de transição S_A é dado por $\Sigma \cup \mathbb{R}$. Há dois tipos de transições em S_A, o estado pode mudar devido:

- ao passar do tempo – para um estado (s, v) e um valor real de incremento de tempo $\delta \geq 0$, $(s,v) \xrightarrow{\delta} (s, v+\delta)$, se para todo $0 \leq \delta' \leq \delta$, $v + \delta'$ satisfaz a restrição invariável $I(s)$;
- à transição de estado no autômato temporizado – para um estado (s, v) e a uma transição $\langle s, a, \varphi, \lambda, s' \rangle$, tal que v satisfaz φ, $(s, v) \xrightarrow{a} (s', v[\lambda :=0])$;

Por exemplo, o autômato da figura 6.1, definido como o autômato temporizado $A=(\{q_0, q_1\}, \{q_0\}, \{a, b\}, \{x\}, I, \{ (q_0, a, verdadeiro, \{x\}, q_1), (q_1, b, x \geq 1, \{\}, q_0) \})$, onde $I(q_1) = x \leq 2$, pode ser descrito pelo sistema de transição $S_A = (Q_A, Q_A^0, \Sigma_A, T)$, onde $Q_A = \{q_0, q_1\} \times \mathbb{R}$, $Q_A^0 = \{(q_0, 0)\}$, $\Sigma_A = \{a, b\} \cup \mathbb{R}$ e T é conjunto de transições que respeitam os tipos definidos acima. A seqüência

$$(q_0, 0) \xrightarrow{3} (q_0, 3) \xrightarrow{a} (q_1, 0) \xrightarrow{0,5} (q_1, 0,5) \xrightarrow{1} (q_1, 1,5) \xrightarrow{b} (q_0, 1,5) \xrightarrow{a} (q_1, 0)$$

representa uma série de transições válidas para o sistema S_A e conseqüentemente descreve o comportamento do autômato temporizado A para uma determinada seqüência de eventos.

6.2.4 Composição de Autômatos Temporizados

A composição de autômatos temporizados é essencialmente uma composição de sistemas de transição, que além da combinação de estados e transições, realiza também a conjunção das restrições e união dos relógios. Formalmente, a composição de um autômato temporizado $A_1=(L_1, L_1^0, \Sigma_1, X_1, I_1, E_1)$ com $A_2=(L_2, L_2^0, \Sigma_2, X_2, I_2, E_2)$, denotado por $A_1||A_2$, é o autômato temporizado $A_r=(L_1 \times L_2, L_1^0 \times L_2^0, \Sigma_1 \cup \Sigma_2, X_1 \cup X_2, I, E)$, onde $I(s_1, s_2)=I(s_1) \wedge I(s_2)$. Assume-se que $X_1 \cap X_2=\varnothing$. As transições de E são definidas por:

- para $a \in \Sigma_1 \cap \Sigma_2$, para todo $\langle s_1, a, \varphi_1, \lambda_1, s'_1 \rangle$ em E_1 e $\langle s_2, a, \varphi_2, \lambda_1, s'_2 \rangle$ em E_2, E possui $\langle (s_1, s_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (s'_1, s'_2) \rangle$;
- para $a \in \Sigma_1 - \Sigma_2$, para todo $\langle s, a, \varphi, \lambda, s' \rangle$ em E_1 e todo t em L_2, E possui $\langle (s, t), a, \varphi, \lambda, (s', t) \rangle$;
- para $a \in \Sigma_2 - \Sigma_1$, para todo $\langle s, a, \varphi, \lambda, s' \rangle$ em E_2 e todo t em L_1, E possui $\langle (t, s), a, \varphi, \lambda, (t, s') \rangle$.

A composição de autômatos temporizados determina o sincronismo a partir dos símbolos comuns do alfabeto de entrada. Esta operação possibilita que sistemas complexos possam ser definidos a partir do produto dos sistemas componentes.

6.2.5 Exemplo[1] de Composição de Autômatos Temporizados

O exemplo utilizado nesta seção ilustra a composição entre uma cancela de bloqueio para passagem de trem e um aparelho eletrônico que a controla. O controlador eletrônico é sensível aos movimentos de chegada e partida do trem em frente da cancela.

O autômato temporizado $P=(\{t_0, t_1, t_2, t_3\}, \{t_0\}, \{abaixar, descida, levantar, subida\}, \{y\}, I_P, E_P)$ é criado para modelar o comportamento da cancela de bloqueio quanto ao tempo de resposta aos sinais de *levantar* e *abaixar*. O estado t_0 indica a cancela aberta, enquanto t_2 representa a cancela fechada. Os estados t_1 e t_3 indicam a espera entre a requisição de uma tarefa e sua resposta com a ação. As restrições invariáveis $I_P(t_1)=y \leq 1$ e $I_P(t_2)=y \leq 2$ em I_P determinam, respectivamente, que a cancela pode demorar no máximo 1 minuto depois de emitido o sinal *abaixar* para bloquear (*descida*), e pode esperar no máximo 2 minutos para abrir (*subida*) mediante o sinal de *levantar*. O conjunto E_P pode ser visto através da presença das arestas na figura 6.2. A restrição $y \geq 1$ na transição de t_3 para t_0 indica o tempo de espera mínimo para determinar a subida do portão. Devido a essa restrição e a restrição invariável em t_3, após o sinal de *levantar*, o evento de *subida* deve ocorrer entre 1 e 2 minutos. Os controles temporais são realizados sobre o único relógio y.

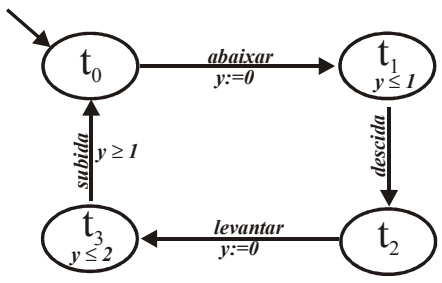


FIGURA 6.2 – Autômato temporizado da cancela

[1] Este exemplo é uma versão simplificada do exemplo apresentado em [ALU 96].

O autômato temporizado *P* se comunica com o controlador eletrônico através dos símbolos *abaixar* e *levantar*. O controlador é definido como o autômato temporizado $C=(\{u_0, u_1, u_2\}, \{u_0\}, \{aproximação, saída, abaixar, levantar\}, \{z\}, I_C, E_C)$ que modela o comportamento desse aparelho quando registra a aproximação e saída do trem. Os estados u_1 e u_2 simbolizam, respectivamente, o tempo de reposta para os eventos de aproximação e saída do trem. As restrições invariáveis $I_C(u_1)=I_C(u_2)=z\leq 1$ determinam que o tempo de resposta aos eventos deve ser de no máximo 1 minuto. Devido à presença da restrição $z=1$ na aresta de u_1 para u_0, o tempo de resposta para abaixar a cancela fica fixado em exatamente 1 minuto. O conjunto E_c pode ser visto através da presença das arestas na figura 6.3.

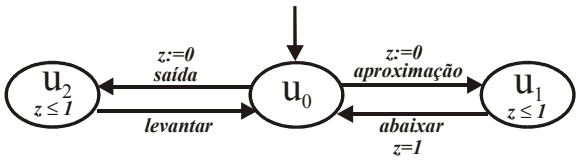


FIGURA 6.3 – Autômato temporizado do controlador eletrônico

A computação conjunta dos dois autômatos interagindo pode ser vista através da composição. Esta operação utiliza os símbolos comuns, *levantar* e *abaixar*, para estabelecer o sincronismo entre os dois autômatos. A composição temporizada dos autômatos é definida como $PC = P||C = (L_{PC}, L^0_{PC}, \Sigma_{PC}, X_{PC}, I_{PC}, E_{PC})$, onde:

$L_{PC} = \{t_0, t_1, t_2, t_3\} \times \{u_0, u_1, u_2\} = \{ t_0u_0, t_0u_1, t_0u_2, t_1u_0, t_1u_1, t_1u_2, t_2u_0, t_2u_1, t_2u_2, t_3u_0, t_3u_1, t_3u_2 \}$,

$L^0_{PC} = \{t_0\} \times \{u_0\} = \{t_0u_0\}$,

$E_{PC} = \{abaixar, descida, levantar, subida\} \cup \{aproximação, saída, abaixar, levantar\} = \{aproximação, saída, abaixar, levantar, descida, subida\}$,

$X_{PC}=\{y\} \cup \{z\}=\{y, z\}$,

I_{PC} é o conjunto de restrições invariáveis obtido pela conjunção das restrições dos autômatos originais, por exemplo, $I_{PC}(t_1u_1)=I_P(t_1) \wedge I_C(u_1) = y\leq 1 \wedge z \leq 1$. Os elementos desse conjunto podem ser observados em cada estado na figura 6.4.

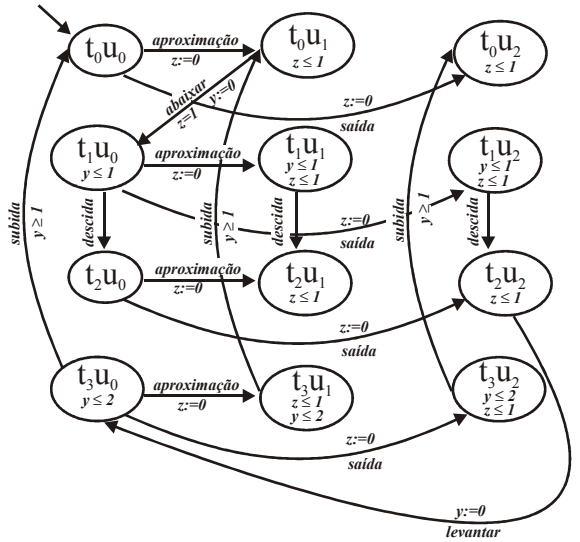


FIGURA 6.4 – Composição dos autômatos temporizados da cancela e controlador

E_{PC} é o conjunto de transições definidas de acordo com sua participação no autômato original. Para as transições de símbolos de sincronismo é preciso manter os

estados de origem e destino originais, e compor a conjunção das restrições e inicializações. Por exemplo, a partir de $\langle t_0,abaixar,\{\},\{y\},t_1\rangle$ em E_P e $\langle u_1,abaixar,\{z=1\},\{\},u_0\rangle$ em E_C é produzida a transição $\langle t_0u_1,\ abaixar,\ \{z=1\},\{y\},t_1u_0\rangle$. O conjunto completo pode ser observado através das arestas na figura 6.4.

A figura 6.5 ilustra a seqüência de eventos α que respeita as restrições temporais da composição *PC*. A seqüência de eventos α = *(aproximação, 3)(abaixar, 4)(descida, 5)(saída, 9)(levantar, 10)(subida, 11,5)*, é representada pelos pares *(a, t)*, onde $a \in \Sigma_{PC}$ e $t \in \mathbb{R}$. As caixas em cinza na figura, demonstram a evolução do tempo para os relógios *z* e *y*, assim como suas inicializações. As linhas verticais indicam a ocorrência do evento, e suas interseções com a linha do tempo descrevem os instantes (em minutos) que ocorreram.

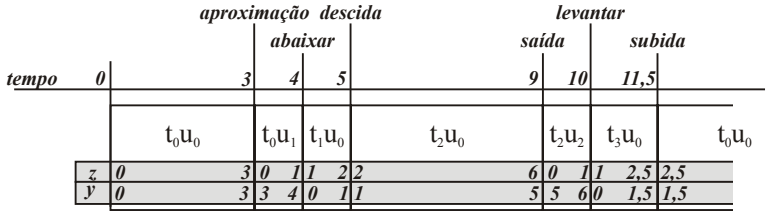


FIGURA 6.5 – Comportamento da composição PC

O estado inicial é t_0u_0 e os relógios estão inicializados com *0*. O primeiro evento indica a aproximação do trem aos 3 minutos e causa a transição instantânea que inicializa novamente *z* com *0*. A transição de t_0u_1 para t_1u_0 ocorre exatamente com $z=1$, devido à restrição imposta na aresta. A transição provocada pela *subida* respeita a invariável $I_{PC}(t_3u_0)=y\leq 2$ e a restrição da aresta $y\geq 1$, ocorrendo a *1,5* do evento *levantar*.

Na composição *PC*, todos os estados são acessíveis por alguma seqüência, porém há composições que podem compor estados que não são alcançáveis devido às restrições de tempo combinadas. Isto ocorre porque a composição é realizada a partir de uma operação sintática onde não é realizada a análise das restrições [ALU 99].

6.3 Modelo AGA-S

O modelo AGA-S conserva a estrutura baseada em autômatos finitos definida para o modelo AGA, porém, estende a definição da unidade básica do modelo, o ator, a partir do autômato temporizado proposto por Alur e Dill.

O autômato temporizado de Alur e Dill, apresentado na seção 6.2, é utilizado para adicionar restrições temporais à estrutura do ator, chamado neste modelo, de ator AGA-S. Essas restrições são utilizadas principalmente com o objetivo de atribuir propriedades de temporização ao padrão de comportamento do ator.

O ator AGA-S também tem o alfabeto de saída estendido para emitir não só objetos discretos[1], como no modelo AGA (imagens estáticas), mas também objetos contínuos, como, por exemplo, sons. Essa extensão foi proposta com o objetivo de modelar animações para *Web* que utilizam vários tipos de mídias sincronizadas.

Para atender de forma adequada essa última característica, foi definida a operação de composição para os atores AGA-S com o propósito de sincronizar a apresentação dos objetos por eles coordenados. Essa operação é definida a partir de composição de autômatos temporizados descrita na seção 6.2.4.

[1] O termo *discreto* é adotado no texto para indicar que a apresentação do objeto é instantânea, enquanto o termo *contínuo* é adotado para objetos que necessitam de tempo para a apresentação.

No modelo AGA, a interação entre o observador e a animação é limitada aos botões do painel de controle do AGA Player, os quais são responsáveis apenas em controlar a reprodução. No AGA-S, porém, essa funcionalidade é ampliada para que o observador possa interagir diretamente com cada ator. Para prover este recurso o modelo AGA-S conta com um componente adicional, denominado autômato de controle, o qual coordena as atividades de interação entre os atores e o observador durante a apresentação da animação.

6.3.1 Ator[1] AGA-S

O ator AGA-S é definido como uma 6-upla : $A = (At, \Delta, \lambda_c, \sigma, F, D)$, onde:

At autômato temporizado $At = (L, L^0, \Sigma, X, I, E)$;
Δ alfabeto de símbolos de saída;
λ_c função de saída contextual (função parcial $\lambda_c : E \times F^* \rightarrow \Delta^F$);
σ função descrição (função parcial $\sigma : L \rightarrow D$);
F conjunto finito de funções de transformação de Δ^F;
D conjunto finito de descrições;

As restrições temporais dos relógios em At definem o comportamento dinâmico do ator em relação ao tempo de permanência nos estados e os intervalos disponíveis para transição. A figura 6.6, por exemplo, ilustra o ator *bicho*, apresentado inicialmente no capítulo 3, modelado a partir da definição do ator AGA-S. Nota-se que, agora, esse ator possui restrições temporais vinculadas aos estados e transições definidas sobre um relógio y.

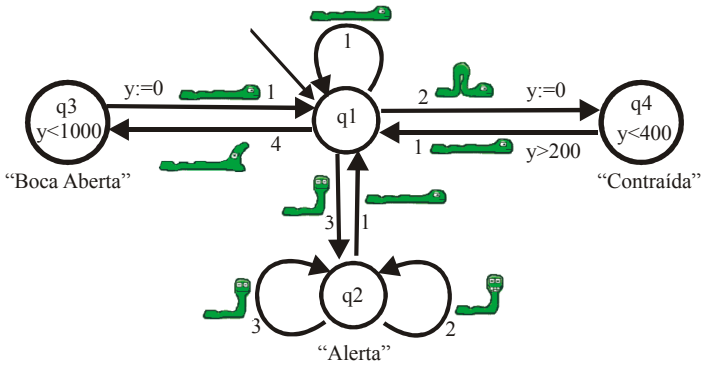


FIGURA 6.6 – Ator Bicho temporizado

A restrição invariável atribuída ao estado *q4* do ator *Bicho* determina que a imagem dele contraído apenas será emitida no máximo por um tempo inferior a 400 milésimos de segundo, enquanto a restrição vinculada à aresta *(q4,q1)* determina que essa mesma imagem poderá ser apresentada no mínimo por um tempo superior a 200 milésimos de segundo. Nota-se que, em toda fita de entrada especificada para esse ator, essas restrições deverão ser respeitadas e, portanto, todas as animações em que ele participar esse padrão de comportamento dinâmico será mantido.

O alfabeto de saída Δ é um conjunto finito de objetos para apresentação. Esses objetos podem ser discretos ou contínuos. Para esta definição do modelo AGA-S são estabelecidos 4 tipos primitivos de objetos: texto, som, gráfico (imagem) e vídeo. A

[1] No artigo [ACC 01a] escrito pelo autor, o termo *elemento cênico* foi adotado ao invés de *ator*, porém, o termo *ator* foi conservado nesta dissertação por ser mais adequado à nomenclatura utilizada na área de Computação Gráfica.

eleição desses 4 tipos de objetos está baseada na nomenclatura definida em [HEL 2001] para os tipos de mídia. Tipos correspondentes aos adotados são utilizados pelo Flash e o SVG.

Nota-se que a função de saída contextual λ_c permanece a mesma do modelo AGA, apenas foi adequada para vincular a saída às transições do autômato temporizado. O conjunto de funções de transformação *F*, utilizado pela função de saída contextual, tem o mesmo propósito do definido no modelo original, porém, pode ser explorado para a definição de funções para tratamento de mídia contínua, como, por exemplo, selecionar parte da mídia para apresentação.

O comportamento de emissão da saída para os objetos discretos permanece o mesmo do modelo AGA. O objeto vinculado à aresta do autômato é transformado pelas funções determinadas na fita de entrada e apresentado no momento da transição. Para os objetos contínuos, em especial, a transição dispara o início da apresentação, e o tempo de duração da mídia avança durante a ocupação do estado. Esse tipo de comportamento já é adotado na implementação do AGA Player para a emissão de sons.

A função descrição σ e o conjunto de descrições *D* mantêm as mesmas definições do modelo AGA com o propósito de estabelecer à vinculação de descrições semânticas aos estados do autômato temporizado.

É importante ressaltar, que embora as características da fita de entrada para os atores AGA-S permaneçam as mesmas do modelo AGA, durante a especificação do conteúdo das células, as restrições temporais devem ser respeitadas. Caso contrário, durante a simulação dos autômatos ou checagem prévia[1], o conteúdo será dado como inconsistente (não reconhecido).

6.3.2 Sincronização de Atores AGA-S

A sincronização dos atores AGA-S é estabelecida pelos símbolos comuns aos alfabetos de entrada dos respectivos autômatos temporizados. A ocorrência de um símbolo comum determina que as transições correspondentes nesses autômatos devam ocorrer de forma sincronizada. Como a saída está vinculada à transição dos autômatos temporizados, a sincronização garante que dois ou mais atores iniciem a apresentação dos objetos de saída ao mesmo tempo.

A sincronização dos atores é definida formalmente a partir da composição de autômatos temporizados, apresentada na seção 6.2.4, estendida para o modelo do ator AGA-S. Portanto, a composição de um ator $A_1 = ((L_1, L_{01}, \Sigma_1, X_1, I_1, E_1), \Delta_1, \lambda_{c1}, \sigma_1, F_1, D_1)$ com $A_2 = ((L_2, L_{02}, \Sigma_2, X_2, I_2, E_2), \Delta_2, \lambda_{c2}, \sigma_2, F_2, D_2)$, denotado por $A_1||A_2$, é o ator $A_r = ((L_1 \times L_2, L_{01} \times L_{02}, \Sigma_1 \cup \Sigma_2, X_1 \cup X_2, I, E), \Delta_1 \cup \Delta_2, \lambda_c, \sigma, F_1 \cup F_2, D_1 \cup D_2)$, onde $I(s1, s2)=I(s1)\wedge I(s2)$ e $\sigma(s1, s2)=\sigma(s1) \cup \sigma(s2)$. Assume-se que $X_1 \cap X_2 = \varnothing$. As transições *E* e a função λ_c são definidas por:

- para $a \in \Sigma_1 - \Sigma_2$, para todo $e_1=\langle s,a,\varphi, \chi, s' \rangle$ em E_1 e todo *t* em L_2, *E* possui $e=\langle (s,t), a, \varphi, \chi, (s',t) \rangle$ e $\lambda(e, f)=\lambda_1(e_1, f)$;
- para $a \in \Sigma_2 - \Sigma_1$, para todo $e_2=\langle s,a,\varphi, \chi, s' \rangle$ em E_2 e todo *t* em L_1, *E* possui $e=\langle (t,s), a, \varphi, \chi, (t,s') \rangle$ e $\lambda(e, a, f)=\lambda_2(e_2, f)$;
- para $a \in \Sigma_1 \cap \Sigma_2$, para todo $e_1=\langle s_1, a, \varphi_1, \chi_1, s'_1 \rangle$ em E_1 e $e_2=\langle s_2, a, \varphi_2, \chi_2, s'_2 \rangle$ em E_2, *E* possui $e=\langle (s_1,s_2), a, \varphi_1 \wedge \varphi_2, \chi_1 \cup \chi_2, (s'_1, s'_2) \rangle$ e $\lambda(e, f) = \lambda_1 (e_1, f_1) \cup \lambda_2(e_2, f_2)$ onde $f=f_1 f_2$.

[1] A checagem prévia diz respeito a verificações realizadas futuramente com o desenvolvimento do ambiente para construção de animações em AGA-S.

O ator A_r resultante de $A_1||A_2$ pode produzir estados inalcançáveis tanto pela inexistência de caminhos quanto pelo aparecimento de limites impossíveis de serem satisfeitos causados pela conjunção das restrições temporais. Desta maneira estes estados podem ser eliminados sem prejuízo à composição dos atores.

A figura 6.7 ilustra 2 atores AGA-S: *Maçã* e *Efeito Sonoro*. O primeiro modela o comportamento de uma maçã que pode ter suas imagens alteradas na saída para indicar sucessivas mordidas. O estado *q2* é associado à descrição *"mordida"* para indicar esta condição.

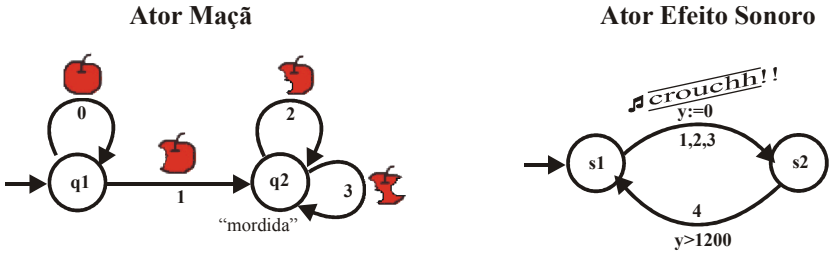


FIGURA 6.7 – Ator Maçã e Efeito Sonoro

O segundo ator controla a emissão de um efeito sonoro que simula o som de uma mordida. Para esse ator é criado o relógio *y* com o propósito de controlar o tempo de duração do som. Nota-se que a restrição *y>1200,* vinculada à transição *4,* impede que o som seja disparado novamente antes de 1200 milésimos de segundo, ou seja, o tempo de duração do efeito sonoro é respeitado para sua reprodução sucessiva.

A sincronização dos dois atores é proposta nesse exemplo para que a imagem da maçã mordida apareça para o observador acompanhada do efeito sonoro da mordida. Para isso, toda transição do ator *Maçã,* que projeta a maçã mordida, é sincronizada com a transição do ator *Efeito Sonoro,* que emite o som, utilizando símbolos do alfabeto de entrada comuns (1,2 e 3).

A figura 6.8 ilustra o ator resultante da composição dos atores *Maçã* e *Efeito Sonoro*. O estado *q1s2*, embora participe do resultado da operação de composição, é inalcançável, sendo assim, pode ser excluído do ator resultante. A descrição do estado *q2* do ator *Maçã* é propagada para todos os novos estados que ele participa. As saídas dos atores são conservadas nas novas transições de acordo com os estados de origem e destino dos atores componentes. Nota-se, portanto, que o ator resultante reflete a sincronização dos atores componentes respeitando suas propriedades originais.

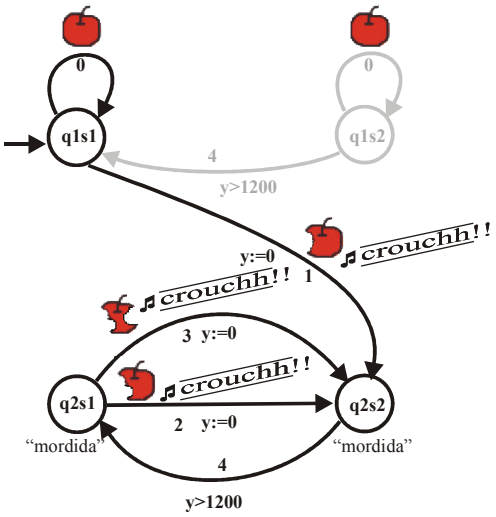


FIGURA 6.8 – Composição dos atores Maçã e Efeito Sonoro

Na sincronização obtida, a imagem da maçã mordida sempre virá acompanhada do efeito sonoro e, no caso de sucessivas mordidas, o tempo entre uma mordida e outra respeitará a duração do som. A elaboração da fita de entrada para o ator resultante deve ser realizada tendo em vista as restrições temporais da composição, pois caso contrário, durante a simulação, a entrada pode ser dada como inconsistente.

6.3.3 Interação

O modelo AGA-S conta com um autômato de controle agregado a cada ator AGA-S para prover o controle das atividades de interação com o observador. Esse autômato também é estendido a partir do autômato temporizado e efetua as transições mediante a ocorrência de algum evento externo.

Portanto, o autômato de controle é definido como uma 3-upla $AC = (At , \alpha , A)$, onde :

At autômato temporizado $At = (L, L^0, \Sigma, X, I, E)$;

α função execução (função parcial $\alpha : E \rightarrow 2^A$);

A conjunto finito de ações $at \# ft$ ($at \in AS$ e $ft \in T$, onde AS é um conjunto de atores AGA-S e T um conjunto de fitas de entrada).

O alfabeto de entrada Σ do autômato temporizado At tem cada um dos seus símbolos associados a eventos externos. Esses eventos externos podem ser provocados tanto pelo observador da animação como pelo sistema. Por exemplo, o observador pode gerar eventos de *mouse* como pressionar o botão direito ou esquerdo, enquanto o sistema pode gerar eventos informando que a reprodução de uma determinada mídia foi concluída, ou ainda, que alguma restrição invariável do autômato temporizado foi excedida.

Quando um evento externo ocorre, a transição do autômato temporizado cujo rótulo corresponde ao evento é efetuada. Se não há mapeamento de transição correspondente ao evento, o autômato simplesmente descarta a entrada.

A função execução α mapeia as transições do autômato temporizado para um conjunto de partes de A. O conjunto A possui ações do tipo $at \# ft$, as quais determinam que o ator at deve processar a fita de entrada ft. Desta maneira, quando uma transição é realizada, as ações vinculadas a ela são executadas, trocando as fitas de entrada dos atores AGA-S. No momento da troca, a fita corrente é descartada e o ator é restaurado para o estado inicial com os seus relógios inicializados.

Cada ator AGA-S pode estar associado a um autômato de controle. Essa associação é importante para classificar o destino do evento externo. Por exemplo, quando um evento de *mouse* é produzido, a localização da imagem de saída do ator AGA-S determina se o evento foi efetuado em sua região, e então, repassa o evento ao autômato de controle. A relação de ordem que indica as camadas de apresentação dos atores AGA-S determina a precedência no atendimento dos eventos.

A figura 6.9 ilustra um autômato de controle associado ao ator *Bicho*. Esse autômato possui dois eventos externos associados, o evento *"click"*, registrado quando o botão esquerdo do mouse é pressionado e liberado, e o evento *"overflow"*, gerado pelo sistema quando a restrição invariável é excedida.

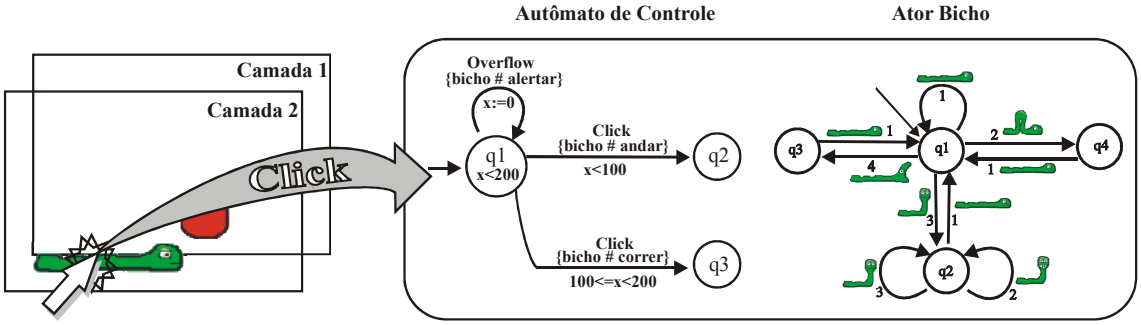


FIGURA 6.9 – Autômato de controle associado ao ator Bicho

Quando o observador pressiona o botão com o cursor sobre a figura do ator *Bicho*, o evento *"click"* é repassado para o autômato de controle. Caso esse evento ocorra num instante inferior a 100 milésimos de segundo, a fita *"andar"* é atribuída ao ator para provocar o comportamento do bicho andando até a maçã. Caso o mesmo evento ocorra dentro do intervalo de 100 a 200 milésimos de segundo, a fita *"correr"* será adotada. Se nenhum evento de *mouse* ocorrer dentro de 200 milésimos de segundo, o sistema gera um evento *"overflow"* indicando que a restrição invariável do estado *q1* foi excedida. Conseqüentemente, o autômato de controle efetua a transição correspondente, inicializando o relógio *x* e atribuindo a fita *"alertar"* ao ator.

É importante ressaltar, que as ações vinculadas às transições podem trocar a fita de outros atores além do associado. Assim, é possível construir atores AGA-S com a semântica de botões, que ao serem estimulados por eventos externos, mudam seus próprios comportamentos. Da mesma forma, podem disparar as animações de diversos outros atores pela atribuição das fitas de entrada.

6.4 Considerações Finais

Embora o modelo AGA-S estenda o ator a partir do autômato temporizado proposto por Alur e Dill, diferentemente do modelo AGA, a estrutura básica do modelo não é alterada, apenas é adicionada a possibilidade de atribuição de restrições temporais às especificações dos atores.

Deste modo, os benefícios apresentados para o modelo AGA quanto ao espaço de armazenamento, reusabilidade e manutenibilidade das animações e suporte à consulta, são preservados no modelo AGA-S.

A operação de sincronização definida para os atores AGA-S baseada na proposta de Alur e Dill para composição de autômatos temporizados, garante que o ator resultante mantenha as propriedades dos atores componentes além da sincronização da apresentação dos objetos de saída. Essa operação ilustra a possibilidade de definição de outras operações formais para a construção de atores mais complexos a partir de componentes simples.

A definição do autômato de controle suporta a especificação de interação para o modelo AGA-S. De maneira semelhante aos atores AGA e AGA-S, a mesma estrutura utilizada para especificação do autômato de controle pode ser simulada para coordenar as ações durante a apresentação da animação. A utilização de relógios para estabelecer as restrições temporais dos autômatos é bastante conveniente para implementação desse recurso no AGA Player.

7 Conclusão e Trabalhos Futuros

7.1 Conclusões

Na *World Wide Web*, grande parte do conteúdo é composto por informações visuais. Entre elas, são encontradas as animações 2D, amplamente utilizadas para diversos fins. Apesar de existirem várias alternativas para prover esse tipo de animação na *Web*, algumas questões quanto ao espaço de armazenamento necessário para representá-las, manutenção de seu conteúdo e suporte à recuperação de informação, que tem motivado esforços para melhorar os processos de criação, manutenção, apresentação e consulta de animações.

Os modelos AGA e AGA-S propostos pelo autor neste texto são novas alternativas para prover animações 2D na *Web* que possuem características que contribuem em vários aspectos com os processos citados.

A definição do alfabeto de saída como um conjunto de objetos básicos utilizados na animação contribui para a redução do espaço de armazenamento, pois um mesmo objeto pode ser apresentado em diversos momentos da animação sem a necessidade de duplicação. A forma de codificação de cada objeto é independente da estrutura dos modelos.

O comparativo realizado entre o AGA e GIF quanto ao espaço de armazenamento demonstra os benefícios que esta característica agrega. As animações apresentadas no estudo de caso, foram representadas pelo AGA com uma quantidade de bytes bem inferior ao GIF (menos da metade em média). Como ficou demonstrado pela análise das características do modelo, o pior caso é dado por animações formadas por seqüências de quadros completamente distintos entre si. Mesmo no pior caso, o crescimento do tamanho das representações em AGA em função do número de quadros é assintomaticamente igual ao GIF. Os experimentos quantitativos realizados no estudo de caso ilustram os resultados teóricos obtidos por Carlos A. P. Campani utilizando a Complexidade de Kolmogorov, que provam que o AGA é mais eficiente para compressão de dados do que o GIF. Esses resultados também podem ser atribuídos ao AGA-S, posto que a estrutura básica do modelo não é alterada.

As características da linguagem AgaML contribuem ainda mais com a redução do espaço de armazenamento no servidor HTTP no desenvolvimento de várias animações, pois as estruturas definidas para especificação de atores, fitas e instâncias podem ser compartilhadas por diversas animações. Os resultados obtidos no estudo de caso ilustram essa vantagem, uma vez que o total de espaço ocupado no servidor pelas animações é menor que a soma dos gastos das 3 animações individualmente devido ao compartilhamento de vários componentes comuns às animações.

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.

O AGA Player explora essa característica do modelo para prover o recurso de consulta ao conteúdo da animação baseado em palavras-chave. O mecanismo desenvolvido verifica o casamento de palavras-chave determinadas pelo observador com as descrições semânticas associadas aos estados dos atores, proporcionando uma rápida recuperação dos instantes da animação em que essa condição é satisfeita. Esse recurso possibilita que o observador explore o conteúdo da animação de acordo com seu

foco de interesse e não fique preso às tradicionais “barras de busca”, através das quais ele localiza trechos da animação por tentativa e erro na maioria das soluções atuais.

Como o mapeamento semântico está associado aos estados dos atores independentemente da fita de entrada, qualquer animação que eles participem está automaticamente mapeada para consulta. Essa característica é ilustrada no estudo de caso, onde os atores *Estado* e *Unidade* são compartilhados pelas 3 animações através de várias instâncias, possibilitando assim, que observador explore, em qualquer uma delas, condições como estado ativo ou unidade de controle com movimento à direita.

As definições dos atores nos modelos AGA e AGA-S encapsulam as propriedades estéticas e comportamentais dos objetos por eles modelados. Essa característica contribui para que o conteúdo da animação seja facilmente reutilizado por outras, pois uma vez especificado o ator, as animações podem reutilizá-lo atribuindo fitas de entradas diferentes para adaptá-lo aos seus contextos. O estudo de caso ilustra bem essa característica, pois a maior parte dos atores é reutilizada nas 3 animações propostas, o que não acontece com as animações equivalentes em GIF. A fita de entrada é outro componente dos modelos que pode ser reutilizada sempre que o mesmo comportamento for desejado para o ator.

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 AgaML suporta essa possibilidade devido ao mecanismo desenvolvido para a criação de instâncias e os recursos herdados do XML para a criação de entidades externas.

A manutenção das animações é favorecida de várias maneiras pelos modelos propostos. A alteração do comportamento dos atores em cada animação pode ser realizada pela mudança do conteúdo das fitas de entrada, dispensando a necessidade de edição das imagens quadro a quadro como no GIF. A expansão ou alteração dos atores tanto para a estrutura do autômato quanto para os objetos do alfabeto de saída podem ser realizadas de forma independente, conservando inalterados os demais componentes da animação. Como os modelos propostos estão definidos a partir de autômatos finitos, as operações de manutenção das animações podem ser definidas formalmente, garantindo que propriedades desejadas sejam conservadas. Essa característica subsidia a construção de processos automatizados para a manutenção das animações.

Quanto às características específicas do modelo AGA-S, a operação de sincronização definida para os atores baseada na proposta de Alur e Dill para composição de autômatos temporizados, garante que o ator resultante mantenha as propriedades dos atores componentes, além da sincronização da apresentação dos objetos de saída. As restrições temporais providas pelo autômato temporizado são de grande importância para assegurar a apresentação completa dos objetos com tempo de duração. Essa operação ilustra a possibilidade de definição de outras operações formais para a construção de atores mais complexos a partir de componentes simples.

A definição do autômato de controle suporta a especificação de interação para o modelo AGA-S com funcionalidades semelhantes às encontradas na *Web* para sistemas de animação em tempo real. De maneira semelhante aos atores AGA e AGA-S, a mesma estrutura utilizada para especificação do autômato de controle pode ser simulada para coordenar as ações durante a apresentação da animação.

É importante ressaltar que, embora o foco deste trabalho seja a aplicação na *Web*, as características apresentadas para os modelos teóricos AGA e AGA-S independem do ambiente de aplicação e, portanto, podem ser aplicados em animações 2D de maneira geral.

7.2 Trabalhos Futuros

7.2.1 Modelos AGA e AGA-S

Na definição do modelo AGA-S, o alfabeto de saída corresponde a diversos tipos de objetos para apresentação, tais como: texto, som, imagem e vídeo. Essa característica aliada à possibilidade de interação provida pelo autômato de controle proporciona várias funcionalidades comuns às encontradas no Hyper-Automaton [MAC 2000] (ambiente semi-automatizado para o suporte ao gerenciamento de hipertextos, cuja arquitetura também está baseada em autômatos finitos). Essa intersecção de funcionalidades indica uma possível unificação dos modelos AGA-S e Hyper-Automaton.

A operação de sincronização definida no modelo AGA-S indica a possibilidade de criação de outras operações formais para alteração, expansão e composição de atores. Em [MEN 99], a Teoria das Categorias é utilizada como instrumento para estabelecer operações similares aplicadas à composição de hiperdocumentos com resultados que favorecem a reutilização dos conteúdos elaborados. Desta forma, o uso da Teoria das Categorias para definição de novas operações para os atores definidos nos modelos AGA e AGA-S deve ser avaliada.

7.2.2 Implementação

A AgaML 2.0 e o AGA Player formam uma solução totalmente operacional para apresentação de animações baseadas no modelo AGA na *Web*. Essa solução foi utilizada com sucesso na produção das animações do estudo de caso, assim como em diversas outras animações criadas ao longo deste trabalho. Porém, com o desenvolvimento do modelo AGA-S, novas funcionalidades foram definidas, e conseqüentemente, a extensão da AgaML e do AGA Player será necessária a fim de atendê-las. Entre essas características, a possibilidade de interação tem um destaque especial, pois ampliará o escopo de utilização das animações em AGA. Com essa extensão implementada será possível a especificação de atores com semânticas de botões e outros componentes de interface.

A edição direta da linguagem AgaML para especificação dos atores e fitas se mostra pouco amigável para o posicionamento espacial das instâncias e descrição dos comportamentos dos atores. Desta forma, o desenvolvimento de um ambiente visual específico para criação de animações em AGA é bastante oportuno. Através dele, o usuário poderá trabalhar de forma interativa em um nível de abstração mais alto, ocultando assim, os detalhes estruturais do modelo.

A implementação atual do AGA Player utiliza apenas representações matriciais para as imagens do alfabeto de saída. Essa característica, embora útil em alguns casos, muitas vezes, ocupa espaço de armazenamento desnecessário para representação de figuras mais simples, como as primitivas geométricas. O SVG (*Scalable Vector Graphics*) proposto pelo W3C possui vários elementos definidos para representar vetorialmente as figuras, os quais poderiam ser facilmente integrados à linguagem AgaML para prover esse tipo de representação ao alfabeto de saída. Para essa integração seria necessário também estender o AGA Player para manipular os elementos do SVG, o qual tem se tornado um padrão na representação vetorial na *Web*.

7.3 Produção Científica

Esta dissertação de mestrado resultou na publicação dos artigos relacionados abaixo até o momento do fechamento deste texto:

- ACCORSI, Fernando; MENEZES, Paulo Fernando Blauth. **Animação Gráfica Baseada na Teoria de Autômatos**. In: WMF - WORKSHOP ON FORMAL METHODS, 2000, João Pessoa, Paraíba. WMF'2000 - 3rd Workshop on Formal Methods. SBC, 2000. p. 122 – 127.
- ACCORSI, Fernando; MENEZES, Paulo Fernando Blauth; NEDEL, Luciana Porcher. **Animação Gráfica Baseada em Autômatos Temporizados Sincronizados**. In: WMF – WORKSHOP ON FORMAL METHODS, 2001, Rio de Janeiro. Proceedings of IV WMF Workshop on Formal Methods. SBC, 2001.

Anexo 1 Artigo Workshop on Formal Methods 2000

O artigo contido neste anexo foi publicado e apresentado pelo autor no III Workshop on Formal Methods realizado em outubro de 2000 durante o Simpósio Brasileiro de Engenharia de Software na cidade de João Pessoa - PB. Este artigo foi aceito na categoria de artigo curto (até 6 páginas) e descreve o modelo AGA.

ANIMAÇÃO GRÁFICA BASEADA NA TEORIA DE AUTÔMATOS[1]

Fernando Accorsi
Departamento de Engenharia da Computação - Centro Politécnico
UNOPAR – Universidade Norte do Paraná
E-mail: astacus@sercomtel.com.br

Paulo F. Blauth Menezes
Departamento de Informática Teórica - Instituto de Informática
Universidade Federal do Rio Grande do Sul
E-mail: blauth@inf.ufrgs.br

Resumo

Este artigo descreve um modelo baseado na teoria de autômatos para representar animações gráficas. O modelo proposto, AGA (Animação Gráfica baseada em Autômatos), estrutura o conteúdo da animação em autômatos que descrevem o comportamento de atores durante a animação. As características do AGA viabilizam a utilização de linguagens de consulta para recuperação de informação, ampliam as possibilidades de reutilização de partes da animação, assim como contribuem para a redução do espaço de armazenamento.

Palavras-Chave: animação gráfica, autômato com saída, recuperação de informação.

Abstract

This article presents a model to represent graphic animation from automata theory. The model called AGA (Graphic Animation based on Automata) organizes the content of animation from automata. These automata describe the behavior of actors in animation time. The features of AGA provide the utilization of query language for information retrieval, increase the animation reuse and contribute for reducing the storage space.

Key-Words: graphic animation, automata with output, information retrieval.

1. Introdução

O desenvolvimento tecnológico vem possibilitando a utilização cada vez maior de conteúdos multimídia no ambiente digital. Os limites como: espaço de armazenamento, poder de processamento e largura de banda de transmissão vem sendo ampliados a todo momento. Porém são restrições que acompanham o desenvolvimento de projetos multimídia. Um recurso como a apresentação de animações gráficas demanda grande espaço de armazenamento, poder de processamento e se estiver sendo

[1] Este trabalho é parcialmente suportado por: FAPERGS (Projeto QaP-For), CNPq (Projeto HoVer-CAM, GRAPHIT, MEFIA (CNPq/NSF)) e CAPES (Projeto TEIA) no Brasil.

transmitida pela Internet por exemplo, uma satisfatória largura de banda. Várias soluções vêm sendo criadas para animações gráficas através de representações que visam utilizar de maneira eficiente os recursos disponíveis, tanto formatos de arquivos baseados em quadros como: *GIF* (*Graphics Interchange Format*) (ver [HEL96]), *AVI* (*Audio Video Interleave*) (ver [AVI00]) e *MPEG* (*Moving Picture Expert Group*) (ver [MPE00]), quanto formatos baseados em conteúdo como por exemplo, o *Flash,* desenvolvido pela Macromidea (ver [MAC99]) (classificação proposta por [LEE98]).

As informações disponibilizadas na forma de animações gráficas necessitam muitas vezes de alguma estrutura, pois buscas podem ser necessárias assim como visualizações parciais do conteúdo. Outra característica importante em animações é a possibilidade de reutilização, ou seja, reutilizar uma seqüência de quadros ou de partes específicas da animação para compor outras.

Este artigo define um modelo para animação gráfica baseado na teoria de autômatos. O modelo proposto *AGA* (Animação Gráfica baseada em Autômatos) busca através da utilização de máquinas seqüências eliminar redundâncias de representação da animação, assim como estruturar seu conteúdo de tal forma a proporcionar a recuperação de informação, visualização parcial do conteúdo e reutilização. O AGA é demonstrado neste artigo para modelar animações gráficas 2D sem áudio. Na seção 2 são apresentadas as características gerais do *AGA* como também a teoria de autômatos e extensões utilizadas para defini-lo. Um exemplo é mostrado na seção 2.3 a fim de ilustrar as possibilidades de utilização do modelo. Na seção 3 são discutidas possibilidades de utilização e trabalhos futuros potenciais.

2. Visão Geral do AGA

Os formatos de armazenamento de animação gráfica como o *GIF*, *AVI* e *MPEG* organizam a animação em quadros. Estas representações buscam através de algoritmos de compressão de dados eliminar redundâncias internas a um quadro e entre os quadros. O *GIF*, por exemplo, utiliza o algoritmo *LZW* (ver [HEL96]) para comprimir o quadro e apenas armazena as diferenças do quadro anterior. Como os modelos citados acima têm o objetivo de suportar vídeos, onde a forma de aquisição é linear, a organização do documento é disposta como uma coleção linear de quadros. Desta maneira não há uma estruturação do conteúdo dentro do quadro, nem registro de eventos ao longo da animação. O *Flash*, por outro lado, organiza a animação em unidades que têm um comportamento dinâmico durante a animação. A coleção destas unidades se desenvolvendo no tempo juntas formam a animação gráfica.

O modelo proposto *AGA* organiza a animação como um conjunto de atores. Cada ator é modelado como um autômato finito determinístico com saída (ver [HOP79]), no qual cada estado está vinculado a uma imagem de saída. O comportamento dinâmico do ator durante a animação é decorrente dos símbolos contidos na fita de entrada, quando um símbolo é lido, uma transição para um outro estado é provocada e uma nova imagem do ator é mostrada. A animação é estruturada em camadas que unificadas formam uma imagem resultante que a descreve naquele instante. Cada camada é coordenada por um autômato com fita independente. A Figura 1 ilustra um exemplo onde há 3 atores, onde o primeiro descreve o comportamento da ave e atua na primeira camada. Estas máquinas seqüenciais (termo utilizado em [SAL69]) trabalhando juntas ao longo do tempo compõem a animação gráfica.

A organização do *AGA* proporciona a possibilidade de compressão interna de cada imagem do ator assim como elimina a redundância de armazenamento de imagens. Por exemplo, a ave descrita na Figura 1 possui apenas três imagens distintas associadas aos estados do autômato. Estas imagens podem ser emitidas em diferentes combinações

com os outros atores e com a ordem determinada pelos símbolos da fita. O *AGA* possibilita também a documentação dos estados, viabilizando a identificação de quando um determinado ator está em um estado específico. Outra característica relevante do *AGA* é a possibilidade de reutilização dos atores, pois uma vez modelados podem participar de várias animações apenas alterando os símbolos da fita e os agrupando com outros.

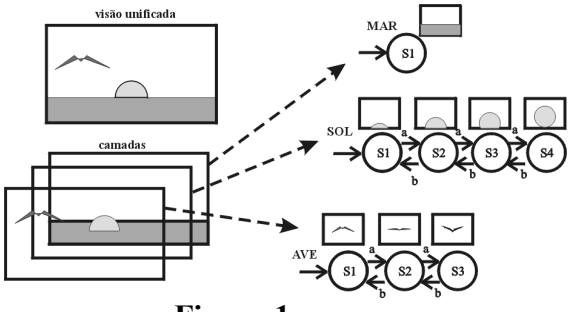


Figura 1

2.1 Autômato Finito com Saída

O autômato com saída utilizado como base para modelar o ator no *AGA*, é a máquina de Moore, proposta por E.F. Moore, e pode ser vista em [MEN00], [SAL69] e [HOP79]. A máquina de Moore é representada em [HOP79] como uma 6-upla: $M = (Q, \Sigma , \Delta, \delta, \lambda, q0)$, onde: Q é o conjunto de estados, Σ o alfabeto de entrada, Δ o alfabeto de saída, δ função de transição (função parcial $\delta: Q x \Sigma \rightarrow Q$), λ função de saída (função total $\lambda: Q \rightarrow \Delta^*$) e $q0$ estado inicial. A máquina seqüencial de Moore associa cada estado a uma palavra sobre o alfabeto de saída, e quando o estado é alcançado, esta palavra é concatenada a uma cadeia de saída. O ator do *AGA* é definido sobre esta formalização e algumas extensões são propostas para isto.

2.1.1 Imagens como Alfabeto de Saída

A primeira extensão é realizada quanto ao alfabeto e a função de saída. A componente Δ, para o ator AGA, é um conjunto de imagens, onde cada elemento é uma imagem estática. O contradomínio da função λ é o conjunto Δ^*, que denota o conjunto de todas as imagens que podem ser produzidas pela composição de várias imagens de Δ. Desta maneira cada estado está vinculado a uma imagem produzida pela composição de imagens de Δ. O funcionamento da máquina pode ser visto como um selecionador de imagem, onde não há uma concatenação de imagens na saída, mas sim a projeção de uma única imagem associada ao estado corrente naquele instante. A Figura 2 descreve um ator com apenas três estados e a cada um deles é associada uma imagem formada por um único elemento de Δ.

A animação do ator é decorrente da fita de entrada, que dependendo dos símbolos contidos nela, produz um padrão de comportamento ao longo da animação. A Figura 2 mostra a seqüência de imagens produzidas quando colocada a palavra 1111112 na fita de entrada.

Figura 2

O alfabeto de entrada Σ é composto por dois símbolos no exemplo da Figura 2, pois o maior grau de saída nos nós do grafo orientado é 2. Duas características

importantes podem ser evidenciadas a partir do exemplo. A primeira é a manutenção da animação, se fosse necessário alterar o comportamento do ator ao longo da animação bastaria alterar os símbolos da fita de entrada, como também poderia ser ampliada aumentando a quantidade destes. A segunda característica é o espaço de armazenamento, apenas são armazenadas as imagens distintas, o autômato e a cadeia de símbolos da fita de entrada.

2.1.2 Função de Transição Temporal

A animação pode ser vista como a projeção de várias imagens estáticas no tempo. O tempo de cada projeção pode variar de acordo com o efeito desejado. Devido a esta característica outra extensão é proposta para o autômato finito, o tempo de transição entre os estados. Este tempo não pode ser associado aos arcos da função de transição de forma estática, pois assim limitaria as possibilidades do modelo. Portanto uma extensão é proposta para a fita de entrada e para a função de transição. Cada célula da fita passa a conter o par ordenado (α,β), onde $\alpha \in \Sigma \cup \{\varepsilon\}$ e β é o tempo em centésimos de segundo que pondera o arco na função de transição. Desta maneira quando o par é lido, é esperado um tempo β para a aplicação de $\delta(q, \alpha)$. A função de transição é denotada como $\delta^t(q,\alpha,\beta)$ e sua definição estendida para $\delta^t: Qx(\Sigma\cup\{\varepsilon\})xN \rightarrow Q$, onde N é conjunto dos naturais. Embora transições espontâneas não sejam permitidas no ator, o símbolo ε é inserido para indicar a permanência no mesmo estado, isto evita a criação de uma transição para indicar o laço no estado. Logo $\delta^t(q,\varepsilon,\beta) = q$ para todo $q \in Q$.

2.1.3 Função de Saída Contextual

Outra característica comum encontrada em animações são as transformações aplicadas a uma mesma imagem em contextos diferentes, como por exemplo: rotação, translação, escalonamento, rebatimento entre outras. Para comportar esta flexibilidade a célula da fita é ampliada para o triplo ordenado (α,β,φ), onde $\varphi \in F^*$ e F é o conjunto de todas as funções que transformam o conjunto Δ*. A função denotada π é introduzida e definida como $\pi(i) = f_1(f_k(\ldots f_{|\varphi|}(i)))$ tal que f_k é o k-ésimo símbolo de φ ($1 <= k <= |\varphi|$) *e* $i \in \Delta^*$. Portanto a função de saída é estendida para $\lambda^c(q,\varphi)$ e é definida como $\lambda^c: QxF^* \rightarrow \Delta^{*F}$, onde Δ^{*F} é o conjunto de todas imagens produzidas pela aplicação de zero ou mais funções de F. Logo $\lambda^c(q,\varphi) = \pi(\lambda(q))$.

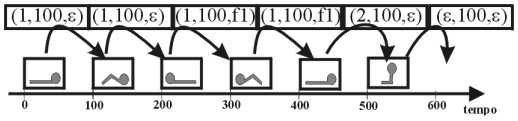


Figura 3

A Figura 3 ilustra a aplicação da função *f1* para provocar o espelho da imagem de saída. Note que quando a célula da fita (1,100,f1) é lida ocorre a aplicação da função $\lambda^c(s1,f1)$ e depois $\delta^t(s1,1,100)$.

2.1.4 Função Descrição

Com o objetivo de documentar os estados de forma semântica é introduzida a função descrição, denotada por σ, e definida como a função parcial $\sigma: Q \rightarrow D$, onde D é o conjunto de descrições dos estados e pode ser vazio. Desta maneira é possível recuperar o momento exato que um ator ocupa um determinado estado na animação. Esta recuperação é realizada pela análise da fita a procura de um prefixo da palavra de entrada que leve ao estado cuja descrição é procurada. Por exemplo, para o autômato da Figura 2, a função σ poderia ser definida como: $\sigma(s1)$=*estendida,* $\sigma(s2)$=*contraída e*

σ *(s3)=alerta.* Com a descrição agregada aos estados é possível realizar uma pesquisa do tipo: *encontre ATOR em alerta*. Note que se esta pesquisa fosse realizada para a animação descrita pela Figura 3, uma análise seqüencial seria feita na fita, onde os valores de tempo seriam acumulados até encontrar a transição para o estado S3 (500). O mecanismo de busca pode ser explorado para pesquisas complexas onde vários atores estão envolvidos, como é o caso de pesquisas do tipo: *encontre AT1 em alerta E AT2 em estendida*, onde seria necessária a análise das fitas dos atores AT1 e AT2 e a verificação do intervalo de tempo que ocorrem as transições.

2.2 Animação Gráfica como um Conjunto de Atores

Um Ator do AGA é definido como uma 9–upla: AT= (Q, Σ , Δ, δ^t, λ^c, σ , q0, F, D), onde: Q é o conjunto de estados do ator, Σ alfabeto de entrada, Δ conjunto de imagens distintas do ator, δ^t função de transição temporal (função parcial δ^t: Qx($\Sigma \cup \{\varepsilon\}$)xN$\rightarrow$Q), λ^c função de saída contextual (função total λ^c: QxF$^*\rightarrow \Delta^{*F}$), σ função descrição (função parcial σ :Q$\rightarrow$D), q0 estado inicial, F conjunto de funções que transformam Δ^* e D conjunto de descrições dos estados.

A animação gráfica representada pelo AGA é definida como um conjunto totalmente ordenado de pares ordenados (ator, fita de entrada) cuja relação de ordem total, denotada por $\leq^C$, é definida como: $AT_1 \leq^C AT_2$ se e somente se AT_1 é desenhado antes ou ao mesmo tempo que AT_2. Portanto AGA = ({ (a,fita de a) | a é ator }, $\leq^C$) (notação descrita em [LIP 67]). Uma animação em AGA pode ser entendida como um conjunto de máquinas seqüências, funcionando em paralelo, lendo fitas independes cujas saídas são apresentadas em um mesmo plano, obedecendo a relação de ordem total $\leq^C$. A relação de ordem indica a distribuição dos atores nas camadas, ou seja, se $AT_1 \leq^C AT_2$ então o autômato AT_1 atua em uma camada mais interna que AT_2. A imagem produzida por AT_1 é complementada pela sobreposição da imagem de AT_2. A Figura 1 ilustra uma animação onde há três atores: MAR, SOL e AVE e a relação de ordem é MAR $\leq^C$ SOL $\leq^C$ AVE.

2.3 Exemplo

O exemplo proposto modela a animação gráfica da Figura 4 representada pelos quadros de 1 a 17. Animação toda ocorre em 17s e cada quadro representa a animação no segundo corrente. A modelo é criado a partir de três atores: *FUNDO*, *SOL* e *AVE* e são definidos como:

FUNDO = ({s1}, {}, Δ^1, δ_1^t, λ_1^c, σ_1, s1,{}, {dia}), onde Δ^1, δ_1^t, λ_1 são descritos na Figura 4. A imagem mais próxima ao estado indica a associação entre eles. A função σ_1 é definida como: σ_1*(s1)=dia.* A fita para a animação deste ator é descrita como: $Fita_1$= *(ε,1700,{})*;

SOL = ({s1,s2,s3,s4}, {a,b}, Δ^2, δ_2^t, λ_2^c, σ_2, s1,{}, {meio, inteiro}), onde Δ^2, δ_2^t, λ_2 são descritos na Figura 4. A função σ_2 é definida como: σ_2*(s2)=meio,* σ_2*(s4)=inteiro.* A fita definida como: $Fita_2$*=(a,400,{}), (a,500,{}), (a,400,{}), (ε,400,{})*;

AVE = ({s1,s2,s3}, {a,b}, Δ^3, δ_3^t, λ_3^c, σ_3, s1,{f1}, {baixo, cima}), onde Δ^3, δ_3^t, λ_3 são descritos na Figura 4 e f_1*=translação*. A função σ_3 é definida como: σ_3*(s1)=baixo,* σ_3*(s3)=cima.* A fita definida como: $Fita_3$*=(a,100,{}), (a,100,{}), (b,100,{}), (b,100,{}), (a,100,{f1}), (a,100,{f1}), (b,100,{f1}), (ε,700,{f1}), (ε,300,{f1}).*

Portanto a animação é definida como: *A=({ (FUNDO, $Fita_1$), (SOL, $Fita_2$), (AVE, $Fita_3$) }, $\leq^C$),* onde *FUNDO* $\leq^C$ *SOL* $\leq^C$ *AVE* . O modelo acima descreve todo comportamento dos atores na animação e possibilita uma consulta aos estados que ocupam. Por exemplo, para animação da Figura 4 poderia ser realiza a consulta do tipo:

"Encontre FUNDO em dia E SOL em meio E AVE em cima", e retornaria o momento do quadro 7.

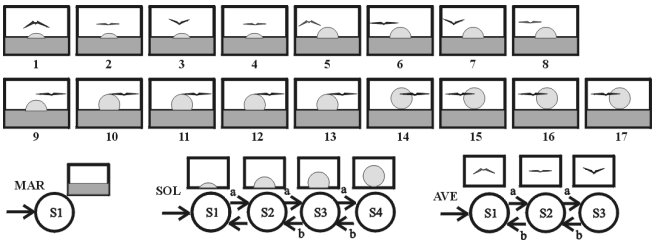


Figura 4

3. Conclusões

A estruturação do conteúdo da animação proposta pela AGA viabiliza tanto a recuperação de informações orientada aos estados dos atores, quanto a visualização parcial das camadas. Devido a estas características é possível a definição de uma linguagem de consulta que proporcione o resgate do conteúdo baseado no comportamento dos atores na animação. O encapsulamento das propriedades estéticas e comportamentais dos elementos da animação em atores amplia a reutilização, como também contribui para a construção de animações mais complexas. Outra característica relevante do modelo é a descrição do comportamento do ator pelos símbolos contidos na fita de entrada, o que viabiliza o envio destas informações sob demanda. Esta propriedade pode ser explorada para animações longas executadas em ambientes de rede por clientes a partir de um servidor. A representação apenas de imagens distintas que caracterizam o comportamento do ator reduzem a necessidade de espaço de armazenamento. Este espaço não aumenta proporcionalmente ao tamanho da animação, como é o caso das animações baseadas em quadros, mas acompanha a complexidade estética e comportamental dos atores.

4. Referências Bibliográficas

[AVI 00] AVI OVERVIEW. **AVI overview**. Disponível por WWW em http://www.jmcgowan.com/avi.html (10 jan. 2000)

[HEL 96] HELD, Gilbert. **Data and Image compression: tools and tecniques.** John Wiley & Sons Ltd, 1996. p311-343.

[HOP 79] HOPCROFT, J. E. e ULLMAN, J.D**. Introduction to Automata Theory, Languages and Computation.** Addison-Wesley, 1969. p.42- 45.

[LEE 98] LEE, G. S. **A Classification of File Formats for Animation**. Disponível por WWW em http://www.cs.ubc.ca /labs/imager/tr/ps/lee.1998.ps.gz. (10 jan. 200)

[MAC 99] MACROMIDEA Inc. **Macromedia Flash**. Disponível por WWW em http:// www.macromedia.com /software/flash/ (20 nov. 1999)

[MEN 00] MENEZES, Paulo Fernando Blauth. **Linguagens Formais e Autômatos**. Porto Alegre: Instituto de Informática da UFGRS: Editora Sagra Luzatto, 2000. Cap 2.

[MPE 00] MPEG ORG. **Mpeg Pointers e Resoucers.** Disponível por WWW em http://www.mpeg.org (10 jan. 2000)

[SAL 69] SALOMAA, Arto. **Theory of Automata**. Pergamon Press, 1969. p31-39.

Anexo 2 Artigo Workshop on Formal Methods 2001

O artigo contido neste anexo foi publicado e apresentado pelo autor no IV Workshop on Formal Methods realizado em outubro de 2001 durante o Simpósio Brasileiro de Engenharia de Software na cidade do Rio de Janeiro - RJ. Este artigo foi aceito na categoria de artigo curto (até 6 páginas) e descreve o modelo AGA-S.

Animação Gráfica Baseada em Autômatos Temporizados Sincronizados[1]

Fernando Accorsi
UNOPAR – Departamento de Engenharia da Computação
E-mail: fernando.accorsi@prof.unopar.br

Paulo F. Blauth Menezes, Luciana Porcher Nedel
UFRGS – Instituto de Informática
E-mail: {blauth, nedel}@inf.ufrgs.br

Resumo

Este artigo descreve um modelo baseado em autômatos temporizados para especificar animações gráficas. O modelo proposto AGA-S estrutura o conteúdo da animação em autômatos temporizados que descrevem o comportamento de elementos cênicos durante a animação. A utilização de autômatos temporizados proporciona a especificação formal do comportamento dinâmico dos elementos cênicos e provê mecanismos de sincronização.

Abstract

This article presents a model based on timed automata for specifying graphic animation. The proposed model, AGA-S, organizes the content of animation from timed automata. These timed automata describe the behavior of scenic elements in animation time. Utilization of timed automata provides formal specification for dynamic behavior of scenic elements and synchronizing mechanisms.

1. Introdução

A utilização de animações gráficas em ambientes digitais vem aumentando a cada dia devido ao panorama favorável obtido pelo desenvolvimento tecnológico. Elas podem ser vistas em sistemas com diferentes propósitos e conteúdos que reúnem informações gráficas, sonoras e textuais.

Desta maneira, vários trabalhos têm sido desenvolvidos com o propósito de criar modelos que especifiquem formalmente estes conteúdos e seus comportamentos durante a animação. Muitos destes modelos são definidos a partir de formalismos clássicos, como por exemplo, redes de Petri [7] e autômatos [6][1], favorecendo a verificação formal de aspectos quantitativos, como o tempo de permanência de determinadas situações na animação, e aspectos qualitativos, como o sincronismo entre os elementos.

Este artigo apresenta o modelo AGA-S (Animação Gráfica baseada em Autômatos Temporizados Sincronizados) que tem como objetivo especificar os componentes da animação gráfica e coordená-los durante o processo de animação possibilitando sincronização entre seus elementos. Na seção 2 são apresentados trabalhos relacionados à estrutura proposta pelo AGA-S, em especial, o modelo AGA proposto pelos autores em [1]. Na seção 3, o modelo é descrito e formalizado a partir da

[1] Este trabalho é parcialmente suportado por: FAPERGS (Projeto QaP-For), CNPq (Projeto HoVer-CAM, GRAPHIT, MEFIA (CNPq/NSF)) e CAPES (Projeto TEIA).

extensão dos autômatos temporizados. A sincronização utilizada no modelo é ilustrada na seção 3.3 através do produto de autômatos temporizados. Na seção 4 são apresentados as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Em [6] é apresentado um modelo baseado em autômatos temporizados para construção de animações gráficas direcionadas à visualização dos comportamentos de sistemas. Essas animações são limitadas a expressar esquematicamente o comportamento do sistema modelado. Uma abordagem semelhante é utilizada em [5] para a ilustração de sistemas especificados com autômatos temporizados.

A utilização de autômatos com saída para coordenar animações gráficas é apresentada em [1] pelos autores deste artigo. Nessa abordagem é definido o modelo AGA (Animação Gráfica baseada em Autômatos) a partir de extensões propostas para as máquinas de Moore [4]. Esse modelo tem como alfabeto de saída apenas imagens estáticas que são selecionadas durante a animação. Os autômatos utilizam fitas independentes e não possuem mecanismos explícitos de sincronização nem restrições temporais. O modelo proposto AGA-S pode ser visto como uma extensão do modelo AGA com o objetivo de modelar animações gráficas mais complexas que necessitam de sincronização entre os elementos cênicos.

3. Modelo AGA-S

O modelo AGA-S especifica a animação gráfica como um conjunto de elementos cênicos coordenados por autômatos temporizados. Os elementos cênicos são partes estruturais da animação, separados com o propósito de terem seus comportamentos modelados. Essas partes podem ser personagens com características gráficas, ou mesmo estruturas não gráficas, como por exemplo, efeitos sonoros e trilhas musicais.

O comportamento do elemento cênico é modelado através de um autômato temporizado que coordena suas ações durante a animação. Esse autômato possui restrições temporais que determinam as características dinâmicas do elemento. A especificação dessas restrições é utilizada tanto na execução da animação, para seleção do comportamento do elemento, quanto no seu processo de composição, para estabelecer sincronização entre os elementos cênicos.

As características estéticas dos elementos são vinculadas às transições do autômato, ou seja, de forma semelhante à máquina de Mealy[4], quando uma transição é realizada uma imagem é emitida ou uma música é iniciada. Essas características podem ser submetidas a transformações durante a animação através de funções descritas no momento da transição, mudando por exemplo, cor, posição, volume, etc.

Os estados de cada autômato são documentados com informações relativas ao comportamento do elemento cênico. Essas informações subsidiam mecanismos de consulta para identificar eventos durante a animação, os quais são revelados através da avaliação dos estados correntes dos autômatos e suas informações correspondentes.

A execução de uma animação em AGA-S pode ser vista como um conjunto de autômatos temporizados com saída trabalhando em paralelo e obtendo informações de uma única fita. Nesta fita, estão os símbolos de entrada que provocam as transições, os momentos em que devem ocorrer e as funções de transformações dos elementos cênicos.

3.1 Autômato Temporizado

O modelo de autômato temporizado utilizado no AGA-S é proposto por Alur e Dill em [2]. Esse modelo pode ser visto como um sistema de transição, o qual possui um grafo de transição finito com restrições de temporização associadas a suas arestas e estados.

As restrições são definidas a partir da verificação de um conjunto finito de relógios, os quais armazenam valores reais que determinam o tempo decorrido desde a sua última inicialização. Este incremento dos valores dos relógios é realizado de acordo com uma referência global de tempo. As transições entre os estados são consideradas instantâneas e o tempo passa durante a ocupação de um estado. Uma transição só pode ser realizada se avaliação do relógio satisfazer a restrição da aresta. Da mesma maneira, um estado pode ser ocupado apenas pelo tempo tolerado pela restrição associada ao estado, também chamada de restrição invariável.

Um autômato temporizado *At* é uma tupla $(Q, Q_0, \Sigma, X, I, E)$, onde: Q é um conjunto finito de estados; Q_0 é um conjunto de estados iniciais ($Q_0 \subseteq Q$); Σ é um alfabeto de símbolos de entrada; X é um conjunto finito de relógios; I é um mapeamento de designa cada estado s em Q com alguma restrição de relógio em $\Phi(X)$; E é um conjunto de transições $(E \subseteq Q \times \Sigma \times 2^X \times \Phi(X) \times Q)$.

Uma transição $(s, a, \varphi, \chi, s')$ representa uma transição do estado s para s' mediante o símbolo a. A restrição de relógio φ determina quando a transição está habilitada, enquanto o conjunto $\chi \subseteq X$ determina os relógios a serem inicializados na transição.

O conjunto de restrições, denotado por $\Phi(X)$, possui as restrições de relógio φ, definidas pela gramática $\varphi := x \leq c \mid c \leq x \mid x<c \mid c < x \mid \varphi_1 \wedge \varphi_1$, onde x é um relógio em X e c é uma constante pertencente aos racionais.

3.2 Elemento Cênico AGA-S

O elemento cênico é a unidade básica em uma animação AGA-S, seu objetivo é descrever as características estéticas e comportamentais de cada parte estrutural da animação. Esse elemento é definido a partir da extensão do autômato temporizado para prover saídas vinculadas às transições e a documentações vinculadas aos estado.

Um elemento cênico AGA-S é uma tupla $A = (At , \Delta, F, \lambda, D, \sigma)$, onde : *At* é um autômato temporizado; Δ alfabeto de saída; F conjunto de funções de transformação de Δ; λ função de saída contextual; D conjunto de descrições dos estados; σ função que mapeia a documentação dos estados.

O autômato *At* descreve o comportamento do elemento cênico através dos estados que pode ocupar e as transições entre eles. As restrições temporais dos relógios definem o comportamento dinâmico quanto ao tempo de permanência nos estados e os instantes disponíveis para transição. A figura 1, por exemplo, ilustra um elemento responsável em coordenar uma personagem cujo comportamento tem quatro estados, as transições indicam que este apenas pode *apontar* (*p2*) ou *apagar* (*p3*) se estiver previamente *parado* (*p1*). A utilização do relógio *y* restringe o tempo máximo e mínimo que o elemento pode ficar no estado *p3*, neste caso, pode permanecer no máximo até 7 segundos e no mínimo 5.

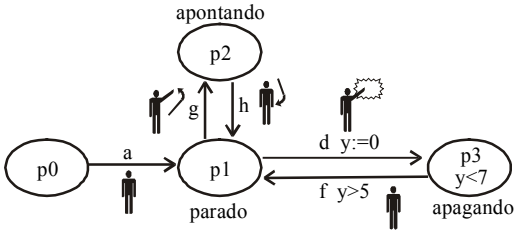


FIGURA 1 – Elemento Cênico AGA-S

O alfabeto de saída Δ é um conjunto finito de objetos para apresentação. Esses objetos podem ser imagens estáticas, vídeos, seqüências sonoras, ou outros objetos pertinentes à animação. Desta maneira, quando uma transição é realizada, o objeto vinculado a esta aresta é apresentado naquele momento na animação. Caso seja um objeto que tenha tempo de duração como um vídeo ou uma seqüência sonora, a transição indica o início de sua execução. O elemento cênico da figura 1, por exemplo, dispara seqüências de animações que produzem o efeito da personagem apontar, apagar um quadro ou ficar parado. Desta forma, quando o autômato ocupa o estado *p1* e um símbolo *g* é lido, a personagem realiza o movimento de apontar.

Em animações é comum o mesmo elemento ser apresentado com características diferentes em vários momentos durante a execução. Uma imagem, por exemplo, pode ser apresentada em posições diferentes, ou mesmo uma música pode ser iniciada com diversos níveis de volume. Com o objetivo de prover estas transformações, o elemento cênico possui o conjunto de funções de transformação *F*, estas funções são do tipo $f(\vec{a}, e) = e^t$, onde $\vec{a} = a_1 a_2 ... a_n$ é um vetor de argumentos, *e* e e^t são elementos de Δ^F, conjunto de elementos transformados por zero ou mais funções de *F*. A indicação de qual função e argumentos utilizar no momento da transição é especificada na fita de entrada. Desta maneira, a função de saída contextual λ é definida por $\lambda: Q \times \Sigma \times F^* \rightarrow \Delta^F$, ou seja, a partir de um estado, do símbolo da fita e uma palavra que descreve as transformações, apresenta o objeto vinculado à transição transformado pelas funções. Para o elemento cênico da figura 1, por exemplo, poderia ser definida a função *sentido*(a_1), onde a_1 representa o sentido da imagem do elemento, à direita ou à esquerda, desta forma a entrada *(p1, g, sentido(E))* na fita provocaria a inversão das imagens emitindo a personagem apontando à esquerda.

O conjunto de descrições dos estados *D* e a função $\sigma(\sigma: Q \rightarrow D)$ compõem uma forma de documentação dos elementos cênicos, onde há o mapeamento dos estados para descrições contidas em *D*. O objetivo principal das descrições é subsidiar mecanismos de consulta para a animação, pois a partir da função σ é possível recuperar as informações do estado do elemento cênico. Por exemplo, na figura 1, os estados *p1*, *p2* e *p3* estão vinculados às descrições: *parado*, *apontando* e *apagando*, respectivamente, o que facilitaria um consulta na fita a procura do momento em que o elemento cênico ocupe a tarefa de apontar.

3.3 Animação em AGA-S com Elementos Cênicos Sincronizados

O modelo proposto AGA-S proporciona uma maneira de especificar de forma independente o comportamento e as características estéticas de unidades da animação, e depois agrupá-las compondo uma seqüência animada através da especificação da fita. Esta característica favorece o reaproveitamento dos elementos cênicos em outras animações, como também colabora com a manutenção da seqüência animada por trocas de símbolos na fita de entrada.

A execução da animação consiste na avaliação da fita pelo conjunto de elementos cênicos participantes em paralelo. A cada entrada *(a, t, f)* na fita, esses elementos verificam se o símbolo *a* é pertinente a seus alfabetos e realizam a transição no momento determinado *t* utilizando as funções descritas em *f*. Caso o símbolo não pertença ao alfabeto de entrada de um elemento cênico, nenhuma ação é desencadeada, porém, se pertencer ao alfabeto e a transição for indefinida, a fita é considerada inconsistente. A verificação da inconsistência não se limita aos símbolos, mas também considera as determinações dos tempos, os quais devem respeitar as restrições dos relógios descritas nos elementos cênicos.

A sincronização dos elementos é obtida pelos símbolos comuns aos alfabetos de dois ou mais autômatos. A ocorrência de um símbolo comum aos autômatos provoca transições de estados sincronicamente nos elementos cênicos. Como a função de saída está vinculada a transição nos autômatos, a sincronização pelo símbolo determina que dois ou mais elementos iniciem determinadas tarefas ao mesmo tempo, por exemplo, um vídeo e uma música disparados simultaneamente.

A sincronização dos elementos cênicos é definida formalmente a partir do produto de autômatos temporizados proposto em [3] e estendido para o modelo AGA-S. O produto de um elemento $A_1 = ((Q_1, Q_{01}, \Sigma_1, X_1, I_1, E_1), \Delta_1, F_1, \lambda_1, D_1, \sigma_1)$ com $A_2 = ((Q_2, Q_{02}, \Sigma_2, X_2, I_2, E_2), \Delta_2, F_2, \lambda_2, D_2, \sigma_2)$, denotado por $A_1||A_2$, é o elemento $A = ((Q_1 \times Q_2, Q_{01} \times Q_{02}, \Sigma_1 \cup \Sigma_2, X_1 \cup X_2, I, E), \Delta_1 \cup \Delta_2, F_1 \cup F_2, \lambda, D_1 \cup D_2, \sigma)$, onde $I(s1, s2)=I(s1)\wedge I(s2)$ e $\sigma(s1, s2)=\sigma(s1)\cup\sigma(s2)$. Assume-se que $X_1 \cap X_2 = \varnothing$. As transições E e a função λ são definidas por:

a) para $a \in \Sigma_1-\Sigma_2$, para todo (s,a,φ, χ, s') em E_1 e todo t em Q_2, E contém $((s,t), a, \varphi, \chi, (s',t))$ e $\lambda((s,t), a, f)=\lambda_1(s, a, f)$;
b) para $a \in \Sigma_2-\Sigma_1$, para todo (s,a,φ, χ, s') em E_2 e todo t em Q_1, E contém $((t,s), a, \varphi, \chi, (t,s'))$ e $\lambda((t,s), a, f)=\lambda_2(s, a, f)$;
c) para $a \in \Sigma_1 \cap \Sigma_2$, para todo $(s_1, a, \varphi_1, \chi_1, s'_1)$ em E_1 e $(s_2, a, \varphi_2, \chi_2, s'_2)$ em E_2, E contém $((s_1,s_2), a, \varphi_1\wedge\varphi_2, \chi_1\cup\chi_2, (s'_1, s'_2))$ e $\lambda((s,t), a, f) = \lambda_1 (s, a, f_1)\cup\lambda_2(t, a, f_2)$ onde $f=f_1f_2$.

O elemento cênico A resultante de $A_1//A_2$ pode produzir estados inalcançáveis tanto pela falta de caminhos até estes a partir dos estados iniciais, como também pela conjunção das restrições temporais causando limites impossíveis de serem satisfeitos. Desta maneira estes estados podem ser eliminados sem prejuízo a composição dos elementos.

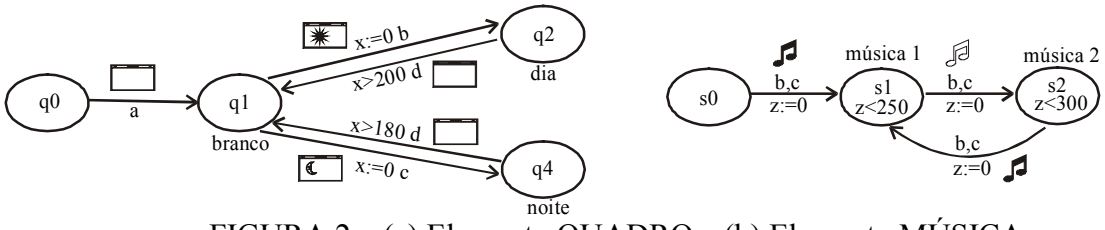


FIGURA 2 – (a) Elemento QUADRO e (b) Elemento MÚSICA

A figura 2 ilustra dois elementos cênicos, QUADRO e MÚSICA, o primeiro modela o comportamento de um quadro explicativo que possui informações sobre atividades diurnas e noturnas, e o segundo modela o comportamento de uma trilha sonora composta por duas músicas. Os símbolos *b* e *c* sincronizam os dois elementos para que as músicas sejam revezadas sempre que uma explicação diferente no quadro seja solicitada, como também o espaço de tempo entre as escolhas não demore mais que a duração das músicas.

A figura 3 mostra o produto entre os dois elementos cênicos com os estados não alcançáveis já excluídos. A sincronização dos elementos determina, por exemplo, que o quadro pode estar com as explicações sobre o dia e a trilha sonora na música 1 apenas durante um tempo de no máximo 250 segundos e no mínimo 200 (estado *q2s1*).

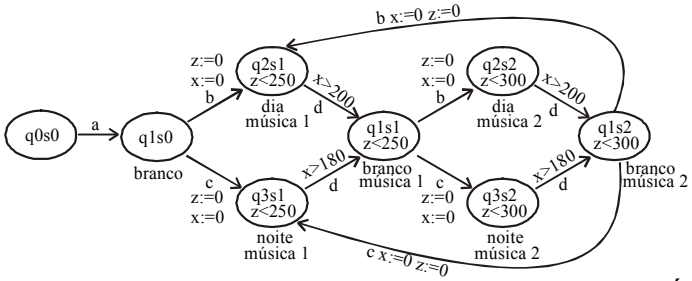


FIGURA 3 – Produto entre os elementos QUADRO e MÚSICA

4. Conclusão e Trabalhos Futuros

Embora o modelo AGA-S defina o elemento cênico a partir da extensão de autômatos temporizados com saída nas transições, diferentemente do proposto pelo AGA em [1], ainda assim preserva os mesmos benefícios apresentados pelo modelo, como por exemplo, o favorecimento de mecanismos de consulta baseados nos estados dos elementos cênicos e o encapsulamento das propriedades estéticas e comportamentais em unidades independentes.

A utilização de autômatos temporizados proporciona a especificação do comportamento dinâmico dos elementos cênicos e provê mecanismos de sincronização. Desta maneira, o AGA-S possibilita verificações formais de aspectos quantitativos e qualitativos da animação gráfica.

A definição do AGA-S como coordenador da animação gráfica mediante a leitura de uma fita seqüencial, proporciona ao modelo uma implementação bastante oportuna para aplicações direcionadas à Internet.

Baseado no AGA-S está sendo desenvolvido um protótipo em JAVA para execução de animações gráficas em páginas WEB. Como trabalho futuro, pretende-se estender o modelo AGA-S para modelar a interatividade entre a animação e o observador.

5. Referências Bibliográficas

[1] ACCORSI, Fernando, MENEZES, Paulo Fernando Blauth. *Animação Gráfica Baseada na Teoria de Autômatos.* In: WMF - WORKSHOP ON FORMAL METHODS, 2000, João Pessoa, Paraíba. WMF'2000 - 3rd Workshop on Formal Methods. João Pessoa: SBC, 2000. p. 122 – 127.

[2] ALUR, Rajeev, DILL, David L. A theory of timed automata. *Theoretical Computer Science*, v.126, 1994. p. 183-235.

[3] ALUR, Rajeev. Timed automata. *Lectures Notes in Computer Science*. Computer Aided Verification. Springer Verlag, v.1633, 1999. p.8-22. Trabalho apresentado no 11th International Conference, CAV'99, 1999, Trento, Itália.

[4] HOPCROFT, J. E., ULLMAN, J.D**.** *Introduction to Automata Theory, Languages and Computation***.** Addison-Wesley, 1969. p.42- 45.

[5] LARSEN, K. G., PETTERSSON, P., YI, W. UPPALL in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, v.1, n.1, out. 1997. p.134-152.

[6] MAGEE, J. et al. Graphical Animation of Behavior Models. INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING. *Proceedings…* 2000. p. 499-508.

[7] PORTO, A. M. C., COSTA, R.C. Especificando formalmente o comportamento de apresentações multimídia interativas. *Revista Alcance*. Editora da UNIVALI, ano VIII, n.1, 2001. p.31-37.

Anexo 3 Artigo Submetido a IEEE Transactions on Information Theory

O artigo contido neste anexo foi submetido à revista IEEE Transactions on Information Theory. O autor desta dissertação contribui no artigo com as descrições referentes às características do modelo AGA e as informações quantitativas reais utilizadas para ilustrar os resultados teóricos.

Comparing Data Compression in Web-based Animation Models using Kolmogorov Complexity

Carlos A. P. Campani, Fernando Accorsi, Paulo Blauth Menezes, and Luciana Porcher Nedel

***Abstract*— In the last few years, the internet use is growing and web pages are being populated by multimedia objects. At the same time, traffic over the internet is growing and, consequently, the importance of data compression. This paper presents formal definitions of "compression algorithm", "better compression algorithm", "best compression algorithm", "animation", "better animation" and a methodology based on Kolmogorov complexity for comparing computer animation models. The comparison is related to data compression. Kolmogorov complexity is defined as the length of the smallest effective description of a binary string and it is a suitable quantitative measure of the string information contents. The method is theoretical and more qualitative than empirical and quantitative. We show an example of application, comparing GIF-based animations with AGA-based animations. This paper focuses on bitmap-oriented animation and data compression without loss, although the method may be applied to the general case.**

I. Introduction

The main goal of data compression is to minimize the amount of data to be transmitted over a communication channel or to be stored in data files. It is usual to refer data compression as *coding* [8], [16].

It belongs to a more general field called *information theory* [8], [12], which is related to communication theory, probability theory and statistics.

Data compression is becoming an important subject since the internet use grows and web pages are becoming enriched by multimedia objects. Image, video, sound and animation demand large file transfers and good use of the band-width to reduce the cost of transmission. For this purpose, multimedia formats have built-in data compression.

This paper introduces a methodology for comparing, and in some sense evaluating, computer animation models based on a formal definition of data compression. It is important to note that, in this paper, we do not cover data compression with loss.

The main results shown in this paper are:

- Formal definitions of "compression algorithm", "better compression algorithm" and "best compression algorithm" based on Kolmogorov complexity;
- There are better compression algorithms (better than a set of given compression algorithms);
- We can not reach the best compression ratio (by reduction to the noncomputability of Kolmogorov complexity);
- More time spent on compression means a higher compression ratio (compression may be approximated from above by a recursive function);
- Formal definitions of "animation" and "better animation" (only bitmap-oriented animations);
- Applying a methodology for comparing animation models, we show that AGA-based animations are better than GIF-based animations ("better" means that AGA is more efficient to compress data than GIF).

AGA (Automata-based Graphical Animation) is an animation model based on automata theory and aims to provide reuse, reduced storage space and the use of a query language to allow information retrieval [1] .

Kolmogorov complexity is based on the theory of computation [2]. It is defined as the length of the smallest effective description of a binary string and defines a new approach to information theory, called *algorithmic information theory* [6], [17].

Sections II, III and IV briefly introduce some basic definitions and well known results about data compression, classical information theory, Kolmogorov complexity and randomness.

In Section II, we show informally why data can be compressed, and what is the relation between entropy and data compression. We also present the definition of compression ratio. In Section III, we define Kolmogorov complexity. Theorem 1 proves that the Kolmogorov complexity of a string is invariably related to the universal Turing machine used.

Later, we introduce prefix complexity, which is a more robust definition of complexity. We argue that entropy and prefix complexity are formally equivalent, thus, identifying Kolmogorov complexity with the best code (maximum data compression). Prefix coding is also introduced, because it is used in the proof of Theorem 9 for delimiting data blocks.

Section IV shows, using Kolmogorov complexity, why regular binary strings can be compressed and what is the relation between compressibility and randomness.

Theorem 5 is a well known result presented in [21]. Section VI-A briefly presents GIF protocol and Lempel-Ziv compression algorithm.

The remaining sections are the authors' main contribution, specially Sections V-C, VI-B, VII and Theorem 9.

In Section V, we formalize data compression based on Kolmogorov complexity. Moreover, we show that the problem of achieving the best coding is an open problem without exact solution, which can only be approximated.

In Section VII, we formalize "animation" and "better animation", which are related to data compression. We define "animation" as a machine and the proofs are related to machine

This work is partially supported by: CNPq (Projects HoVer-CAM, MEFIA), UFRGS (Project Hyper-Automaton), FAPERGS (Project QaP-For) and CAPES/UFPel in Brazil. (e-mails: {campani, blauth, nedel}@inf.ufrgs.br, fernando.accorsi@prof.unopar.br).

Message	Codeword
a_1	00
a_2	01
a_3	10
a_4	11

TABLE I
ENUMERATIVE CODE

Message	Codeword
a_1	0
a_2	10
a_3	110
a_4	111

TABLE II
PREFIX CODE

simulation.

Finally, Theorem 9 is the main result of the paper. It is a proof that AGA is better than GIF. Section IX shows empirical data from some tests performed to compare AGA and GIF. These empirical data do not have the purpose to explore in depth AGA and GIF file sizes, but to show some data that only illustrate the theoretical results. Section X shows a further result, exploring the possibilities of the method.

II. PRELIMINARY DEFINITIONS

We will be concerned with a communication channel linking two communication devices, a *transmitter* and a *receiver* [8], [19]. Suppose you want to communicate to the receiver an information about a sequence of results of an experiment. The set of outcomes is called *set of messages*. We will suppose also that the channel is noiseless, that is, it is error free.

For example, let $\alpha = \{a_1, a_2, a_3, a_4\}$ be a set of equiprobable messages, with $\Pr(a_i) = 1/4$, for $1 \leq i \leq 4$. It is known that we can communicate such messages using two bits, coding the messages as shown in Table I.

But, if the set of messages has the following probability distribution, $\Pr(a_1) = 1/2$, $\Pr(a_2) = 1/4$, $\Pr(a_3) = 1/8$ and $\Pr(a_4) = 1/8$, we can define a more efficient code, reserving shorter codewords for more frequent messages. We do it using the prefix code shown in Table II, with the average codeword length (asymptotically)

$$L = \sum_{i=1}^{\overline{\overline{\alpha}}} |E(a_i)| \Pr(a_i) = \frac{1}{2} + 2\frac{1}{4} + 3\frac{1}{8} + 3\frac{1}{8} = 1.75\,,$$

where $E(x)$ denotes the codeword of x, $\overline{\overline{\alpha}}$ denotes the diameter of the set α and $|.|$ means the length of a binary string. As we can see, the new code needs fewer bits to communicate messages. It means that the prefix code, exploring regularities in the source, is more efficient than the enumerative code, achieving some degree of data compression.

Let $\{0,1\}^*$ be the set of binary strings with length greater than or equal to zero, and let $\{0,1\}^+$ be the set of binary strings with length greater than zero.

Definition 1: A *binary code* $E : \alpha \to \{0,1\}^+$ is a mapping from the set α of messages to the set of binary strings called *codewords*.

A code is *uniquely decodable* if we can decode any codeword without ambiguity when it is immersed in a sequence of codewords. A code is *instantaneously decodable* if we can decode any codeword without lookahead.

Definition 2: Let $x, y \in \{0,1\}^*$ be two binary strings. We call x *prefix* of y if exists z in such a way that $y = xz$, where xz denotes the concatenation of x and z.

Definition 3: A set $\beta \subseteq \{0,1\}^+$ is *prefix free* if there is not $x, y \in \beta$ with x prefix of y.

Definition 4: A binary code is prefix free if it generates a set of prefix free codewords.

A prefix free code is uniquely decodable and instantaneously decodable.

The problem involved on the compression and decompression of data files is similar to the above mentioned encoding and decoding problem of data communication, and the same formalisms are applied to both.

Coding is related to the concept of *entropy* as a measure of disorder [8], [12], [19].

Definition 5: (Shannon's Entropy) Suppose a discrete probability distribution P,

$$P = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_n \\ p_1 & p_2 & p_3 & \cdots & p_n \end{pmatrix},$$

where p_i, for $1 \leq i \leq n$, are probabilities, each p_i is the probability of the occurrence of the outcome a_i, with $\sum_{i=1}^{n} p_i = 1$. *Entropy* is defined as

$$H(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i\,,$$

where log represents the base two logarithm.

A "good" code is the optimal one, that is, the one that has minimum redundancy [16]. Redundancy is formally defined as $L - H$. An interesting result shows that entropy coincides with L_b, the average length of the best code [8], that is, a code without redundancy.

The amount of compression yielded by a coding can be measured by a *compression ratio* [16]. We define the compression ratio as

$$R = \frac{\text{average message length}}{\text{average codeword length}}\,.$$

Example 1: The compression ratio yielded by the prefix coding showed in Table II is $R = 2/1.75 = 1.143$.

Data compression may be evaluated by the compression ratio and the computational complexity, that is, the time necessary to compress data [16].

III. KOLMOGOROV COMPLEXITY

A typical problem in computer science is the evaluation of the complexity, that is, the evaluation of the computational re-

sources needed to solve a given problem. The concept of complexity belongs to both categories: dynamic complexity and static complexity.

The dynamic complexity is related to the time execution complexity and to the space complexity [11]. The static complexity, on the other hand, is related to the quantitative measure of an object information content. It is an important concept for computer science because, instead of dealing with the time or space needed to run a given program, it deals with the complexity of the object itself [17].

We accept that exists a specification method f that associates at most one object x with one description y. Let $X \subseteq \{0,1\}^*$ be the set of objects and $Y \subseteq \{0,1\}^*$ the set of descriptions, then $f(y) = x$, where $x \in X$ and $y \in Y$. We call f^{-1} the *coding method*.

There is an enumeration of the binary strings by the lexicographycal order

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ \Lambda & 0 & 1 & 00 & 01 & 10 & 11 & 000 & 001 & \cdots \end{pmatrix},$$

where Λ is the empty string. The natural number i represents the i-th binary string (the string s_i). If $|.|$ denotes the string length, then

$$|s_i| = \lfloor \log(i+1) \rfloor , \tag{1}$$

where $\lfloor . \rfloor$ represents the biggest integer number smaller or equal than a number, while log represents the base two logarithm.

In this paper, we will not distinguish between binary strings and natural numbers. Hence, we can define the "size" of an integer number. Moreover, we can define functions over naturals $\phi : \mathbb{N} \to \mathbb{N}$ as functions over binary strings $\phi : \{0,1\}^* \to \{0,1\}^*$. Then, we can represent objects and descriptions as both natural numbers or binary strings.

A. Plain Complexity

We define the complexity $C_f(.)$ over binary strings (or natural numbers), relating to some partial function f (specification method), as

$$C_f(x) = \min_{f(y)=x} |y| ,$$

where x is an object and y is a description. Hence, this complexity is defined over all the set of partial functions. But, in this way, it would not be an objective notion because the complexity of an object would depend on the specification method adopted.

Kolmogorov complexity, on the other hand, uses the Turing machine [2] as specification method (partial recursive functions).

We consider a Turing machine $\mathcal{M}$ which, with a given binary string p and natural number y, computes the output $\mathcal{M}_{p,y} = x$. We say that $\mathcal{M}$ interprets p as a description of x in the presence of the side information y. The side information y interprets the role of a "hint" to compute x.

We will show how to construct a concrete computer to deal with our definition of complexity. The machine has two tapes. The first one is called *program tape* or *input tape* and is an one-way finite read-only tape. The second tape is called *work tape* and is a two-way infinite read/write tape. Initially, the first tape stores the description p while the second stores the side information y literally. All other fields on the work tape are filled with blanks. The machine has two heads, one per tape, and can read or write a symbol (0 or 1), move the head left or right, or delete symbols (all these operations are defined on the work tape). After a finite amount of time, the machine eventually halts with the output stored on the work tape.

Definition 6: The *conditional complexity* $C_{\mathcal{M}}(x|y)$ of a number x with respect to a given number y is the size of the smallest description p in such a way that $\mathcal{M}_{p,y} = x$,

$$C_{\mathcal{M}}(x|y) = \min_{\mathcal{M}_{p,y}=x} |p| .$$

If does not exist a description p of x, then we say, by definition, $C_{\mathcal{M}}(x|y) = \infty$.

In a first view, it seems that the conditional complexity $C_{\mathcal{M}}(.|.)$ remains dependent on the machine $\mathcal{M}$. But, we can prove that it depends little, up to a constant, because there are universal Turing machines, capable of simulating any other Turing machine whose description is supplied (for a proof of the existence of universal Turing machines see [9], [11]). This is the important *invariance theorem* [4], [10], [14], [15], [17].

Let $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots$ be an enumeration of the Turing machines (see [17]). We can do it because the set of programs is enumerable. Let $p_0, p_1, p_2, \ldots$ be an enumeration of the set of programs. Each machine $\mathcal{M}_i$ is defined by an "internal" program p_i. We will call it the "hardware program" [10].

In such a way, we code the i-th machine as

$$\overbrace{111\cdots1}^{i \text{ times}}0p = 1^i 0p ,$$

which means that the universal machine $\mathcal{U}$ will simulate the i-th Turing machine running the program p.

Theorem 1: (Invariance Theorem) There is a machine $\mathcal{U}$, called *universal*, in such a way that for any machine $\mathcal{M}$ and numbers x and y, and some $c > 0$, $C_{\mathcal{U}}(x|y) \leq C_{\mathcal{M}}(x|y) + c$, where c depends only on $\mathcal{M}$.

Proof: For $C_{\mathcal{M}}(x|y) = \infty$ the proposition is trivially truthful. We can show an universal Turing machine simulating any other Turing machine. For example, suppose the machine $\mathcal{U}$ in such a way that $\mathcal{U}_{1^i 0 p,y} = \mathcal{M}_{i_{p,y}}$, where $\mathcal{M}_i$ is the i-th machine in the Turing machines enumeration. Suppose $\mathcal{M}$ is the n-th machine in the enumeration, that is, $\mathcal{M}_n = \mathcal{M}$, then

$$C_{\mathcal{M}}(x|y) = \min_{\mathcal{M}_{p,y}=x} |p|$$

and

$$\begin{aligned} C_{\mathcal{U}}(x|y) &= \min_{\mathcal{U}_{1^n 0 p,y}=x} |1^n 0 p| = \\ &= \min_{\mathcal{U}_{1^n 0 p,y}=x} |p| + n + 1 = \\ &= C_{\mathcal{M}}(x|y) + n + 1 . \end{aligned}$$

That is, the upper limit of the complexity expressed with $\mathcal{U}$ is $C_{\mathcal{M}}(x|y) + n + 1$, and possibly there is a program p' in such

a way that $\mathcal{U}_{p',y} = x$ and $|p'| < |p| + n + 1$. Then,

$$C_{\mathcal{U}}(x|y) \leq C_{\mathcal{M}}(x|y) + n + 1\,.$$

Taking $c = n + 1$ completes de proof, with c depending on $\mathcal{M}$.

Let f and g be two partial functions. If, for all x and fixed y, $C_f(x|y) \leq C_g(x|y) + c$, we say that f *minorizes* g. If $C_f(x|y) \leq C_g(x|y) + c$, for every g in a subset of the partial functions, we say that f is an *universal* element of the subset. The invariance theorem proves that there exists an universal element in the set of the partial recursive functions (the universal partial recursive function).

Restricting the descriptions only to the effective (computable) descriptions permits that exists an universal specification method (additively optimum method) that obtains the minimal-size description with respect to all other methods (minorizes all other specification methods). As a consequence the complexity of an object is an intrinsic attribute of the object itself independently of a particular specification method [17].

From Theorem 1 we can trivially prove Corollary 1, which define an equivalence relation between additively optimum methods.

Corollary 1: Let $\mathcal{U}$ and $\mathcal{V}$ be two universal Turing machines. For all x and some fixed y,

$$\mathrm{abs}\left(C_{\mathcal{U}}(x|y) - C_{\mathcal{V}}(x|y)\right) \leq c_{\mathcal{U},\mathcal{V}}\,,$$

where $\mathrm{abs}(.)$ denotes the absolute value of an integer, and $c_{\mathcal{U},\mathcal{V}}$ does not depend on x.

That is, two optimum specification methods, defining the minimal description of an object, differ only by a constant. It is true because both $\mathcal{U}$ and $\mathcal{V}$ simulate each other [17].

It is interesting to note that the $c_{\mathcal{U},\mathcal{V}}$ constant, perhaps big, is asymptotically negligible, because it does not depend on x. It is very important because we are only interested in the asymptotical behaviour of the string complexity.

Fixing an universal Turing machine $\mathcal{U}$, called *reference machine*, we can say $C(x|y) = C_{\mathcal{U}}(x|y)$, called *conditional complexity*. If no side information is supplied, we can define the unconditional complexity of the number x

$$C(x) = C(x|\Lambda)\,,$$

called *plain complexity*.

In few words, the plain complexity $C(x)$ of a number or binary string x is defined as the size, expressed in bits, of the smallest program for the universal Turing machine $\mathcal{U}$ that computes x (see [3] for an extensive presentation of the soundness and suitability of Kolmogorov complexity applied to computer science problems).

Kolmogorov complexity may be identified with the "size of the compressed version of the string" [4].

Theorem 2: There is a constant $c > 0$ in such a way that $C(x) \leq |x| + c$ for all binary strings x.

Proof: There is a machine $\mathcal{M}$ in such a way that $\mathcal{M}_p = p$ for all p programs. Then, for all binary strings x, applying the Theorem 1, $C(x) \leq C_{\mathcal{M}}(x) + c = |x| + c$.

It is true because a machine that performs the copy of the program itself to the output exists [4]. The upper-bound of the string complexity is the length of the string.

It means that we can compute a string by a program which contains the string itself literally stored inside the program.

B. Prefix Complexity

We define a *prefix code*, a coding of binary strings, which has the remarkable property that strings coded in such way, called *self-delimiting strings*, know their own length. Hence, if we code the string x as a self-delimiting string $\overline{x}$, it is simple to recover from the concatenation $\overline{x}y$ both strings x and y [17].

We can code a string by the following scheme. In the beginning of the string we put the length of the string x coded as a sequence of 1's, $1^{|x|}$, followed by the terminating symbol 0,

$$E_1(x) = \overline{x} = \overbrace{111\cdots1}^{|x| \text{ times}}0x = 1^{|x|}0x\,,$$

with length $|E_1(x)| = 2|x| + 1$.

We decode x by counting the 1's before the first 0 and recovering x using its length.

We can improve the code repeting the procedure for the length of x, obtaining a more compact coding,

$$E_2(x) = \overline{|x|}x = 1^{||x||}0|x|x\,, \tag{2}$$

with length $|E_2(x)| = |x| + 2\log|x| + 1$, by Equation (1).

It is important to note that, by construction, E_1 and E_2 are prefix free codes (see Definition 4).

Example 2: Let $x = 11010100111010$ be a binary string. Hence,

$$E_1(x) = 11111111111111011010100111010$$

with $|E_1(x)| = 29$, and

$$E_2(x) = 11110111011010100111010$$

with $|E_2(x)| = 23$.

Using prefix coding we can define another complexity measure called *prefix complexity*. Prefix complexity represents a more realistic "program length based complexity" because all real-life computer languages are self-delimiting. For example, we know that PASCAL programs always end with a `END.` command (terminating symbol).

We should change our definition of Turing machine to allow programs coded as self-delimiting strings. First, the machine reads the prefix with the length of the program and then, knowing where the code ends, performs the computation of the program. This kind of Turing machine is called *prefix Turing machine*.

Definition 7: Let $\mathcal{M}$ be a prefix Turing machine. The conditional prefix complexity $K_{\mathcal{M}}(x|y)$ of the number x with respect to the side information y is the length of the smallest self-delimiting description p, in such a way that $\mathcal{M}_{p,y} = x$,

$$K_{\mathcal{M}}(x|y) = \min_{\mathcal{M}_{p,y}=x} |p|\,.$$

If there is not a description p of x then we say, by definition, $K_{\mathcal{M}}(x|y) = \infty$.

Theorem 3: There is a prefix universal Turing machine $\mathcal{U}$, in such a way that for any prefix Turing machine $\mathcal{M}$ and numbers

x and y, and some $c > 0$, $K_{\mathcal{U}}(x|y) \leq K_{\mathcal{M}}(x|y) + c$, where c depends only on $\mathcal{M}$.

Proof: Similar to Theorem 1.

As with the plain complexity, we can define an unconditional version of the prefix complexity.

Of course, both complexities K and C are asymptotically equivalent.

Suppose binary strings generated by a stochastic process (ergodic source). We can prove that K complexity is formally equivalent (asymptotically) to the entropy of the source [8], as defined in Definition 5.

We can define the K complexity as a Shannon-Fano code, based on the algorithmic probability $P_{\mathcal{U}}$, which is optimal (has the best compression) [8], [17], [20].

$P_{\mathcal{U}}(x)$ is defined as the probability of a string x to be computed by a random self-delimiting binary program in the universal Turing machine $\mathcal{U}$ [5], [8],

$$P_{\mathcal{U}}(x) = \sum_{\mathcal{U}_p = x} 2^{-|p|} \approx 2^{-K(x)} \,. \tag{3}$$

The equivalence (3) is proven in [8].

Let $\langle .,. \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be an effective pairing function. Hence, we can define $K(\langle x, y \rangle) = K(x, y)$ as the complexity of both x and y.

The K complexity has a remarkable property, called *subadditive* [5], [17], defined as

$$K(x, y) \leq K(x) + K(y) + O(1) \,. \tag{4}$$

IV. Incompressibility of Binary Strings and Randomness

The original proposal of Kolmogorov complexity concept was to define random sequences by a formal way, supporting a mathematical theory of probability. At that time, when Kolmogorov proposed this concept, only empirical evidence from Monte Carlo games supported the theory of limiting frequency [22].

Limiting frequency states that, in a long run, relative frequencies of ocurrences of success or failure in an experiment must converge to a limit,

$$\lim_{n \to \infty} \frac{\lambda(n)}{n} = p \,,$$

where λ counts the ocurrences of success in the experiment and p is the limiting frequency. For example, if a fair coin is tossed a large number of times, we expect that, in a *sufficiently* long run, $p = 1/2$.

A difficulty in this approach is to define what is "a long run", because any sequence of trials is, obviously, limited in size.

Kolmogorov complexity proposes a new approach, first defining finite random sequences and after, infinite random sequences.

Kolmogorov realized that many strings may be algorithmically compressed. Today, it is a simple concept, used for data and image compression purposes by formats like GIF and programs like winzip, gzip, compress, etc. These programs try to identify regularities in the file to be compressed and use the regularities, substituting regular sequences by shorter descriptions.

We identify strings that can be compressed as *compressible* or *regular* and strings that can not be compressed to a size much smaller than the original as *incompressible* or *random* [10], [14], [15].

Definition 8: For some constant $c > 0$, we say that a finite string x is *c-incompressible* if $C(x) \geq |x| - c$.

The constant c, in the Definition 8, plays the role of a "compression ratio".

Suppose the uniform distribution (Lebesgue measure). A simple counting argument shows that the majority of strings are incompressible [4], and it is the profound reason that the fair coin has a limiting frequency and its behaviour is *unpredictable* or random.

Theorem 4: There are more binary strings with high Kolmogorov complexity than binary strings with low complexity.

Proof: Follows from the fact that there are $2^n - 1$ binary programs with length smaller than n, because $\sum_{i=0}^{n-1} 2^i = 2^n - 1$, but there are much more strings with length smaller or equal than n ($2^{n+1} - 1$).

Despite Theorem 4, it is interesting to note that the majority of the strings that appear in data produced by human beings and Nature have much redundancy and may be compressed (see [4], [13]).

If $x = x_1 x_2 x_3 x_4 \cdots$, each x_i for $i \geq 1$ a binary digit, is an infinite binary string, then $x_{1:n} = x_1 x_2 \cdots x_n$.

Definition 9: An infinite binary string is *random* if

$$\forall n K(x_{1:n}) \geq n - c$$

for some $c > 0$.

In other words, an infinite string x is random if all its prefixes are random or incompressible by the K complexity. It is important to note that the use of K complexity instead of C is to avoid some problems with the definition. The problems are related to the fluctuation of complexity [17].

We define the *mutual information*, $I(x : y) \equiv_{\text{def}} K(x) + K(y) - K(x, y) + O(1)$, as a measure of the quantitative information one string knows about the other [6]. By Equation (4), we realize that $I(x : y) \geq 0$. It means that the bigger is $I(x : y)$, less distance exists between the strings and easier it is to compute both together. It is an important concept for video and animation compression, because contiguous frames may be more compressed using the similarities between them.

V. Formalizing Data Compression

Remain two fundamental questions about compression: Can we get better algorithms to compress data? And, if yes, is there the best one?

If the answer for the first question is no, any effort to compare and evaluate algorithms for data, video, image and animation compression is without meaning, we must only pick up a good algorithm to use. If the answer for the second question is yes, probably it is more important to concentrate efforts to develop such "perfect algorithm".

To answer these questions we must first define *compression algorithm*, based on the formal definition of compressibility

by Kolmogorov complexity. It is important to note that Kolmogorov complexity is related to the Turing machine as a "decompression machine" and does not concern about the "compression algorithm". Moreover, we must formalize what we mean by "better compression" and "best compression" [21].

We will understand compression algorithm as a computable function over binary strings that shrinks data.

A. There Are Better Compression Algorithms

Definition 10: Let $s, s' \in \{0,1\}^*$ be binary strings. We call δ a *decompression algorithm* and, for all s' exists some s, in such a way that $s' = \delta(s)$, where s is the code of s', with $|s| \leq |s'| + c$. We call γ a *compression algorithm* if, for all s, $s = \delta(\gamma(s))$, and γ must be an injective function. We call the pair $\Gamma = (\gamma, \delta)$ a *compression scheme*.

Definition 11: Let $\Gamma_1 = (\gamma_1, \delta_1)$ and $\Gamma_2 = (\gamma_2, \delta_2)$ be two compression schemes. We say that Γ_1 is *better than* Γ_2 if, for all s and some $c > 0$,

$$|\gamma_1(s)| \leq |\gamma_2(s)| + c\,. \tag{5}$$

Here we find the link between Kolmogorov complexity and our problem. $|\gamma(s)|$ is the size of the code of s in a compression scheme (γ, δ). A compression scheme Γ_1 is better than Γ_2 if and only if Γ_1 *minorizes* Γ_2.

Now, we can proof the result about the existence of better compression algorithms [21].

Theorem 5: Let $\Sigma = \{(\gamma_1, \delta_1), (\gamma_2, \delta_2), \ldots, (\gamma_n, \delta_n)\}$ be a set of compression schemes. There exists a compression scheme (γ, δ) that is better than all of them.

Proof: Suppose Σ above defined. We can construct a better compression scheme (γ, δ) by the following procedure. First, for any string s, apply all compression algorithms γ_i, $1 \leq i \leq n$, to the string s and choose the best compression in Σ, identified by γ_b,

$$|\gamma_b(s)| = \min\{|\gamma_1(s)|, |\gamma_2(s)|, \ldots, |\gamma_n(s)|\}\,.$$

Then, place b in the first $\lfloor \log(n+1) \rfloor$ bits and place $\gamma_b(s)$ in the final of the code. This is $\gamma(s)$.

We can prove that γ is better by construction, according to Definition 11, because

$$|\gamma(s)| = \lfloor \log(n+1) \rfloor + |\gamma_b(s)|\,.$$

That is, the upper limit of the size of $\gamma(s)$ is $|\gamma_b(s)| + c$, where $c = \lfloor \log(n+1) \rfloor$. Then,

$$|\gamma(s)| \leq |\gamma_i(s)| + c$$

for all i, $1 \leq i \leq n$, as required by Equation (5), showing that γ is better than all the other compression algorithms in Σ.

Finally, we can construct the decompression algorithm δ. δ must look the first $\lfloor \log(n+1) \rfloor$ bits of the code s', where is stored b and reconstruct s applying δ_b to the remaining part of the code s'.

B. We Can Not Reach the Best Compression Ratio

Definition 12: We say that a compression scheme Γ is *the best* if it is better than any possible compression scheme, in the sense of Definition 11.

We realize that the best compression scheme agrees with the Kolmogorov complexity concept of minimal-size description, that is, if Γ_b is the best compression scheme then, for all s,

$$|\gamma_b(s)| = C(s) + O(1)\,. \tag{6}$$

So, we identify δ, the decompression algorithm, with the universal Turing machine $\mathcal{U}$. It means that a compression scheme is the best one if it minorizes all other schemes.

We think about $C(.)$ as a function over naturals, $C : \mathbb{N} \to \mathbb{N}$. We want to prove that we can not reach the best compression ratio by reduction to the problem of computing Kolmogorov complexity.

Theorem 6: The function C is not computable.

Proof: For some x, we compute $C(x)$ running all programs p with $|p| \leq |x| + c$, for some c, in a parallel fashion, because it is known that $C(x) \leq |x| + c$ by the Theorem 2. We do this by generating all programs in an increasing size lexicographycal order and providing increasing run time to the programs by the following procedure.

1) Set $i := 0$ and $j := -1$;
2) Let $i := i + 1$;
3) If $j = -1$ then generate the first i programs in the lexicographycal order. If $j \neq -1$ then generate only the programs p with $|p| \leq j$;
4) Simulate i steps of the computation of these programs in the universal Turing machine $\mathcal{U}$. If a program p halts with $\mathcal{U}_p = x$ and $|p| \leq j \vee j = -1$ do $j := |p|$;
5) If all programs halted and $j \neq -1$ or there is not a program p that does not halt with $|p| < j \wedge j \neq -1$ then go to 6, otherwise delete all generated programs and go to 2;
6) Return j and stop.

While the programs are halting, we store the length of the smallest program that halted computing x. When all programs will have halted we have the value of $C(x)$. But, by the halting problem, we can not decide if a program halts or not. Hence, by reduction to the halting problem, the process is not effective.

The same result holds for K complexity.

The problem is that if we can not decide if a given program halts or not, we can not decide what is the smallest program that *halts* and compute a given binary string. Then, by (6), we can not decide if a given compress scheme Γ is the best.

Moreover, by Theorems 5 and 6 we conclude that it is impossible to find, among all possible compression algorithms, one which is the best, because we always can construct another compression scheme better than all given schemes and can not prove formally which one is the best.

Theorem 7: There is not the best compression algorithm.

Proof: Follows from the discussion above.

C. More Time Means More Compression Ratio

We want to develop a more practical point of view. If we can not achieve the best compression, what about a non-optimum

but good solution? It means that we want to have the best possible compression.

A function $f : \mathbb{N} \to \mathbb{N}$ is semi-computable if there exists a (total) recursive function $\phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, with $\phi(t, x)$ monotonic on t, in such a way that $\lim_{t\to\infty} \phi(t, x) = f(x)$.

Let p be a program and t a natural number, then $\overline{\mathcal{U}}_p^t$ defines the predicate that is true if the computation of t steps of p in $\mathcal{U}$ halts and computes the value Φ_p and false otherwise with Φ_p undefined. Using the Turing-Church thesis is easy to show that $\overline{\mathcal{U}}_p^t$ is decidable and Φ_p is computable [10].

Corollary 2: (bounded halting problem) $\overline{\mathcal{U}}_p^t$ is decidable and Φ_p is computable.

Theorem 8: The function C is semi-computable.

Proof: We know that $C(x) < \log(x) + c$, because a natural number may be represented by its binary code. Let $\mathcal{U}_p^t$ be the universal Turing machine which computes t steps of the program p simulating $\mathcal{U}$. Let $C^t(x)$ be a value smaller than $\log(x) + c$ and

$$C^t(x) = \min\{|p| \leq t : \mathcal{U}_p^t = \text{true}, \Phi_p = x\}\,.$$

Then, the function C^t is computable, monotonic on t and $\lim_{t\to\infty} C^t(x) = C(x)$.

By Theorem 8, there is a (total) recursive function $\phi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, in such a way that $\lim_{t\to\infty} \phi(t, x) = C(x)$.

C^t is a monotonic non-increasing function on t, implying that C is a *co-enumerable* function. A co-enumerable function is a function which may be approximated from above [17]. C^t is a resource bounded complexity measure.

Hence, remembering the link between Kolmogorov complexity and data compression, we conclude that the more time we have to compute the minimal description, more compression we will get. That is, the solution for the problem of obtaining the best compression may be approximated from above by a (total) recursive function. We must warn that this procedure may be a hard problem and a time consuming one.

VI. Web-based Animation Models

Multimedia formats provide resources to handle a wide range of elements like images, sound, animation and video, bringing life to web pages.

Computer animation models usually supply tools to move and deform objects, cameras and light sources disposed on a scene. In the last 20 years, many different animation methods are being developed [18] to provide objects behaviour changing along time. Some of them search for producing movement with a natural look, others try to combine several animation techniques in one, to enlarge the data compression ratio and so on.

From the point of view of multimedia applications on the web, movement sensing is produced, in general, by concatenating sequences of images that are displayed on the screen one after the other, in the same way used to simulate movement in cinema or TV. While in these cases we need about 24 or 30 pictures per second to have the sensation of motion, by multimedia applications, the display of 15 images per second is considered an acceptable rate.

In this context, animation can be defined as a sequence of images displayed at intervals of time. Hence, we can define an animation s as

$$s = s_1 s_2 s_3 \cdots s_n\,,$$

where s_i, for $1 \leq i \leq n$, are called *frames*, each frame corresponding to an image.

An image is composed by a sequence of binary string digits, where each bit or chunk of bits describe the intensity or the color of a single pixel on the screen. Finally, an image can also be defined as a binary string.

One problem with storing and transmitting images is that an image contains a large amount of data. High resolution pictures may contain hundreds of Megabytes of data [21]. The problem is specially difficult in long distance transmition with low quality channel, like communication with deep space probes.

Usually, animation and images are stored compressed to save storage space and transmission time.

For example, pictures of Uranus transmitted by Voyager 2 used a technique of data compression called *difference mapping*, in which the image is represented as an array of differences in brightness and colors between adjacent pixels. In this case, it is reported that 8 bits color was reduced to an average of 3 bits per pixel [16]. This illustrates how important data compression technology is.

The problem of data compression is more complex when applied to animated sequences, because animations are not composed of a single image, but of a set of images. The development of a methodology for comparing and evaluating data compression in animation is a great and usefull achievement.

In the next sections we will compare two web-based animation models regarding data compression ratio. The first format analyzed is GIF-based animations, while the second one corresponds to AGA-based animation model, a format recently developed [1] to provide animation based on the sequencing of images.

A. GIF-Based Animations

GIF (Graphics Interchange Format) is a very popular image format adopted in the development of web pages. It uses a built-in LZW compression algorithm and provides some nice features like animation of multiple images, refered as frames and encoded within a single file.

GIF is defined in terms of blocks and sub-blocks containing, each one, data and control information. The main divisions defined in the GIF protocol are: the *Header*, identifying the beginning of the data stream; the *Logical Screen Descriptor*, defining some parameters for the area of the display device where the images will be rendered, like the number of bits available for color; a *Global Color Table*, to store a color table represented as sequences of bytes of RGB triplets and maintained to handle the colors in the images; the *Image Descriptor*; the *Image Data*; and the *GIF Trailer* to end the data stream [7]. Each image stored in an animated GIF file is composed of the Image Descriptor and the Image Data. Finally, each block has a Block Size field that counts the number of bytes in the block.

Concerning the algorithm used in the compression of data streams, the GIF format uses the LZW algorithm, that is a variation of the Lempel-Ziv compression algorithm [7], [16].

Lempel-Ziv coding, as well as its derivatives, is an *adaptive code*. It means that the behaviour of the compression algorithm is dynamic, changing "on the fly" the mapping from the set of messages to the set of codewords, according to the approximated probabilities computed during the process [16].

The Lempel-Ziv algorithm parses the binary source into a set of phrases. At each step, it searches a phrase that does not belong to the set previously obtained. It means that each new phrase has a prefix belonging to the previous set plus a new binary digit, in an increasing length sequence. In such a way, it is enough to store the position of the previous prefix and the terminating binary digit [8], [16].

Codewords are represented here as pairs (i, x), where i is the position of the prefix in the sequence of codewords and x is the final binary digit of the phrase. $(0, x)$ means that the codeword does not have prefix, it has only the binary digit x (this occurs in the beginning of the process).

Example 3: Let 1001101010011 be a binary source. The Lempel-Ziv compression algorithm will parse the set of phrases as $\{1, 0, 01, 10, 101, 00, 11\}$. The input data will be encoded as $(0, 1)$, $(0, 0)$, $(2, 1)$, $(1, 0)$, $(4, 1)$, $(2, 0)$ and $(1, 1)$.

The compression is achieved because, as the length of the phrases increase, it becomes more efficient to store its positions [8].

Lempel-Ziv compression algorithm brings a poor compression ratio when the length of the input data is short, but we can prove that, applied on a string generated by a stochastic process (ergodic source), the average codeword length approaches asymptotically the entropy of the source [8]. It means that Lempel-Ziv code is optimal. Typical compression, obtained with Lempel-Ziv algorithm, is in the range of 50-60% [16].

It is an *universal code*, that is, a code that does not depend on the distribution of the source (a general purpose compression algorithm). A straightforward implementation takes time $O(n^2)$ to compress a string with length n [16].

LZW algorithm differs from the straighforward implementation of the Lempel-Ziv algorithm in some ways, such as:

- LZW uses a dictionary of phrases;
- LZW manages phrases and codewords in a subtly different way of Lempel-Ziv, obtaining a faster compression.

B. AGA-Based Animations

Accorsi and Menezes describe in [1] a model to represent computer animation based on automata theory [11]. The model, called AGA (Automata-based Graphical Animation), organizes the content of an animation sequence as automata. These automata describe the behaviour of actors in animation time.

The features of AGA provide the use of a query language for information retrieval, increasing the animation reuse and contributing to reduce storage space.

The AGA model organizes animation in a set of actors. These actors are clipped elements from images that have dynamic behaviour and, together, compose the movement. For example, an actor could be the image of a bird or the image of an eye on a face.

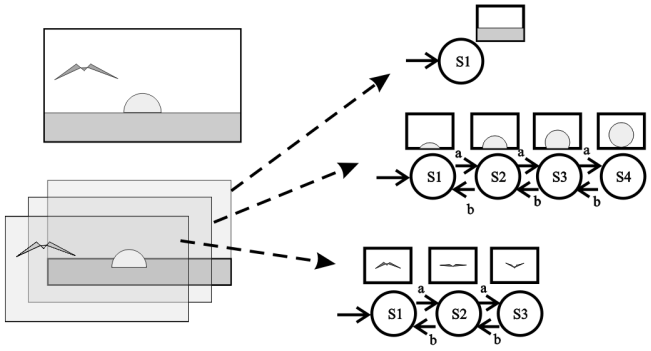


Fig. 1. AGA Animation

Each actor is modeled as a finite automaton with output, whose states are linked with output images. The dynamic behaviour of the actor during the animation is produced by adding symbols to the input tape. When a symbol is read, the current state is changed and the linked image is shown on a layer. The animation consists of a set of layers. The union of these layers composes the frame at the current instant. Each layer is controlled by one automaton with independent tape.

An example of AGA animation is shown in Figure 1. In the example, we can see the individual behaviour of the actors, each actor modelled by an automata, and the overlapped layers.

The model of actor used in AGA is based on traditional automaton with output (Moore and Mealy machines) [11], but some extensions are proposed to insert timing information, use images and support query facilities.

The first extension is related to the output alphabet and output function. The output function is a mapping from a state to image instead of a state to symbol. Thus, when a state is reached, an output image is shown on the layer controlled by the automaton. The storing of only distinct images of actors is an important feature of this model, these images are selected by the transition function when a symbol is read. Hence the storage space does not increase in the same proportion of the number of frames, but it depends on the number of distinct images and symbols in the tape. The output function is also extended to produce transformations on output images. In this way, it is possible to produce scaling, rotation and other transformations without storing new images. This function is called context output function. So, each tape cell has, besides the input symbol, an associated string of symbols representing which transformations must be done.

Another extension proposed for the automaton is the timed transition function. This function provides the insertion of timing information in the tape to temporize the state changes. Thus, when a tapes cell is read, three kind of information are acquired: the input symbol, the time spent on current state and the string of transformations. The time instructions are in the input tape instead of edge that link the states in a static way.

Figure 2 shows the input tape used in an animation, together with the animation in progress. We can see how the input symbol reading controls the behaviour of the actor.

A new function, called description function, has been introduced to insert a semantic documentation of states. This func-

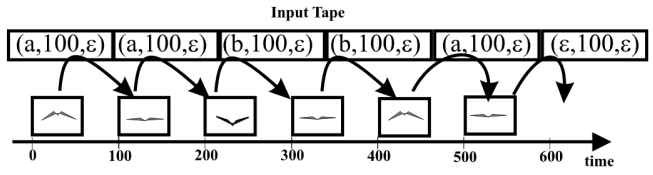


Fig. 2. Input Tape

tion is a mapping from a state to a semantic description. The aim of the function is to support information retrieval. It is possible to retrieve the exact moment when an actor stays on a state by its description. This recovery is performed by tape and transition diagram analysis. This operation can be further explored to produce complex queries using several actors.

In the next two definitions AGA actor and AGA animation are formalized.

Definition 13: An *AGA actor* is defined as

$$\mathrm{ACT} = (Q, \Sigma, \Delta, \delta', \lambda^c, \sigma, q_0, F, D)\,,$$

where

Q – set of actor's states;
Σ – input alphabet;
Δ – set of images of the actor;
$\delta' : Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{N} \to Q$ – timed transition function (partial function);
$\lambda^c : Q \times F^* \to \Delta^{*F}$ – context output function (total function);
$\sigma : Q \to D$ – description function (partial function);
q_0 – initial state;
F – set of transformation functions;
D – set of descriptions.

The production $\delta'(q_1, x, 200) = q_2$ means that if the current state is q_1, with x in the input tape and after 200ms of time waiting, the automaton goes to state q_2. ε allows empty transitions.

The elements of Δ^{*F} are sequences of images transformed by applications of elements of F.

Definition 14: An *AGA animation* is defined as a total order $\leq^C$ of pairs $(\mathrm{ACT}, \mathrm{tape})$, where ACT is an actor and tape is an input tape, and

$$\mathrm{AGA} = (\{(a, \text{tape of } a) | a \text{ is an actor}\}, \leq^C)\,.$$

$\leq^C$ is defined in such a way that $ACT_1 \leq^C ACT_2$ if and only if ACT_1 is behind ACT_2 in the animation.

Animations generated with AGA use LZW compression algorithm and may send images, from the web server to the client, on demand to improve the time of download.

The content structure in AGA model provides information retrieval oriented by state and partial visualization of layers. This new approach contributes to the design of query languages for information retrieval on semistructured data. The encapsulation of actor aesthetic and behaviour properties provides the reusability of images during animation compositions.

VII. Formalizing Animation

As noted, an animation is formed by a sequence of frames displayed on the screen at time intervals. To deal with the concept of "better animation", we need to define "animation" in a formal way.

Definition 15: Let F be a set of frames. A *target animation* is a pair (s, t), where s is a sequence $s = s_1 s_2 s_3 \cdots s_n$, each $s_i \in F$ for $1 \leq i \leq n$, and t is a sequence of intervals of time $t = t_1 t_2 \cdots t_n$, each $t_i \in \mathbb{N}$ for $1 \leq i \leq n$.

In Definition 15, we define an animation as a binary string "computed" in some way. This definition concerns about the "dynamic behaviour" of the animation, that is, the effect of the execution of the animation, but it does not tell anything about how to store and produce the animation. We must define a mechanism that "computes" the animation.

Definition 16: An *animation descriptor* is a pair $\phi = (\alpha, \Delta)$, where α is an indexed set of images (or actors) and Δ is a function, $\Delta : \alpha \mapsto a$, that is, Δ maps α to the target animation a. Δ is a strategy to compute a target animation from α.

For example, in GIF format the strategy Δ is a very simple function. All frames are literally stored in the file and the animation goes on sequentially, from the first frame to the last one.

But, we must remember that usually animation formats use data compression. Definition 17 combines the strategy to compute the animation with compression.

Definition 17: An *animation scheme* is a pair $G = (\Gamma, \phi)$, where Γ is a compression scheme and ϕ is an animation descriptor.

The compression algorithm γ is applied on α, the set of images (or actors) of Definition 16.

Let G be an animation scheme and let a be a target animation. We define $G(a)$ as the animation a performed by the scheme G, and define $|G(a)|$ as the number of bits necessary to store the animation a using the scheme G. It is simple to show that, for some $c > 0$,

$$|G(a)| = |\gamma(\alpha)| + |\Delta| + c\,, \tag{7}$$

where $|\gamma(\alpha)|$ is the number of bits necessary to store all the images (or actors) in α using γ compression, and $|\Delta|$ is the length of function Δ.

The meaning of $|\Delta|$ is that we must *describe* the function Δ in some effective way. Hence, we can count the number of bits of the description.

Definition 18: Let G_1 and G_2 be two animation schemes. G_1 is *better than* G_2 if, for all target animations a and some $c > 0$,

$$|G_1(a)| \leq |G_2(a)| + c\,.$$

That is, G_1 is better than G_2 if G_1 *minorizes* G_2 in the sense of Kolmogorov complexity. We will call G_1 and G_2 respectively G_1-machine and G_2-machine to emphasize this relationship. That is, G_1-machine is better than G_2-machine if G_1-machine *simulates* G_2-machine.

VIII. AGA Is Better Than GIF

From the fact that we formalized the concept of "better animation" as machine simulation, induced by the Kolmogorov complexity concept, we must define both, AGA and GIF, in a formal way as machines.

We realize that the compression ratio of the animation is obtained from two different aspects:

- The compression algorithm used to compress the images the animation is composed of;

- The way the animation is computed from a set of images.

These two parts are represented in Equation (7) as $|\gamma(\alpha)|$ and $|\Delta|$, respectively.

Fortunately, both, GIF and AGA, use the same compression algorithm, LZW, reducing the proof complexity, and avoiding the need of empirical data about compression ratio.

Image and animation formats usually have a header with control information followed by a body with the data stream. The header length is not affected by the body length more than a logarithm term (about prefix coding length see Section III-B), and therefore, it is asymptotically negligible.

Hence, let $G_{\text{AGA}} = (\Gamma_{\text{LZW}}, \phi_{\text{AGA}})$ be the AGA-machine and let $G_{\text{GIF}} = (\Gamma_{\text{LZW}}, \phi_{\text{GIF}})$ be the GIF-machine (animation schemes). Moreover, $\phi_{\text{AGA}} = (A, \Delta_{\text{AGA}})$ and $\phi_{\text{GIF}} = (F, \Delta_{\text{GIF}})$, where A is a set of actors and F is a set of frames.

Formalizing Δ_{AGA} and Δ_{GIF}, we must take into account the way both compute the target animation.

Function Δ_{GIF} is very simple. The GIF file stores all frames literally and performs the animation displaying the frames sequentially, that is, $\Delta_{\text{GIF}} : \{s_1, s_2, \ldots, s_n\} \mapsto (s_1 s_2 \cdots s_n, t_1 t_2 \cdots t_n)$, where $t_1 = t_2 = \cdots = t_n = t \in \mathbb{N}$. This approach is related to the concept of storing the string inside the program shown in Theorem 2.

On the other hand, AGA uses a more sophisticated approach. Δ_{AGA} is defined as a function computed by a finite automata with output [11]. Although automata may be a weak definition, with serious limitations compared with Turing machines, it is good enough to express strings with fewer bits than GIF.

Δ_{AGA} maps an input $x_1 x_2 \cdots x_n$, each $x_i \in \Sigma$ for $1 \leq i \leq n$, to

$$(\lambda^c(q_0, f_1)\lambda^c(q_1, f_2) \cdots \lambda^c(q_n, f_{n+1}), t_1 \cdots t_{n+1}) \,, \qquad (8)$$

where $q_0, q_1, \ldots, q_n$ is the sequence of states such that $\delta'(q_{i-1}, x_i, t_i) = q_i$, for $1 \leq i \leq n$, and t_{n+1} is obtained from the transition $\delta'(q_n, \varepsilon, t_{n+1}) = q_n$ (empty transition) [1], [11]. $f_1, f_2, \ldots, f_{n+1}$ are transformation functions.

Theorem 9: AGA is better than GIF.

Proof: We must show that AGA minorizes GIF, that is, for any target animation a and some $c > 0$,

$$|G_{\text{AGA}}(a)| \leq |G_{\text{GIF}}(a)| + c \,. \qquad (9)$$

It is enough to show that, in a general case, Equation (9) holds.

Let $a = (s, t)$ be any target animation, with

$$s = s_1 s_2 s_3 \cdots s_n$$

and

$$t = t_1 t_2 t_3 \cdots t_n \,,$$

with $t_1 = t_2 = \cdots = t_n = t'$.

Hence, for the GIF format $F = \{s_1, s_2, \ldots, s_n\}$, $|\gamma_{\text{LZW}}(F)|$ is the storage space needed for the frames.

We need to delimit the frames to implement Δ_{GIF}. Using prefix coding E_2 (see Equation (2)), we know that it is necessary $O(\log |F|)$ bits plus $O(\log n)$ bits for delimiting the final frame. Then, $|\Delta_{\text{GIF}}| = O(\log |F| + \log n)$ and from Equation (7),

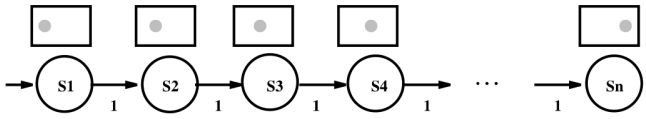


Fig. 3. Simple Automata

$$|G_{\text{GIF}}(a)| = |\gamma_{\text{LZW}}(F)| + O(\log |F| + \log n) + c_{\text{GIF}} \,,$$

for some $c_{\text{GIF}} > 0$.

For AGA format, we define the worst coding, storing literally all frames and defining the automata shown in Figure 3. So, $A = \{s_1, s_2, \ldots, s_n\}$. The automata has n states, each state outputs one frame (actor). Note that, in Equation (8), we set $t_1 = t_2 = \cdots = t_n = t'$, because in GIF protocol all frames are displayed with the same time delay, and $f_1 = f_2 = \cdots = f_n = \varepsilon$.

We need to store the input tape $111 \cdots 1$, which are n consecutive 1's. It is know that we need $\log n$ bits to store it plus $O(\log |A| + \log n)$ for the prefix coding of the frames. Hence,

$$|G_{\text{AGA}}(a)| = |\gamma_{\text{LZW}}(A)| + O(\log |A| + \log n) + c_{\text{AGA}} \,,$$

for some $c_{\text{AGA}} > 0$.

It is straightforward to conclude that (9) holds, because the upper limit of $|G_{\text{AGA}}(a)|$ is $|G_{\text{GIF}}(a)| + c$, for some $c > 0$.

Theorem 9 follows from the fact that GIF-machine stores the animation literally and AGA-machine computes the animation from the set of actors and the input tape (GIF compression is the upper limit of AGA compression, in the sense of Theorem 2). That is, AGA-machine simulates GIF-machine.

IX. Empirical Data

This Section shows empirical data from tests performed to compare AGA and GIF. The purpose of this Section is not to explore in depth empirical data about AGA and GIF, but to show some data that illustrate the theoretical results of the paper.

The Table III shows that AGA file sizes are smaller than GIF file sizes. Good results are obtained with very short and simple animations. Better results may be obtained with larger animations.

In Table III, "Improvement" means the percentage of compression obtained with AGA compared with GIF.

The animations used in the tests are not artificial, but real animations used in a formal languages course web page.

GIF with partial frames is an improvement of full frames, available in GIF protocol, achieving more compression ratio.

X. A Model Better Than AGA

We can do a stronger definition of $\phi = (\alpha, \Delta)$, the animation descriptor, allowing Δ to be any partial recursive function. We call it $\phi_{\mathcal{U}}$, the *universal descriptor*.

Hence, $\phi_{\mathcal{U}}$ is based on the universal Turing machine $\mathcal{U}$ instead of a finite automata like AGA.

	Animation 1	Animation 2	Animation 3
AGA			
Images	25,750	26,190	44,648
Automaton	6,745	7,353	9,820
Total	32,495	33,543	54,468
GIF			
Full frames	111,305	176,020	655,151
Partial frames	64,558	93,508	278,513
Improvement			
Full frames	29.2%	19.1%	8.3%
Partial frames	50.3%	35.9%	19.6%

TABLE III
AGA AND GIF COMPARATIVE SIZES (BYTES)

A straightforward argument shows that an animation scheme $G_{\mathcal{U}}$, based on $\phi_{\mathcal{U}}$, would be better than AGA.

Theorem 10: $G_{\mathcal{U}}$ is better than AGA.

Proof: Trivially, by the simulation of finite automata in the Turing machine [11].

We must note that in Theorem 10 we are not concerned about the compression algorithm $\gamma_{\mathcal{U}}$. We supposed it is LZW, that is, the same of AGA.

Changing our definition of Δ from automata to Turing machines, brings us a problem. $\Delta_{\mathcal{U}}$ may "hang" during the animation (partial function). Apart from our proof that $G_{\mathcal{U}}$ is better that AGA, it is an argument to use AGA instead of $G_{\mathcal{U}}$.

XI. CONCLUSIONS

This paper has shown a formalism to define "compression algorithm", "better compression algorithm" and "best compression algorithm". Based on the formalism, we proved that there are better compression algorithms, but we can not reach, in the general case, the best compression ratio. The solution for the optimal compression problem can only be approximated.

We also formally defined "animation" and "better animation", based on machine simulation. Moreover, we have defined a method of comparing computer animation models regarding data compression. The method is theoretical and more qualitative than empirical and quantitative. The approach is based on algorithmic information theory.

We showed an example, comparing GIF-based animations with AGA-based animations.

The method works with data compression without loss, as we can see in Definition 10 that states $s = \delta(\gamma(s))$.

Moreover, Section X shows a further result, defining another animation model and comparing with AGA model.

Future works should cover ways of formally defining data compression with loss, applied to formats like JPEG and MPEG.

Another possible future work is to extend the formalism developed here to other kinds of animation models.

REFERENCES

[1] F. Accorsi and P. Menezes, Computer Animation Based on Automata Theory (in portuguese, "Animação Gráfica Baseada na Teoria de Autômatos"). In: *WMF - Workshop on Formal Methods*, 2000, João Pessoa, Paraíba. WMF'2000 - 3rd Workshop on Formal Methods, SBC, 2000, pp. 122-127.

[2] R. Bird, *Programs and Machines: An Introduction to the Theory of Computation*. Wiley, London, 1976.

[3] C. Campani and P. Menezes, Characterizing the Software Development Process: A New Approach Based on Kolmogorov Complexity. In: *Computer Aided Systems Theory - EUROCAST'2001*, 8th International Workshop on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, feb. 19-23, 2001, Lecture Notes in Computer Science, Springer, vol. 2178, pp. 242-256, 2001.

[4] G. Chaitin, Information-Theoretic Computational Complexity. *IEEE Transactions on Information Theory*, vol. 20, pp. 10-15, 1974.

[5] G. Chaitin, A Theory of Program Size Formally Identical to Information Theory. *Journal of the ACM*, vol. 22, pp. 329-340, 1975.

[6] G. Chaitin, Algorithmic Information Theory. In: *Encyclopedia of Statistical Sciences*, Wiley, New York, vol. 1, pp. 38-41, 1982.

[7] CompuServe Inc., Graphics Interchange Format Programming Reference. Version 89a. CompuServe Inc., jul. 1990.

[8] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley, New York, 1991.

[9] M. Davis, A Note On Universal Turing Machines. In: C. Shannon and J. McCarthy, editors, *Automata Studies*, Princeton University Press, pp. 167-175, 1956.

[10] P. Gács, Lecture Notes on Descriptional Complexity and Randomness. Technical Report, Boston University, Computer Science Dept., Boston, 1988. available: http://www.cs.bu.edu/ftp/gacs/papers/ait-notes.ps.Z.

[11] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, 1979.

[12] A. Khinchin, *Mathematical Foundations of Information Theory*. Dover, New York, 1957.

[13] W. Kirchherr and M. Li and P. Vitányi, The Miraculous Universal Distribution. *Mathematical Intelligencer*, vol. 19, n. 4, pp. 7-15, 1997.

[14] A. Kolmogorov, Three Approaches to the Quantitative Definition of Information. *Problems of Information Transmission*, vol. 1, pp. 4-7, 1965.

[15] A. Kolmogorov, Logical Basis for Information Theory and Probability Theory. *IEEE Transactions on Information Theory*, vol. 14, pp. 662-664, 1968.

[16] D. Lelewer and D. Hirschberg, Data Compression. *ACM Computing Surveys*, vol. 19, n. 3, pp. 261-296, sep. 1987.

[17] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*. Springer, New York, 1997.

[18] R. Parent, *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann Publishers, ISBN: 1558605797.

[19] C. Shannon, A Mathematical Theory of Communication. *Bell Sys. Tech. Journal*, vol. 27, pp. 379-423, 623-656, 1948.

[20] R. Solomonoff, The Discovery of Algorithmic Probability. *Journal of Computer and System Sciences*, vol. 55, n. 1, pp. 73-88, 1997.

[21] S. Subbaramu and A. Gates and V. Kreinovich, Application of Kolmogorov Complexity to Image Compression: It Is Possible to Have a Better Compression, But It Is Not Possible to Have the Best One. In: *Bulletin of the European Association for Theoretical Computer Science*, vol. 69, pp. 150-154, oct. 1998.

[22] P. Vitányi, Randomness. In: *Matematica, Logica, Informatica*, Storia del XX Secolo, vol. 12. available: http://www.cwi.nl/~paulv/ency.ps.

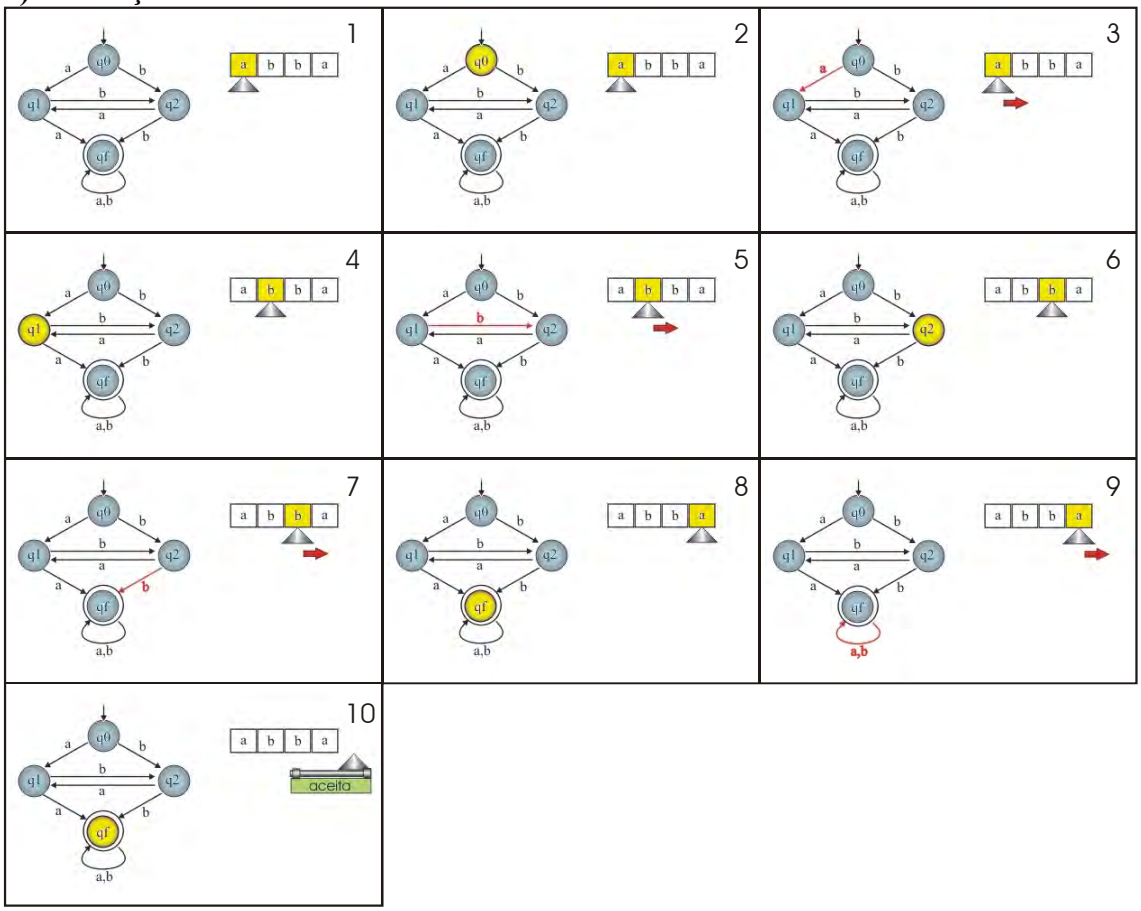
Anexo 4 DTD da Linguagem AgaML 2.0

```
<!ELEMENT AGA (HEAD, ACTOR+, TAPE+, INSTANCE+)>
<!ATTLIST AGA
        VERSION CDATA #FIXED “2.0”
        WIDTH CDATA #REQUIRED
        HEIGHT CDATA #REQUIRED
        BACKGROUND CDATA #REQUIRED
        FRAMERATE CDATA #REQUIRED
        REPEAT CDATA #IMPLIED>

<!ELEMENT HEAD (TITLE, AUTHOR, SUBJECT)>

<!ELEMENT TITLE (#PCDATA)>

<!ELEMENT AUTHOR (#PCDATA)>

<!ELEMENT SUBJECT (#PCDATA)>

<!ELEMENT ACTOR (OUTPUT*, DESCF?, TRANSF)>
<!ATTLIST ACTOR
        ID CDATA #REQUIRED
        TYPE CDATA #REQUIRED
        STATES CDATA #REQUIRED
        SYMBOLS CDATA #REQUIRED>

<!ELEMENT OUTPUT EMPTY>
<!ATTLIST OUTPUT
        ID CDATA #REQUIRED>
        SOURCE CDATA #REQUIRED>
        X CDATA #REQUIRED>
        Y CDATA #REQUIRED>

<!ELEMENT DESCF (DESCRIPTION)+>
<!ELEMENT DESCRIPTION>
<!ATTLIST DESCRIPTION
        STATE CDATA #REQUIRED>

<!ELEMENT TRANSF (FROM+)>

<!ELEMENT FROM (TO+)>
<!ATTLIST FROM
        STATE CDATA #REQUIRED>

<!ELEMENT TO EMPTY>
<!ATTLIST TO
        STATE CDATA #REQUIRED
        SYMBOL CDATA #REQUIRED
        OUTPUT CDATA #IMPLIED>
```

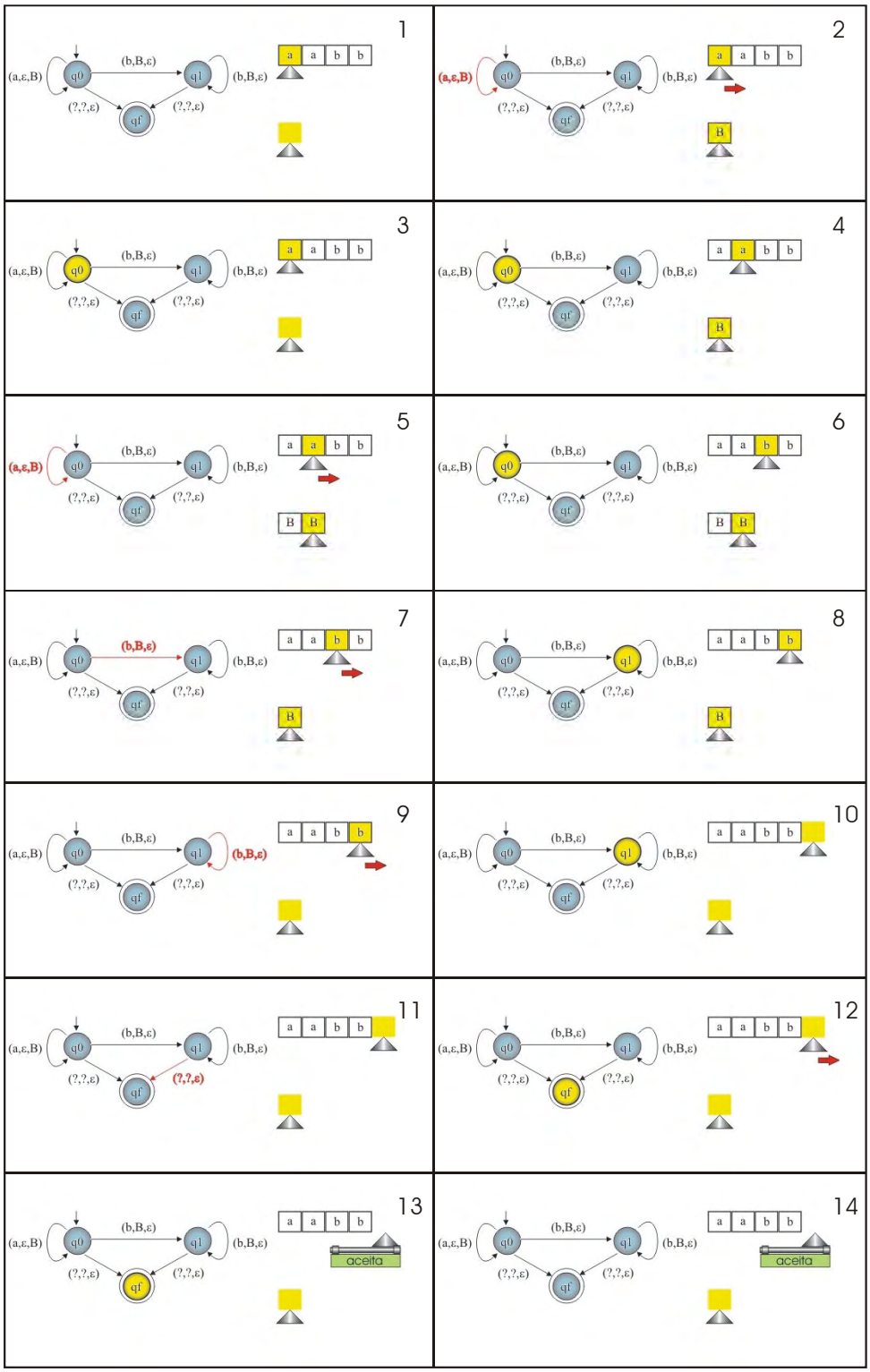
```
<!ELEMENT TAPE (CEL|GROUP)+>
<!ATTLIST TAPE
        ID CDATA #REQUIRED>

<!ELEMENT CEL EMPTY>
<!ATTLIST CEL
        SYMBOL CDATA #REQUIRED
        TIME CDATA #REQUIRED
        FN CDATA #IMPLIED>

<!ELEMENT GROUP (CEL|GROUP)+>
<!ATTLIST GROUP
        ITERATION CDATA #REQUIRED>

<!ELEMENT INSTANCE (USE+)>
<!ATTLIST INSTANCE
        ID CDATA #IMPLIED
        ACTOR CDATA #REQUIRED
        ORDER CDATA #IMPLIED>

<!ELEMENT USE EMPTY>
<!ATTLIST USE
        TAPE CDATA #REQUIRED>
```

Anexo 5 Animações do Estudo de Caso

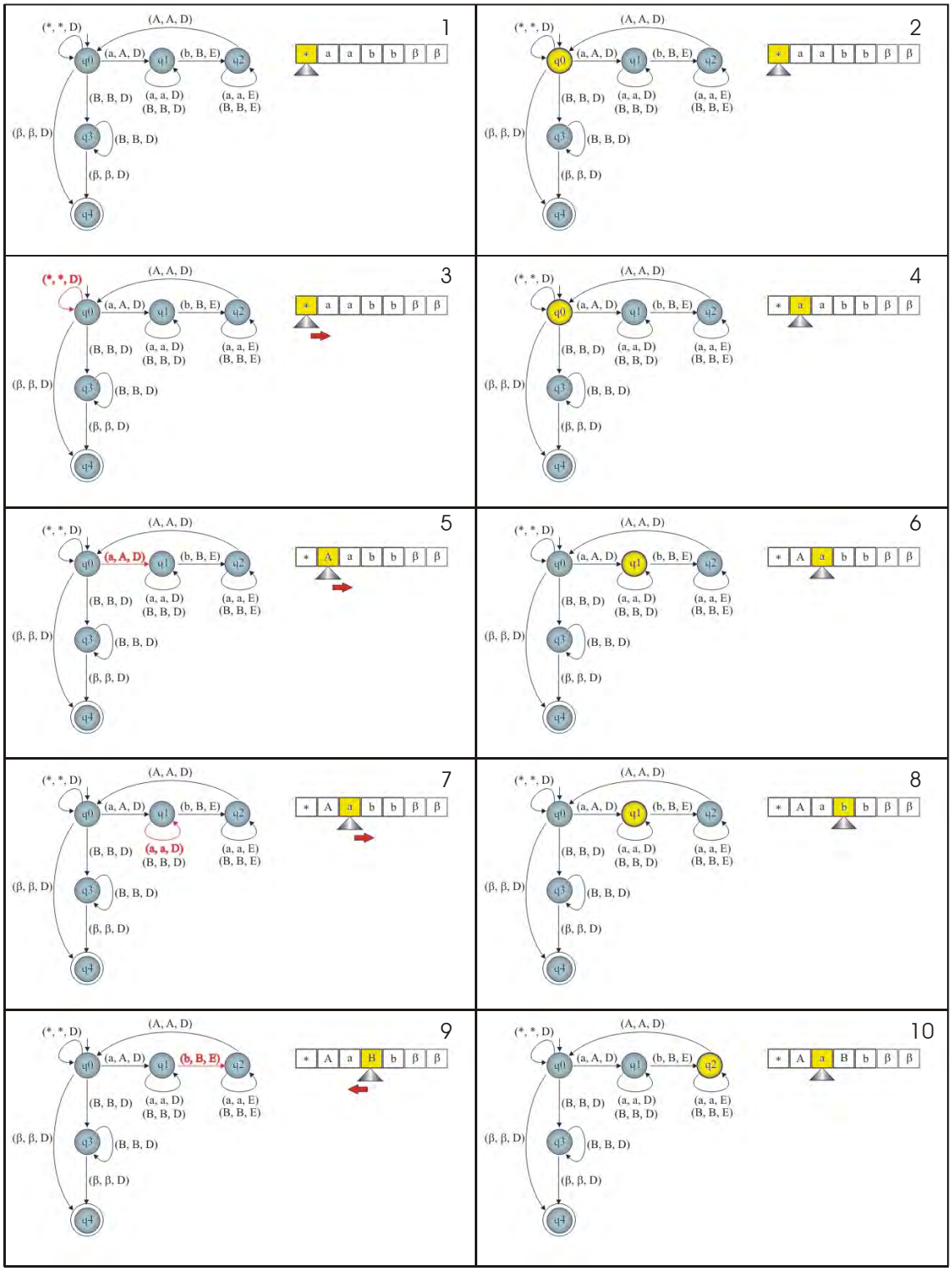
a) Animação Autômato Finito Determinístico

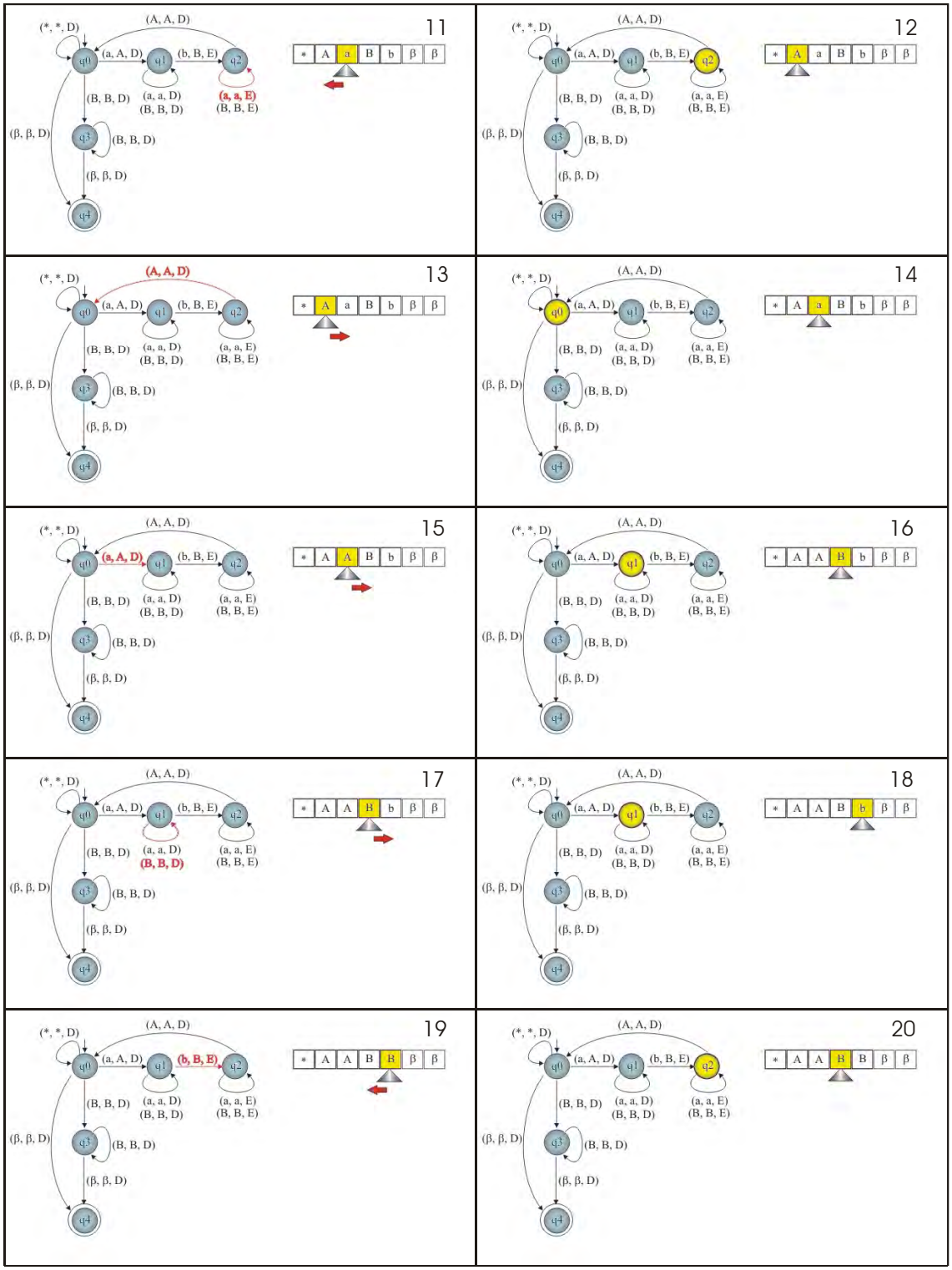


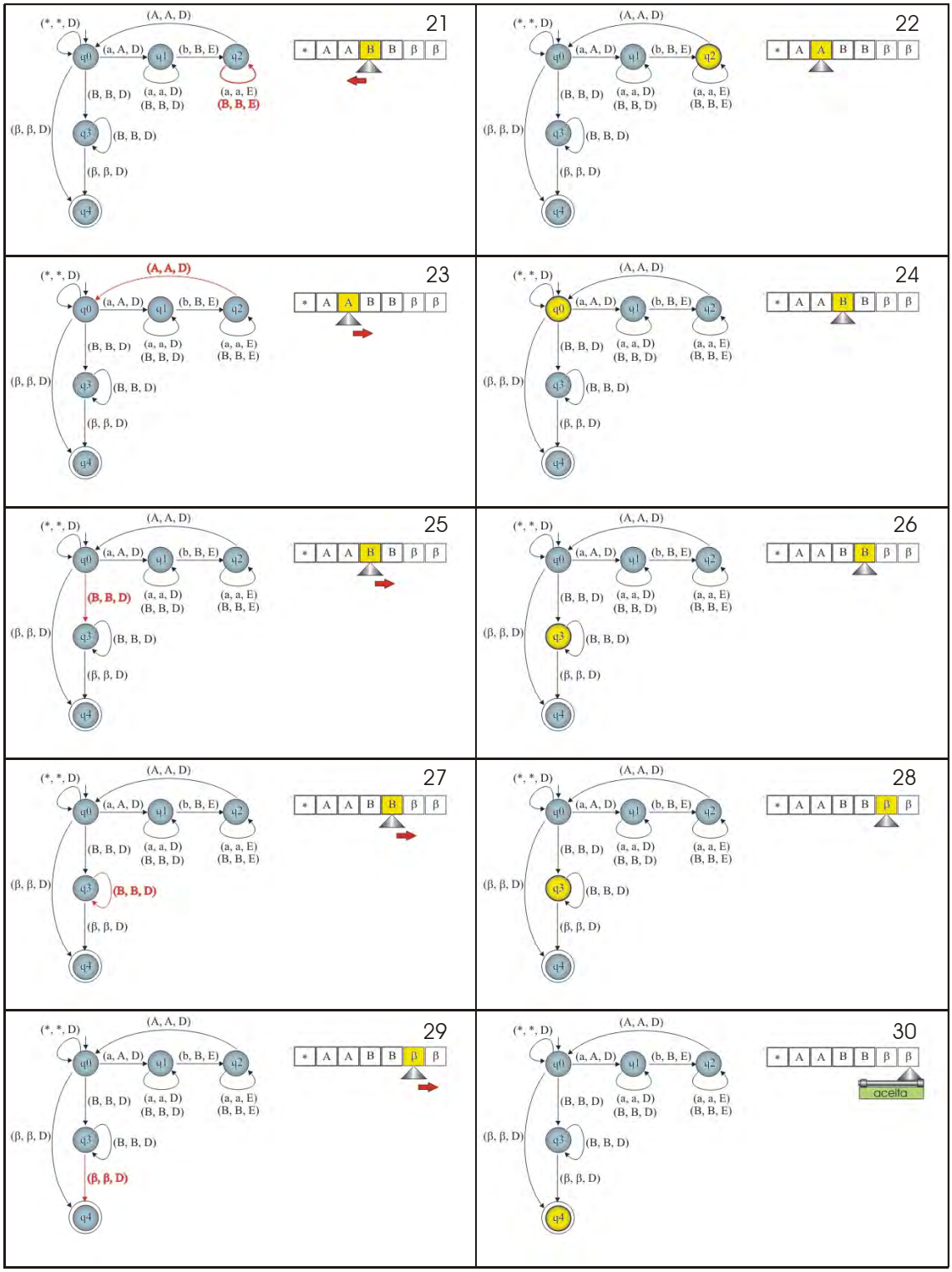
b) Animação Autômato com Pilha



c) **Animação Máquina de Turing**




11
12
13
14
15
16
17
18
19
20
(*, *, D)
(A, A, D)
(a, A, D)
(b, B, E)
(B, B, D)
(a, a, D)
(B, B, D)
(a, a, E)
(B, B, E)
(β, β, D)
(B, B, D)
(β, β, D)
q0
q1
q2
q3
q4
* A a B b β β
* A A B b β β
* A a B b β β
* A A B b β β
* A A B B β β


21
22
23
24
25
26
27
28
29
30
aceita

Bibliografia

[ACC 2000] ACCORSI, Fernando; MENEZES, Paulo Fernando Blauth. Animação Gráfica Baseada na Teoria de Autômatos. In: WORKSHOP ON FORMAL METHODS, 3., 2000, João Pessoa. **Proceedings…** João Pessoa: SBC, 2000. p. 122 – 127.

[ACC 2001a] ACCORSI, Fernando; MENEZES, Paulo Fernando Blauth; NEDEL, Luciana Porcher. Animação Gráfica Baseada em Autômatos Temporizados Sincronizados. In: WORKSHOP ON FORMAL METHODS, 4., 2001, Rio de Janeiro. **Proceedings...** Rio de Janeiro: SBC, 2001. Paginação irregular.

[ACC 2001b] ACCORSI, Fernando. **Autômatos Temporizados**: modelos Merrit-Modugno-Tuttle e Alur-Dill. 2001. Trabalho Individual (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[ACH 98] ACHARYA, Soam; SMITH, Brian. Experiment to Characterize Videos Stored on the Web. In: PHOTONICS WEST, 1998, San Jose, CA**. Proceedings of SPIE**. [S.l.]: SPIE Press, 1997. p. 166-178.

[ALU 92] ALUR, Rajeev; HENZINGER, Thomas A. Logics and models of real time: a survey. In: REX WORKSHOP, 1991. **Real-Time Theory in Pratice**. Berlin: Springer-Verlag, 1992. p. 74-106. (Lecture Notes in Computer Science, v.600).

[ALU 94] ALUR, Rajeev; DILL, David L. A theory of timed automata. **Theoretical Computer Science**, Amsterdam, v.126, n.2, p. 183-235, Apr. 1994.

[ALU 96] ALUR, Rajeev; DILL, David L. Automata-theoretic verification of real-time systems. In: HEITMEYER, C.; MANDRIOLI, D. (Org.). **Formal Methods for Real-Time Computing**. Great Britain: John Wiley & Sons Publishers, 1996. p. 55-82.

[ALU 99] ALUR, Rajeev. Timed automata. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, CAV, 11., 1999, Trento, It. **Computer Aided Verification**: proceedings. Berlin: Springer-Verlag, 1999. p. 8-22. (Lecture Notes in Computer Science, v.1633).

[AND 2001] ANDERSON, Richard et al. **Professional XML**. Rio de Janeiro: Ciência Moderna, 2001. 1266 p.

[APP 2000] APPLE COMPUTER INC. **QuickTime File Format**. [S.l.]: Apple Computer, 2000. Disponível em: <http://developer.apple.com /documentation/QuickTime/QTFF/index.html>. Acesso em: 10 abr. 2000.

[ARA 2000] ARAÚJO, Arnaldo de A.; GUIMARÃES, Sílvio 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.

[BUC 62] BÜCHI, R. On a decision method in restricted second-order arithmetic. In: INTERNATIONAL CONGRESS ON LOGIC, METHODOLOGY AND PHILOSOPHY OF SCIENCE, 1960. **Proceedings…** [S.l.]: Stanford University Press, 1962. p. 1-12.

[CHE 2001] CHESIRE, Maureen; WOLMAN, Alec; VOELKER, Geoffrey M.; LEVY, Henry M. Measurement and Analysis of a Streaming-Media Workload. In: USENIX SYMPOSIUM ON INTERNET TECHNOLOGIES AND SYSTEMS, 3., 2001, San Francisco. **Proceedings…** San Francisco, CA: USENIX, 2001. Disponível em: <http:/www.usenix.org/publications/library/proceedings/usits01/chesire/chesire.pdf>. Acesso em: 10 maio 2002.

[CHO 74] CHOUEKA, Yaacov. Theories of automata on w-tapes: a simplified approach. **Journal of Computer and System Sciences**, New York, v.8, n.2, p. 117-141, Apr. 1974.

[COM 90] COMPUSERVE INCORPORATED. **Graphics Interchange Format Programming Reference Version 89a**. [S.l.: s.n.], 1990.

[DAW 96] DAWS, C. et al. The tool KRONOS. In: DIMACS/SYCON WORKSHOP ON VERIFICATION AND CONTROL OF HYBRID SYSTEMS, 1995. **Hybrid Systems III**: Verification and Control: proceedings. Berlin: Springer-Verlag, 1996. p. 208-219. (Lectures Notes in Computer Science, v.1066).

[DUB 2002] DUBLIN CORE METADATA INITIATIVE. **Dublin Core Metadata Initiative**. Disponível em: <http://dublincore.org>. Acesso em: 10 maio 2002.

[FOL 90] FOLEY, J. et al. **Computer Graphics**: Principles and Practice. Reading, MA: Addison-Wesley, 1990. 1174 p.

[GEV 99] GEVERS, Theo; SMEULDERS, Arnold. The PicToSeek WWW Image Search System. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA COMPUTING AND SYSTEMS, 1999, Florence. **Proceedings…** New York: IEEE, 1999. v.1, p. 264-269.

[GIB 98] GIBSON, Jerry D. et al. **Digital compression for multimedia**: principles and standards. San Francisco: Morgan Kaufmann Publishers, 1998.

[GUD 97] GUDIVADA, V. N. et al. Information Retrieval on the World Wide Web. **IEEE Internet Computing**, New York, v.1, n.5, p. 58-68, Sept. 1997.

[HAR 96] HARDIN, R. et al. COSPAN. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, CAV, 8., 1996. **Computer Aided Verification:** proceedings. Berlin: Springer-Verlag, 1996. p. 423-427. (Lecture Notes in Computer Science, v. 1102).

[HEL 2001] HELLER, Rachelle S. et al. Using a Theoretical Multimedia Taxonomy Framework. **ACM Journal of Education Resouces in Computing**, New York, v.1, n.1, 2001.

[HOL 2001] HOLZNER, Steve. **Desvendando XML**. Rio de Janeiro: Campus, 2001. 858 p.

[HOP 79] HOPCROFT, J. E.; ULLMAN, J. D**. Introduction to Automata Theory, Languages and Computation.** Reading, MA: Addison-Wesley, 1979. 418 p.

[INT 2001] INTERNATIONAL ORGANISATION FOR STANDARDISATION. **Overview of the MPEG-7 Standard**. [S.l.: s.n.], 2001. Disponível em: <http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm>. Acesso em: 10 maio 2002.

[INT 96] INTERNATIONAL ORGANISATION FOR STANDARDISATION. **Short MPEG-1 description**. [S.l.: s.n.], 1996. Disponível em:

<http://mpeg.telecomitalialab.com/standards/mpeg-1/mpeg-1.htm>. Acesso em: 10 maio 2002.

[JAV 2002] JAVA.SUN.COM. **The source for Java(TM) Technology**. Disponível em: <http://java.sun.com>. Acesso em: 10 maio 2002.

[JDO 2002] JDOM ORG. **JDOM**. Disponível em: <http://www.jdom.org>. Acesso em: 10 maio 2002.

[KIR 2000] KIRK, Cheryl; PITTS-MOULTIS, Natanya. **XML Black Book**. São Paulo: Makron Books, 2000. 627 p.

[LAR 97] LARSEN, K. G.; PETTERSSON, P.; YI, W. UPPALL in a nutshell. **International Journal of Software Tools for Technology Transfer**, Berlin, v.1, n.1, p. 134-152, Oct. 1997.

[LAW 98] LAWRENCE, Steve; GILES, C. Lee. Searching the World Wide Web. **Science**, New York, v.280, n.5360, p. 98-100, Apr. 1998.

[LEE 98] LEE, G. S. A Classification of File Formats for Animation. In: WESTERN SYMPOSIUM ON INTERACTIVE GRAPHICS, 1998, Whistler. **Proceedings…** [S.l.: s.n.], 1998. Disponível em: <http://www.cs.ubc.ca/labs/imager/tr/ps/lee.1998b.ps.gz>. Acesso em: 10 maio 2002.

[LEW 2000] LEW, Michael S. Next-Generation Web Searches for Visual Content. **Computer**, New York, v.33, n.11, p. 46-53, Nov. 2000.

[LYN 87] LYNCH, Nancy A.; TUTTLE, Mark R. Hierarchical correctness proofs for distributed algorithms. In: ANNUAL SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 6., 1987. **Proceedings…** Vancouver: ACM Press, 1987. p.137-151.

[MAC 2000] MACHADO, Júlio H. A. P. **Hyper-Automaton:** hipertextos e cursos na Web usando autômatos finitos com saída. 2000. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

[MAC 2002] MACROMEDIA. **Macromedia Flash Application Development Center**. Disponível em: <http://www.macromedia.com/ desdev/ mx/ flash>. Acesso em: 12 maio 2002.

[MAG 2000] MAGEE, J. et al. Graphical Animation of Behavior Models. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 22., 2000, Limerick. **Proceedings…** New York: ACM Press, 2000. p. 499-508.

[MAR 99] MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento digital de imagens**. Rio de Janeiro: Brasport Livros e Multimídia, 1999.

[MCN 66] MCNAUGHTON, Robert. Testing and Generating infinite sequences by a finite automaton. **Information and Control**, [S.l.], v.9, n.5, p. 512-530, Oct. 1966.

[MEN 2000] MENEZES, Paulo Fernando Blauth. **Linguagens Formais e Autômatos**. Porto Alegre: Sagra Luzatto, 2000.

[MEN 99] MENEZES, Paulo Blauth; MACHADO, Júlio Pereira. Web Courses are Automata: a Categorial Framework. In: WORKSHOP ON FORMAL METHODS, 2., 1999, Florianópolis. **Proceedings…** Florianópolis: SBC, 1999. p. 79-88.

[MER 91] MERRIT, M.; MODUGNO, F.; TUTLE, M.R. Time-constrained automata. In: INTERNATIONAL CONFERENCE ON CONCURRENCY THEORY, CONCUR, 2., 1991. **Concurrency**

Theory. Berlin: Springer-Verlag, 1991. p. 408-423. (Lecture Notes in Computer Science, v. 527).

[MIC 2002] MICROSOFT CORPORATION. **AVI RIFF file reference**. Disponível em: <http:// msdn.microsoft.com/ library/ en-us/ wcedshow/ htm/ _dxce_dshow_avi_riff_file_reference.asp>. Acesso em: 10 fev. 2002.

[POR 2001] PORTO, A. M. C.; COSTA, R. C. Especificando formalmente o comportamento de apresentações multimídia interativas. **Revista Alcance**, Itajaí, SC, v.8, n.1, p. 31-37, 2001.

[PUE 88] PUEYO, Xavier; TOST, Daniela. A Survey of Computer Animation. **Computer Graphics Forum**, Amsterdam, v.7, p. 281-300, 1988.

[REJ 99] REJAIE, Reza; HANDLEY, Mark; YU, Haobo; ESTRIN, Deborah. Proxy Caching Mechanism for Multimedia Playback Streams in the Internet. In: INTERNATIONAL WEB CACHING WORKSHOP, 4., 1999, San Diego. **Proceedings…** San Diego: NLANR, 1999. Disponível em: <http://workshop99.ircache.net/Papers/rejaie-abstract.html>. Acesso em: 15 maio 2002.

[SAL 69] SALOMAA, Arto. **Theory of Automata**. New York: Pergamon Press, 1969. 263 p.

[SCL 97] SCLAROFF, S.; TAYCHER, L.; LACASCIA, M. ImageRover: Content-Based Image Browser for the World Wide Web. In: WORKSHOP ON CONTENT-BASED ACCESS OF IMAGE AND VIDEO LIBRARIES, 1997, Puerto Rico. **Proceedings…** New York: IEEE, 1997. p 2-9.

[SMI 97] SMITH, John R.; CHANG, Shih-Fu. Visually Searching the Web for Content. **IEEE Multimedia**, New York, v.4, n.3, p. 12-20, July 1997.

[TAN 97] TANENBAUM, A. S. A World Wide Web. In: TANENBAUM, A. S. **Redes de Computadores**. Rio de Janeiro: Campus, 1997. p. 776-821.

[THA 85] THALMANN, Nadia M.; THALMANN, Daniel. **Computer Animation**: Theory and Practice. Tokyo: Springer-Verlag, 1985. 239 p.

[THO 90] THOMAS, W. Automata on infinite objects. In: LEEUWEN, J. (Org.). **Handbook of Theoretical Computer Science:** Formal Models and Semantics. Amsterdam: Elsevier, 1990. p.133-191.

[VET 94] VETTER, Ronald J.; SPELL, Chris; WARD, Charles. Mosaic and the World-Wide Web. **IEEE Computer Magazine**, New York, v.27, n.10, p. 49-57, Oct. 1994.

[W3C 2002a] WORLD WIDE WEB CONSORTIUM. **Scalable Vector Graphics**. Disponível em: <http://www.w3.org/ Graphics/ SVG/ Overview.htm>. Acesso em: 10. maio 2002.

[W3C 2002b] WORLD WIDE WEB CONSORTIUM. **W3C Synchronized Multimedia Home Page**. Disponível em: <http://www.w3.org/ AudioVideo>. Acesso em: 10 maio 2002.

[ZEL 85] ZELTZER, D. Towards an integrated view of 3-D computer animation. **The Visual Computer,** New York, v.1, n.4, p. 249-259, 1985.