Animação de Algoritmos tornada Sistemática

Maria João Varanda
GEPL / Departamento de Informática
Universidade do Minho
mjp@di.uminho.pt

Pedro Rangel Henriques
GEPL / Departamento de Informática
Universidade do Minho
prh@di.uminho.pt

1. SUMÁRIO

Neste artigo vamos propor a arquitectura do sistema Alma, um sistema para animar algoritmos programados em diferentes linguagens, oferecendo uma interface de visualização controlável pelo utilizador. Assim, o que na nossa opinião caracteriza esta proposta é: a independência relativamente à aplicação e à linguagem de programação; a existência de uma linguagem de visualização versátil (permitindo adaptá-lo às necessidades do utilizador).

Como aplicações possíveis do Alma, destacamos: a animação de algoritmos, como apoio ao ensino da programação e como instrumento da didáctica da matemática; a análise de resposta, para apoio à correcção de provas de avaliação; a interpretação (visual) de documentos anotados.

1.1 Palavras-chave

Animação, Algoritmos, Visualização, Simulação, EAC.

2. INTRODUCÃO

Neste artigo pretendemos defender a ideia de que é possível fazer a animação dos algoritmos subjacentes aos programas de uma forma sistemática, isto é, sem depender nem de um programa (ou classe de programas) específico, nem tão pouco de uma linguagem de programação determinada. Para tal, propomo-nos recorrer a métodos, técnicas e ferramentas tradicionalmente usadas para reconhecer, representar e manipular o significado dos programas, no contexto do desenvolvimento formal e automático de compiladores.

Para corroborar a nossa ideia, iremos apresentar (secção 3) a arquitectura de um sistema, que designamos por Alma, o qual é capaz de analisar o programa fonte, que se pretende animar —ou seja, visualizar o fluxo de controlo e/ou dos dados durante uma simulação da sua execução — e fornecer ao seu utilizador as facilidades necessárias para indicar que secções de código e que subconjunto de variáveis deseja acompanhar durante uma pseuda-execução do programa.

Como se dirá na secção 4, tal ambiente deve ser versátil na forma de visualisar os objectos em análise, para ser facilmente adaptado a diferentes contextos.

Para além do interesse teórico de investigação na área do processamento formal de linguagens, o grande motivo que nos levou a conceber o Alma foi, sem dúvida, fornecer uma ajuda aos programadores e aos formadores (professores) de informática —permitindo-lhes validar pragmaticamente os algoritmos que desenharam para resolver determinado problema, bem como a própria implementação (na LP escolhida) que escreveram.

Contudo, parece-nos que a utilidade de um tal ambiente de animação, visualização, pode ser igualmente aproveitada em outras áreas didáticas, como por exemplo no ensino da matemática —onde o principal uso, poderá ser na explicação visual dos princípios de cálculo descritos pelo algoritmo em análise. A discussão deste campo de aplicação, bem como a apresentação de exemplos do seu uso no campo da programação, constituirão o assunto da secção 5.

Para situar o leitor na área e poder comparar a abordagem aqui proposta com as soluções várias existentes, dedicaremos a secção 6 à apresentação de alguns outros sistemas de animação de algoritmos de que temos tido conhecimento.

3. O SISTEMA ALMA

Nesta secção vamos apresentar o sistema Alma, referindo:

- o princípio que liderou a sua concepção/desenvolvimento e que faz a distinção em relação aos outros
- a arquitectura do sistema
- as finalidades
- as fases do seu desenvolvimento

A animação de um algoritmo é um tipo de visualização dinâmica das principais abstracções expressas pelo algoritmo subjacente a um programa.

O objectivo é perceber o funcionamento de aplicações cuja animação vai sendo progressivamente despoletada como resposta a acções do utilizador, tornando a interacção mais realista.

Quando pensamos em visualizar um programa (inspeccionar o fluxo de controlo e o comportamento de variáveis) temos, logo à partida, duas grandes escolhas: fazê-lo durante a execução do código (debugging); ou simular a execução num outro ambiente.

O executável do programa, por si só, em nada ajuda a entender o funcionamento do programa. Por outro lado, a execução em tempo real de um programa é demasiadamente rápida para haver percepção do fluxo de controlo e até da evolução dos valores das variáveis. Tal análise só é possível com a ajuda de um debugger, mas

para isso o compilador tem de ser prevenido, de modo a gerar código adicional apropriado.

Face ao exposto, parece ser uma boa solução recorrer a simulações do comportamento do programa num ambiente que permita estudar o seu comportamento.

Uma segunda escolha que tem de ser feita de seguida é se se deve, ou não, incluir no programa fonte instruções e/ou marcas especiais a indicar que blocos de código e que variáveis é que se pretende estudar durante a animação. Em caso de escolha afirmativa, será necessário definir a linguagem de animação que o programador deve conhecer para incluir nos seus programas.

Aqui a escolha foi, claramente, pela negativa —não pretendemos que haja uma linguagem de marcação de animação extra, pois não desejamos que os programas a animar tenham de ser previamente alterados. Tal decisão implica que o ambiente de animação tenha capacidade para: analisar o programa fonte —identificando o seu fluxo de controlo e as suas variáveis; permitir ao programador controlar, na hora da animação, os blocos e variáveis a estudar.

Uma tentativa de implementação do primeiro *case-study* demonstrou que existe, em certas linguagens, uma grande dificuldade em aceder a certos tipos de informações sobre o programa fonte sem que este seja modificado. Nessa primeira tentativa de animação de programas —que teve por objectivo dar-nos uma boa percepção dos requisitos e dificuldades do problema— foram criados dois programas em Java: um programa fonte, de pequenas dimensões, que tinha como objectivo a inserção de valores num array; e um outro, o simulador/animador, que servia para inspeccionar os conteúdos das variáveis do primeiro, de tanto em tanto tempo, e construía uma janela de visualização dessas variáveis. Neste caso, o programa de animação era muito específico porque apenas permitia a visualização de variáveis com determinados nomes e tipos.

Tornava-se então necessário generalizar este programa de animação de modo a ele conseguir inspeccionar todas as variáveis de qualquer programa fonte. Verificou-se, então, que o problema se tornava bastante penoso de ser resolvido desta forma tão específica, baseado só em programas Java preocupados com a interface para interacção com o utilizador.

Conclui-se, no entanto, que um sistema de animação que tenha acesso a certas informações (nomes e tipos de variáveis e estruturas de controlo de fluxo) pode criar interfaces de animação, permitindo que o utilizador não só programe a animação, mas também, interactue com a própria simulação da execução do programa.

Optou-se, assim e no domínio da animação de linguagens textuais, por uma solução mais genérica, descrita diagramaticamente na figura 1.

Para simular a execução, permitindo um controlo total sobre as variáveis e blocos a inspeccionar, decidimos construir um animador baseado na árvore de sintaxe abstrata do programa fonte decorada com valores de atributos. A ideia chave é escolher o conjunto de atributos significativos para uma tarefa específica de visualização.

Assim, pretende-se usar o front-end de um compilador[8] para reconhecer a estrutura do programa fonte (reconhecer símbolos, as regras de derivação) e construir a árvore decorada (calculando o valor dos atributos associados a cada nó da árvore).

Inspirados na tecnologia tradicional da compilação (bem conhecida e bem fundada), o princípio em que se vai basear o sistema Alma é de que a Árvore de Sintaxe Decorada (ASD) juntamente com o Diagrama do Fluxo de Controlo (DFC) e o Grafo de Vida das Variáveis (DAG, Directed Aciclic Graph) constituem o conjunto de estruturas de dados adequado para manter a informação relevante (nomes e endereços de variáveis e estruturas de controlo de fluxo) a um sistema que vai permitir a animação do algoritmo e a visualização do conteúdo de variáveis.

Se pretendermos animar um algoritmo implementado em qualquer tipo de linguagem de programação (imperativa, funcional, lógica, orientada ao objecto, ou até linguagens visuais), o *front-end* deverá ser alterado de forma a conseguir extrair dessa linguagem fonte a informação necessária para a animação. Mas tal alteração é sistematica, tendo por base a gramática de atributos da linguagem fonte a analisar, e pode ser automatizada recorrendo às tradicionais ferramentas de geração de compiladores.

O sistema de animação constitui o *back-end* desse mesmo compilador e deve ser construído de forma genérica de modo a poder funcionar com qualquer *front-end*.

Os nomes e tipos das variáveis são utilizados para criar um interface de programação de animação. Essa interface deve permitir escolher quais as variáveis, estruturas, funções, etc. a animar/visualizar e também os aspectos (representações visuais) a dar a cada uma dessas entidades.

Por outro lado, vai ser necessário criar uma interface para permitir a inserção de dados no programa. Essa interface precisa de ser notificada sempre que uma instrução do tipo *read* surge no programa.

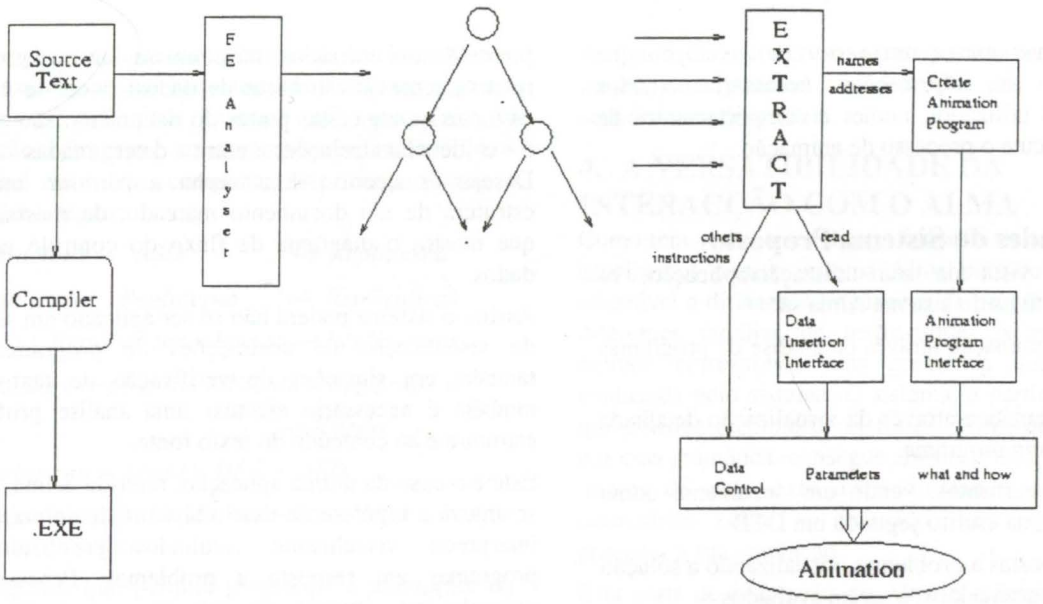

Figura 1: Esquema de blocos do sistema Alma

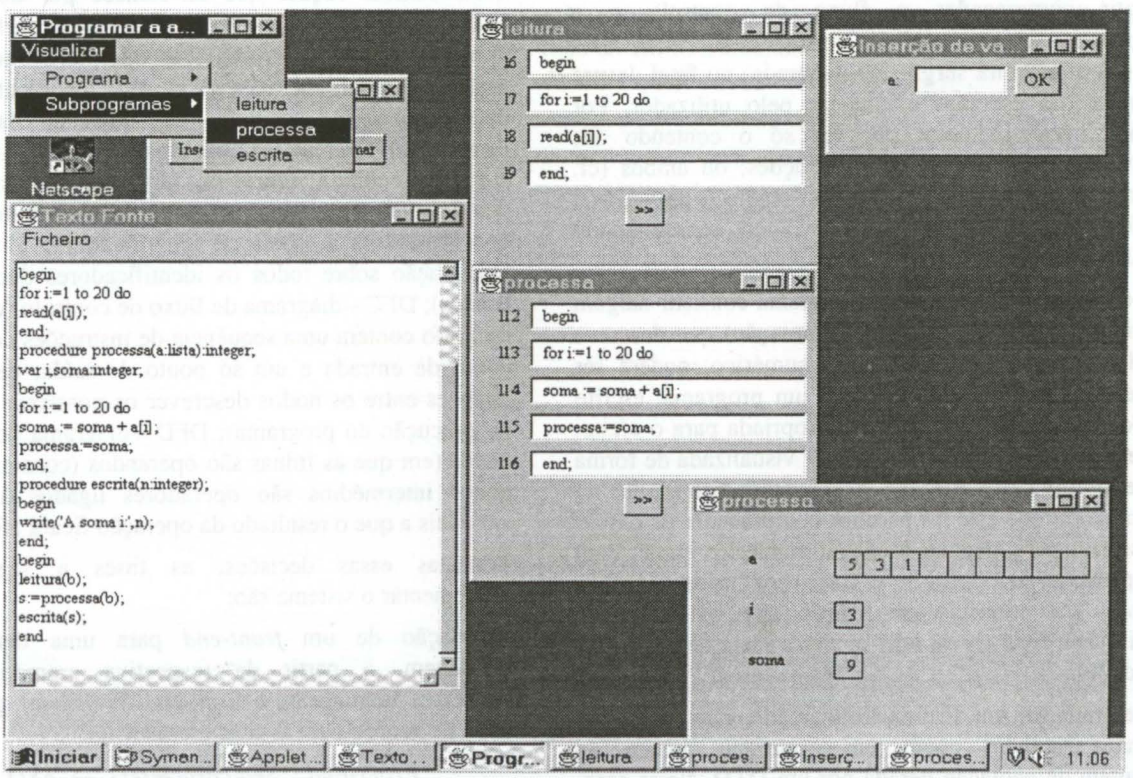

Figura2: Aspecto Geral das Interfaces usadas para Visualização

Por último, dever existir um *gestor de animação* que recolha todas as informações necessárias (dados fornecidos pelo utilizador, nomes e comportamentos de variáveis) e execute o processo de animação.

3.1 Finalidades do Sistema Proposto

Do ponto de vista da sua utilização/aplicação, os principais objectivos do sistema Alma são:

- ensino da programação através da análise de programas / algoritmos,
- ensino da matemática através da visualização detalhada dos cálculos de um programa,
- análise de documentos, vendo um documento como sendo um programa escrito segundo um DTD
- análise de respostas a problemas, visualizando a solução gerada pelos programadores a serem avaliados.

A finalidade do sistema proposto é analisar um texto de entrada (que poderá ser um programa clássico, um algoritmo, um documento, etc.) e extrair a informação considerada importante que, depois de visualizada de uma forma clara e sugestiva, permite entender melhor o significado do texto fonte.

Se o texto de entrada for um programa escrito numa linguagem imperativa, a informação a extrair terá que permitir compreender o fluxo de controlo e o comportamento das estruturas de dados. A forma como essa informação irá surgir (ser mostrada) no final deverá depender das escolhas efectuadas pelo utilizador. Por exemplo, pode-se optar por ver só o conteúdo das variáveis, ou só o fluxo das instruções, ou ambos (cf. figura 2); poder-se-á ver o valor instantâneo da variável (cf. figura 3), mas também a sua evolução representando os valores no eixo do tempo.

Se, por outro lado, a entrada do sistema consistir nalgum algoritmo (escrito numa qualquer notação) que descreva um determinado tipo de cálculo numérico, poderá ser também interpretado como sendo um programa, escrito em Fortran ou outra linguagem apropriada para o efeito. Neste caso, a informação deverá ser visualizada de forma a que o utilizador perceba o algoritmo subjacente ao programa porque este irá permitir compreender os passos necessários para efectuar o cálculo numérico em si. Para isso procurar-se-á mostrar o traço dos valores que as variáveis vão tomando em ligação com os passos do algoritmo (a figura 4 dá uma modesta ideia deste tipo de facilidade).

Se a entrada fôr um documento marcado, este deverá ser encarado também como um programa escrito na linguagem de anotação gerada por um DTD. Neste caso, o sistema de animação também pode reconhecer a estrutura do documento —de acordo com o seu tipo, definido pelo DTD subjacente a esse documento— e, então, mostrar a sua forma. Assim será possivel evidenciar determinado tipo de anotações, segundo a pretensão do utilizador, tal como se faz nos programas relativamente às estruturas de dados; poder-se-á ilustrar os locais aonde certas partes do documento são referidas e evidenciar relações entre determinadas marcas. Desejamos que o Alma venha a permitir analisar a estrutura de um documento marcado, da mesma forma que mostra o diagrama de fluxo do controlo e/ou dos dados.

Assim, o sistema poderá não só ser aplicado em situações de visualização de abstracções de programas, mas também em situações de verificação de textos onde também é necessário efectuar uma análise profunda à estrutura e ao conteúdo do texto fonte.

Este é o caso da última aplicação, referida acima, em que se antevê a hipótese de usar o sistema de animação para interpretar visualmente resultados produzidos por programas em resposta a problemas. Pense-se, por exemplo, na situação em que se quer verificar um conjunto de programas cujo fim é listar um caminho entre 2 vértices de um grafo, ou gerar uma sequência de passos para recorte de uma figura.

3.2 Desenvolvimento do Sistema

Para sermos consequentes com o princípio subjacente ao Alma, que foi apresentado e se encontra esquematizado no diagrama de blocos da figura 1, o desenvolvimento deste sistema requer que se comece por definir com exactidão a informação genérica necessária para qualquer animação e a sua representação interna. Como se disse atrás, o Alma irá basear o seu funcionamento na manipulação das seguintes estruturas de dados (que representam a semântica do texto fonte): ASD --árvore de sintaxe decorada (com os atributos de cada nodo devidamente valorados), contendo um atributo (TabId) que representa a tabela de identificadores global (com informação sobre todos os identificadores declarados e usados); DFC --diagrama de fluxo de controlo (cada nodo do grafo contém uma sequência de instruções com um só ponto de entrada e um só ponto de saída, devendo as ligações entre os nodos descrever os possíveis caminhos na execução do programa); DFD --diagrama de fluxo de dados (em que as folhas são operandos (constantes) e os nodos intermédios são operadores ligados à lista de variáveis a que o resultado da operação ficará afecto).

Tomadas essas decisões, as fases a seguir para implementar o sistema são:

- Criação de um *front-end* para uma determinada linguagem, a partir da respectiva gramática (para diferentes linguagens, o front-end deverá ser gerado de novo de acordo com as suas gramáticas).

$$Front\text{-}End : Seq(caracteres) \rightarrow ASD$$

- Criação de um *back-end* que, a partir da informação extraída do resultado produzido pelo *front-end*, forneça a necessária interface para programação da animação e realize a visualização pretendida (este back-end será independente da linguagem fonte). Este *back-end* será

implementado com base no algoritmo de um *Tree-Walker Evaluator* que ao visitar cada nodo da ASD (segundo um percusros pré-definido) aplicará as operações de visualização, filtragem e animação adequadas. Dividimos o *back-end* em quatro componentes principais, que se especificam a seguir:

$$\textit{Extractor}: \quad \textit{ASD} \rightarrow \textit{RepInterna}$$

$$\textit{Visualizador}: \textit{RepInterna} \rightarrow \textit{RepGraficas}$$

$$\textit{Filtro}: \textit{RepGraficas} \times \textit{Estado} \rightarrow \textit{RepGraficas}$$

$$\textit{Animador}: \textit{RepInterna} \times \textit{Estado} \times \textit{RepGraficas} \rightarrow \textit{Estado}$$

sendo $\textit{RepInterna} = \textit{TabId} \times \textit{DFC} \times \textit{DFD}$

Assim o desenvolvimento do back-end envolve a criação das seguintes interfaces:

- uma interface que permita programar a animação, ou seja, indicar os subprogramas (dos quais queremos visualizar a sequência de instruções) e as variáveis (das quais queremos visualizar o conteúdo); essa interface terá ainda de permitir a definição do aspecto com que os objectos seleccionados serão apresentados
- uma interface para visualizar a sequência de instruções e o conteúdo das variáveis, de acordo com os requisitos expressos no item anterior

Para enriquecer o sistema de animação Alma, pensámos que uma terceira interface, agora ao nível do front-end, poderá ainda ser incluida. Trata-se de criar uma interface que permita a inserção/edição do texto fonte e o seu armazenamento.

A figura 2, retirada do protótipo que desenvolvemos, mostra uma possível concretização das interfaces que acabámos de descrever.

4. A VERSATIBILIDADE DA INTERACÇÃO COM O ALMA

Como tem vindo a ser dito, a filosofia de construção que escolhemos para o sistema Alma permite que ele seja adaptável a diferentes linguagens de programação e até a diferentes famílias de textos-fonte. A cada texto de entrada corresponde uma gramática que deverá ser conhecida pelo sistema. O sistema, a partir do momento que consiga reconhecer as frases da linguagem definida por essa gramática, consegue construir uma representação interna de cada texto disponibilizando as informações necessárias sobre o conteúdo dos programas para proceder à sua animação.

Esta parte de adaptação do Alma a diferentes situações, embora não seja tarefa a ser executada pelo utilizador final, é um trabalho que pode ser realizado sistemática e facilmente pelo implementador (se este fôr especialista em desenvolver processadores de linguagens).

Como se disse acima, era desejável que o Alma fornecesse um editor de texto. Usando novamente a tecnologia da compilação, verifica-se que a gramática que se define para gerar o reconhecedor (que constitui o front-end) pode ser, igualmente, usada para gerar um Editor estruturado Dirigido pela Sintaxe. Se tal fôr feito, a interacção com o animador é francamente melhorada (não só em funcionalidade, como também em facilidade de utilização).

O sistema que queremos construir deve também permitir escolher alguns tipos de visualizações a obter. O tipo de visualização depende do objectivo que se pretende atingir ao fazer a análise de um programa. Sabendo que um programa tem sempre subjacente uma determinada estrutura, as visualizações irão basear-se em diagramas hierárquicos, em grafos, em diagramas de controlo de fluxo, em representações visuais das estruturas de dados e outros esquemas que de alguma forma contribuirão para o objectivo da análise que se quer efectuar. Os esquemas que são visualizados devem ser dinâmicos, ou seja, devem ser alterados à medida que a simulação da execução do programa vai progredindo. Para fazer tal escolha, torna-se necessário que o back-end forneça uma interface adequada a qual tem de permitir escolher facilmente os blocos a analisar e as variavéis, ou a textos anotados inspeccionar. Porém esta interface precisa de dar alguma liberdade para que o utilizador possa seleccionar a representação visual com que pretende analisar o seu texto.

O desenvolvimento da interface que respeite estes requisitos é assumidamente um dos maiores desafios do projecto, com que estamos actualmente a lidar, mas para o qual antevemos uma solução mais ou menos formal com base no tipo dos objectos com que se quer trabalhar.

5. DISCUSSÃO DE APLICAÇÕES; EXEMPLOS

Como já foi dito em secções anteriores, o sistema Alma vai ser aplicado a várias classes de textos de entrada produzindo, como resultado, as visualizações julgadas apropriadadas para cada caso —nesta secção apresentamos alguns exemplos.

Pretende-se ilustrar a ideia que vimos a defender, mostrando o género de resultados que o sistema Alma deverá vir a produzir face aos tipos de entradas que na secção identificámos como possíveis finalidades para aplicação desse animador de programas.

5.1 A aplicação do Alma à animação de programas imperativos

O primeiro exemplo que mostramos ilustra a situação prevista inicialmente: a aplicação do Alma à animação de programas, neste caso escritos em Pascal. A figura 3 exemplifica diversos ecrans que o Alma poderia apresentar para se inspeccionar o comportamento do

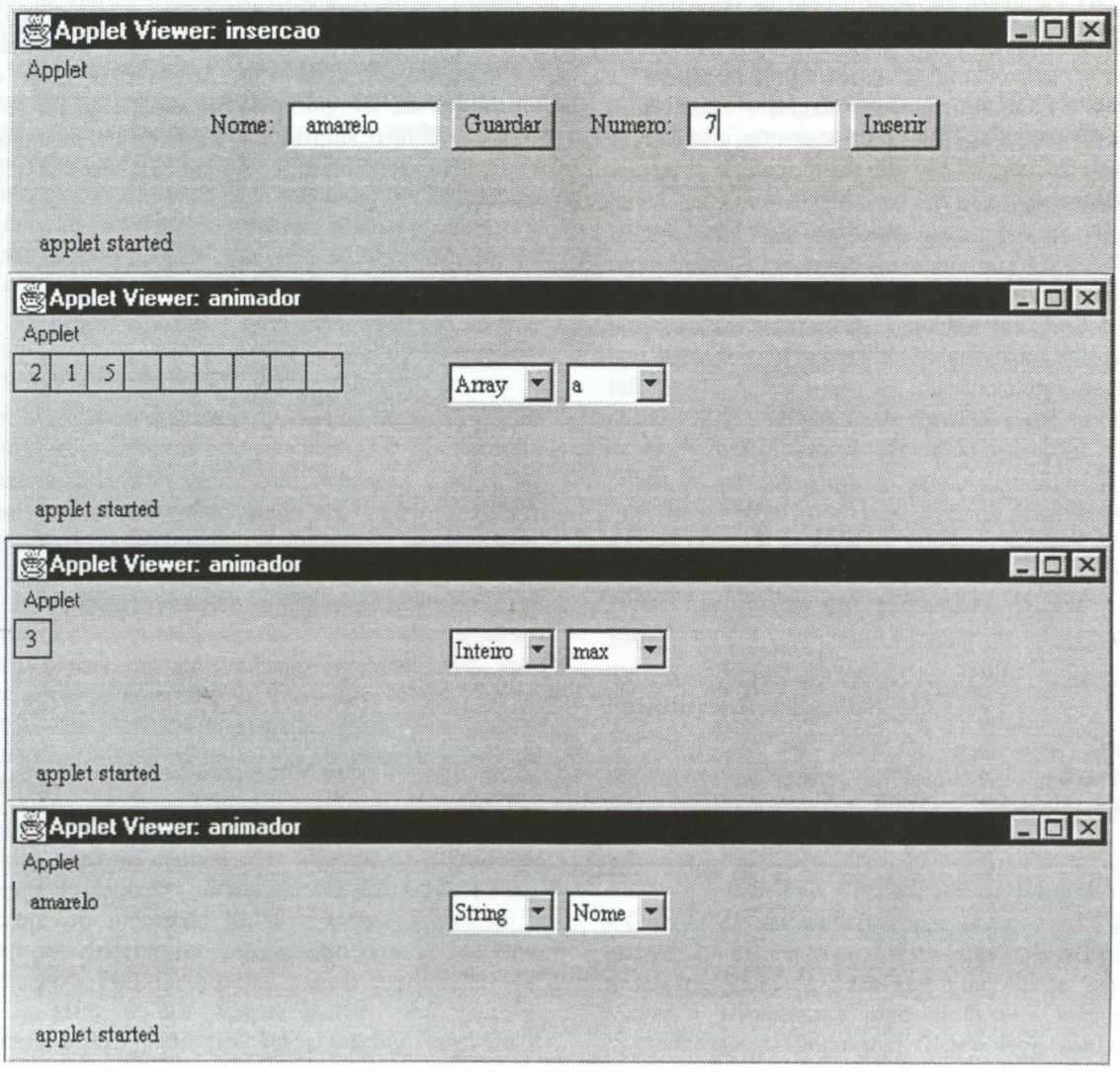

Figura 3: Animação de um programa em Pascal

programa Pascal seguinte

```
PROGRAM insercao;
   VAR Nome:string[10];   max, Numero:integer;
     a:ARRAY[1..10] of integer;
   BEGIN
    max:=0; read(nome);
    repeat
     max:=max + 1;
     read(num);
     a[max]:=num;
    until max=10;
   END.
```

Neste caso, muito simples (para efeitos de exposição) e portanto pouco interesante, o programador poderá ver a evolução dos valores contidos nas variáveis escolhidas à medida que a introdução de dados vai progredindo, tal como se passaria na animação de programas mais longos e complexos envolvendo ordenações de arrays, pesquisas, etc.

5.2 A aplicação do sistema Alma à animação de cálculos matemáticos

Para exemplificar a ideia de usar a aplicação do sistema Alma à compreensão de cálculos matemáticos, mostramos o caso da análise e visualização de um possível algoritmo para o cálculo do factorial, que no exemplo está implementado num programa escrito em FORTRAN.

Segue-se a listagem do programa:

```
FACT=1;
READ*, N
FACTORIAL:  DO I=1,N
            FACT = FACT * I
            END DO FACTORIAL
PRINT*, FACT
```

Na figura 4 pode apreciar-se uma forma de visualizar o respectivo algoritmo, sendo apresentados o diagrama de fluxo de controlo (que permite compreender a ordem de execução das operações) e o diagrama de fluxo de dados em que se pode analisar a forma como os valores são afectados às variáveis e depois usados.

Em cada caixa do diagrama surge uma instrução e um valor. Esse valor corresponde ao valor actual (após ter sido executada essa instrução) da variável definida. À medida que a simulação da execução do programa vai correndo, será marcada a instrução que está a ser efectuada e o valor da variável é actualizado. O utilizador

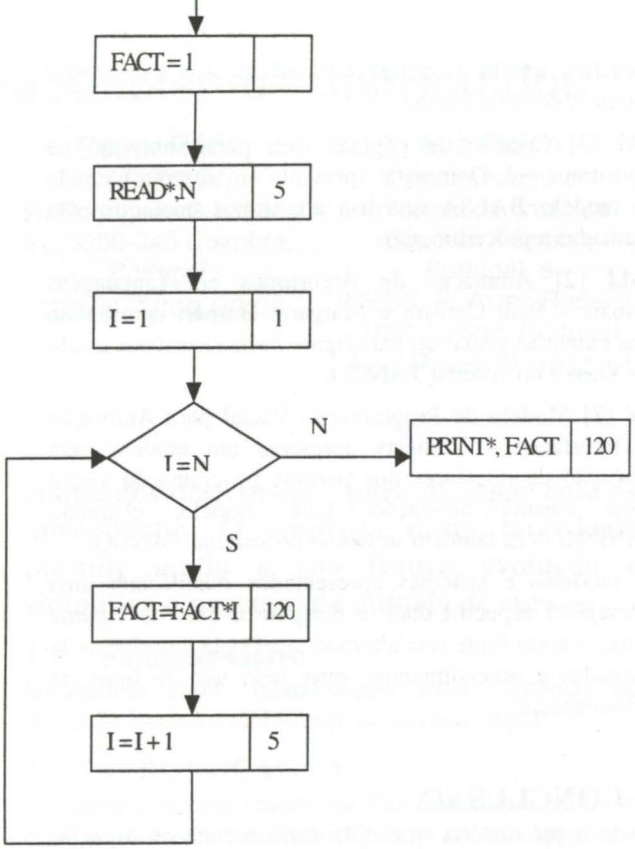

Figura 4: Animação de um programa em Fortran

poderá controlar a visualização premindo um botão que vai passando de instrução a instrução.

6. OUTROS MODELOS E SISTEMAS

Para que se possa fazer um estudo comparativo, apresentamos nesta secção alguns modelos de ferramentas e/ou sistemas de animação que têm sido propostos por diversos autores.

BALSA (Brown ALgoritm Simulator and Animator) foi o primeiro sistema textual de animação de algoritmos, surgiu em 1981 e tornou-se um modelo a seguir nos trabalhos posteriores. Em 1987, surgiu o sistema ANIMUS [4] que sendo também um sistema textual, inclui restrições temporais na construção de animações de algoritmos. Mais tarde, em 1989, o sistema ALADDIN [5] permite especificar visualmente a animação de programas textuais. TANGO [6] surge em 1990 com um modelo de animação com uma semântica precisa baseada no paradigma *Path Transition*.

Mais recentemente, foram apresentados alguns trabalhos que têm vindo a dar o seu contributo no sentido de criar um sistema que evite uma programação exaustiva e inflexível de animações.

BNR [1] Animação de Algoritmos Java usando Tipos de Dados Auto-animados – M. H. Brown apresenta um sistema chamado JELIOT que, embora não recorra a código adicional, usa tipos de dados especiais. O código

Java fica sujeito a uma pré-compilação e a animação é gerada automaticamente.

BM [3] Criação de páginas web para animação de algoritmos – J. Domingue apresenta um sistema baseado no modelo BALSA que usa algoritmos anotados com chamadas a procedimentos.

PMJ [2] Animação de Algoritmos em Linguagens Visuais – Paul Carlson e Margaret Burnett apresentam uma extensão visual do paradigma *Path Transition* usado por Stasko no sistema TANGO.

DV [7] Modelo de Programação Visual para Animação de Interfaces – Vodislav apresenta um modelo para animação de interfaces que permite programação visual dessa mesma animação. Este modelo, chamado HANDMOVE, também se baseia no sistema TANGO.

Os modelos e sistemas apresentados manifestam uma indesejável especificidade e obrigam a que, de alguma forma, o texto fonte seja alterado, quer pela introdução de chamadas a procedimentos, quer pelo uso de tipos de dados especiais.

7. CONCLUSÃO

Desde a pré-história que o Homem reconhece especial importância à expressão visual como forma de comunicação —recordem-se as gravuras de Foz Côa e tantas outras mostras de arte rupestre.

Na era informática que se vive desde meados deste século, tem por isso vindo a ser preocupação constante possibilitar o uso de interfaces gráficas que permitam estabelecer entre o utilizador e o computador uma comunicação não limitada à forma textual.

Neste contexto, a importância das linguagens (de programação) visuais é totalmente reconhecida. Um grande empenho tem sido dedicado ao seu processamento e até à especificação formal com vista a possibilitar a automatização do seu tratamento.

É neste quadro que propomos o sistema Alma, que foi discutido ao longo do presente artigo, como o intuito de permitir automatizar a interpretação visual de textos estruturados.

Feito o primeiro protótipo e traçadas as directivas exactas que queremos imprimir ao desenvolvimento do projecto, resta-nos decidir as ferramentas de geração automática de compiladores que queremos usar e a maneira mais adequada de representar as estruturas de dados necessárias, para passarmos à construção do back-end do Alma e à criação de alguns front-end para testes.

8. REFERÊNCIAS

[1] M. H. Brown, M. A. Najork, and R. Raisamo. A java-based implementation of collaborative active textbooks, DEC Systems Research Center, 1997.

[2] Paul Carlson, Magaret Burnett, and Jonathan Cadiz. A seamless integration of algorithm animation into a visual programming language. Department of Computer Science, Oregon state University, May 1996.

[3] J. Domingue and P. Mulholland. Staging software visualizations on the web, The Open University, 1997.

[4] R. A. Duisberg. Animation using temporal constraints: An overview of the animus system. Human-Computer Interaction 275-307, Volume 3, Number 3, August 1988.

[5] E. Helttula, A. Hyrskykari, and K. Raiha. Graphical specifications of algorithm animations with aladdin. 22nd Hawaii International Conference on System Sciences, Kailua-Kona, HI, January 1989.

[6] John T. Stasko. Simplifying algoritm animation with tango, IEEE Workshop on Visual Languages, Skokie, IL, October 1990.

[7] D. Vodislav. A visual programming model for user interface animation. IEEE International Symposium on Visual Languages, Capri, 1997.

[8] William M. Waite and Gerhard Goos. *Compiler Construction*. Springer-Verlag, 1990.